Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

January 20, 2020

Contents

theory *DPLL-W*
imports
Entailment-Definition.Partial-Herbrand-Interpretation
Entailment-Definition.Partial-Annotated-Herbrand-Interpretation
Weidenbach-Book-Base.Wellfounded-More
begin

0.1 Weidenbach's DPLL

0.1.1 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* $\times$ $'v$ *clauses*

abbreviation *trail* :: $'v$ $dpll_W$*-state* $\Rightarrow$ $'v$ $dpll_W$*-ann-lits* **where**
trail $\equiv$ *fst*
abbreviation *clauses* :: $'v$ $dpll_W$*-state* $\Rightarrow$ $'v$ *clauses* **where**
clauses $\equiv$ *snd*

inductive $dpll_W$ :: $'v$ $dpll_W$*-state* $\Rightarrow$ $'v$ $dpll_W$*-state* $\Rightarrow$ *bool* **where**
propagate: *add-mset* L C $\in\#$ *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)

0.1.2 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 dest*: *no-dup-appendD*)
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 no-dup-def*)
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 *undef*: *undefined-lit* M (*lit-of* L)
using *no-dup backtrack-split-list-eq*[*of trail* S, *symmetric*] **unfolding** *extracted* **by** *force*
moreover have *lit-of* $L \notin$ *lits-of-l* M
using *undef* **by** (*auto simp*: *Decided-Propagated-in-iff-in-lits-of-l*)
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* (*unmark* ‘ *c*) = *atm-of* ‘ *lit-of* ‘ *c*
unfolding *atms-of-ms-def* **using** *image-iff* **by** *force*

theorem 2.8.3 page 86 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 L C 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 unmark (trail S)) ∪ set-mset (clauses S)
  have ?I ⊨p add-mset L C 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 L C] 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 unmark 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 unmark a) ∪ set-mset (clauses S))
            and
          atms-of-ms (unmark-l c) ⊆ atms-of-ms (set (map unmark a)
            ∪ set-mset (clauses S))
           using atms-incl cnot
           apply (auto simp: atms-of-def dest!: true-annots-CNot-all-atms-defined; fail)[]
          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#}
          using inS true-clss-cls-plus-CNot true-clss-clss-in-imp-true-clss-cls union-trus-clss-clss
          by blast
      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 unmark* (*L* # *M*)) ∪ *set-mset* (*clauses S*) ⊨*ps CNot D*
using *true-clss-clss-left-right* **by** *fastforce*
then have *set* (*map unmark* (*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*) ⇒ *unmark-l Ls* ∪ *set-mset* (*clauses S*)
⊨*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*)) ≠ []
proof −
have *f1*: ⋀*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*[*of* ‹*L* # *M*›] *decided* **by** (*cases* ‹*get-all-ann-decomposition* (*M'* @ *L* # *M*)›; *cases L*) *auto*
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 (*smt is-decided-ex-Decided lit-of.simps*(*1*) *local.decided old.unit.exhaust*)
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* ∪ *set-mset* (*clauses S*)
∪ *unmark-l M0'* ⊨*ps* {{#− *lit-of L*#}}
using *IL* **by** (*simp add*: *Un-commute Un-left-commute image-Un*)
moreover have *H*: *unmark-l M0* ∪ *set-mset* (*clauses S*)
⊨*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*) ⇒ *unmark-l Ls* ∪ *set-mset* (*clauses S*)
⊨*ps unmark-l seen*
using *true-clss-clss-left-right* **unfolding** *x''* **by** *auto*
}
ultimately show *case x of* (*Ls*, *seen*) ⇒
unmark-l Ls ∪ *set-mset* (*snd* (*?M'*, *clauses S*))
⊨*ps unmark-l seen*
unfolding *snd-conv* **by** *blast*
qed
qed

theorem 2.8.4 page 86 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*) ⊆ *atms-of-mm* (*clauses S*)
shows *set-mset* (*clauses S'*) ∪ {{#*lit-of L*#} |*L*. *is-decided L* ∧ *L* ∈ *set* (*trail S'*)}
⊨*ps unmark* ‘ ⋃(*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 2.8.5 page 86 of Weidenbach's book

lemma *only-propagated-vars-unsat*:
assumes *decided*: ∀ *x* ∈ *set M*. ¬ *is-decided x*
and *DN*: *D* ∈ *N* **and** *D*: *M* ⊨*as CNot D*
and *inv*: *all-decomposition-implies N* (*get-all-ann-decomposition M*)
and *atm-incl*: *atm-of* ‘ *lits-of-l M* ⊆ *atms-of-ms N*
shows *unsatisfiable N*
proof (*rule ccontr*)
assume ¬ *unsatisfiable N*
then obtain *I* **where**
I: *I* ⊨*s N* **and**

cons: *consistent-interp I* **and**
tot: *total-over-m I N*
unfolding *satisfiable-def* **by** *auto*
then have *I-D*: $I \models D$
using *DN* **unfolding** *true-clss-def* **by** *auto*

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

then have *total-over-m I* $(N \cup \textit{unmark}\ `\ (\textit{set } M))$
using *tot* **unfolding** *total-over-m-def* **by** *auto*
then have $I \models s\ \textit{unmark}\ `\ (\textit{set } M)$
using *all-decomposition-implies-propagated-lits-are-implied*[*OF inv*] *cons I*
unfolding *true-clss-clss-def l0* **by** *auto*
then have *IM*: $I \models s\ \textit{unmark-l } M$ **by** *auto*
{
fix *K*
assume $K \in\# D$
then have $-K \in \textit{lits-of-l } M$
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 $\textit{dpll}_W$*-same-clauses*:
assumes $\textit{dpll}_W\ S\ S'$
shows *clauses* $S = \textit{clauses } S'$
using *assms* **by** (*induct rule*: $\textit{dpll}_W$*.induct*, *auto*)

lemma *rtranclp-*$\textit{dpll}_W$*-inv*:
assumes *rtranclp* $\textit{dpll}_W\ S\ S'$
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* $S = \textit{clauses } S'$
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*) $\subseteq$ *atms-of-mm* (*clauses S*) **and**
clauses $S = \textit{clauses } S$ **and**
consistent-interp (*lits-of-l* (*trail S*)) **and**
no-dup (*trail S*) **using** *assms* **by** *auto*
next
case (*step S′ S″*) **note** $\textit{dpll}_W\textit{Star} = \textit{this}(1)$ **and** $\textit{IH} = \textit{this}(3,4,5,6,7)$ **and**
$\textit{dpll}_W = \textit{this}(2)$

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*) $\subseteq$ *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* $\in$ *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* $\subseteq$ *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*) $\longleftrightarrow$
(*trail S* $\models$*asm clauses S* $\vee$ (($\forall$ *L* $\in$ *set* (*trail S*). $\neg$*is-decided L*)
$\wedge$ ($\exists$ *C* $\in$# *clauses S*. *trail S* $\models$*as CNot C*)))

theorem 2.8.7 page 87 of Weidenbach's book

lemma *dpll$_W$-strong-completeness*:
assumes *set M* $\models$*sm N*
and *consistent-interp* (*set M*)
and *distinct M*
and *atm-of* ‘ (*set M*) $\subseteq$ *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* $\models$*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 2.8.6 page 86 of Weidenbach's book

lemma *dpll$_W$-sound*:
assumes
rtranclp dpll$_W$ ([], *N*) (*M*, *N*) **and**
$\forall$ *S*. $\neg$*dpll$_W$* (*M*, *N*) *S*
shows *M* $\models$*asm N* $\longleftrightarrow$ *satisfiable* (*set-mset N*) (**is** *?A* $\longleftrightarrow$ *?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 (∃ *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 (*smt mem-Collect-eq union-single-eq-member*)
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

0.1.3 Termination

definition $dpll_W$*-mes M n* =
map (λ*l*. *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 no-dup-def*)

lemma *Cons-lexn-iff*:
shows ‹(*x* # *xs*, *y* # *ys*) ∈ *lexn R n* ⟷ (*length* (*x* # *xs*) = *n* ∧ *length* (*y* # *ys*) = *n* ∧
((*x*,*y*) ∈ *R* ∨ (*x* = *y* ∧ (*xs*, *ys*) ∈ *lexn R* (*n* − *1*))))›
unfolding *lexn-conv* **apply** (*rule iffI*; *clarify*)
subgoal for *xys xa ya xs′ ys′*
by (*cases xys*) (*auto simp*: *lexn-conv*)
subgoal by (*auto 5 5 simp*: *lexn-conv simp del*: *append-Cons simp*: *append-Cons*[*symmetric*])
done
declare *append-same-lexn*[*simp*] *prepend-same-lexn*[*simp*] *Cons-lexn-iff*[*simp*]
declare *lexn.simps*(*2*)[*simp del*]

lemma $dpll_W$*-card-decrease*:
assumes
dpll: $dpll_W$ *S S′* **and**
[*simp*]: *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 less-than* (*card vars*)
using *assms*
proof (*induct rule*: $dpll_W$*.induct*)
case (*propagate C L S*)
then have *m*: *card vars* − *length* (*trail S*) = *Suc* (*card vars* − *Suc* (*length* (*trail S*)))
by *fastforce*
then show ‹($dpll_W$*-mes* (*trail* (*Propagated C* () # *trail S*, *clauses S*)) (*card vars*),
$dpll_W$*-mes* (*trail S*) (*card vars*)) ∈ *lexn less-than* (*card vars*)›
unfolding $dpll_W$*-mes-def* **by** *auto*
next
case (*decided S L*)
have *m*: *card vars* − *length* (*trail S*) = *Suc* (*card vars* − *Suc* (*length* (*trail S*)))
using *decided.prems*[*simplified*] **using** *Suc-diff-le* **by** *fastforce*
then show ‹($dpll_W$*-mes* (*trail* (*Decided L* # *trail S*, *clauses S*)) (*card vars*),
$dpll_W$*-mes* (*trail S*) (*card vars*)) ∈ *lexn less-than* (*card vars*)›
unfolding $dpll_W$*-mes-def* **by** *auto*
next
case (*backtrack S M′ L M D*)
moreover have *S*: *trail S* = *M′* @ *L* # *M*
using *backtrack.hyps*(*1*) *backtrack-split-list-eq*[*of trail S*] **by** *auto*
ultimately show ‹($dpll_W$*-mes* (*trail* (*Propagated* (− *lit-of L*) () # *M*, *clauses S*)) (*card vars*),
$dpll_W$*-mes* (*trail S*) (*card vars*)) ∈ *lexn less-than* (*card vars*)›
using *backtrack-split-list-eq*[*of trail S*] **unfolding** $dpll_W$*-mes-def* **by** *fastforce*
qed

theorem 2.8.8 page 87 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 less-than*
proof −
have *finite* (*atms-of-mm* (*clauses S*)) **unfolding** *atms-of-ms-def* **by** *auto*
then have *1*: *length* (*trail S*) ≤ *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 less-than* (*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 less-than*
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 less-than*
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 *wf-*$dpll_W$:
wf {(S', S). $dpll_W$*-all-inv* S $\wedge$ $dpll_W$ S S'}
apply (*rule wf-wf-if-measure'*[*OF wf-lex-less*, *of - -*
λS. $dpll_W$*-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 (*trancl-into-trancl* S S' S'')
then have (S', S) $\in$ {*a*. *case a of* (S', S) $\Rightarrow$ $dpll_W$*-all-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$_W$-all-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$_W$-all-inv* $S \wedge$ *dpll$_W$* $S\ S'\}^+$› **by** *auto*
qed
}
then show *?B* $\subseteq$ *?A* **by** *blast*
qed

lemma *wf-dpll$_W$-tranclp*: *wf* $\{(S', S).$ *dpll$_W$-all-inv* $S \wedge$ *dpll$_W$*$^{++}$ $S\ S'\}$
unfolding *dpll$_W$-tranclp-star-commute*[*symmetric*] **by** (*simp add*: *wf-dpll$_W$ wf-trancl*)

lemma *wf-dpll$_W$-plus*:
wf $\{(S', ([], N))|\ S'.$ *dpll$_W$*$^{++}$ $([], N)\ S'\}$ (**is** *wf ?P*)
apply (*rule wf-subset*[*OF wf-dpll$_W$-tranclp*, *of ?P*])
unfolding *dpll$_W$-all-inv-def* **by** *auto*

0.1.4 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*: $\forall s \in$ *atms-of-mm* (*clauses* S). $s \in$ *atm-of* ‘ *lits-of-l* (*trail* S)
proof (*rule ccontr*)
assume ¬ ($\forall s\in$*atms-of-mm* (*clauses* S). $s \in$ *atm-of* ‘ *lits-of-l* (*trail* S))
then obtain L **where**
L-in-atms: $L \in$ *atms-of-mm* (*clauses* S) **and**
L-notin-trail: $L \notin$ *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**
($\exists L\in$*set* (*trail* S). *is-decided* L) ∨ ($\forall C\in\#$*clauses* S. ¬*trail* S ⊨*as CNot* C)
unfolding *conclusive-dpll$_W$-state-def* **by** *auto*
moreover {
assume $\exists L\in$*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** $D \in\#$ *clauses* S **and** ¬ *trail* S ⊨*a* D
using ‹¬ *trail* S ⊨*asm clauses* S› **unfolding** *true-annots-def* **by** *auto*
then have $\forall s\in$*atms-of-ms* $\{D\}$. $s \in$ *atm-of* ‘ *lits-of-l* (*trail* S)
using *vars* **unfolding** *atms-of-ms-def* **by** *auto*
then have *trail* S ⊨*as CNot* D
using *all-variables-defined-not-imply-cnot*[*of* D] ‹¬ *trail* S ⊨*a* D› **by** *auto*

moreover have *is-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* ⊨*as CNot D*› **by** *blast*
}
moreover {
assume *tr*: ∀ *C*∈#*clauses S*. ¬*trail S* ⊨*as CNot C*
obtain *C* **where** *C-in-cls*: *C* ∈# *clauses S* **and** *trC*: ¬ *trail S* ⊨*a C*
using ‹¬ *trail S* ⊨*asm clauses S*› **unfolding** *true-annots-def* **by** *auto*
have ∀ *s*∈*atms-of-ms* {*C*}. *s* ∈ *atm-of* ‘ *lits-of-l* (*trail S*)
using *vars* ‹*C* ∈# *clauses S*› **unfolding** *atms-of-ms-def* **by** *auto*
then have *trail S* ⊨*as CNot C*
by (*meson C-in-cls tr trC all-variables-defined-not-imply-cnot*)
then have *False* **using** *tr C-in-cls* **by** *auto*
}
ultimately show *False* **by** *blast*
qed
qed

lemma $dpll_W$*-conclusive-state-correct*:
assumes $dpll_W{}^{**}$ ([], *N*) (*M*, *N*) **and** *conclusive-*$dpll_W$*-state* (*M*, *N*)
shows *M* ⊨*asm N* ⟷ *satisfiable* (*set-mset N*) (**is** *?A* ⟷ *?B*)
proof
let *?M′*= *lits-of-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 *no-mark*: ∀ *L*∈*set M*. ¬ *is-decided L* ∃ *C* ∈# *N*. *M* ⊨*as CNot C*
using *n assms*(*2*) **unfolding** *conclusive-*$dpll_W$*-state-def* **by** *auto*
moreover obtain *D* **where** *DN*: *D* ∈# *N* **and** *MD*: *M* ⊨*as CNot D* **using** *no-mark* **by** *auto*
ultimately have *unsatisfiable* (*set-mset N*)
using *only-propagated-vars-unsat rtranclp-*$dpll_W$*-all-inv*[*OF assms*(*1*)]
unfolding $dpll_W$*-all-inv-def* **by** *force*
then show *False* **using** ‹*?B*› **by** *blast*
qed
qed

lemma $dpll_W$*-trail-after-step1*:
assumes ‹$dpll_W$ *S T*›
shows
‹∃ *K′ M1 M2′ M2″*.
(*rev* (*trail T*) = *rev* (*trail S*) @ *M2′* ∧ *M2′* ≠ []) ∨
(*rev* (*trail S*) = *M1* @ *Decided* (−*K′*) # *M2′* ∧
rev (*trail T*) = *M1* @ *Propagated K′* () # *M2″* ∧
Suc (*length M1*) ≤ *length* (*trail S*))›
using *assms*
apply (*induction S T rule*: $dpll_W$*.induct*)
subgoal for *L C T*

    by *auto*
  subgoal
    by *auto*
  subgoal for $S\ M'\ L\ M\ D$
    using *backtrack-split-snd-hd-decided*[*of* ‹*trail S*›]
      backtrack-split-list-eq[*of* ‹*trail S*›, *symmetric*]
    apply − **apply** (*rule exI*[*of* - ‹−*lit-of L*›], *rule exI*[*of* - ‹*rev M*›], *rule exI*[*of* - ‹*rev M'*›], *rule exI*[*of* - ‹[]›])
    by (*cases L*)
      auto
  done

lemma *tranclp-dpll*$_W$*-trail-after-step*:
  assumes ‹$dpll_W{}^{++}$ *S T*›
  shows
    ‹$\exists\,K'\ M1\ M2'\ M2''.$
      (*rev* (*trail T*) = *rev* (*trail S*) @ $M2' \wedge M2' \neq []$) ∨
      (*rev* (*trail S*) = *M1* @ *Decided* (−K') # $M2' \wedge$
        rev (*trail T*) = *M1* @ *Propagated* K' () # $M2'' \wedge$ *Suc* (*length M1*) ≤ *length* (*trail S*))›
  using *assms*(*1*)
proof (*induction rule*: *tranclp-induct*)
  case (*base y*)
  then show *?case* **by** (*auto dest*!: *dpll*$_W$*-trail-after-step1*)
next
  case (*step y z*)
  then consider
    (*1*) $M2'$ **where**
      ‹*rev* (*DPLL-W.trail y*) = *rev* (*DPLL-W.trail S*) @ $M2'$› ‹$M2' \neq []$› |
    (*2*) $K'\ M1\ M2'\ M2''$ **where** ‹*rev* (*DPLL-W.trail S*) = *M1* @ *Decided* (− K') # $M2'$›
      ‹*rev* (*DPLL-W.trail y*) = *M1* @ *Propagated* K' () # $M2''$› **and** ‹*Suc* (*length M1*) ≤ *length* (*trail S*)›
    by *blast*
  then show *?case*
  proof *cases*
    case (*1* $M2'$)
    consider
      (*a*) $M2'$ **where**
        ‹*rev* (*DPLL-W.trail z*) = *rev* (*DPLL-W.trail y*) @ $M2'$› ‹$M2' \neq []$› |
      (*b*) $K''\ M1'\ M2''\ M2'''$ **where** ‹*rev* (*DPLL-W.trail y*) = $M1'$ @ *Decided* (− K'') # $M2''$›
        ‹*rev* (*DPLL-W.trail z*) = $M1'$ @ *Propagated* K'' () # $M2'''$› **and**
        ‹*Suc* (*length* $M1'$) ≤ *length* (*trail y*)›
      using *dpll*$_W$*-trail-after-step1*[*OF step*(*2*)]
      by *blast*
    then show *?thesis*
    proof *cases*
      case *a*
      then show *?thesis* **using** *1* **by** *auto*
    next
      case *b*
      have *H*: ‹*rev* (*DPLL-W.trail S*) @ $M2' = M1'$ @ *Decided* (− K'') # $M2'' \Longrightarrow$
          length $M1' \neq$ *length* (*DPLL-W.trail S*) ⟹
          length $M1' <$ *Suc* (*length* (*DPLL-W.trail S*)) ⟹ *rev* (*DPLL-W.trail S*) =
          $M1'$ @ *Decided* (− K'') # *drop* (*Suc* (*length* $M1'$)) (*rev* (*DPLL-W.trail S*))›
        apply (*drule arg-cong*[*of* - - ‹*take* (*length* (*trail S*))›])
        by (*auto simp*: *take-Cons'*)
      show *?thesis* **using** *b 1* **apply** −

```
      apply (rule exI[of - ‹K''›])
      apply (rule exI[of - ‹M1'›])
      apply (rule exI[of - ‹if length (trail S) ≤ length M1' then drop (length (DPLL-W.trail S)) (rev
(DPLL-W.trail z)) else
            drop (Suc (length M1')) (rev (DPLL-W.trail S))›])
      apply (cases ‹length (trail S) < length M1'›)
      subgoal
        apply auto
        by (simp add: append-eq-append-conv-if)
      apply (cases ‹length M1' = length (trail S)›)
      subgoal by auto
      subgoal
        using H
        apply (clarsimp simp: )
        done
      done
    qed
  next
    case (2 K'' M1' M2'' M2''')
    consider
      (a) M2' where
        ‹rev (DPLL-W.trail z) = rev (DPLL-W.trail y) @ M2'› ‹M2' ≠ []› |
      (b) K'' M1' M2'' M2''' where ‹rev (DPLL-W.trail y) = M1' @ Decided (− K'') # M2''›
         ‹rev (DPLL-W.trail z) = M1' @ Propagated K'' () # M2'''› and
        ‹Suc (length M1') ≤ length (trail y)›
      using dpll_W-trail-after-step1[OF step(2)]
      by blast
    then show ?thesis
    proof cases
      case a
      then show ?thesis using 2 by auto
    next
      case (b K''' M1'' M2'''' M2''''')
      have [iff]: ‹M1' @ Propagated K'' () # M2''' = M1'' @ Decided (− K''') # M2'''' ⟷
        (∃ N1''. M1'' = M1' @ Propagated K'' () # N1'' ∧ M2''' = N1'' @ Decided (− K''') # M2'''')›
if ‹length M1' < length M1''›
        using that apply (auto simp: append-eq-append-conv-if)
        by (metis (no-types, lifting) Cons-eq-append-conv append-take-drop-id drop-eq-Nil leD)
      have [iff]: ‹M1' @ Propagated K'' () # M2''' = M1'' @ Decided (− K''') # M2'''' ⟷
        (∃ N1''. M1' = M1'' @ Decided (− K''') # N1'' ∧ M2'''' = N1'' @ Propagated K'' () # M2''')›
if ‹¬length M1' < length M1''›
        using that apply (auto simp: append-eq-append-conv-if)
      by (metis (no-types, lifting) Cons-eq-append-conv append-take-drop-id drop-eq-Nil le-eq-less-or-eq)

      show ?thesis using b 2 apply −
        apply (rule exI[of - ‹if length M1' < length M1'' then K'' else K'''›])
        apply (rule exI[of - ‹if length M1' < length M1'' then M1' else M1''›])
        apply (cases ‹length (trail S) < min (length M1') (length M1'')›)
        subgoal
          by auto
        apply (cases ‹min (length M1') (length M1'') = length (trail S)›)
        subgoal by auto
        subgoal
          by (auto simp: )
        done
     qed
```

  qed
qed

This theorem is an important (although rather obvious) property: the model induced by trails are not repeated.

lemma *tranclp-dpll$_W$-no-dup-trail*:
  assumes ‹$dpll_W{}^{++}$ *S T*› **and** ‹*dpll$_W$-all-inv S*›
  shows ‹*set* (*trail S*) $\neq$ *set* (*trail T*)›
proof −
  have [*simp*]: ‹$A = B \cup A \longleftrightarrow B \subseteq A$› **for** *A B*
    by *auto*
  have [*simp*]: ‹*rev* (*trail U*) = *xs* $\longleftrightarrow$*trail U* = *rev xs*› **for** *xs U*
    by *auto*
  have ‹*dpll$_W$-all-inv T*›
    by (*metis assms*(*1*) *assms*(*2*) *reflclp-tranclp rtranclp-dpll$_W$-all-inv sup2CI*)
  then have *n-d*: ‹*no-dup* (*trail S*)› ‹*no-dup* (*trail T*)›
    using *assms* **unfolding** *dpll$_W$-all-inv-def* **by** (*auto dest*: *no-dup-imp-distinct*)
  have [*simp*]: ‹*no-dup* (*rev M2′* @ *DPLL-W.trail S*) $\Longrightarrow$
          dpll$_W$-all-inv S $\Longrightarrow$
          set M2′ $\subseteq$ *set* (*DPLL-W.trail S*) $\longleftrightarrow$ *M2′* = []› **for** *M2′*
    by (*cases M2′ rule*: *rev-cases*)
      (*auto simp*: *undefined-notin*)
  show *?thesis*
    using *n-d tranclp-dpll$_W$-trail-after-step*[*OF assms*(*1*)] *assms*(*2*) **apply** *auto*
    by (*metis* (*no-types*, *lifting*) *Un-insert-right insertI1 list.simps*(*15*) *lit-of.simps*(*1*,*2*)
      n-d(*1*) *no-dup-cannot-not-lit-and-uminus set-append set-rev*)
qed

end
theory *CDCL-W-Level*
imports
  Entailment-Definition.Partial-Annotated-Herbrand-Interpretation
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.

definition *count-decided* :: (*′v*, *′b*, *′m*) *annotated-lit list* ⇒ *nat* **where**
count-decided l = *length* (*filter is-decided l*)

definition *get-level* :: (*′v*, *′m*) *ann-lits* ⇒ *′v literal* ⇒ *nat* **where**
get-level S L = *length* (*filter is-decided* (*dropWhile* (λS. *atm-of* (*lit-of S*) $\neq$ *atm-of L*) *S*))

lemma *get-level-uminus*[*simp*]: ‹*get-level M* (−*L*) = *get-level M L*›
  by (*auto simp*: *get-level-def*)

lemma *get-level-Neg-Pos*: ‹*get-level M* (*Neg L*) = *get-level M* (*Pos L*)›
  unfolding *get-level-def* **by** *auto*

lemma *count-decided-0-iff*:
  ‹*count-decided M* = *0* $\longleftrightarrow$ ($\forall L \in$ *set M*. ¬*is-decided L*)›
  by (*auto simp*: *count-decided-def filter-empty-conv*)

lemma
shows
count-decided-nil[*simp*]: ‹*count-decided* [] = *0*› **and**
count-decided-cons[*simp*]:
‹*count-decided* (*a* # *M*) = (*if is-decided a then Suc* (*count-decided M*) *else count-decided M*)› **and**
count-decided-append[*simp*]:
‹*count-decided* (*M* @ *M′*) = *count-decided M* + *count-decided M′*›
by (*auto simp*: *count-decided-def*)

lemma *atm-of-notin-get-level-eq-0*[*simp*]:
assumes *undefined-lit M L*
shows *get-level M L* = *0*
using *assms* **by** (*induct M rule*: *ann-lit-list-induct*) (*auto simp*: *get-level-def defined-lit-map*)

lemma *get-level-ge-0-atm-of-in*:
assumes *get-level M L* > *n*
shows *atm-of L* ∈ *atm-of* ‘ *lits-of-l M*
using *atm-of-notin-get-level-eq-0*[*of M L*] *assms* **unfolding** *defined-lit-map*
by (*auto simp*: *lits-of-def simp del*: *atm-of-notin-get-level-eq-0*)

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-level-skip*[*simp*]:
assumes *undefined-lit M L*
shows *get-level* (*M* @ *M′*) *L* = *get-level M′ L*
using *assms* **by** (*induct M rule*: *ann-lit-list-induct*) (*auto simp*: *get-level-def defined-lit-map*)

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

lemma *get-level-skip-end*[*simp*]:
assumes *defined-lit M L*
shows *get-level* (*M* @ *M′*) *L* = *get-level M L* + *count-decided M′*
using *assms* **by** (*induct M′ rule*: *ann-lit-list-induct*)
(*auto simp*: *lits-of-def get-level-def count-decided-def defined-lit-map*)

lemma *get-level-skip-beginning*[*simp*]:
assumes *atm-of L′* ≠ *atm-of* (*lit-of K*)
shows *get-level* (*K* # *M*) *L′* = *get-level M L′*
using *assms* **by** (*auto simp*: *get-level-def*)

lemma *get-level-take-beginning*[*simp*]:
assumes *atm-of L′* = *atm-of* (*lit-of K*)
shows *get-level* (*K* # *M*) *L′* = *count-decided* (*K* # *M*)
using *assms* **by** (*auto simp*: *get-level-def count-decided-def*)

lemma *get-level-cons-if*:
‹*get-level* (*K* # *M*) *L′* =
(*if atm-of L′* = *atm-of* (*lit-of K*) *then count-decided* (*K* # *M*) *else get-level M L′*)›
by *auto*

lemma *get-level-skip-beginning-not-decided*[*simp*]:
assumes *undefined-lit S L*
and ∀ *s*∈*set S*. ¬*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 ∈ atm-of ‘ lits-of-l M*)
 apply (*auto simp*: *image-iff lits-of-def filter-empty-conv count-decided-def defined-lit-map dest*: *set-dropWhileD*)
done

lemma *get-level-skip-all-not-decided*[*simp*]:
fixes *M*
assumes ∀ *m*∈*set M*. ¬ *is-decided m*
shows *get-level M L* = *0*
using *assms* **by** (*auto simp*: *filter-empty-conv get-level-def dest*: *set-dropWhileD*)

the {#*0*::′*a*#} is there to ensures that the set is not empty.

definition *get-maximum-level* :: (′*a*, ′*b*) *ann-lits* ⇒ ′*a clause* ⇒ *nat*
where
get-maximum-level M D = *Max-mset* ({#*0*#} + *image-mset* (*get-level M*) *D*)

lemma *get-maximum-level-ge-get-level*:
L ∈# *D* ⟹ *get-maximum-level M D* ≥ *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 ≠ {#} ⟹ ∃ *L*∈# *D*. *get-level M L* = *get-maximum-level M D*
unfolding *get-maximum-level-def*
apply (*induct D*)
 apply *simp*
by (*rename-tac x D*, *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-add-mset*:
get-maximum-level M (*add-mset L D*) = *max* (*get-level M L*) (*get-maximum-level M D*)
unfolding *get-maximum-level-def* **by** *simp*

lemma *get-level-append-if*:
‹*get-level* (*M* @ *M*′) *L* = (*if defined-lit M L then get-level M L* + *count-decided M*′
else get-level M′ *L*)›
by (*auto*)

Do mot activate as [simp] rules. It breaks everything.

lemma *get-maximum-level-single*:
‹*get-maximum-level M* {#*x*#} = *get-level M x*›
by (*auto simp*: *get-maximum-level-add-mset*)

lemma *get-maximum-level-plus*:
get-maximum-level M (*D* + *D*′) = *max* (*get-maximum-level M D*) (*get-maximum-level M D*′)
by (*induction D*) (*simp-all add*: *get-maximum-level-add-mset*)

lemma *get-maximum-level-cong*:
assumes ‹∀ *L* ∈# *D*. *get-level M L* = *get-level M*′ *L*›
shows ‹*get-maximum-level M D* = *get-maximum-level M*′ *D*›

using *assms* **by** (*induction D*) (*auto simp*: *get-maximum-level-add-mset*)

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

lemma *get-maximum-level-skip-first*[*simp*]:
assumes *atm-of* (*lit-of K*) ∉ *atms-of D*
shows *get-maximum-level* (*K* # *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 lit-of.simps*(*2*)
multiset.map-cong0)

lemma *get-maximum-level-skip-beginning*:
assumes *DH*: ∀ *x* ∈# *D*. *undefined-lit c x*
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*; *fail*)
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-union-mset*:
get-maximum-level M (*A* ∪# *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*: *count-decided-def rev-filter*[*symmetric*])

lemma *count-decided-ge-get-level*:
count-decided M ≥ *get-level M L*
by (*induct M rule*: *ann-lit-list-induct*)
(*auto simp add*: *count-decided-def le-max-iff-disj get-level-def*)

lemma *count-decided-ge-get-maximum-level*:
count-decided M ≥ *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*)

lemma *get-level-last-decided-ge*:
‹*defined-lit* (*c* @ [*Decided K*]) *L′* ⟹ *0* < *get-level* (*c* @ [*Decided K*]) *L′*›

by (*induction c*) (*auto simp*: *defined-lit-cons get-level-cons-if*)

lemma *get-maximum-level-mono*:
⟨$D \subseteq\# D' \Longrightarrow$ *get-maximum-level M D* $\leq$ *get-maximum-level M D'*⟩
unfolding *get-maximum-level-def* **by** *auto*

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*

lemma *get-all-mark-of-propagated-tl-proped*:
⟨$M \neq [] \Longrightarrow$ *is-proped* (*hd M*) $\Longrightarrow$ *get-all-mark-of-propagated* (*tl M*) = *tl* (*get-all-mark-of-propagated M*)⟩
by (*induction M 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*

Before I try yet another time to prove that I can remove the assumption *no-dup M*: It does not work. The problem is that *get-level M K* = *Suc i* peaks the first occurrence of the literal *K*. This is for example an issue for the trail *replicate n* (*Decided K*). An explicit counter-example is below.

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 M K*] **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 simp*: *get-level-def defined-lit-map*)
next
case *False*
show *?thesis*
apply (*rule exI*[*of - []*])
apply (*rule exI*[*of - L*])
apply (*rule exI*[*of - M*])
using *False i* **by** (*auto simp*: *get-level-def count-decided-def*)
qed
next
case (*Propagated L mark' M*) **note** *i* = *this*(*2*) **and** *IH* = *this*(*1*) **and** *n-d* = *this*(*3*)
then obtain *c K c'* **where**
M: *M* = *c* @ *Decided K # c'* **and** *lev-K*: *get-level M K* = *Suc i*
by (*auto simp*: *count-decided-def*)
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 get-level-def defined-lit-map*)
qed
qed

The counter-example if the assumption *no-dup M*.

lemma
fixes *K*
defines ‹*M* ≡ *replicate 3* (*Decided K*)›
defines ‹*i* ≡ *1*›
assumes ‹*i* < *count-decided M* ⟷ (∃ *c K c'*. *M* = *c* @ *Decided K # c'* ∧ *get-level M K* = *Suc i*)›
shows *False*
using *assms*(*3*−) **unfolding** *M-def i-def numeral-3-eq-3*
by (*auto simp*: *Cons-eq-append-conv*)

lemma *Suc-count-decided-gt-get-level*:
‹*get-level M L* < *Suc* (*count-decided M*)›
by (*induction M rule*: *ann-lit-list-induct*) (*auto simp*: *get-level-cons-if*)

lemma *get-level-neq-Suc-count-decided*[*simp*]:
‹*get-level M L* ≠ *Suc* (*count-decided M*)›
using *Suc-count-decided-gt-get-level*[*of M L*] **by** *auto*

lemma *length-get-all-ann-decomposition*: ‹*length* (*get-all-ann-decomposition M*) = *1*+*count-decided M*›
by (*induction M rule*: *ann-lit-list-induct*) *auto*

lemma *get-maximum-level-remove-non-max-lvl*:
‹*get-level M a* < *get-maximum-level M D* ⟹
get-maximum-level M (*remove1-mset a D*) = *get-maximum-level M D*›
by (*cases* ‹*a* ∈# *D*›)
(*auto dest*!: *multi-member-split simp*: *get-maximum-level-add-mset*)

lemma *exists-lit-max-level-in-negate-ann-lits*:

⟨*negate-ann-lits M* ≠ {#} ⟹ ∃ *L*∈#*negate-ann-lits M*. *get-level M L* = *count-decided M*⟩
by (*cases* ⟨*M*⟩) (*auto simp*: *negate-ann-lits-def*)

end
theory *CDCL-W*
imports *CDCL-W-Level Weidenbach-Book-Base.Wellfounded-More*
begin

Chapter 1

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, based on the book.
- `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). This is useful for the refinement from NOT.
- `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. There are two different refinement: on from NOT's to Weidenbach's CDCL and another to W's CDCL with 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, since the conflict and analyse are merged.
- `CDCL_W_Restart.thy` adds restart and forget while restarting. It is built on the top of `CDCL_W_Merge.thy`.

1.1 Weidenbach's CDCL with Multisets

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

1.1.1 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
    state :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* × *'v clauses* × *'v clauses* × *'v clause option* ×
      'b **and**
    trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **and**
    init-clss :: *'st* ⇒ *'v clauses* **and**
    learned-clss :: *'st* ⇒ *'v clauses* **and**
    conflicting :: *'st* ⇒ *'v clause option* **and**

    cons-trail :: (*'v*, *'v clause*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
    tl-trail :: *'st* ⇒ *'st* **and**

    add-learned-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
    remove-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
    update-conflicting :: *'v clause option* ⇒ *'st* ⇒ *'st* **and**

    init-state :: *'v clauses* ⇒ *'st*
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* :: ‹*'a literal* ⇒ *'a clause* ⇒ *'a clause* ⇒ *'a clause*› **where**
resolve-cls L D' E ≡ *remove1-mset* (−*L*) *D'* ∪# *remove1-mset L E*

abbreviation *state-butlast* :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* × *'v clauses* × *'v clauses*
  × *'v clause option* **where**
state-butlast S ≡ (*trail S*, *init-clss S*, *learned-clss S*, *conflicting S*)

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

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 conflicting clause (if any has been found so far).

Contrary to the original version, we have removed the maximum level of the trail, since the information is redundant and required an additional invariant.

There are two different clause representation: one for the conflicting clause (*'v clause*, standing for conflicting clause) and one for the initial and learned clauses (*'v clause*, standing for clause).

The representation of the clauses annotating literals in the trail is slightly different: being able to convert it to *'v 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 $state_W$*-no-state* =
  $state_W$*-ops*
    state
    — functions about the state:
      — getter:
    trail init-clss learned-clss conflicting
      — setter:
    cons-trail tl-trail add-learned-cls remove-cls
    update-conflicting

      — Some specific states:
    init-state
  for
    state-eq :: $'st \Rightarrow 'st \Rightarrow bool$ (**infix** $\sim$ *50*) **and**
    state :: $'st \Rightarrow ('v, 'v\ clause)\ ann\text{-}lits \times 'v\ clauses \times 'v\ clauses \times 'v\ clause\ option \times$
      $'b$ **and**
    trail :: $'st \Rightarrow ('v, 'v\ clause)\ ann\text{-}lits$ **and**
    init-clss :: $'st \Rightarrow 'v\ clauses$ **and**
    learned-clss :: $'st \Rightarrow 'v\ clauses$ **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-conflicting :: $'v\ clause\ option \Rightarrow 'st \Rightarrow 'st$ **and**

    init-state :: $'v\ clauses \Rightarrow 'st$ +
  assumes
    state-eq-ref[*simp*, *intro*]: ‹$S \sim S$› **and**
    state-eq-sym: ‹$S \sim T \longleftrightarrow T \sim S$› **and**
    state-eq-trans: ‹$S \sim T \Longrightarrow T \sim U' \Longrightarrow S \sim U'$› **and**
    state-eq-state: ‹$S \sim T \Longrightarrow state\ S = state\ T$› **and**

    cons-trail:
      $\bigwedge S'.\ state\ st = (M, S') \Longrightarrow$
        $state\ (cons\text{-}trail\ L\ st) = (L \# M, S')$ **and**

    tl-trail:
      $\bigwedge S'.\ state\ st = (M, S') \Longrightarrow state\ (tl\text{-}trail\ st) = (tl\ M, S')$ **and**

    remove-cls:
      $\bigwedge S'.\ state\ st = (M, N, U, S') \Longrightarrow$
        $state\ (remove\text{-}cls\ C\ st) =$
          $(M, removeAll\text{-}mset\ C\ N, removeAll\text{-}mset\ C\ U, S')$ **and**

    add-learned-cls:
      $\bigwedge S'.\ state\ st = (M, N, U, S') \Longrightarrow$
        $state\ (add\text{-}learned\text{-}cls\ C\ st) = (M, N, \{\#C\#\} + U, S')$ **and**

    update-conflicting:

$\bigwedge S'.\ state\ st = (M,\ N,\ U,\ D,\ S') \Longrightarrow$
$state\ (update\text{-}conflicting\ E\ st) = (M,\ N,\ U,\ E,\ S')$ **and**

init-state:
$state\text{-}butlast\ (init\text{-}state\ N) = ([],\ N,\ \{\#\},\ None)$ **and**

cons-trail-state-eq:
$\langle S \sim S' \Longrightarrow cons\text{-}trail\ L\ S \sim cons\text{-}trail\ L\ S'\rangle$ **and**

tl-trail-state-eq:
$\langle S \sim S' \Longrightarrow tl\text{-}trail\ S \sim tl\text{-}trail\ S'\rangle$ **and**

add-learned-cls-state-eq:
$\langle S \sim S' \Longrightarrow add\text{-}learned\text{-}cls\ C\ S \sim add\text{-}learned\text{-}cls\ C\ S'\rangle$ **and**

remove-cls-state-eq:
$\langle S \sim S' \Longrightarrow remove\text{-}cls\ C\ S \sim remove\text{-}cls\ C\ S'\rangle$ **and**

update-conflicting-state-eq:
$\langle S \sim S' \Longrightarrow update\text{-}conflicting\ D\ S \sim update\text{-}conflicting\ D\ S'\rangle$ **and**

tl-trail-add-learned-cls-commute:
$\langle tl\text{-}trail\ (add\text{-}learned\text{-}cls\ C\ T) \sim add\text{-}learned\text{-}cls\ C\ (tl\text{-}trail\ T)\rangle$ **and**
tl-trail-update-conflicting:
$\langle tl\text{-}trail\ (update\text{-}conflicting\ D\ T) \sim update\text{-}conflicting\ D\ (tl\text{-}trail\ T)\rangle$ **and**

update-conflicting-update-conflicting:
$\langle \bigwedge D\ D'\ S\ S'.\ S \sim S' \Longrightarrow$
$update\text{-}conflicting\ D\ (update\text{-}conflicting\ D'\ S) \sim update\text{-}conflicting\ D\ S'\rangle$ **and**
update-conflicting-itself:
$\langle \bigwedge D\ S'.\ conflicting\ S' = D \Longrightarrow update\text{-}conflicting\ D\ S' \sim S'\rangle$

locale $state_W$ =
$state_W$*-no-state*
state-eq state
— functions about the state:
— getter:
trail init-clss learned-clss conflicting
— setter:
cons-trail tl-trail add-learned-cls remove-cls
update-conflicting

— Some specific states:
init-state
for
$state\text{-}eq :: 'st \Rightarrow 'st \Rightarrow bool$ (**infix** $\sim$ *50*) **and**
$state :: 'st \Rightarrow ('v,\ 'v\ clause)\ ann\text{-}lits \times 'v\ clauses \times 'v\ clauses \times 'v\ clause\ option \times$
$'b$ **and**
$trail :: 'st \Rightarrow ('v,\ 'v\ clause)\ ann\text{-}lits$ **and**
$init\text{-}clss :: 'st \Rightarrow 'v\ clauses$ **and**
$learned\text{-}clss :: 'st \Rightarrow 'v\ clauses$ **and**
$conflicting :: 'st \Rightarrow 'v\ clause\ option$ **and**

$cons\text{-}trail :: ('v,\ 'v\ clause)\ ann\text{-}lit \Rightarrow 'st \Rightarrow 'st$ **and**
$tl\text{-}trail :: 'st \Rightarrow 'st$ **and**
$add\text{-}learned\text{-}cls :: 'v\ clause \Rightarrow 'st \Rightarrow 'st$ **and**

remove-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
update-conflicting :: *'v clause option* ⇒ *'st* ⇒ *'st* **and**

init-state :: *'v clauses* ⇒ *'st* +
assumes
state-prop[*simp*]:
‹*state S* = (*trail S*, *init-clss S*, *learned-clss S*, *conflicting S*, *additional-info S*)›
begin

lemma
trail-cons-trail[*simp*]:
trail (*cons-trail L st*) = *L* # *trail st* **and**
trail-tl-trail[*simp*]: *trail* (*tl-trail st*) = *tl* (*trail st*) **and**
trail-add-learned-cls[*simp*]:
trail (*add-learned-cls C st*) = *trail st* **and**
trail-remove-cls[*simp*]:
trail (*remove-cls C st*) = *trail st* **and**
trail-update-conflicting[*simp*]: *trail* (*update-conflicting E st*) = *trail st* **and**

init-clss-cons-trail[*simp*]:
init-clss (*cons-trail M st*) = *init-clss st*
and
init-clss-tl-trail[*simp*]:
init-clss (*tl-trail st*) = *init-clss st* **and**
init-clss-add-learned-cls[*simp*]:
init-clss (*add-learned-cls C st*) = *init-clss st* **and**
init-clss-remove-cls[*simp*]:
init-clss (*remove-cls C st*) = *removeAll-mset C* (*init-clss st*) **and**
init-clss-update-conflicting[*simp*]:
init-clss (*update-conflicting E st*) = *init-clss st* **and**

learned-clss-cons-trail[*simp*]:
learned-clss (*cons-trail M st*) = *learned-clss st* **and**
learned-clss-tl-trail[*simp*]:
learned-clss (*tl-trail st*) = *learned-clss st* **and**
learned-clss-add-learned-cls[*simp*]:
learned-clss (*add-learned-cls C st*) = {#*C*#} + *learned-clss st* **and**
learned-clss-remove-cls[*simp*]:
learned-clss (*remove-cls C st*) = *removeAll-mset C* (*learned-clss st*) **and**
learned-clss-update-conflicting[*simp*]:
learned-clss (*update-conflicting E st*) = *learned-clss st* **and**

conflicting-cons-trail[*simp*]:
conflicting (*cons-trail M st*) = *conflicting st* **and**
conflicting-tl-trail[*simp*]:
conflicting (*tl-trail st*) = *conflicting st* **and**
conflicting-add-learned-cls[*simp*]:
conflicting (*add-learned-cls C st*) = *conflicting st*
and
conflicting-remove-cls[*simp*]:
conflicting (*remove-cls C st*) = *conflicting st* **and**
conflicting-update-conflicting[*simp*]:
conflicting (*update-conflicting E st*) = *E* **and**

init-state-trail[*simp*]: *trail* (*init-state N*) = [] **and**
init-state-clss[*simp*]: *init-clss* (*init-state N*) = *N* **and**

init-state-learned-clss[*simp*]: *learned-clss* (*init-state* N) = {#} **and**
init-state-conflicting[*simp*]: *conflicting* (*init-state* N) = *None*
using *cons-trail*[*of st*] *tl-trail*[*of st*] *add-learned-cls*[*of st* - - - - C]
update-conflicting[*of st* - - - - - -]
remove-cls[*of st* - - - - C]
init-state[*of* N]
by *auto*

lemma
shows
clauses-cons-trail[*simp*]:
clauses (*cons-trail* M S) = *clauses* S **and**

clss-tl-trail[*simp*]: *clauses* (*tl-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-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-init-state[*simp*]: *clauses* (*init-state* N) = N
by (*auto simp*: *ac-simps replicate-mset-plus clauses-def intro*: *multiset-eqI*)

lemma *state-eq-trans'*: ‹$S \sim S' \Longrightarrow T \sim S' \Longrightarrow T \sim S$›
by (*meson state-eq-trans state-eq-sym*)

abbreviation *backtrack-lvl* :: *'st* ⇒ *nat* **where**
‹*backtrack-lvl* S ≡ *count-decided* (*trail* S)›

named-theorems *state-simp* ‹*contains all theorems of the form* @{*term* ‹$S \sim T \Longrightarrow P\ S = P\ T$›}.
These theorems can cause a signefecant blow−up of the simp−space›

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

lemma *state-eq-conflicting-None*:
$S \sim T \Longrightarrow$ *conflicting* T = *None* $\Longrightarrow$ *conflicting* S = *None*
using *state-eq-state* **unfolding** *clauses-def* **by** *auto*

We combine all simplification rules about ($\sim$) 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.

declare *state-simp*[*simp*]

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 *reduce-trail-to-induct*:
  assumes
    ‹⋀*F S*. *length* (*trail S*) = *length F* ⟹ *P F S*› **and**
    ‹⋀*F S*. *trail S* = [] ⟹ *P F S*› **and**
    ‹⋀*F S*. *length* (*trail S*) ≠ *length F* ⟹ *trail S* ≠ [] ⟹ *P F* (*tl-trail S*) ⟹ *P F S*›
  shows
    ‹*P F S*›
  apply (*induction rule*: *reduce-trail-to.induct*)
  subgoal for *F S* **using** *assms*
    by (*cases* ‹*length* (*trail S*) = *length F*›; *cases* ‹*trail S* = []›) *auto*
  done

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 *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*)

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-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-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*) ≥ *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*; *fail*)
  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*) ∈ *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 *reduce-trail-to-state-eq*:
  ‹*S* ∼ *S′* ⟹ *length M* = *length M′* ⟹ *reduce-trail-to M S* ∼ *reduce-trail-to M′ S′*›
  apply (*induction M S arbitrary*: *M′ S′ rule*: *reduce-trail-to-induct*)
   apply ((*auto*;*fail*)+)[*2*]
  by (*simp add*: *reduce-trail-to-length-ne tl-trail-state-eq*)

lemma *conflicting-cons-trail-conflicting*[*iff*]:
  conflicting (*cons-trail L S*) = *None* ⟷ *conflicting S* = *None*
  using *conflicting-cons-trail*[*of L S*] *map-option-is-None* **by** *fastforce+*

lemma *conflicting-add-learned-cls-conflicting*[*iff*]:
  conflicting (*add-learned-cls C S*) = *None* ⟷ *conflicting S* = *None*
  by *fastforce+*

lemma *reduce-trail-to-compow-tl-trail-le*:
  assumes ‹*length M* < *length* (*trail M′*)›
  shows ‹*reduce-trail-to M M′* = (*tl-trail*^^(*length* (*trail M′*) − *length M*)) *M′*›
proof −
  have [*simp*]: ‹(∀ *ka*. *k* ≠ *Suc ka*) ⟷ *k* = *0*› **for** *k*
    by (*cases k*) *auto*
  show *?thesis*
    using *assms*
    apply (*induction M*≡*M S*≡*M′ arbitrary*: *M M′ rule*: *reduce-trail-to.induct*)
    subgoal for *F S*
      by (*subst reduce-trail-to.simps*; *cases* ‹*length F* < *length* (*trail S*) − *Suc 0*›)
        (*auto simp*: *less-iff-Suc-add funpow-swap1*)
    done

qed

lemma *reduce-trail-to-compow-tl-trail-eq*:
‹*length* M = *length* (*trail* M') $\Longrightarrow$ *reduce-trail-to* M M' = (*tl-trail*^^(*length* (*trail* M') − *length* M)) M'›
by *auto*

lemma *reduce-trail-to-compow-tl-trail*:
‹*length* $M \leq$ *length* (*trail* M') $\Longrightarrow$ *reduce-trail-to* M M' = (*tl-trail*^^(*length* (*trail* M') − *length* M)) M'›
using *reduce-trail-to-compow-tl-trail-eq*[*of* M M']
reduce-trail-to-compow-tl-trail-le[*of* M M']
by (*cases* ‹*length* M < *length* (*trail* M')›) *auto*

lemma *tl-trail-reduce-trail-to-cons*:
‹*length* (L # M) < *length* (*trail* M') $\Longrightarrow$ *tl-trail* (*reduce-trail-to* (L # M) M') = *reduce-trail-to* M M'›
by (*auto simp*: *reduce-trail-to-compow-tl-trail-le funpow-swap1*
reduce-trail-to-compow-tl-trail-eq less-iff-Suc-add)

lemma *compow-tl-trail-add-learned-cls-swap*:
‹(*tl-trail* ^^ n) (*add-learned-cls* D S) ∼ *add-learned-cls* D ((*tl-trail* ^^ n) S)›
by (*induction* n)
(*auto intro*: *tl-trail-add-learned-cls-commute state-eq-trans*
tl-trail-state-eq)

lemma *reduce-trail-to-add-learned-cls-state-eq*:
‹*length* $M \leq$ *length* (*trail* S) $\Longrightarrow$
reduce-trail-to M (*add-learned-cls* D S) ∼ *add-learned-cls* D (*reduce-trail-to* M S)›
by (*cases* ‹*length* M < *length* (*trail* S)›)
(*auto simp*: *compow-tl-trail-add-learned-cls-swap reduce-trail-to-compow-tl-trail-le*
reduce-trail-to-compow-tl-trail-eq)

lemma *compow-tl-trail-update-conflicting-swap*:
‹(*tl-trail* ^^ n) (*update-conflicting* D S) ∼ *update-conflicting* D ((*tl-trail* ^^ n) S)›
by (*induction* n)
(*auto intro*: *tl-trail-add-learned-cls-commute state-eq-trans*
tl-trail-state-eq tl-trail-update-conflicting)

lemma *reduce-trail-to-update-conflicting-state-eq*:
‹*length* $M \leq$ *length* (*trail* S) $\Longrightarrow$
reduce-trail-to M (*update-conflicting* D S) ∼ *update-conflicting* D (*reduce-trail-to* M S)›
by (*cases* ‹*length* M < *length* (*trail* S)›)
(*auto simp*: *compow-tl-trail-add-learned-cls-swap reduce-trail-to-compow-tl-trail-le*
reduce-trail-to-compow-tl-trail-eq compow-tl-trail-update-conflicting-swap)

lemma
additional-info-cons-trail[*simp*]:
‹*additional-info* (*cons-trail* L S) = *additional-info* S› **and**
additional-info-tl-trail[*simp*]:
additional-info (*tl-trail* S) = *additional-info* S **and**
additional-info-add-learned-cls-unfolded:
additional-info (*add-learned-cls* U S) = *additional-info* S **and**
additional-info-update-conflicting[*simp*]:
additional-info (*update-conflicting* D S) = *additional-info* S **and**
additional-info-remove-cls[*simp*]:
additional-info (*remove-cls* C S) = *additional-info* S **and**

additional-info-add-learned-cls[*simp*]:
additional-info (*add-learned-cls C S*) = *additional-info S*
unfolding *additional-info-def*
using *tl-trail*[*of S*] *cons-trail*[*of S*] *add-learned-cls*[*of S*]
update-conflicting[*of S*] *remove-cls*[*of S*]
by (*cases* ‹*state S*›; *auto*; *fail*)+

lemma *additional-info-reduce-trail-to*[*simp*]:
‹*additional-info* (*reduce-trail-to F S*) = *additional-info S*›
by (*induction F S rule*: *reduce-trail-to.induct*)
(*metis additional-info-tl-trail reduce-trail-to.simps*)

lemma *reduce-trail-to*:
state (*reduce-trail-to F S*) =
((*if length* (*trail S*) ≥ *length F*
then drop (*length* (*trail S*) − *length F*) (*trail S*)
else []), *init-clss S*, *learned-clss S*, *conflicting S*, *additional-info S*)
proof (*induction F S rule*: *reduce-trail-to.induct*)
case (*1 F S*) **note** *IH* = *this*
show *?case*
proof (*cases trail S*)
case *Nil*
then show *?thesis* **using** *IH* **by** (*subst state-prop*) *auto*
next
case (*Cons L M*)
show *?thesis*
proof (*cases Suc* (*length M*) > *length F*)
case *True*
then have *Suc* (*length M*) − *length F* = *Suc* (*length M* − *length F*)
by *auto*
then show *?thesis*
using *Cons True reduce-trail-to-length-ne*[*of S F*] *IH* **by** (*auto simp del*: *state-prop*)
next
case *False*
then show *?thesis*
using *IH reduce-trail-to-length-ne*[*of S F*] **apply** (*subst state-prop*)
by (*simp add*: *trail-reduce-trail-to-drop*)
qed
qed
qed

end — end of $state_W$ locale

1.1.2 CDCL Rules

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

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

update-conflicting

— get state:
init-state
for
state-eq :: $'st \Rightarrow 'st \Rightarrow bool$ (**infix** $\sim$ *50*) **and**
state :: $'st \Rightarrow ('v, 'v\ clause)\ ann\text{-}lits \times 'v\ clauses \times 'v\ clauses \times 'v\ clause\ option \times$
$'b$ **and**
trail :: $'st \Rightarrow ('v, 'v\ clause)\ ann\text{-}lits$ **and**
init-clss :: $'st \Rightarrow 'v\ clauses$ **and**
learned-clss :: $'st \Rightarrow 'v\ clauses$ **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-conflicting :: $'v\ clause\ option \Rightarrow 'st \Rightarrow 'st$ **and**

init-state :: $'v\ clauses \Rightarrow 'st$
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-lit (*trail* S) $L \Longrightarrow$
$T \sim$ *cons-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-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* (*add-mset* L 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) (*add-mset* L D') $\Longrightarrow$
get-maximum-level (*trail* S) $D' \equiv i \Longrightarrow$
get-level (*trail* S) $K = i + 1 \Longrightarrow$
$D' \subseteq\# D \Longrightarrow$
clauses $S \models pm$ *add-mset* L $D' \Longrightarrow$
$T \sim$ *cons-trail* (*Propagated* L (*add-mset* L D'))
(*reduce-trail-to* *M1*
(*add-learned-cls* (*add-mset* L D')

$(update\text{-}conflicting\ None\ S))) \Longrightarrow$
$backtrack\ S\ T$

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

Here is the normal backtrack rule from Weidenbach's book:

inductive *simple-backtrack* :: $'st \Rightarrow 'st \Rightarrow bool$ **for** S :: $'st$ **where**
simple-backtrack-rule:
$conflicting\ S = Some\ (add\text{-}mset\ L\ D) \Longrightarrow$
$(Decided\ K\ \#\ M1,\ M2) \in set\ (get\text{-}all\text{-}ann\text{-}decomposition\ (trail\ S)) \Longrightarrow$
$get\text{-}level\ (trail\ S)\ L = backtrack\text{-}lvl\ S \Longrightarrow$
$get\text{-}level\ (trail\ S)\ L = get\text{-}maximum\text{-}level\ (trail\ S)\ (add\text{-}mset\ L\ D) \Longrightarrow$
$get\text{-}maximum\text{-}level\ (trail\ S)\ D \equiv i \Longrightarrow$
$get\text{-}level\ (trail\ S)\ K = i + 1 \Longrightarrow$
$T \sim cons\text{-}trail\ (Propagated\ L\ (add\text{-}mset\ L\ D))$
$(reduce\text{-}trail\text{-}to\ M1$
$(add\text{-}learned\text{-}cls\ (add\text{-}mset\ L\ D)$
$(update\text{-}conflicting\ None\ S))) \Longrightarrow$
$simple\text{-}backtrack\ S\ T$

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

This is a generalised version of backtrack: It is general enough te also include OCDCL's version.

inductive *backtrackg* :: $'st \Rightarrow 'st \Rightarrow bool$ **for** S :: $'st$ **where**
backtrackg-rule:
$conflicting\ S = Some\ (add\text{-}mset\ L\ D) \Longrightarrow$
$(Decided\ K\ \#\ M1,\ M2) \in set\ (get\text{-}all\text{-}ann\text{-}decomposition\ (trail\ S)) \Longrightarrow$
$get\text{-}level\ (trail\ S)\ L = backtrack\text{-}lvl\ S \Longrightarrow$
$get\text{-}level\ (trail\ S)\ L = get\text{-}maximum\text{-}level\ (trail\ S)\ (add\text{-}mset\ L\ D') \Longrightarrow$
$get\text{-}maximum\text{-}level\ (trail\ S)\ D' \equiv i \Longrightarrow$
$get\text{-}level\ (trail\ S)\ K = i + 1 \Longrightarrow$
$D' \subseteq\#\ D \Longrightarrow$
$T \sim cons\text{-}trail\ (Propagated\ L\ (add\text{-}mset\ L\ D'))$
$(reduce\text{-}trail\text{-}to\ M1$
$(add\text{-}learned\text{-}cls\ (add\text{-}mset\ L\ D')$
$(update\text{-}conflicting\ None\ S))) \Longrightarrow$
$backtrackg\ S\ T$

inductive-cases *backtrackgE*: *backtrackg S T*

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

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

inductive *skip* :: $'st \Rightarrow 'st \Rightarrow bool$ **for** S :: $'st$ **where**
skip-rule:
$trail\ S = Propagated\ L\ C'\ \#\ M \Longrightarrow$
$conflicting\ S = Some\ E \Longrightarrow$
$-L \notin\#\ E \Longrightarrow$
$E \neq \{\#\} \Longrightarrow$

$T \sim tl\text{-}trail\ S \Longrightarrow$
$skip\ S\ T$

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

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

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

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

Christoph's version restricts restarts to the the case where $\neg M \models N+U$. While it is possible to implement this (by watching a clause), This is an unnecessary restriction.

inductive $restart :: 'st \Rightarrow 'st \Rightarrow bool$ **for** $S :: 'st$ **where**
$restart$: $state\ S = (M,\ N,\ U,\ None,\ S') \Longrightarrow$
$U' \subseteq\#\ U \Longrightarrow$
$state\ T = ([],\ N,\ U',\ None,\ S') \Longrightarrow$
$restart\ S\ T$

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

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

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

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

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

inductive $cdcl_W\text{-}bj :: 'st \Rightarrow 'st \Rightarrow bool$ **where**
$skip$: $skip\ S\ S' \Longrightarrow cdcl_W\text{-}bj\ S\ S'\ |$
$resolve$: $resolve\ S\ S' \Longrightarrow cdcl_W\text{-}bj\ S\ S'\ |$
$backtrack$: $backtrack\ S\ S' \Longrightarrow cdcl_W\text{-}bj\ S\ S'$

inductive-cases $cdcl_W\text{-}bjE$: $cdcl_W\text{-}bj\ S\ T$

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

lemma *rtranclp-propagate-is-rtranclp-*$cdcl_W$*-restart*:
  *propagate*** *S S'* ⟹ $cdcl_W$*-restart*** *S S'*
  apply (*induction rule*: *rtranclp-induct*)
    apply (*simp*; *fail*)
  apply (*frule propagate*)
  using *rtranclp-trans*[*of* $cdcl_W$*-restart*] **by** *blast*

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

lemma $cdcl_W$*-*$cdcl_W$*-restart*:
  $cdcl_W$ *S T* ⟹ $cdcl_W$*-restart S T*
  by (*induction rule*: $cdcl_W$*.induct*) (*auto intro*: $cdcl_W$*-restart.intros simp del*: *state-prop*)

lemma *rtranclp-*$cdcl_W$*-*$cdcl_W$*-restart*:
  ‹$cdcl_W$*** *S T* ⟹ $cdcl_W$*-restart*** *S T*›
  apply (*induction rule*: *rtranclp-induct*)
   apply (*auto*; *fail*)[]
  by (*meson* $cdcl_W$*-*$cdcl_W$*-restart rtranclp.rtrancl-into-rtrancl*)

lemma $cdcl_W$*-restart-all-rules-induct*[*consumes 1*, *case-names propagate conflict forget restart decide skip resolve backtrack*]:
  fixes *S* :: *'st*
  assumes
    $cdcl_W$*-restart*: $cdcl_W$*-restart S S'* **and**
    propagate: ⋀*T*. *propagate S T* ⟹ *P S T* **and**
    conflict: ⋀*T*. *conflict S T* ⟹ *P S T* **and**
    forget: ⋀*T*. *forget S T* ⟹ *P S T* **and**
    restart: ⋀*T*. *restart S T* ⟹ *P S T* **and**
    decide: ⋀*T*. *decide S T* ⟹ *P S T* **and**
    skip: ⋀*T*. *skip S T* ⟹ *P S T* **and**
    resolve: ⋀*T*. *resolve S T* ⟹ *P S T* **and**
    backtrack: ⋀*T*. *backtrack S T* ⟹ *P S T*
  shows *P S S'*
  using *assms*(*1*)
proof (*induct S' rule*: $cdcl_W$*-restart.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 $cdcl_W$*-restart-all-induct*[*consumes 1*, *case-names propagate conflict forget restart decide skip resolve backtrack*]:
fixes *S* :: *'st*
assumes
$cdcl_W$*-restart*: $cdcl_W$*-restart S S'* **and**
propagateH: $\bigwedge C\ L\ T$. *conflicting S = None* $\Longrightarrow$
C ∈# *clauses S* $\Longrightarrow$
L ∈# *C* $\Longrightarrow$
trail S ⊨*as CNot* (*remove1-mset L C*) $\Longrightarrow$
undefined-lit (*trail S*) *L* $\Longrightarrow$
T ∼ *cons-trail* (*Propagated L C*) *S* $\Longrightarrow$
P S T **and**
conflictH: $\bigwedge D\ T$. *conflicting S = None* $\Longrightarrow$
D ∈# *clauses S* $\Longrightarrow$
trail S ⊨*as CNot D* $\Longrightarrow$
T ∼ *update-conflicting* (*Some D*) *S* $\Longrightarrow$
P S T **and**
forgetH: $\bigwedge C\ T$. *conflicting S = None* $\Longrightarrow$
C ∈# *learned-clss S* $\Longrightarrow$
¬(*trail S*) ⊨*asm clauses S* $\Longrightarrow$
C ∉ *set* (*get-all-mark-of-propagated* (*trail S*)) $\Longrightarrow$
C ∉# *init-clss S* $\Longrightarrow$
removeAll-mset C (*clauses S*) ⊨*pm C* $\Longrightarrow$
T ∼ *remove-cls C S* $\Longrightarrow$
P S T **and**
restartH: $\bigwedge T\ U$. *conflicting S = None* $\Longrightarrow$
state T = ([], *init-clss S*, *U*, *None*, *additional-info S*) $\Longrightarrow$
U ⊆# *learned-clss S* $\Longrightarrow$
P S T **and**
decideH: $\bigwedge L\ T$. *conflicting S = None* $\Longrightarrow$
undefined-lit (*trail S*) *L* $\Longrightarrow$
atm-of L ∈ *atms-of-mm* (*init-clss S*) $\Longrightarrow$
T ∼ *cons-trail* (*Decided L*) *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 ∉# *E* $\Longrightarrow$ *E* ≠ {#} $\Longrightarrow$
T ∼ *tl-trail S* $\Longrightarrow$
P S T **and**
resolveH: $\bigwedge L\ E\ M\ D\ T$.
trail S = Propagated L E # M $\Longrightarrow$

$L \in\# E \Longrightarrow$
$hd\text{-}trail\ S = Propagated\ L\ E \Longrightarrow$
$conflicting\ S = Some\ D \Longrightarrow$
$-L \in\# D \Longrightarrow$
$get\text{-}maximum\text{-}level\ (trail\ S)\ ((remove1\text{-}mset\ (-L)\ D)) = backtrack\text{-}lvl\ S \Longrightarrow$
$T \sim update\text{-}conflicting$
$(Some\ (resolve\text{-}cls\ L\ D\ E))\ (tl\text{-}trail\ S) \Longrightarrow$
$P\ S\ T$ **and**
$backtrackH$: $\bigwedge L\ D\ K\ i\ M1\ M2\ T\ D'.$
$conflicting\ S = Some\ (add\text{-}mset\ L\ D) \Longrightarrow$
$(Decided\ K\ \#\ M1,\ M2) \in set\ (get\text{-}all\text{-}ann\text{-}decomposition\ (trail\ S)) \Longrightarrow$
$get\text{-}level\ (trail\ S)\ L = backtrack\text{-}lvl\ S \Longrightarrow$
$get\text{-}level\ (trail\ S)\ L = get\text{-}maximum\text{-}level\ (trail\ S)\ (add\text{-}mset\ L\ D') \Longrightarrow$
$get\text{-}maximum\text{-}level\ (trail\ S)\ D' \equiv i \Longrightarrow$
$get\text{-}level\ (trail\ S)\ K = i+1 \Longrightarrow$
$D' \subseteq\# D \Longrightarrow$
$clauses\ S \models pm\ add\text{-}mset\ L\ D' \Longrightarrow$
$T \sim cons\text{-}trail\ (Propagated\ L\ (add\text{-}mset\ L\ D'))$
$(reduce\text{-}trail\text{-}to\ M1$
$(add\text{-}learned\text{-}cls\ (add\text{-}mset\ L\ D')$
$(update\text{-}conflicting\ None\ S))) \Longrightarrow$
$P\ S\ T$
shows $P\ S\ S'$
using $cdcl_W\text{-}restart$
proof ($induct\ S\ S'\ rule$: $cdcl_W\text{-}restart\text{-}all\text{-}rules\text{-}induct$)
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*)
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 $cdcl_W\text{-}o\text{-}induct$[*consumes 1*, *case-names decide skip resolve backtrack*]:

fixes S :: *'st*
assumes *cdcl*$_W$*-restart*: *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) 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* $T\ D'$.
conflicting S = *Some* (*add-mset* $L\ D$) $\Longrightarrow$
(*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) (*add-mset* $L\ D'$) $\Longrightarrow$
get-maximum-level (*trail* S) $D' \equiv i \Longrightarrow$
get-level (*trail* S) $K = i{+}1 \Longrightarrow$
$D' \subseteq\#\ D \Longrightarrow$
clauses S $\models$*pm add-mset* $L\ D'$ $\Longrightarrow$
$T \sim$ *cons-trail* (*Propagated* L (*add-mset* $L\ D'$))
(*reduce-trail-to M1*
(*add-learned-cls* (*add-mset* $L\ D'$)
(*update-conflicting None* S))) $\Longrightarrow$
$P\ S\ T$
shows $P\ S\ T$
using *cdcl*$_W$*-restart* **apply** (*induct* T *rule*: *cdcl*$_W$*-o.induct*)
subgoal using *assms*(*2*) **by** (*auto elim*: *decideE*; *fail*)
subgoal apply (*elim cdcl*$_W$*-bjE skipE resolveE backtrackE*)
apply (*frule skipH*; *simp*; *fail*)
apply (*cases trail S*; *auto elim*!: *resolveE intro*!: *resolveH*; *fail*)
apply (*frule backtrackH*; *simp*; *fail*)
done
done

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*)

lemma *backtrack-backtrackg*:
  ‹*backtrack S T* $\Longrightarrow$ *backtrackg S T*›
  unfolding *backtrack.simps backtrackg.simps*
  by *blast*

lemma *simple-backtrack-backtrackg*:
  ‹*simple-backtrack S T* $\Longrightarrow$ *backtrackg S T*›
  unfolding *simple-backtrack.simps backtrackg.simps*
  by *blast*

1.1.3 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.

Nitpicking 0.1. *As one can see in the following proof, the properties of the levels are* required *to prove Item 1 page 94 of Weidenbach's book. However, this point is only mentioned* later*, and only in the proof of Item 7 page 95 of Weidenbach's book.*

lemma *backtrack-lit-skiped*:
  assumes
    L: *get-level* (*trail S*) *L* = *backtrack-lvl S* **and**
    M1: (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*)) **and**
    no-dup: *no-dup* (*trail S*) **and**
    lev-K: *get-level* (*trail S*) *K* = *i* + *1*
  shows *undefined-lit M1 L*
proof (*rule ccontr*)
  let *?M* = *trail S*
  assume *L-in-M1*: ¬ *?thesis*
  obtain *M2'* **where**
    Mc: *trail S* = *M2'* @ *M2* @ *Decided K* # *M1*
    using *M1* **by** *blast*
  have *Kc*: ‹*undefined-lit M2' K*› **and** *KM2*: ‹*undefined-lit M2 K*› ‹*atm-of L* ≠ *atm-of K*› **and**
    ‹*undefined-lit M2' L*› ‹*undefined-lit M2 L*›
    using *L-in-M1 no-dup* **unfolding** *Mc* **by** (*auto simp*: *atm-of-eq-atm-of dest*: *defined-lit-no-dupD*)
  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 M1 L*] *KM2 lev-K Kc* **unfolding** *Mc* **by** *auto*
  moreover have *Suc i* $\leq$ *backtrack-lvl S* **using** *KM2 lev-K Kc* **unfolding** *Mc* **by** (*simp add*: *Mc*)
  ultimately show *False* **using** *L g-M-eq-g-M1* **by** *auto*
qed

lemma $cdcl_W$*-restart-distinctinv-1*:
  assumes
    $cdcl_W$*-restart S S'* **and**
    n-d: *no-dup* (*trail S*)
  shows *no-dup* (*trail S'*)
  using *assms*(*1*)
proof (*induct rule*: $cdcl_W$*-restart-all-induct*)
  case (*backtrack L D K i M1 M2 T D'*) **note** *decomp* = *this*(*2*) **and** *L* = *this*(*3*) **and** *lev-K* = *this*(*6*)
and
    T = *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** (*auto dest*: *no-dup-appendD simp*: *defined-lit-map image-Un*)
  moreover have *L-M1*: *undefined-lit M1 L*
    using *backtrack-lit-skiped*[*of S L K M1 M2 i*] *L decomp lev-K n-d*
    unfolding *defined-lit-map lits-of-def* **by** *fast*
  ultimately show *?case* **using** *decomp T n-d* **by** (*auto dest*: *no-dup-appendD*)
qed (*use n-d* **in** *auto*)

Item 1 page 94 of Weidenbach's book

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

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*)

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

lemma *rtranclp-cdcl$_W$-restart-consistent-inv*:
  assumes
    cdcl$_W$-restart** 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$_W$-restart-consistent-inv*)

lemma *tranclp-cdcl$_W$-restart-consistent-inv*:
  assumes
    cdcl$_W$-restart^{++} 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$_W$-restart-consistent-inv*)

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

lemma *backtrack-ex-decomp*:
  assumes
    M-l: *no-dup* (*trail 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* **by** *auto*
  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*
  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

lemma *backtrack-lvl-backtrack-decrease*:
  assumes *inv*: *cdcl$_W$-M-level-inv S* **and** *bt*: *backtrack S T*
  shows *backtrack-lvl T* < *backtrack-lvl S*
  using *inv bt le-count-decided-decomp*[*of trail S backtrack-lvl T*]
  unfolding *cdcl$_W$-M-level-inv-def*
  by (*fastforce elim*!: *backtrackE simp*: *append-assoc*[*of - - -# -*, *symmetric*]
    simp del: *append-assoc*)

Compatibility with (∼)

declare *state-eq-trans*[*trans*]

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* ∈# *clauses S′*
using *SS′ C*
by (*auto simp*: *clauses-def*)
have *T′*: ‹*T′* ∼ *cons-trail* (*Propagated L C*) *S′*›
using *state-eq-trans*[*of T′ T*] *SS′ TT′*
by (*meson T cons-trail-state-eq state-eq-sym state-eq-trans*)
show *?thesis*
apply (*rule propagate-rule*[*of - C*])
using *SS′ conf C′ L tr undef TT′ T T′* **by** *auto*
qed

lemma *conflict-state-eq-compatible*:
assumes
confl: *conflict S T* **and**
TT′: *T* ∼ *T′* **and**
SS′: *S* ∼ *S′*
shows *conflict S′ T′*
proof −
obtain *D* **where**
conf: *conflicting S* = *None* **and**
D: *D* ∈# *clauses S* **and**
tr: *trail S* ⊨*as CNot D* **and**
T: *T* ∼ *update-conflicting* (*Some D*) *S*
using *confl* **by** (*elim conflictE*) *auto*

have *D′*: *D* ∈# *clauses S′*
using *D SS′* **by** *fastforce*

have *T′*: ‹*T′* ∼ *update-conflicting* (*Some D*) *S′*›
using *state-eq-trans*[*of T′ T*] *SS′ TT′*
by (*meson T update-conflicting-state-eq state-eq-sym state-eq-trans*)
show *?thesis*
apply (*rule conflict-rule*[*of - D*])
using *SS′ conf D′ tr TT′ T T′* **by** *auto*
qed

lemma *backtrack-state-eq-compatible*:
assumes
bt: *backtrack S T* **and**
SS′: *S* ∼ *S′* **and**
TT′: *T* ∼ *T′*
shows *backtrack S′ T′*
proof −
obtain *D L K i M1 M2 D′* **where**
conf: *conflicting S* = *Some* (*add-mset L D*) **and**
decomp: (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*)) **and**
lev: *get-level* (*trail S*) *L* = *backtrack-lvl S* **and**
max: *get-level* (*trail S*) *L* = *get-maximum-level* (*trail S*) (*add-mset L D′*) **and**

```
    max-D: get-maximum-level (trail S) D' ≡ i and
    lev-K: get-level (trail S) K = Suc i and
    D'-D: ‹D' ⊆# D› and
    NU-DL: ‹clauses S ⊨pm add-mset L D'› and
    T: T ∼ cons-trail (Propagated L (add-mset L D'))
              (reduce-trail-to M1
                (add-learned-cls (add-mset L D')
                  (update-conflicting None S)))
    using bt by (elim backtrackE) metis
  let ?D = ‹add-mset L D›
  let ?D' = ‹add-mset L D'›
  have D': conflicting S' = Some ?D
    using SS' conf by (cases conflicting S') auto

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

lemma decide-state-eq-compatible:
  assumes
    dec: decide S T and
    SS': S ∼ S' and
    TT': T ∼ T'
  shows decide S' T'
  using assms
proof -
  obtain L :: 'v literal where
    f4: undefined-lit (trail S) L
      atm-of L ∈ atms-of-mm (init-clss S)
      T ∼ cons-trail (Decided L) S
    using dec decide.simps by blast
  have cons-trail (Decided L) S' ∼ T'
    using f4 SS' TT' by (metis (no-types) cons-trail-state-eq state-eq-sym
        state-eq-trans)
  then show ?thesis
    using f4 SS' TT' dec by (auto simp: decide.simps state-eq-sym)
```

qed

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*
have T': ‹$T' \sim$ *tl-trail* S'›
by (*meson* SS' T TT' *state-eq-sym state-eq-trans tl-trail-state-eq*)
show *?thesis*
apply (*rule skip-rule*)
using *tr raw* L E T SS' **apply** (*auto*; *fail*)[]
using E' **apply** (*simp*; *fail*)
using E' SS' L *raw* E **apply** ((*auto*; *fail*)+)[*2*]
using T' **by** *auto*
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*
have T': ‹$T' \sim$ *update-conflicting* (*Some* (*remove1-mset* ($-$ L) D' $\cup\#$ *remove1-mset* L E))
(*tl-trail* S')›
proof −
have *tl-trail* $S \sim$ *tl-trail* S'

using *SS'* **by** (*auto simp*: *tl-trail-state-eq*)
then show *?thesis*
using *T T'T* ‹*D* = *D'*› *state-eq-trans update-conflicting-state-eq* **by** *blast*
qed
show *?thesis*
apply (*rule resolve-rule*)
using *tr SS'* **apply** (*simp*; *fail*)
using *hd SS'* **apply** (*simp*; *fail*)
using *L* **apply** (*simp*; *fail*)
using *D'* **apply** (*simp*; *fail*)
using *D' SS' raw LD* **apply** (*auto*; *fail*)[]
using *D' SS' raw LD i* **apply** (*auto*; *fail*)[]
using *T'* **by** *auto*
qed

lemma *forget-state-eq-compatible*:
assumes
forget: *forget S T* **and**
SS': *S* ∼ *S'* **and**
TT': *T* ∼ *T'*
shows *forget S' T'*
proof −
obtain *C* **where**
conf: *conflicting S* = *None* **and**
C: *C* ∈# *learned-clss S* **and**
tr: ¬(*trail S*) ⊨*asm clauses S* **and**
C1: *C* ∉ *set* (*get-all-mark-of-propagated* (*trail S*)) **and**
C2: *C* ∉# *init-clss S* **and**
ent: ‹*removeAll-mset C* (*clauses S*) ⊨*pm C*› **and**
T: *T* ∼ *remove-cls C S*
using *forget* **by** (*elim forgetE*) *simp*
have *T'*: ‹*T'* ∼ *remove-cls C S'*›
by (*meson SS' T TT' remove-cls-state-eq state-eq-sym state-eq-trans*)
show *?thesis*
apply (*rule forget-rule*)
using *SS' conf* **apply** (*simp*; *fail*)
using *C SS'* **apply** (*simp*; *fail*)
using *SS' tr* **apply** (*simp*; *fail*)
using *SS' C1* **apply** (*simp*; *fail*)
using *SS' C2* **apply** (*simp*; *fail*)
using *ent SS'* **apply** (*simp*; *fail*)
using *T'* **by** *auto*
qed

lemma *cdcl$_W$-restart-state-eq-compatible*:
assumes
cdcl$_W$-restart S T **and** ¬*restart S T* **and**
S ∼ *S'*
T ∼ *T'*
shows *cdcl$_W$-restart S' T'*
using *assms* **by** (*meson backtrack backtrack-state-eq-compatible bj cdcl$_W$-restart.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
$T \sim T'$
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
$S \sim S'$ **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)]*
have *$cdcl_W$-restart^{++} S T*
using *tranclp-mono[of $cdcl_W$-bj $cdcl_W$-restart] step.hyps(1) $cdcl_W$-restart.other $cdcl_W$-o.bj* **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

lemma *skip-unique*:
skip S T $\Longrightarrow$ skip S T' $\Longrightarrow$ $T \sim T'$
by (*auto elim!: skipE intro: state-eq-trans'*)

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

The same holds for backtrack, but more invariants are needed.

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!: tranclp-$cdcl_W$-restart-consistent-inv

dest: *tranclp-mono-explicit*[*of* $cdcl_W$*-o* - - $cdcl_W$*-restart*] $cdcl_W$*-o-no-more-init-clss*
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 $cdcl_W$*-restart-init-clss*:
assumes
$cdcl_W$*-restart* S T
shows *init-clss* S = *init-clss* T
using *assms* **by** (*induction rule*: $cdcl_W$*-restart-all-induct*)
(*auto simp*: *not-in-iff*)

lemma *rtranclp-*$cdcl_W$*-restart-init-clss*:
$cdcl_W$*-restart*** S T $\Longrightarrow$ *init-clss* S = *init-clss* T
by (*induct rule*: *rtranclp-induct*) (*auto dest*: $cdcl_W$*-restart-init-clss* *rtranclp-*$cdcl_W$*-restart-consistent-inv*)

lemma *tranclp-*$cdcl_W$*-restart-init-clss*:
$cdcl_W$*-restart*$^{++}$ S T $\Longrightarrow$ *init-clss* S = *init-clss* T
using *rtranclp-*$cdcl_W$*-restart-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 belong to the clauses. We could also restrict it to entailment by the clauses, to allow forgetting this clauses.

definition (**in** $state_W$*-ops*) *reasons-in-clauses* :: ‹$'st \Rightarrow$ *bool*› **where**
‹*reasons-in-clauses* (S :: $'st$) $\longleftrightarrow$
(*set* (*get-all-mark-of-propagated* (*trail* S)) $\subseteq$ *set-mset* (*clauses* S))›

definition (**in** $state_W$*-ops*) $cdcl_W$*-learned-clause* :: ‹$'st \Rightarrow$ *bool*› **where**
$cdcl_W$*-learned-clause* (S :: $'st$) $\longleftrightarrow$
(($\forall$ T. *conflicting* S = *Some* T $\longrightarrow$ *clauses* S $\models$*pm* T)
$\wedge$ *reasons-in-clauses* S)

lemma $cdcl_W$*-learned-clause-alt-def*:
‹$cdcl_W$*-learned-clause* (S :: $'st$) $\longleftrightarrow$
(($\forall$ T. *conflicting* S = *Some* T $\longrightarrow$ *clauses* S $\models$*pm* T)
$\wedge$ *set* (*get-all-mark-of-propagated* (*trail* S)) $\subseteq$ *set-mset* (*clauses* S))›
by (*auto simp*: $cdcl_W$*-learned-clause-def* *reasons-in-clauses-def*)

lemma *reasons-in-clauses-init-state*[*simp*]: ‹*reasons-in-clauses* (*init-state* N)›
by (*auto simp*: *reasons-in-clauses-def*)

Item 3 page 95 of Weidenbach's book for the inital state and some additional structural properties about the trail.

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

Item 4 page 95 of Weidenbach's book

lemma *cdcl$_W$-restart-learned-clss*:
  assumes
    cdcl$_W$-restart 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*)
proof (*induct rule*: *cdcl$_W$-restart-all-induct*)
  case (*backtrack L D K i M1 M2 T D'*) **note** *decomp* = *this*(*2*) **and** *confl* = *this*(*1*) **and** *lev-K* = *this* (*6*)
    and *T* = *this*(*9*)
  show *?case*
    using *decomp learned confl T* **unfolding** *cdcl$_W$-learned-clause-alt-def reasons-in-clauses-def*
    by (*auto dest*!: *get-all-ann-decomposition-exists-prepend*)
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 *clauses S* ⊨*pm add-mset L C*
      using *trail learned* **unfolding** *cdcl$_W$-learned-clause-alt-def clauses-def reasons-in-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-alt-def clauses-def reasons-in-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*
    by (*auto*
      simp: *clauses-def cdcl$_W$-learned-clause-alt-def reasons-in-clauses-def*
      dest: *true-clss-clssm-subsetE*)
next
  case *propagate*
  then show *?case* **using** *learned* **by** (*auto simp*: *cdcl$_W$-learned-clause-alt-def reasons-in-clauses-def*)
next
  case *conflict*
  then show *?case* **using** *learned*
    by (*fastforce simp*: *cdcl$_W$-learned-clause-alt-def clauses-def*
      true-clss-clss-in-imp-true-clss-cls reasons-in-clauses-def)
next
  case (*forget U*)
  then show *?case* **using** *learned*
    by (*auto simp*: *cdcl$_W$-learned-clause-alt-def clauses-def reasons-in-clauses-def*
      split: *if-split-asm*)
qed (*use learned* **in** ‹*auto simp*: *cdcl$_W$-learned-clause-alt-def clauses-def reasons-in-clauses-def*›)

lemma *rtranclp-cdcl$_W$-restart-learned-clss*:
  assumes
    cdcl$_W$-restart** *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$_W$-restart-learned-clss intro*: *rtranclp-cdcl$_W$-restart-consistent-inv*)

lemma *cdcl$_W$-restart-reasons-in-clauses*:
  assumes
    cdcl$_W$-restart S S' **and**
    learned: *reasons-in-clauses S*
  shows *reasons-in-clauses S'*
  using *assms*(*1*) *learned*
  by (*induct rule*: *cdcl$_W$-restart-all-induct*)
    (*auto simp*: *reasons-in-clauses-def dest*!: *get-all-ann-decomposition-exists-prepend*)

lemma *rtranclp-cdcl$_W$-restart-reasons-in-clauses*:
  assumes
    cdcl$_W$-restart** *S S'* **and**
    learned: *reasons-in-clauses S*
  shows *reasons-in-clauses S'*
  using *assms*(*1*) *learned*
  by (*induct rule*: *rtranclp-induct*)
    (*auto simp*: *cdcl$_W$-restart-reasons-in-clauses*)

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 *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 rule*: *propagate.induct*)
  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* ⊆ *atms-of-mm* (*init-clss S*)
    using *C alien* **unfolding** *no-strange-atm-def*
    by (*auto simp*: *clauses-def dest*!: *multi-member-split*)
  show *?case*
    unfolding *no-strange-atm-def*
  proof (*intro conjI allI impI*, *goal-cases*)
    case (*1 C*)
    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 *atms-of-ms-learned-clss-restart-state-in-atms-of-ms-learned-clssI*:
  atms-of-mm (*learned-clss S*) ⊆ *atms-of-mm* (*init-clss S*) ⟹
   x ∈ *atms-of-mm* (*learned-clss T*) ⟹
   learned-clss T ⊆# *learned-clss S* ⟹
   x ∈ *atms-of-mm* (*init-clss S*)
  by (*meson atms-of-ms-mono contra-subsetD set-mset-mono*)

lemma (**in** −) *atms-of-subset-mset-mono*: ‹*D′* ⊆# *D* ⟹ *atms-of D′* ⊆ *atms-of D*›
  by (*auto simp*: *atms-of-def*)

lemma $cdcl_W$*-restart-no-strange-atm-explicit*:
  assumes
    $cdcl_W$*-restart S S′* **and**
    lev: $cdcl_W$*-M-level-inv S* **and**
    conf: ∀ *T*. *conflicting S* = *Some T* ⟶ *atms-of T* ⊆ *atms-of-mm* (*init-clss S*) **and**
    decided: ∀ *L mark*. *Propagated L mark* ∈ *set* (*trail S*)
      ⟶ *atms-of mark* ⊆ *atms-of-mm* (*init-clss S*) **and**
    learned: *atms-of-mm* (*learned-clss S*) ⊆ *atms-of-mm* (*init-clss S*) **and**
    trail: *atm-of* ‘ (*lits-of-l* (*trail S*)) ⊆ *atms-of-mm* (*init-clss S*)
  shows
    (∀ *T*. *conflicting S′* = *Some T* ⟶ *atms-of T* ⊆ *atms-of-mm* (*init-clss S′*)) ∧
    (∀ *L mark*. *Propagated L mark* ∈ *set* (*trail S′*) ⟶ *atms-of mark* ⊆ *atms-of-mm* (*init-clss S′*)) ∧
    atms-of-mm (*learned-clss S′*) ⊆ *atms-of-mm* (*init-clss S′*) ∧
    atm-of ‘ (*lits-of-l* (*trail S′*)) ⊆ *atms-of-mm* (*init-clss S′*)
    (**is** *?C S′* ∧ *?M S′* ∧ *?U S′* ∧ *?V S′*)
  using *assms*(*1*)
proof (*induct rule*: $cdcl_W$*-restart-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 ⊆ ⋃(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 ⊆ (⋃x∈set-mset (init-clss S). atms-of x)
      ∪ (⋃x∈set-mset (learned-clss S). atms-of x)
    assume a2:
      (⋃x∈set-mset (learned-clss S). atms-of x) ⊆ (⋃x∈set-mset (init-clss S). atms-of x)
    assume xa ∈# D
    then have atm-of xa ∈ UNION (set-mset (init-clss S)) atms-of
      using a2 a1 by (metis (no-types) Un-iff atm-of-lit-in-atms-of atms-of-def subset-Un-eq)
    then have ∃ m∈set-mset (init-clss S). atm-of xa ∈ atms-of m
      by blast
    } note H = this
  ultimately show ?case using conflict.prems T learned 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 intro: atms-of-ms-learned-clss-restart-state-in-atms-of-ms-learned-clssI)
next
  case (forget C T) note confl = this(1) and C = this(4) and C-le = this(5) and
    T = this(7)
  have H: ⋀L mark. Propagated L mark ∈ set (trail S) ⟹ atms-of mark ⊆ 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 D') note confl = this(1) and decomp = this(2) and
    lev-K = this(6) and D-D' = this(7) and M1-D' = this(8) and T = this(9)
  have ?C T
    using conf T decomp lev lev-K by (auto simp: cdcl_W-M-level-inv-decomp)
  moreover have set M1 ⊆ set (trail S)
    using decomp by auto
  then have M: ?M T
    using decided conf confl T decomp lev lev-K D-D'
    by (auto simp: image-subset-iff clauses-def cdcl_W-M-level-inv-decomp
        dest!: atms-of-subset-mset-mono)
  moreover have ?U T
```

using *learned decomp conf confl T lev lev-K D-D'* **unfolding** *clauses-def*
by (*auto simp*: *cdcl*$_W$*-M-level-inv-decomp dest*: *atms-of-subset-mset-mono*)
moreover have *?V T*
using *M conf confl trail T decomp lev 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-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-minusD*)
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*$_W$*-restart-no-strange-atm-inv*:
assumes *cdcl*$_W$*-restart S S'* **and** *no-strange-atm S* **and** *cdcl*$_W$*-M-level-inv S*
shows *no-strange-atm S'*
using *cdcl*$_W$*-restart-no-strange-atm-explicit*[*OF assms*(*1*)] *assms*(*2*,*3*) **unfolding** *no-strange-atm-def*
by *fast*

lemma *rtranclp-cdcl*$_W$*-restart-no-strange-atm-inv*:
assumes *cdcl*$_W$*-restart*** *S S'* **and** *no-strange-atm S* **and** *cdcl*$_W$*-M-level-inv S*
shows *no-strange-atm S'*
using *assms* **by** (*induction rule*: *rtranclp-induct*)
(*auto intro*: *cdcl*$_W$*-restart-no-strange-atm-inv rtranclp-cdcl*$_W$*-restart-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*
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* **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$_W$-restart*[*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$_W$-restart 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$_W$-restart-all-induct*)
case (*backtrack L D K i M1 M2 D'*)
then show *?case*
using *lev-inv* **unfolding** *distinct-cdcl$_W$-state-def*
by (*auto dest*: *get-all-ann-decomposition-incl distinct-mset-mono 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* **by** *auto*
next
case *resolve*
then show *?case*
by (*auto simp add*: *distinct-cdcl$_W$-state-def distinct-mset-set-def clauses-def*)
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$_W$-restart** 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$_W$-restart-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* $\Rightarrow$ *bool* **where**
every-mark-is-a-conflict S $\equiv$
$\forall$ *L mark a b. a* @ *Propagated L mark* # *b* = (*trail S*)
$\longrightarrow$ (*b* $\models$*as CNot* (*mark* − {#*L*#}) $\wedge$ *L* $\in$# *mark*)

definition *cdcl$_W$-conflicting* :: *'st* $\Rightarrow$ *bool* **where**
cdcl$_W$-conflicting S $\longleftrightarrow$
($\forall$ *T. conflicting S* = *Some T* $\longrightarrow$ *trail S* $\models$*as CNot T*) $\wedge$ *every-mark-is-a-conflict S*

lemma *backtrack-atms-of-D-in-M1*:
fixes *S T* :: *'st* **and** *D D'* :: ‹*'v clause*› **and** *K L* :: ‹*'v literal*› **and** *i* :: *nat* **and**
M1 M2:: ‹(*'v*, *'v clause*) *ann-lits*›

assumes
inv: *no-dup* (*trail S*) **and**
i: *get-maximum-level* (*trail S*) *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*) (*add-mset L D'*) **and**
S-confl: *conflicting S* = *Some D* **and**
lev-K: *get-level* (*trail S*) *K* = *Suc i* **and**
T: *T* ∼ *cons-trail K''*
(*reduce-trail-to M1*
(*add-learned-cls* (*add-mset L D'*)
(*update-conflicting None S*))) **and**
confl: ∀ *T*. *conflicting S* = *Some T* ⟶ *trail S* ⊨*as CNot T* **and**
D-D': ‹*D'* ⊆# *D*›
shows *atms-of D'* ⊆ *atm-of* ‘ *lits-of-l* (*tl* (*trail T*))
proof (*rule ccontr*)
let *?k* = *get-maximum-level* (*trail S*) (*add-mset L D'*)

have *trail S* ⊨*as CNot D* **using** *confl S-confl* **by** *auto*
then have *trail S* ⊨*as CNot D'*
using *D-D'* **by** (*auto simp*: *true-annots-true-cls-def-iff-negation-in-model*)
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 *S-lvl* **unfolding** *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*)

obtain *L''* **where**
L'' ∈# *D'* **and**
L'': *L'* = *atm-of L''*
using *L' L'-notin-M1* **unfolding** *atms-of-def* **by** *auto*
then have *L'-in*: *defined-lit* (*M0* @ *M2* @ *Decided K* # []) *L''*
using *vars-of-D L'-notin-M1 L'* **unfolding** *M*
by (*auto dest*: *in-atms-of-minusD*
simp: *defined-lit-append defined-lit-map lits-of-def image-Un*)
have *L''-M1*: ‹*undefined-lit M1 L''*›
using *L'-notin-M1 L''* **by** (*auto simp*: *defined-lit-map lits-of-def*)
have ‹*undefined-lit* (*M0* @ *M2*) *K*›
using *inv* **by** (*auto simp*: $cdcl_W$*-M-level-inv-def M*)
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''-M1 L'' get-level-skip-end*[*OF L'-in*[*unfolded L''*], *of M1*] *M* **by** *auto*
moreover {
consider
(*M0*) *defined-lit M0 L''* |
(*M2*) *defined-lit M2 L''* |

(K) $L' = $ *atm-of K*
using *inv L'-in* **unfolding** L''
by (*auto simp*: *cdcl$_W$-M-level-inv-def defined-lit-append*)
then have *get-level* (*M0* @ *M2* @ *Decided K* # []) $L'' \geq$ *Suc 0*
proof *cases*
case *M0*
then have $L' \neq$ *atm-of K*
using ‹*undefined-lit* (*M0* @ *M2*) *K*› **unfolding** L'' **by** (*auto simp*: *atm-of-eq-atm-of*)
then show *?thesis* **using** *M0* **unfolding** L'' **by** *auto*
next
case *M2*
then have *undefined-lit* (*M0* @ *Decided K* # []) L''
by (*rule defined-lit-no-dupD*(*1*))
(*use inv* **in** ‹*auto simp*: *M* L'' *cdcl$_W$-M-level-inv-def no-dup-def*›)
then show *?thesis* **using** *M2* **unfolding** L'' **by** (*auto simp*: *image-Un*)
next
case *K*
have *undefined-lit* (*M0* @ *M2*) L''
by (*rule defined-lit-no-dupD*(*3*)[*of* ‹[*Decided K*]› - *M1*])
(*use inv K* **in** ‹*auto simp*: *M* L'' *cdcl$_W$-M-level-inv-def no-dup-def*›)
then show *?thesis* **using** *K* **unfolding** L'' **by** (*auto simp*: *image-Un*)
qed }
ultimately have *get-level* (*trail S*) $L'' \geq i + 1$
using *lev-*L'' **unfolding** *M* **by** *simp*
then have *get-maximum-level* (*trail S*) $D' \geq i + 1$
using *get-maximum-level-ge-get-level*[*OF* ‹$L'' \in\#$ D'›, *of trail S*] **by** *auto*
then show *False* **using** *i* **by** *auto*
qed

lemma *distinct-atms-of-incl-not-in-other*:
assumes
a1: *no-dup* (*M* @ M') **and**
a2: *atms-of D* $\subseteq$ *atm-of* ‘ *lits-of-l* M' **and**
a3: $x \in$ *atms-of D*
shows $x \notin$ *atm-of* ‘ *lits-of-l M*
using *assms* **by** (*auto simp*: *atms-of-def no-dup-def atm-of-eq-atm-of lits-of-def*)

lemma *backtrack-M1-CNot-D'*:
fixes *S T* :: *'st* **and** *D D'* :: ‹*'v clause*› **and** *K L* :: ‹*'v literal*› **and** *i* :: *nat* **and**
M1 M2:: ‹(*'v*, *'v clause*) *ann-lits*›
assumes
inv: *no-dup* (*trail S*) **and**
i: *get-maximum-level* (*trail S*) $D' \equiv i$ **and**
decomp: (*Decided K* # *M1*, *M2*)
$\in$ *set* (*get-all-ann-decomposition* (*trail S*)) **and**
S-lvl: *backtrack-lvl S* = *get-maximum-level* (*trail S*) (*add-mset L D'*) **and**
S-confl: *conflicting S* = *Some D* **and**
lev-K: *get-level* (*trail S*) *K* = *Suc i* **and**
T: *T* ∼ *cons-trail* K''
(*reduce-trail-to M1*
(*add-learned-cls* (*add-mset L D'*)
(*update-conflicting None S*))) **and**
confl: ∀ *T*. *conflicting S* = *Some T* ⟶ *trail S* ⊨*as CNot T* **and**
D-D': ‹$D' \subseteq\#$ *D*›
shows *M1* ⊨*as CNot D'* **and**
‹*atm-of* (*lit-of* K'') = *atm-of L* ⟹ *no-dup* (*trail T*)›

proof −
obtain *M0* **where** *M*: *trail S* = *M0* @ *M2* @ *Decided K* # *M1*
using *decomp* **by** *auto*
have *vars-of-D*: *atms-of D'* $\subseteq$ *atm-of* ‘ *lits-of-l M1*
using *backtrack-atms-of-D-in-M1*[*OF assms*] *decomp T* **by** *auto*
have *no-dup* (*trail S*) **using** *inv* **by** (*auto simp*: *cdcl*$_W$*-M-level-inv-decomp*)
then have *vars-in-M1*: $\forall$ *x* $\in$ *atms-of D'*. *x* $\notin$ *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* $\models$*as CNot D*
using *S-confl confl* **unfolding** *M true-annots-true-cls-def-iff-negation-in-model*
by (*auto dest*!: *in-diffD*)
then have *trail S* $\models$*as CNot D'*
using *D-D'* **unfolding** *true-annots-true-cls-def-iff-negation-in-model* **by** *auto*
then show *M1-D'*: *M1* $\models$*as CNot D'*
using *true-annots-remove-if-notin-vars*[*of M0* @ *M2* @ *Decided K* # [] *M1 CNot D'*]
vars-in-M1 S-confl confl **unfolding** *M lits-of-def* **by** *simp*
have *M1*: ‹*count-decided M1* = *i*›
using *lev-K inv i vars-in-M1* **unfolding** *M*
by *simp*
have *lev-L*: ‹*get-level* (*trail S*) *L* = *backtrack-lvl S*› **and** ‹*i* < *backtrack-lvl S*›
using *S-lvl i lev-K*
by (*auto simp*: *max-def get-maximum-level-add-mset*)
have ‹*no-dup M1*›
using *T decomp inv* **by** (*auto simp*: *M dest*: *no-dup-appendD*)
moreover have ‹*undefined-lit M1 L*›
using *backtrack-lit-skiped*[*of S L*, *OF - decomp*]
using *lev-L inv i M1* ‹*i* < *backtrack-lvl S*› **unfolding** *M*
by (*auto simp*: *split*: *if-splits*)
moreover have ‹*atm-of* (*lit-of K''*) = *atm-of L* $\Longrightarrow$
undefined-lit M1 L $\longleftrightarrow$ *undefined-lit M1* (*lit-of K''*)›
by (*simp add*: *defined-lit-map*)
ultimately show ‹*atm-of* (*lit-of K''*) = *atm-of L* $\Longrightarrow$ *no-dup* (*trail T*)›
using *T decomp inv* **by** *auto*
qed

Item 5 page 95 of Weidenbach's book

lemma *cdcl*$_W$*-restart-propagate-is-conclusion*:
assumes
cdcl$_W$*-restart S S'* **and**
inv: *cdcl*$_W$*-M-level-inv S* **and**
decomp: *all-decomposition-implies-m* (*clauses 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* (*clauses S'*) (*get-all-ann-decomposition* (*trail S'*))
using *assms*(*1*)
proof (*induct rule*: *cdcl*$_W$*-restart-all-induct*)
case *restart*
then show *?case* **by** *auto*
next
case (*forget C T*) **note** *C* = *this*(*2*) **and** *cls-C* = *this*(*6*) **and** *T* = *this*(*7*)
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 S) ⊨ps unmark-l b
      using decomp T by (auto simp add: all-decomposition-implies-def)
    moreover {
      have a1:C ∈# clauses S
        using C by (auto simp: clauses-def)
      have clauses T = clauses (remove-cls C S)
        using T by auto
      then have clauses T ⊨psm clauses S
        using a1 by (metis (no-types) clauses-remove-cls 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 T) ⊨ps unmark-l b
      using true-clss-clss-generalise-true-clss-clss by blast
  qed
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 S-CNot-C = this(4) 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 unm-ay: unmark-l a ∪ set-mset (clauses 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 (clauses 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 (clauses S) ⊨p {#L#} (is ?I ⊨p -)
  proof (rule true-clss-cls-plus-CNot)
    show ?I ⊨p add-mset L (remove1-mset L C)
```

```
      apply (rule true-clss-clss-in-imp-true-clss-cls[of - set-mset (clauses S)])
      using learned propa L by (auto simp: cdclW-learned-clause-alt-def true-annot-CNot-diff)
    next
      have unmark-l (trail S) ⊨ps CNot (remove1-mset L C)
        using S-CNot-C by (blast dest: true-annots-true-clss-clss)
      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 (clauses S) ⊨ps unmark-l seen ⟹
          (aa, b) ∈ set (tl (get-all-ann-decomposition (y @ a))) ⟹
          unmark-l aa ∪ set-mset (clauses 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 unm-ay ay by auto
  next
    case (backtrack L D K i M1 M2 T D') note conf = this(1) and decomp' = this(2) and
      lev-L = this(3) and lev-K = this(6) and D-D' = this(7) and NU-LD' = this(8)
      and T = this(9)
    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
    let ?D = ‹add-mset L D›
    let ?D' = ‹add-mset L D'›
    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' inv by (simp add: cdclW-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 (clauses 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 M
          unfolding all-decomposition-implies-def cdclW-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': $x = (M1', \textit{Propagated } L\ ?D' \# M1'')$
using ‹$x = ?hd$› **by** *auto*
have $(M1', M1'') \in$ *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* (*clauses S*) ⊨*ps unmark-l M1''*
using *decomp* **unfolding** *all-decomposition-implies-def* **by** *auto*

have ‹*no-dup* (*trail S*)›
using *inv* **unfolding** $cdcl_W$*-M-level-inv-def*
by *blast*
then have *M1-D'*: *M1* ⊨*as CNot D'* **and** ‹*no-dup* (*trail T*)›
using *backtrack-M1-CNot-D'*[*of S D'* ‹*i*› *K M1 M2 L* ‹*add-mset L D*› *T* ‹*Propagated L* (*add-mset L D'*)›]
confl inv backtrack **by** (*auto simp*: *subset-mset-trans-add-mset*)
have *M1* = *M1''* @ *M1'* **by** (*simp add*: *M1 get-all-ann-decomposition-decomp*)
have *TT*: *unmark-l M1'* ∪ *set-mset* (*clauses 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 *T'*: *unmark-l M1'* ∪ *set-mset* (*clauses S*) ⊨*p ?D'* **using** *NU-LD'* **by** *auto*
moreover have *unmark-l M1'* ∪ *set-mset* (*clauses S*) ⊨*p* {#*L*#}
using *true-clss-cls-plus-CNot*[*OF T' TT*] **by** *auto*
ultimately show *?thesis*
using *T' T decomp' inv 1* **unfolding** *x'* **by** (*simp add*: $cdcl_W$*-M-level-inv-decomp*)
qed
qed
qed

lemma $cdcl_W$*-restart-propagate-is-false*:
assumes
$cdcl_W$*-restart S S'* **and**
lev: $cdcl_W$*-M-level-inv S* **and**
learned: $cdcl_W$*-learned-clause S* **and**
decomp: *all-decomposition-implies-m* (*clauses S*) (*get-all-ann-decomposition* (*trail S*)) **and**
confl: ∀ *T*. *conflicting S* = *Some T* ⟶ *trail S* ⊨*as CNot T* **and**
alien: *no-strange-atm S* **and**
mark-confl: *every-mark-is-a-conflict S*
shows *every-mark-is-a-conflict S'*
using *assms*(*1*)
proof (*induct rule*: $cdcl_W$*-restart-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* ⊨*as CNot* (*mark* − {#*L'*#}) ∧ *L'* ∈# *mark*
using *mark-confl confl LC* **by** *cases auto*
qed
next
case (*decide L*) **note** *undef*[*simp*] = *this*(*2*) **and** *T* = *this*(*4*)
have ‹*tl a* @ *Propagated La mark* # *b* = *trail S*›

if ‹a @ Propagated La mark # b = Decided L # trail S› **for** a La mark b
using that **by** (cases 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 ‹b ⊨as CNot (mark − {#La#}) ∧ La ∈# mark›
if a @ Propagated La mark # b = Propagated L C′ # M **for** La mark a b
using mark-confl that **unfolding** skip.hyps(1) **by** simp
ultimately show b ⊨as CNot (mark − {#L′#}) ∧ L′ ∈# mark **by** blast
qed
next
case (conflict D)
then show ?case **using** mark-confl **by** simp
next
case (resolve L C M D T) **note** tr-S = this(1) **and** T = this(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 D′) **note** conf = this(1) **and** decomp = this(2) **and**
lev-K = this(6) **and** D-D′ = this(7) **and** M1-D′ = this(8) **and** T = this(9)
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-conflicting None S))) = M1
using decomp lev **by** (auto simp: $cdcl_W$-M-level-inv-decomp)
let ?D = add-mset L D
let ?D′ = add-mset L D′
have M1-D′: M1 ⊨as CNot D′
using backtrack-M1-CNot-D′[of S D′ ‹i› K M1 M2 L ‹add-mset L D› T ‹Propagated L (add-mset L D′)›]
confl lev backtrack **by** (auto simp: subset-mset-trans-add-mset $cdcl_W$-M-level-inv-def)

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*)
show *b* ⊨*as CNot* (*mark* − {#*La*#}) ∧ *La* ∈# *mark*
using *P M1-D' 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
$cdcl_W$*-restart 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*: $cdcl_W$*-restart-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*)
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*
using *M-lev tr-S* **by** (*fastforce dest*: $cdcl_W$*-M-level-inv-decomp*(*2*)
simp: *Decided-Propagated-in-iff-in-lits-of-l*)
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*) $\models$*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* **using** ‹*distinct-mset* (*?D* + {#− *L*#})› **by** *auto*
have *Propagated L* (*?C* + {#*L*#}) # *M* $\models$*as CNot ?D* ∪ *CNot* {#− *L*#}
using *confl tr confl-inv LC* **by** (*metis CNot-plus LD insert-DiffM2*)
then have *M* $\models$*as CNot ?D*
using *M-lev* ‹− *L* ∉# *?D*› *tr*
unfolding *cdcl*$_W$*-M-level-inv-def* **by** (*force intro*: *true-annots-lit-of-notin-skip*)
moreover assume *conflicting T* = *Some* T'
ultimately show *trail T* $\models$*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* $\models$*as CNot T* **and**
∀ *L mark a b*. *a* @ *Propagated L mark* # *b* = (*trail S*) ⟶
(*b* $\models$*as CNot* (*mark* − {#*L*#}) ∧ *L* ∈# *mark*)
using *assms* **unfolding** *cdcl*$_W$*-conflicting-def* **by** *blast*+

lemma *cdcl*$_W$*-conflicting-decomp2*:
assumes *cdcl*$_W$*-conflicting S* **and** *conflicting S* = *Some T*
shows *trail S* $\models$*as CNot T*
using *assms* **unfolding** *cdcl*$_W$*-conflicting-def* **by** *blast*+

lemma *cdcl*$_W$*-conflicting-S0-cdcl*$_W$*-restart*[*simp*]:
cdcl$_W$*-conflicting* (*init-state N*)
unfolding *cdcl*$_W$*-conflicting-def* **by** *auto*

definition *cdcl*$_W$*-learned-clauses-entailed-by-init* **where**
‹*cdcl*$_W$*-learned-clauses-entailed-by-init S* ⟷ *init-clss S* $\models$*psm learned-clss S*›

lemma *cdcl*$_W$*-learned-clauses-entailed-init*[*simp*]:
‹*cdcl*$_W$*-learned-clauses-entailed-by-init* (*init-state N*)›
by (*auto simp*: *cdcl*$_W$*-learned-clauses-entailed-by-init-def*)

lemma *cdcl*$_W$*-learned-clauses-entailed*:
assumes
cdcl$_W$*-restart*: *cdcl*$_W$*-restart S* S' **and**
2: *cdcl*$_W$*-learned-clause S* **and**
9: ‹*cdcl*$_W$*-learned-clauses-entailed-by-init S*›
shows ‹*cdcl*$_W$*-learned-clauses-entailed-by-init* S'›
using *cdcl*$_W$*-restart 9*
proof (*induction rule*: *cdcl*$_W$*-restart-all-induct*)
case *backtrack*
then show *?case*
using *assms* **unfolding** *cdcl*$_W$*-learned-clause-alt-def cdcl*$_W$*-learned-clauses-entailed-by-init-def*
by (*auto dest*!: *get-all-ann-decomposition-exists-prepend*
simp: *clauses-def cdcl*$_W$*-M-level-inv-decomp dest*: *true-clss-clss-left-right*)
qed (*auto simp*: *cdcl*$_W$*-learned-clauses-entailed-by-init-def elim*: *true-clss-clssm-subsetE*)

lemma *rtranclp-cdcl$_W$-learned-clauses-entailed*:
assumes
cdcl$_W$-restart: *cdcl$_W$-restart** S S'* **and**
2: *cdcl$_W$-learned-clause S* **and**
4: *cdcl$_W$-M-level-inv S* **and**
9: ‹*cdcl$_W$-learned-clauses-entailed-by-init S*›
shows ‹*cdcl$_W$-learned-clauses-entailed-by-init S'*›
using *assms* **apply** (*induction rule*: *rtranclp-induct*)
apply (*simp*; *fail*)
using *cdcl$_W$-learned-clauses-entailed rtranclp-cdcl$_W$-restart-learned-clss* **by** *blast*

Putting all the Invariants Together

lemma *cdcl$_W$-restart-all-inv*:
assumes
cdcl$_W$-restart: *cdcl$_W$-restart S S'* **and**
1: *all-decomposition-implies-m* (*clauses 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 (*clauses 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* (*clauses S'*) (*get-all-ann-decomposition* (*trail S'*))
using *cdcl$_W$-restart-propagate-is-conclusion*[*OF cdcl$_W$-restart 4 1 2 - 5*] *8*
unfolding *cdcl$_W$-conflicting-def* **by** *blast*
show *S2*: *cdcl$_W$-learned-clause S'* **using** *cdcl$_W$-restart-learned-clss*[*OF cdcl$_W$-restart 2 4*] **.**
show *S4*: *cdcl$_W$-M-level-inv S'* **using** *cdcl$_W$-restart-consistent-inv*[*OF cdcl$_W$-restart 4*] **.**
show *S5*: *no-strange-atm S'* **using** *cdcl$_W$-restart-no-strange-atm-inv*[*OF cdcl$_W$-restart 5 4*] **.**
show *S7*: *distinct-cdcl$_W$-state S'* **using** *distinct-cdcl$_W$-state-inv*[*OF cdcl$_W$-restart 4 7*] **.**
show *S8*: *cdcl$_W$-conflicting S'*
using *cdcl$_W$-conflicting-is-false*[*OF cdcl$_W$-restart 4 - - 7*] *8*
cdcl$_W$-restart-propagate-is-false[*OF cdcl$_W$-restart 4 2 1 - 5*] **unfolding** *cdcl$_W$-conflicting-def*
by *fast*
qed

lemma *rtranclp-cdcl$_W$-restart-all-inv*:
assumes
cdcl$_W$-restart: *rtranclp cdcl$_W$-restart S S'* **and**
1: *all-decomposition-implies-m* (*clauses 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 (*clauses 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$_W$-restart-all-inv*[*OF* $H(2)$]
H **by** *presburger*
case *2* **with** $H(3-7)$[*OF this*(*1*−*6*)] **show** *?case* **using** *cdcl$_W$-restart-all-inv*[*OF* $H(2)$]
H **by** *presburger*
case *3* **with** $H(3-7)$[*OF this*(*1*−*6*)] **show** *?case* **using** *cdcl$_W$-restart-all-inv*[*OF* $H(2)$]
H **by** *presburger*
case *4* **with** $H(3-7)$[*OF this*(*1*−*6*)] **show** *?case* **using** *cdcl$_W$-restart-all-inv*[*OF* $H(2)$]
H **by** *presburger*
case *5* **with** $H(3-7)$[*OF this*(*1*−*6*)] **show** *?case* **using** *cdcl$_W$-restart-all-inv*[*OF* $H(2)$]
H **by** *presburger*
case *6* **with** $H(3-7)$[*OF this*(*1*−*6*)] **show** *?case* **using** *cdcl$_W$-restart-all-inv*[*OF* $H(2)$]
H **by** *presburger*
qed

lemma *all-invariant-S0-cdcl$_W$-restart*:
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**
$\forall T$. *conflicting* (*init-state* N) = *Some* $T \longrightarrow$ (*trail* (*init-state* N))$\models$*as CNot* T **and**
no-strange-atm (*init-state* N) **and**
consistent-interp (*lits-of-l* (*trail* (*init-state* N))) **and**
$\forall L$ *mark* a b. a @ *Propagated* L *mark* # b = *trail* (*init-state* N) $\longrightarrow$
(b $\models$*as CNot* (*mark* − {#L#}) $\wedge$ L ∈# *mark*) **and**
distinct-cdcl$_W$-state (*init-state* N)
using *assms* **by** *auto*

Item 6 page 95 of Weidenbach's book

lemma *cdcl$_W$-only-propagated-vars-unsat*:
assumes
decided: $\forall x \in$ *set* M. ¬ *is-decided* x **and**
DN: D ∈# *clauses* S **and**
D: M $\models$*as CNot* D **and**
inv: *all-decomposition-implies-m* (N + U) (*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 + U))
proof (*rule ccontr*)
assume ¬ *unsatisfiable* (*set-mset* (N + U))

then obtain *I* **where**
I: *I* $\models$*s set-mset N I* $\models$*s set-mset U* **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 *tot-N*: *total-over-m I* (*set-mset N*) **using** *tot* **unfolding** *total-over-m-def* **by** *auto*
moreover have *total-over-m I* (*set-mset* (*learned-clss S*))
using *atm-incl state tot-N* **unfolding** *no-strange-atm-def total-over-m-def total-over-set-def*
by *auto*
ultimately have *I-D*: *I* $\models$ *D*
using *I DN cons state* **unfolding** *true-clss-clss-def true-clss-def Ball-def*
by (*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*+*U*) ∪ *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*+*U*) ∪ *unmark-l M*)
using *tot* **unfolding** *total-over-m-def* **by** *auto*
then have *IM*: *I* $\models$*s unmark-l M*
using *all-decomposition-implies-propagated-lits-are-implied*[*OF inv*] *cons I*
unfolding *true-clss-clss-def l0* **by** *auto*
have −*K* ∈ *I* **if** *K* ∈# *D* **for** *K*
proof −
have −*K* ∈ *lits-of-l M*
using *D that* **unfolding** *true-annots-def* **by** *force*
then show −*K* ∈ *I* **using** *IM true-clss-singleton-lit-of-implies-incl* **by** *fastforce*
qed
then have ¬ *I* $\models$ *D* **using** *cons* **unfolding** *true-cls-def true-lit-def consistent-interp-def* **by** *auto*
then show *False* **using** *I-D* **by** *blast*
qed

Item 5 page 95 of Weidenbach's book

We have actually a much stronger theorem, namely *all-decomposition-implies-propagated-lits-are-implied*, 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 ∀ *m* ∈ *set M*. ¬*is-decided m*
shows *set-mset N* $\models$*ps unmark-l M*
proof −
have *T*: {*unmark L* |*L*. *is-decided L* ∧ *L* ∈ *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 95 of Weidenbach's book (part 1)

lemma *conflict-with-false-implies-unsat*:
assumes
$cdcl_W$-restart: *$cdcl_W$-restart S S'* **and**
lev: *$cdcl_W$-M-level-inv S* **and**
[*simp*]: *conflicting S'* = *Some* {#} **and**
learned: *$cdcl_W$-learned-clause S* **and**
learned-entailed: ‹*$cdcl_W$-learned-clauses-entailed-by-init S*›

shows *unsatisfiable* (*set-mset* (*clauses S*))
using *assms*
proof −
have $cdcl_W$*-learned-clause S'* **using** $cdcl_W$*-restart-learned-clss* $cdcl_W$*-restart learned lev* **by** *auto*
then have *entail-false*: *clauses S'* ⊨*pm* {#} **using** *assms*(*3*) **unfolding** $cdcl_W$*-learned-clause-alt-def* **by** *auto*
moreover have *entailed*: ‹$cdcl_W$*-learned-clauses-entailed-by-init S'*›
using $cdcl_W$*-learned-clauses-entailed*[*OF* $cdcl_W$*-restart learned learned-entailed*] **.**
ultimately have *set-mset* (*init-clss S'*) ⊨*ps* {{#}}
unfolding $cdcl_W$*-learned-clauses-entailed-by-init-def*
by (*auto simp*: *clauses-def dest*: *true-clss-clss-left-right*)
then have *clauses S* ⊨*pm* {#}
by (*simp add*: $cdcl_W$*-restart-init-clss*[*OF assms*(*1*)] *clauses-def*)
then show *?thesis* **unfolding** *satisfiable-def true-clss-cls-def* **by** *auto*
qed

Item 7 page 95 of Weidenbach's book (part 2)

lemma *conflict-with-false-implies-terminated*:
assumes $cdcl_W$*-restart S S'* **and** *conflicting S* = *Some* {#}
shows *False*
using *assms* **by** (*induct rule*: $cdcl_W$*-restart-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
$cdcl_W$*-restart S S'* **and**
lev: $cdcl_W$*-M-level-inv S* **and**
conflicting: $cdcl_W$*-conflicting S* **and**
no-tauto: ∀ *s* ∈# *learned-clss S*. ¬*tautology s*
shows ∀ *s* ∈# *learned-clss S'*. ¬*tautology s*
using *assms*
proof (*induct rule*: $cdcl_W$*-restart-all-induct*)
case (*backtrack L D K i M1 M2 T D'*) **note** *confl* = *this*(*1*) **and** *D-D'* = *this*(*7*) **and** *M1-D'* = *this*(*8*) **and**
NU-LD' = *this*(*9*)
let *?D* = ‹*add-mset L D*›
let *?D'* = ‹*add-mset L D'*›
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 have ¬*tautology ?D'*
using *D-D' not-tautology-mono*[*of ?D' ?D*] **by** *auto*
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** *state-eq-learned-clss no-tauto*
by (*auto intro*: *atms-of-ms-learned-clss-restart-state-in-atms-of-ms-learned-clssI*)

qed (*auto dest*!: *in-diffD*)

definition *final-cdcl*$_W$*-restart-state* (S :: $'st$)
$\longleftrightarrow$ (*trail* S $\models$*asm* *init-clss* S
$\vee$ (($\forall L \in$ *set* (*trail* S). $\neg$*is-decided* L) $\wedge$
($\exists C \in\#$ *init-clss* S. *trail* S $\models$*as* *CNot* C)))

definition *termination-cdcl*$_W$*-restart-state* (S :: $'st$)
$\longleftrightarrow$ (*trail* S $\models$*asm* *init-clss* S
$\vee$ (($\forall L \in$ *atms-of-mm* (*init-clss* S). $L \in$ *atm-of* ` *lits-of-l* (*trail* S))
$\wedge$ ($\exists C \in\#$ *init-clss* S. *trail* S $\models$*as* *CNot* C)))

1.1.4 CDCL Strong Completeness

lemma *cdcl*$_W$*-restart-can-do-step*:
assumes
consistent-interp (*set* M) **and**
distinct M **and**
atm-of ` (*set* M) $\subseteq$ *atms-of-mm* N
shows $\exists S$. *rtranclp* *cdcl*$_W$*-restart* (*init-state* N) S
$\wedge$ *state-butlast* S = (*map* (λL. *Decided* L) M, N, {#}, *None*)
using *assms*
proof (*induct* M)
case *Nil*
then show *?case* **apply** − **by** (*auto intro*!: *exI*[*of* - *init-state* N])
next
case (*Cons* L M) **note** *IH* = *this*(*1*) **and** *dist* = *this*(*2*)
have *consistent-interp* (*set* M) **and** *distinct* M **and** *atm-of* ` *set* M $\subseteq$ *atms-of-mm* N
using *Cons.prems*(*1*−*3*) **unfolding** *consistent-interp-def* **by** *auto*
then obtain S **where**
st: *cdcl*$_W$*-restart*** (*init-state* N) S **and**
S: *state-butlast* S = (*map* (λL. *Decided* L) M, N, {#}, *None*)
using *IH* **by** *blast*
let *?S*$_0$ = *cons-trail* (*Decided* L) S
have *undef*: *undefined-lit* (*map* (λL. *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* $L \in$ *atms-of-mm* N **using** *Cons.prems*(*3*) **by** *auto*
moreover have *undef*: *undefined-lit* (*trail* S) L
using S *dist undef* **by** (*auto simp*: *defined-lit-map*)
ultimately have *cdcl*$_W$*-restart* S *?S*$_0$
using *cdcl*$_W$*-restart.other*[*OF* *cdcl*$_W$*-o.decide*[*OF* *decide-rule*[*of* S L *?S*$_0$]]] S
by *auto*
then have *cdcl*$_W$*-restart*** (*init-state* N) *?S*$_0$
using *st* **by** *auto*
then show *?case*
using S *undef* **by** (*auto intro*!: *exI*[*of* - *?S*$_0$] *simp del*: *state-prop*)
qed

theorem 2.9.11 page 98 of Weidenbach's book

lemma *cdcl*$_W$*-restart-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*) ⊆ *atms-of-mm N*
obtains *S* **where**
state-butlast S = (*map* (*λL. Decided L*) *M*, *N*, {#}, *None*) **and**
rtranclp $cdcl_W$*-restart* (*init-state N*) *S* **and**
final-$cdcl_W$*-restart-state S*
proof −
obtain *S* **where**
st: *rtranclp* $cdcl_W$*-restart* (*init-state N*) *S* **and**
S: *state-butlast S* = (*map* (*λL. Decided L*) *M*, *N*, {#}, *None*)
using $cdcl_W$*-restart-can-do-step*[*OF cons dist atm*] **by** *auto*
have *lits-of-l* (*map* (*λL. Decided L*) *M*) = *set M*
by (*induct M*, *auto*)
then have *map* (*λL. Decided L*) *M* ⊨*asm N* **using** *MN true-annots-true-cls* **by** *metis*
then have *final-*$cdcl_W$*-restart-state S*
using *S* **unfolding** *final-*$cdcl_W$*-restart-state-def* **by** *auto*
then show *?thesis* **using** *that st S* **by** *blast*
qed

1.1.5 Higher level strategy

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

- do propagate and conflict when possible;
- otherwise, do another rule (except forget and restart).

Definition

lemma *tranclp-conflict*:
tranclp conflict S S′ ⟹ *conflict S S′*
by (*induct rule*: *tranclp.induct*) (*auto elim*!: *conflictE*)

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 *tranclp-conflict-iff*:
full1 conflict S S′ ⟷ *conflict S S′*
by (*auto simp*: *full1-def dest*: *tranclp-conflict no-chained-conflict*)

lemma *no-conflict-after-conflict*:
conflict S T ⟹ ¬*conflict T U*
by (*auto elim*!: *conflictE simp*: *conflict.simps*)

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

inductive $cdcl_W$*-stgy* :: *′st* ⇒ *′st* ⇒ *bool* **for** *S* :: *′st* **where**
conflict′: *conflict S S′* ⟹ $cdcl_W$*-stgy S S′* |

propagate': *propagate* S S' $\Longrightarrow$ $cdcl_W$*-stgy* S S' |
other': *no-step conflict* S $\Longrightarrow$ *no-step propagate* S $\Longrightarrow$ $cdcl_W$*-o* S S' $\Longrightarrow$ $cdcl_W$*-stgy* S S'

lemma $cdcl_W$*-stgy-*$cdcl_W$: $cdcl_W$*-stgy* S T $\Longrightarrow$ $cdcl_W$ S T
by (*induction rule*: $cdcl_W$*-stgy.induct*) (*auto intro*: $cdcl_W$*.intros*)

lemma $cdcl_W$*-stgy-induct*[*consumes 1*, *case-names conflict propagate decide skip resolve backtrack*]:
assumes
‹$cdcl_W$*-stgy* S T› **and**
‹$\bigwedge T$. *conflict* S T $\Longrightarrow$ P T› **and**
‹$\bigwedge T$. *propagate* S T $\Longrightarrow$ P T› **and**
‹$\bigwedge T$. *no-step conflict* S $\Longrightarrow$ *no-step propagate* S $\Longrightarrow$ *decide* S T $\Longrightarrow$ P T› **and**
‹$\bigwedge T$. *no-step conflict* S $\Longrightarrow$ *no-step propagate* S $\Longrightarrow$ *skip* S T $\Longrightarrow$ P T› **and**
‹$\bigwedge T$. *no-step conflict* S $\Longrightarrow$ *no-step propagate* S $\Longrightarrow$ *resolve* S T $\Longrightarrow$ P T› **and**
‹$\bigwedge T$. *no-step conflict* S $\Longrightarrow$ *no-step propagate* S $\Longrightarrow$ *backtrack* S T $\Longrightarrow$ P T›
shows
‹P T›
using *assms*(*1*) **by** (*induction rule*: $cdcl_W$*-stgy.induct*)
(*auto simp*: *assms*(*2*−) $cdcl_W$*-o.simps* $cdcl_W$*-bj.simps*)

lemma $cdcl_W$*-stgy-cases*[*consumes 1*, *case-names conflict propagate decide skip resolve backtrack*]:
assumes
‹$cdcl_W$*-stgy* S T› **and**
‹*conflict* S T $\Longrightarrow$ P› **and**
‹*propagate* S T $\Longrightarrow$ P› **and**
‹*no-step conflict* S $\Longrightarrow$ *no-step propagate* S $\Longrightarrow$ *decide* S T $\Longrightarrow$ P› **and**
‹*no-step conflict* S $\Longrightarrow$ *no-step propagate* S $\Longrightarrow$ *skip* S T $\Longrightarrow$ P› **and**
‹*no-step conflict* S $\Longrightarrow$ *no-step propagate* S $\Longrightarrow$ *resolve* S T $\Longrightarrow$ P› **and**
‹*no-step conflict* S $\Longrightarrow$ *no-step propagate* S $\Longrightarrow$ *backtrack* S T $\Longrightarrow$ P›
shows
‹P›
using *assms*(*1*) **by** (*cases rule*: $cdcl_W$*-stgy.cases*)
(*auto simp*: *assms*(*2*−) $cdcl_W$*-o.simps* $cdcl_W$*-bj.simps*)

Invariants

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* **by** (*induct rule*: $cdcl_W$*-stgy.induct*) (*blast intro*: $cdcl_W$*-restart-consistent-inv* $cdcl_W$*-restart.intros*)+

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 $cdcl_W$*-stgy-no-more-init-clss*:
assumes $cdcl_W$*-stgy* S S'
shows *init-clss* S = *init-clss* S'
using *assms* $cdcl_W$*-*$cdcl_W$*-restart* $cdcl_W$*-restart-init-clss* $cdcl_W$*-stgy-*$cdcl_W$ **by** *blast*

lemma *rtranclp-*$cdcl_W$*-stgy-no-more-init-clss*:
assumes $cdcl_W$*-stgy*** S 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*)

Literal of highest level in conflicting clauses

One important property of the $cdcl_W$*-restart* 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.

definition *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*)

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

Literal of highest level in decided literals

definition *mark-is-false-with-level* :: *'st* ⇒ *bool* **where**
mark-is-false-with-level S' ≡
∀ *D M1 M2 L*. *M1* @ *Propagated L D* # *M2* = *trail S'* ⟶ *D* − {#*L*#} ≠ {#}
⟶ (∃ *L*. *L* ∈# *D* ∧ *get-level* (*trail S'*) *L* = *count-decided M1*)

lemma $backtrack_W$*-rule*:
assumes
confl: ‹*conflicting S* = *Some* (*add-mset 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**
max-lev: ‹*get-level* (*trail S*) *L* = *get-maximum-level* (*trail S*) (*add-mset L D*)› **and**
max-D: ‹*get-maximum-level* (*trail S*) *D* ≡ *i*› **and**
lev-K: ‹*get-level* (*trail S*) *K* = *i* + *1*› **and**
T: ‹*T* ∼ *cons-trail* (*Propagated L* (*add-mset L D*))
(*reduce-trail-to M1*
(*add-learned-cls* (*add-mset L D*)
(*update-conflicting None S*)))› **and**
lev-inv: $cdcl_W$*-M-level-inv S* **and**
conf: ‹$cdcl_W$*-conflicting S*› **and**
learned: ‹$cdcl_W$*-learned-clause S*›
shows ‹*backtrack S T*›
using *confl decomp lev-L max-lev max-D lev-K*
proof (*rule backtrack-rule*)
let *?i* = *get-maximum-level* (*trail S*) *D*
let *?D* = ‹*add-mset L D*›
show ‹*D* ⊆# *D*›
by *simp*
obtain *M3* **where**
M3: ‹*trail S* = *M3* @ *M2* @ *Decided K* # *M1*›
using *decomp* **by** *auto*
have *trail-S-D*: ‹*trail S* ⊨*as CNot ?D*›
using *conf confl* **unfolding** $cdcl_W$*-conflicting-def* **by** *auto*
then have *atms-E-M1*: ‹*atms-of D* ⊆ *atm-of* ‘ *lits-of-l M1*›
using *backtrack-atms-of-D-in-M1*[*OF - - decomp*, *of D ?i L ?D*
‹*cons-trail* (*Propagated L ?D*) (*reduce-trail-to M1* (*add-learned-cls ?D* (*update-conflicting None S*)))›
‹*Propagated L* (*add-mset L D*)›]
conf lev-K decomp max-lev lev-L confl T max-D lev-inv **unfolding** $cdcl_W$*-M-level-inv-def*
by *auto*
have *n-d*: ‹*no-dup* (*M3* @ *M2* @ *Decided K* # *M1*)›

using *lev-inv no-dup-rev*[*of* ‹*rev M1* @ *rev M2* @ *rev M3*›, *unfolded rev-append*]
by (*auto simp*: $cdcl_W$*-M-level-inv-def M3*)
then have *n-d'*: ‹*no-dup* (*M3* @ *M2* @ *M1*)›
by *auto*
have *atm-L-M1*: ‹*atm-of L* ∉ *atm-of* ‘ *lits-of-l M1*›
using *lev-L n-d defined-lit-no-dupD*(*2*−*3*)[*of M1 L M3 M2*] *count-decided-ge-get-level*[*of* ‹*Decided K* # *M1*› *L*]
unfolding *M3*
by (*auto simp*: *atm-of-eq-atm-of Decided-Propagated-in-iff-in-lits-of-l get-level-cons-if split*: *if-splits*)

have ‹*La* ≠ *L*›‹− *La* ∉ *lits-of-l M3*› ‹− *La* ∉ *lits-of-l M2*› ‹−*La* ≠*K*› **if** ‹*La*∈#*D*› **for** *La*
proof −
have ‹−*La* ∈ *lits-of-l* (*trail S*)›
using *trail-S-D that* **by** (*auto simp*: *true-annots-true-cls-def-iff-negation-in-model dest*!: *get-all-ann-decomposition-exists-prepend*)
moreover have ‹*defined-lit M1 La*›
using *atms-E-M1 that* **by** (*auto simp*: *Decided-Propagated-in-iff-in-lits-of-l atms-of-def dest*!: *atm-of-in-atm-of-set-in-uminus*)
moreover have *n-d'*: ‹*no-dup* (*rev M1* @ *rev M2* @ *rev M3*)›
by (*rule same-mset-no-dup-iff* [*THEN iffD1*, *OF* - *n-d'*]) *auto*
moreover have ‹*no-dup* (*rev M3* @ *rev M2* @ *rev M1*)›
by (*rule same-mset-no-dup-iff* [*THEN iffD1*, *OF* - *n-d'*]) *auto*
ultimately show ‹*La* ≠ *L*›‹− *La* ∉ *lits-of-l M3*› ‹− *La* ∉ *lits-of-l M2*› ‹−*La* ≠ *K*›
using *defined-lit-no-dupD*(*2*−*3*)[*of* ‹*rev M1*› *La* ‹*rev M3*› ‹*rev M2*›]
defined-lit-no-dupD(*1*)[*of* ‹*rev M1*› *La* ‹*rev M3* @ *rev M2*›] *atm-L-M1 n-d*
by (*auto simp*: *M3 Decided-Propagated-in-iff-in-lits-of-l atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set*)
qed

show ‹*clauses S* ⊨*pm add-mset L D*›
using $cdcl_W$*-learned-clause-alt-def confl learned* **by** *blast*

show ‹*T* ∼ *cons-trail* (*Propagated L* (*add-mset L D*)) (*reduce-trail-to M1* (*add-learned-cls* (*add-mset L D*) (*update-conflicting None S*)))›
using *T* **by** *blast*
qed

lemma *backtrack-no-decomp*:
assumes
S: *conflicting S* = *Some* (*add-mset L E*) **and**
L: *get-level* (*trail S*) *L* = *backtrack-lvl S* **and**
D: *get-maximum-level* (*trail S*) *E* < *backtrack-lvl S* **and**
bt: *backtrack-lvl S* = *get-maximum-level* (*trail S*) (*add-mset L E*) **and**
lev-inv: $cdcl_W$*-M-level-inv S* **and**
conf: ‹$cdcl_W$*-conflicting S*› **and**
learned: ‹$cdcl_W$*-learned-clause S*›
shows ∃ *S'*. $cdcl_W$*-o S S'* ∃ *S'*. *backtrack S S'*
proof −
have *L-D*: *get-level* (*trail S*) *L* = *get-maximum-level* (*trail S*) (*add-mset L E*)
using *L D bt* **by** (*simp add*: *get-maximum-level-plus*)
let *?i* = *get-maximum-level* (*trail S*) *E*
let *?D* = ‹*add-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* = *?i* + *1*
using *backtrack-ex-decomp*[*of S ?i*] *D S lev-inv*
unfolding $cdcl_W$*-M-level-inv-def* **by** *auto*

show ‹*Ex* (*backtrack S*)›
  using *backtrack*$_W$*-rule*[*OF S K L L-D - lev-K*] *lev-inv conf learned* **by** *auto*
then show ‹*Ex* (*cdcl*$_W$*-o S*)›
  using *bj* **by** (*auto simp*: *cdcl*$_W$*-bj.simps*)
qed

lemma *no-analyse-backtrack-Ex-simple-backtrack*:
assumes
bt: ‹*backtrack S T*› **and**
lev-inv: *cdcl*$_W$*-M-level-inv S* **and**
conf: ‹*cdcl*$_W$*-conflicting S*› **and**
learned: ‹*cdcl*$_W$*-learned-clause S*› **and**
no-dup: ‹*distinct-cdcl*$_W$*-state S*› **and**
ns-s: ‹*no-step skip S*› **and**
ns-r: ‹*no-step resolve S*›
shows ‹*Ex*(*simple-backtrack S*)›
proof −
obtain *D L K i M1 M2 D'* **where**
confl: *conflicting S* = *Some* (*add-mset L D*) **and**
decomp: (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*)) **and**
lev: *get-level* (*trail S*) *L* = *backtrack-lvl S* **and**
max: *get-level* (*trail S*) *L* = *get-maximum-level* (*trail S*) (*add-mset L D'*) **and**
max-D: *get-maximum-level* (*trail S*) *D'* ≡ *i* **and**
lev-K: *get-level* (*trail S*) *K* = *Suc i* **and**
D'-D: ‹*D'* ⊆# *D*› **and**
NU-DL: ‹*clauses S* ⊨*pm add-mset L D'*› **and**
T: *T* ∼ *cons-trail* (*Propagated L* (*add-mset L D'*))
(*reduce-trail-to M1*
(*add-learned-cls* (*add-mset L D'*)
(*update-conflicting None S*)))
using *bt* **by** (*elim backtrackE*) *metis*
have *n-d*: ‹*no-dup* (*trail S*)›
using *lev-inv* **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *auto*
have *trail-S-Nil*: ‹*trail S* ≠ []›
using *decomp* **by** *auto*
then have *hd-in-annot*: ‹*lit-of* (*hd-trail S*) ∈# *mark-of* (*hd-trail S*)› **if** ‹*is-proped* (*hd-trail S*)›
using *conf that* **unfolding** *cdcl*$_W$*-conflicting-def*
by (*cases* ‹*trail S*›; *cases* ‹*hd* (*trail S*)›) *fastforce+*
have *max-D-L-hd*: ‹*get-maximum-level* (*trail S*) *D* < *get-level* (*trail S*) *L* ∧ *L* = −*lit-of* (*hd-trail S*)›
proof *cases*
assume *is-p*: ‹*is-proped* (*hd* (*trail S*))›
then have ‹−*lit-of* (*hd* (*trail S*)) ∈# *add-mset L D*›
using *ns-s trail-S-Nil confl skip-rule*[*of S* ‹*lit-of* (*hd* (*trail S*))› - - ‹*add-mset L D*›]
by (*cases* ‹*trail S*›; *cases* ‹*hd* (*trail S*)›) *auto*
then have ‹*get-maximum-level* (*trail S*) (*remove1-mset* (− *lit-of* (*hd-trail S*)) (*add-mset L D*)) ≠ *backtrack-lvl S*›
using *ns-r trail-S-Nil confl resolve-rule*[*of S* ‹*lit-of* (*hd* (*trail S*))› ‹*mark-of* (*hd-trail S*)› ‹*add-mset L D*›] *is-p*
hd-in-annot
by (*cases* ‹*trail S*›; *cases* ‹*hd* (*trail S*)›) *auto*
then have *lev-L-D*: ‹*get-maximum-level* (*trail S*) (*remove1-mset* (− *lit-of* (*hd-trail S*)) (*add-mset L D*)) <
backtrack-lvl S›
using *count-decided-ge-get-maximum-level*[*of* ‹*trail S*› ‹*remove1-mset* (− *lit-of* (*hd-trail S*)) (*add-mset L D*)›]
by *auto*

then have ‹$L = -lit\text{-}of\ (hd\text{-}trail\ S)$›
using *get-maximum-level-ge-get-level*[*of L* ‹*remove1-mset* (− *lit-of* (*hd-trail S*)) (*add-mset L D*)›
‹*trail S*›] *lev* **apply** −
by (*rule ccontr*) *auto*
then show *?thesis*
using *lev-L-D lev* **by** *auto*
next
assume *is-p*: ‹¬ *is-proped* (*hd* (*trail S*))›
obtain *L′* **where**
L′: ‹*L′* ∈# *add-mset L D*› **and**
lev-L′: ‹*get-level* (*trail S*) *L′* = *backtrack-lvl S*›
using *lev* **by** *auto*
moreover have *uL′-trail*: ‹−*L′* ∈ *lits-of-l* (*trail S*)›
using *conf confl L′* **unfolding** $cdcl_W$*-conflicting-def true-annots-true-cls-def-iff-negation-in-model*
by *auto*
moreover have ‹*L′* ∉ *lits-of-l* (*trail S*)›
using *n-d uL′-trail* **by** (*blast dest*: *no-dup-consistentD*)
ultimately have *L′-hd*: ‹*L′* = −*lit-of* (*hd-trail S*)›
using *is-p trail-S-Nil* **by** (*cases* ‹*trail S*›; *cases* ‹*hd* (*trail S*)›)
(*auto simp*: *get-level-cons-if atm-of-eq-atm-of*
split: *if-splits*)
have ‹*distinct-mset* (*add-mset L D*)›
using *no-dup confl* **unfolding** *distinct-*$cdcl_W$*-state-def* **by** *auto*
then have ‹*L′* ∉# *remove1-mset L′* (*add-mset L D*)›
using *L′* **by** (*meson distinct-mem-diff-mset multi-member-last*)
moreover have ‹−*L′* ∉# *add-mset L D*›
proof (*rule ccontr*)
assume ‹¬ *?thesis*›
then have ‹*L′* ∈ *lits-of-l* (*trail S*)›
using *conf confl trail-S-Nil* **unfolding** $cdcl_W$*-conflicting-def true-annots-true-cls-def-iff-negation-in-model*
by *auto*
then show *False*
using *n-d L′-hd* **by** (*cases* ‹*trail S*›; *cases* ‹*hd* (*trail S*)›)
(*auto simp*: *Decided-Propagated-in-iff-in-lits-of-l*)
qed
ultimately have ‹*atm-of* (*lit-of* (*Decided* (− *L′*))) ∉ *atms-of* (*remove1-mset L′* (*add-mset L D*))›
using ‹− *L′* ∉# *add-mset L D*› **by** (*auto simp*: *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set*
atms-of-def dest: *in-diffD*)
then have ‹*get-maximum-level* (*Decided* (−*L′*) # *tl* (*trail S*)) (*remove1-mset L′* (*add-mset L D*)) =
get-maximum-level (*tl* (*trail S*)) (*remove1-mset L′* (*add-mset L D*))›
by (*rule get-maximum-level-skip-first*)
also have ‹*get-maximum-level* (*tl* (*trail S*)) (*remove1-mset L′* (*add-mset L D*)) < *backtrack-lvl S*›
using *count-decided-ge-get-maximum-level*[*of* ‹*tl* (*trail S*)› ‹*remove1-mset L′* (*add-mset L D*)›]
trail-S-Nil is-p **by** (*cases* ‹*trail S*›; *cases* ‹*hd* (*trail S*)›) *auto*
finally have *lev-L′-L*: ‹*get-maximum-level* (*trail S*) (*remove1-mset L′* (*add-mset L D*)) < *backtrack-lvl*
S›
using *trail-S-Nil is-p L′-hd* **by** (*cases* ‹*trail S*›; *cases* ‹*hd* (*trail S*)›) *auto*
then have ‹*L* = *L′*›
using *get-maximum-level-ge-get-level*[*of L* ‹*remove1-mset L′* (*add-mset L D*)›
‹*trail S*›] *L′ lev-L′ lev* **by** *auto*
then show *?thesis*
using *lev-L′-L lev L′-hd* **by** *auto*
qed
let *?i* = ‹*get-maximum-level* (*trail S*) *D*›
obtain *K′ M1′ M2′* **where**
decomp′: ‹(*Decided K′* # *M1′*, *M2′*) ∈ *set* (*get-all-ann-decomposition* (*trail S*))› **and**

lev-K′: ⟨*get-level* (*trail S*) *K′* = *Suc ?i*⟩
using *backtrack-ex-decomp*[*of S ?i*] *lev-inv max-D-L-hd*
unfolding *lev* $cdcl_W$*-M-level-inv-def* **by** *blast*

show *?thesis*
apply *standard*
apply (*rule simple-backtrack-rule*[*of S L D K′ M1′ M2′* ⟨*get-maximum-level* (*trail S*) *D*⟩
⟨*cons-trail* (*Propagated L* (*add-mset L D*)) (*reduce-trail-to M1′* (*add-learned-cls* (*add-mset L D*)
(*update-conflicting None S*)))⟩])
subgoal using *confl* **by** *auto*
subgoal using *decomp′* **by** *auto*
subgoal using *lev* **.**
subgoal using *count-decided-ge-get-maximum-level*[*of* ⟨*trail S*⟩ *D*] *lev*
by (*auto simp*: *get-maximum-level-add-mset*)
subgoal .
subgoal using *lev-K′* **by** *simp*
subgoal by *simp*
done
qed

lemma *trail-begins-with-decided-conflicting-exists-backtrack*:
assumes
confl-k: ⟨*conflict-is-false-with-level S*⟩ **and**
conf: ⟨$cdcl_W$*-conflicting S*⟩ **and**
level-inv: ⟨$cdcl_W$*-M-level-inv S*⟩ **and**
no-dup: ⟨*distinct-*$cdcl_W$*-state S*⟩ **and**
learned: ⟨$cdcl_W$*-learned-clause S*⟩ **and**
alien: ⟨*no-strange-atm S*⟩ **and**
tr-ne: ⟨*trail S* ≠ []⟩ **and**
L′: ⟨*hd-trail S* = *Decided L′*⟩ **and**
nempty: ⟨*C* ≠ {#}⟩ **and**
confl: ⟨*conflicting S* = *Some C*⟩
shows ⟨*Ex* (*backtrack S*)⟩ **and** ⟨*no-step skip S*⟩ **and** ⟨*no-step resolve S*⟩
proof −
let *?M* = *trail S*
let *?N* = *init-clss S*
let *?k* = *backtrack-lvl S*
let *?U* = *learned-clss S*
obtain *L D* **where**
E′[*simp*]: *C* = *D* + {#*L*#} **and**
lev-L: *get-level ?M L* = *?k*
using *nempty confl* **by** (*metis* (*mono-tags*) *confl-k insert-DiffM2 conflict-is-false-with-level-def*)

let *?D* = *D* + {#*L*#}
have *?D* ≠ {#} **by** *auto*
have *?M* ⊨*as CNot ?D* **using** *confl conf* **unfolding** $cdcl_W$*-conflicting-def* **by** *auto*
then have *?M* ≠ [] **unfolding** *true-annots-def Ball-def true-annot-def true-cls-def* **by** *force*
define *M′* **where** *M′*: ⟨*M′* = *tl ?M*⟩
have *M*: *?M* = *hd ?M* # *M′* **using** ⟨*?M* ≠ []⟩ *list.collapse M′* **by** *fastforce*

obtain *k′* **where** *k′*: *k′* + *1* = *?k*
using *level-inv tr-ne L′* **unfolding** $cdcl_W$*-M-level-inv-def* **by** (*cases trail S*) *auto*

have *n-s*: *no-step conflict S no-step propagate S*
using *confl* **by** (*auto elim*!: *conflictE propagateE*)

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*
have *L′-L*: $L' = -L$
proof (*rule ccontr*)
  assume ¬ *?thesis*
  moreover {
    have $-L$ ∈ *lits-of-l ?M*
      using *confl conf* **unfolding** $cdcl_W$*-conflicting-def* **by** *auto*
    then have ⟨*atm-of L* ≠ *atm-of L′*⟩
      using $cdcl_W$*-M-level-inv-decomp*(*2*)[*OF level-inv*] *M calculation L′*
      by (*auto simp*: *atm-of-eq-atm-of all-conj-distrib uminus-lit-swap lits-of-def no-dup-def*) }
  ultimately have *get-level* (*hd* (*trail S*) # *M′*) *L* = *get-level* (*tl ?M*) *L*
    using $cdcl_W$*-M-level-inv-decomp*(*1*)[*OF level-inv*] *M* **unfolding** *consistent-interp-def*
    by (*simp add*: *atm-of-eq-atm-of L′ M′*[*symmetric*])
  moreover {
    have *count-decided* (*trail S*) = *?k*
      using *level-inv* **unfolding** $cdcl_W$*-M-level-inv-def* **by** *auto*
    then have *count*: *count-decided M′* = *?k* − *1*
      using *level-inv M* **by** (*auto simp add*: *L′ M′*[*symmetric*])
    then have *get-level* (*tl ?M*) *L* < *?k*
      using *count-decided-ge-get-level*[*of M′ L*] **unfolding** *k′*[*symmetric*] *M′* **by** *auto* }
  finally show *False* **using** *lev-L M* **unfolding** *M′* **by** *auto*
qed
then have *L*: *hd ?M* = *Decided* (−*L*) **using** *L′* **by** *auto*
have *H*: *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 confl*
   by (*metis E′ 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 *get-level* (*tl* (*trail S*)) *L* = *0*
        using *level-inv L′ M* **unfolding** $cdcl_W$*-M-level-inv-def*
        by (*auto simp*: *image-iff L′ L′-L*)
    moreover {
      have ⟨*backtrack-lvl S* = *count-decided* (*hd ?M* # *tl ?M*)⟩
        unfolding *M*[*symmetric*] *M′*[*symmetric*] **..**
      then have *get-level* (*tl* (*trail S*)) *L″* < *backtrack-lvl S*
        using *count-decided-ge-get-level*[*of* ⟨*tl* (*trail S*)⟩ *L″*]
        by (*auto simp*: *image-iff L′ L′-L*) }
    ultimately show *False*
      using *M*[*unfolded L′ M′*[*symmetric*]] *L-k* **by** (*auto simp*: *L′ L′-L*)
  qed
  then have *taut*: *tautology* (*D* + {#*L*#})
    using ⟨*L″* ∈# *D*⟩ **by** (*metis add.commute mset-subset-eqD mset-subset-eq-add-left*
        multi-member-this tautology-minus)
  moreover have *consistent-interp* (*lits-of-l ?M*)

```
      using level-inv unfolding cdcl_W-M-level-inv-def by auto
    ultimately have ¬?M ⊨as CNot ?D
      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 conf unfolding cdcl_W-conflicting-def by auto
    ultimately show False by blast
  qed
  have confl-D: ⟨conflicting S = Some (add-mset L D)⟩
    using confl[unfolded E'] by simp
  have get-maximum-level (trail S) D < get-maximum-level (trail S) (add-mset L D)
    using H by (auto simp: get-maximum-level-plus lev-L max-def get-maximum-level-add-mset)
  moreover have backtrack-lvl S = get-maximum-level (trail S) (add-mset L D)
    using H by (auto simp: get-maximum-level-plus lev-L max-def get-maximum-level-add-mset)
  ultimately show ⟨Ex (backtrack S)⟩
    using backtrack-no-decomp[OF confl-D - ] level-inv alien conf learned
    by (auto simp add: lev-L max-def n-s)

  show ⟨no-step resolve S⟩
    using L by (auto elim!: resolveE)
  show ⟨no-step skip S⟩
    using L by (auto elim!: skipE)
qed

lemma conflicting-no-false-can-do-step:
  assumes
    confl: ⟨conflicting S = Some C⟩ and
    nempty: ⟨C ≠ {#}⟩ and
    confl-k: ⟨conflict-is-false-with-level S⟩ and
    conf: ⟨cdcl_W-conflicting S⟩ and
    level-inv: ⟨cdcl_W-M-level-inv S⟩ and
    no-dup: ⟨distinct-cdcl_W-state S⟩ and
    learned: ⟨cdcl_W-learned-clause S⟩ and
    alien: ⟨no-strange-atm S⟩ and
    termi: ⟨no-step cdcl_W-stgy S⟩
  shows False
proof -
  let ?M = trail S
  let ?N = init-clss S
  let ?k = backtrack-lvl S
  let ?U = learned-clss S
  define M' where ⟨M' = tl ?M⟩
  obtain L D where
    E'[simp]: C = D + {#L#} and
    lev-L: get-level ?M L = ?k
    using nempty confl by (metis (mono-tags) confl-k insert-DiffM2 conflict-is-false-with-level-def)
  let ?D = D + {#L#}
  have ?D ≠ {#} by auto
  have ?M ⊨as CNot ?D using confl conf 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 ≠ []⟩ by fastforce
  then have M: ?M = hd ?M # M' unfolding M'-def .

  have n-s: no-step conflict S no-step propagate S
    using termi by (blast intro: cdcl_W-stgy.intros)+
  have ⟨no-step backtrack S⟩
```

using *termi* **by** (*blast intro*: *cdcl$_W$-stgy.intros cdcl$_W$-o.intros cdcl$_W$-bj.intros*)
then have *not-is-decided*: $\neg$ *is-decided* (*hd ?M*)
using *trail-begins-with-decided-conflicting-exists-backtrack*(*1*)[*OF confl-k conf level-inv no-dup learned alien* ‹*?M* $\neq$ []› *- nempty confl*] **by** (*cases* ‹*hd-trail S*›) (*auto*)
have *g-k*: *get-maximum-level* (*trail S*) *D* $\leq$ *?k*
using *count-decided-ge-get-maximum-level*[*of ?M*] *level-inv* **unfolding** *cdcl$_W$-M-level-inv-def*
by *auto*

let *?D* = *add-mset L D*
have *?D* $\neq$ {#} **by** *auto*
have *?M* $\models$*as CNot ?D* **using** *confl conf* **unfolding** *cdcl$_W$-conflicting-def* **by** *auto*
then have *?M* $\neq$ [] **unfolding** *true-annots-def Ball-def true-annot-def true-cls-def* **by** *force*
then obtain *L′ C* **where** *L′C*: *hd-trail S* = *Propagated L′ C*
using *not-is-decided* **by** (*cases hd-trail S*) *auto*
then have *hd ?M* = *Propagated L′ C*
using ‹*?M* $\neq$ []› **by** *fastforce*
then have *M*: *?M* = *Propagated L′ C* # *M′* **using** *M* **by** *simp*
then have *M′*: *?M* = *Propagated L′ C* # *tl ?M* **using** *M* **by** *simp*
then obtain *C′* **where** *C′*: *C* = *add-mset L′ C′*
using *conf M* **unfolding** *cdcl$_W$-conflicting-def* **by** (*metis append-Nil diff-single-eq-union*)
have *L′D*: $-$*L′* $\in$# *?D*
using *n-s alien level-inv termi skip-rule*[*OF M′ confl*]
by (*auto dest*: *other′ cdcl$_W$-o.intros cdcl$_W$-bj.intros*)

obtain *D′* **where** *D′*: *?D* = *add-mset* ($-$*L′*) *D′* **using** *L′D* **by** (*metis insert-DiffM*)
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 consider
(*D′-max-lvl*) *get-maximum-level* (*trail S*) *D′* = *?k* |
(*D′-le-max-lvl*) *get-maximum-level* (*trail S*) *D′* < *?k*
using *le-neq-implies-less* **by** *blast*
then show *False*
proof *cases*
case *g-D′-k*: *D′-max-lvl*
then have *f1*: *get-maximum-level* (*trail S*) *D′* = *backtrack-lvl S*
using *M* **by** *auto*
then have *Ex* (*cdcl$_W$-o S*)
using *resolve-rule*[*of S L′ C* , *OF* ‹*trail S* $\neq$ []› *- - confl*] *conf*
L′C L′D D′ C′ **by** (*auto dest*: *cdcl$_W$-o.intros cdcl$_W$-bj.intros*)
then show *False*
using *n-s termi* **by** (*auto dest*: *other′ cdcl$_W$-o.intros cdcl$_W$-bj.intros*)
next
case *a1*: *D′-le-max-lvl*
then have *f3*: *get-maximum-level* (*trail S*) *D′* < *get-level* (*trail S*) ($-$*L′*)
using *a1 lev-L D′* **by** (*metis D′ get-maximum-level-ge-get-level insert-noteq-member not-less*)
moreover have *get-level* (*trail S*) *L′* = *get-maximum-level* (*trail S*) (*D′* + {#$-$ *L′*#})
using *a1* **by** (*auto simp add*: *get-maximum-level-add-mset max-def M*)
ultimately show *False*
using *M backtrack-no-decomp*[*of S* $-$*L′ D′*] *confl level-inv n-s termi E′ learned conf*
by (*auto simp*: *D′ dest*: *other′ cdcl$_W$-o.intros cdcl$_W$-bj.intros*)
qed
qed

lemma *cdcl$_W$-stgy-final-state-conclusive2*:

assumes
  termi: *no-step* $cdcl_W$*-stgy S* **and**
  decomp: *all-decomposition-implies-m* (*clauses 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* (*clauses S*)))
    ∨ (*conflicting S* = *None* ∧ *trail S* ⊨*as set-mset* (*clauses S*))
proof −
  let *?M* = *trail S*
  let *?N* = *clauses 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* = {#}
   using *conflicting-no-false-can-do-step*[*of S*, *OF - - confl-k confl level-inv no-dup learned alien*] *termi*
   by (*cases conflicting S*, *simp*) *auto*
  then show *?thesis*
  proof *cases*
   case (*Some-Empty E*)
   then have *conflicting S* = *Some* {#} **by** *auto*
   then have *unsat-clss-S*: *unsatisfiable* (*set-mset* (*clauses S*))
     using *learned* **unfolding** $cdcl_W$*-learned-clause-alt-def true-clss-cls-def*
      conflict-is-false-with-level-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 simp*: *clauses-def*)
  next
   case *None*
   have *?M* ⊨*asm ?N*
   proof (*rule ccontr*)
     assume *MN*: ¬ *?thesis*
     have *all-defined*: *atm-of* ‘ (*lits-of-l ?M*) = *atms-of-mm ?N* (**is** *?A* = *?B*)
     proof
      show *?A* ⊆ *?B* **using** *alien* **unfolding** *no-strange-atm-def clauses-def* **by** *auto*
      show *?B* ⊆ *?A*
      proof (*rule ccontr*)
        assume ¬*?B* ⊆ *?A*
        then obtain *l* **where** *l* ∈ *?B* **and** *l* ∉ *?A* **by** *auto*
        then have *undefined-lit ?M* (*Pos l*)
         using ‹*l* ∉ *?A*› **unfolding** *lits-of-def* **by** (*auto simp add*: *defined-lit-map*)
        then have ∃ *S′*. $cdcl_W$*-o S S′*
         using $cdcl_W$*-o.decide*[*of S*] *decide-rule*[*of S* ‹*Pos l*› ‹*cons-trail* (*Decided* (*Pos l*)) *S*›]
          ‹*l* ∈ *?B*› *None alien* **unfolding** *clauses-def no-strange-atm-def* **by** *fastforce*
        then show *False*
         using *termi* **by** (*blast intro*: $cdcl_W$*-stgy.intros*)
      qed
     qed
     obtain *D* **where** ¬ *?M* ⊨*a D* **and** *D* ∈# *?N*
      using *MN* **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** *all-defined atms-of-ms-def* **by** *auto*

then have *total-over-m* (*lits-of-l ?M*) $\{D\}$
using *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set*
by (*fastforce simp*: *total-over-set-def*)
then have *?M* $\models$*as CNot D*
using ⟨¬ *trail S* $\models$*a D*⟩ **unfolding** *true-annot-def true-annots-true-cls*
by (*fastforce simp*: *total-not-true-cls-true-clss-CNot*)
then have $\exists S'$. *conflict S S'*
using ⟨*trail S* $\models$*as CNot D*⟩ ⟨*D* ∈# *clauses S*⟩
None **unfolding** *clauses-def* **by** (*auto simp*: *conflict.simps clauses-def*)
then show *False*
using *termi* **by** (*blast intro*: $cdcl_W$*-stgy.intros*)
qed
then show *?thesis*
using *None* **by** *auto*
qed
qed

lemma $cdcl_W$*-stgy-final-state-conclusive*:
assumes
termi: *no-step* $cdcl_W$*-stgy S* **and**
decomp: *all-decomposition-implies-m* (*clauses 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* **and**
learned-entailed: ⟨$cdcl_W$*-learned-clauses-entailed-by-init S*⟩
shows (*conflicting S* = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss S*)))
∨ (*conflicting S* = *None* ∧ *trail S* $\models$*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* = {#}
using *conflicting-no-false-can-do-step*[*of S*, *OF - - confl-k confl level-inv no-dup learned alien*] *termi*
by (*cases conflicting S*, *simp*) *auto*
then show *?thesis*
proof *cases*
case (*Some-Empty E*)
then have *conflicting S* = *Some* {#} **by** *auto*
then have *unsat-clss-S*: *unsatisfiable* (*set-mset* (*clauses S*))
using *learned learned-entailed* **unfolding** $cdcl_W$*-learned-clause-alt-def true-clss-cls-def*
conflict-is-false-with-level-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 have *unsatisfiable* (*set-mset* (*init-clss S*))
proof −
have *atms-of-mm* (*learned-clss S*) ⊆ *atms-of-mm* (*init-clss S*)
using *alien no-strange-atm-decomp*(*3*) **by** *blast*
then have *f3*: *atms-of-ms* (*set-mset* (*init-clss S*) ∪ *set-mset* (*learned-clss S*)) =
atms-of-mm (*init-clss S*)
by *auto*

```
      have init-clss S ⊨psm learned-clss S
        using learned-entailed
        unfolding cdclW-learned-clause-alt-def cdclW-learned-clauses-entailed-by-init-def by blast
      then show ?thesis
        using f3 unsat-clss-S
        unfolding true-clss-clss-def total-over-m-def clauses-def satisfiable-def
        by (metis (no-types) set-mset-union true-clss-union)
    qed
    then show ?thesis using Some-Empty by auto
  next
    case None
    have ?M ⊨asm ?N
    proof (rule ccontr)
      assume MN: ¬ ?thesis
      have all-defined: 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)
          then have ∃S'. cdclW-o S S'
            using cdclW-o.decide decide-rule ‹l ∈ ?B› no-strange-atm-def None
            by (metis literal.sel(1) state-eq-ref)
          then show False
            using termi by (blast intro: cdclW-stgy.intros)
        qed
      qed
      obtain D where ¬ ?M ⊨a D and D ∈# ?N
        using MN 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 all-defined atms-of-ms-def by auto
      then have total-over-m (lits-of-l ?M) {D}
        using atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set
        by (fastforce simp: total-over-set-def)
      then have M-CNot-D: ?M ⊨as CNot D
        using ‹¬ trail S ⊨a D› unfolding true-annot-def true-annots-true-cls
        by (fastforce simp: total-not-true-cls-true-clss-CNot)
      then have ∃S'. conflict S S'
        using M-CNot-D ‹D ∈# init-clss S›
          None unfolding clauses-def by (auto simp: conflict.simps clauses-def)
      then show False
        using termi by (blast intro: cdclW-stgy.intros)
    qed
    then show ?thesis
      using None by auto
  qed
qed


lemma cdclW-stgy-tranclp-cdclW-restart:
  cdclW-stgy S S' ⟹ cdclW-restart⁺⁺ S S'
  by (simp add: cdclW-cdclW-restart cdclW-stgy-cdclW tranclp.r-into-trancl)
```

lemma *tranclp-cdcl$_W$-stgy-tranclp-cdcl$_W$-restart*:
cdcl$_W$-stgy^{++} S S' $\Longrightarrow$ *cdcl$_W$-restart^{++} S S'*
apply (*induct rule*: *tranclp.induct*)
using *cdcl$_W$-stgy-tranclp-cdcl$_W$-restart* **apply** *blast*
by (*meson cdcl$_W$-stgy-tranclp-cdcl$_W$-restart tranclp-trans*)

lemma *rtranclp-cdcl$_W$-stgy-rtranclp-cdcl$_W$-restart*:
cdcl$_W$-stgy** S S' $\Longrightarrow$ *cdcl$_W$-restart** S S'*
using *rtranclp-unfold*[*of cdcl$_W$-stgy S S'*] *tranclp-cdcl$_W$-stgy-tranclp-cdcl$_W$-restart*[*of S S'*] **by** *auto*

lemma *cdcl$_W$-o-conflict-is-false-with-level-inv*:
assumes
cdcl$_W$-o S S' **and**
lev: *cdcl$_W$-M-level-inv S* **and**
confl-inv: *conflict-is-false-with-level S* **and**
n-d: *distinct-cdcl$_W$-state S* **and**
conflicting: *cdcl$_W$-conflicting S*
shows *conflict-is-false-with-level S'*
using *assms*(*1*,*2*)
proof (*induct rule*: *cdcl$_W$-o-induct*)
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 $\neg$ *?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 imageI insertCI list.simps*(*15*) *lit-of.simps*(*2*) *lits-of-def no-dup-consistentD*)
qed
}
ultimately have *g-D*: *get-maximum-level* (*Propagated L C # M*) (*remove1-mset* ($-L$) *D*)
= *get-maximum-level M* (*remove1-mset* ($-L$) *D*)
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*
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* $\cup\#$ *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 get-maximum-level-exists-lit-of-max-level*[*of*

remove1-mset $(- L)$ $D \cup\#$ *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 $\neg$ *?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 $\models$*as CNot D*
using *conflicting tr-S D* **unfolding** *cdcl*$_W$*-conflicting-def* **by** *auto*
then have $-L \in$ *lits-of-l M*
using ⟨*La* $\in\#$ 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 *propagate-high-levelE*:
assumes *propagate S T*
obtains M' N' U L C **where**
state-butlast S = (M', N', U, *None*) **and**
state-butlast T = (*Propagated* L (C + {#L#}) # M', N', U, *None*) **and**
C + {#L#} $\in\#$ *local.clauses S* **and**
M' $\models$*as CNot C* **and**
undefined-lit (*trail S*) L
proof $-$
obtain E L **where**
conf: *conflicting S* = *None* **and**
E: $E \in\#$ *clauses S* **and**
LE: $L \in\#$ E **and**
tr: *trail S* $\models$*as CNot* (E $-$ {#L#}) **and**
undef: *undefined-lit* (*trail S*) L **and**
T: $T \sim$ *cons-trail* (*Propagated L E*) S
using *assms* **by** (*elim propagateE*) *simp*
obtain M N U **where**
S: *state-butlast* S = (M, N, U, *None*)
using *conf* **by** *auto*
show *thesis*
using *that*[*of M N U L remove1-mset L E*] *S T LE E tr undef*
by *auto*
qed

lemma $cdcl_W$*-propagate-conflict-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**
    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 C L* **where**
    Y: *state-butlast Y* = (*M'*, *N'*, *U*, *None*) **and**
    Z: *state-butlast Z* = (*Propagated L* (*C* + {#*L*#}) # *M'*, *N'*, *U*, *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 $cdcl_W$*-stgy-strong-completeness-n*:
  assumes
    MN: *set M* $\models$*s set-mset N* **and**

cons: *consistent-interp* (*set M*) **and**
tot: *total-over-m* (*set M*) (*set-mset N*) **and**
atm-incl: *atm-of* ' (*set M*) ⊆ *atms-of-mm N* **and**
distM: *distinct M* **and**
length: $n \leq$ *length M*
shows
∃*M' S*. *length M'* ≥ *n* ∧
lits-of-l M' ⊆ *set M* ∧
no-dup M' ∧
state-butlast S = (*M'*, *N*, {#}, *None*) ∧
$cdcl_W$*-stgy*** (*init-state N*) *S*
using *length*
proof (*induction n*)
case *0*
have *state-butlast* (*init-state N*) = ([], *N*, {#}, *None*)
by *auto*
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*
ultimately show *?case* **by** *blast*
next
case (*Suc n*) **note** *IH* = *this*(*1*) **and** *n* = *this*(*2*)
then obtain *M' 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-butlast S* = (*M'*, *N*, {#}, *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-*$cdcl_W$*-restart rtranclp-*$cdcl_W$*-stgy-consistent-inv st* **apply** *blast*
using $cdcl_W$*-M-level-inv-S0-*$cdcl_W$*-restart no-strange-atm-S0 rtranclp-*$cdcl_W$*-restart-no-strange-atm-inv rtranclp-*$cdcl_W$*-stgy-rtranclp-*$cdcl_W$*-restart st* **by** *blast*

{ **assume** *no-step*: ¬*no-step propagate S*
then obtain *S'* **where** *S'*: *propagate S S'*
by *auto*
have *lev*: $cdcl_W$*-M-level-inv S'*
using *M S' rtranclp-*$cdcl_W$*-restart-consistent-inv rtranclp-propagate-is-rtranclp-*$cdcl_W$*-restart* **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'* $cdcl_W$*-propagate-conflict-completeness*[*OF assms*(*1*−*3*), *of S*] *M' S*
by (*auto simp*: *comp-def*)
moreover have $cdcl_W$*-stgy S S'* **using** *S'* **by** (*simp add*: $cdcl_W$*-stgy.propagate'*)
moreover {
have *trail S* = *M'*
using *S* **by** (*auto simp*: *comp-def rev-map*)
then have *length* (*trail S'*) > *n*
using *S' l-M'* **by** (*auto elim*: *propagateE*) }

```
  moreover {
    have stS': cdcl_W-stgy** (init-state N) S'
      using st cdcl_W-stgy.propagate'[OF S'] by (auto simp: r-into-rtranclp)
    then have init-clss S' = N
      using 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 S S' by (auto elim: propagateE)
    have state-butlast S' = (trail S', N, {#}, None)
      using S by auto }
  moreover
  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 - S'])
    by auto
}
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 defined-lit-map)
      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* $\in$ *atms-of-mm* (*init-clss S*)
using *atm-incl calculation S* **by** *auto*
ultimately have *dec*: *decide S* (*cons-trail* (*Decided L*) *S*)
using *decide-rule*[*of S - cons-trail* (*Decided L*) *S*] *S* **by** *auto*
let *?S'* = *cons-trail* (*Decided L*) *S*
have *lits-of-l* (*trail ?S'*) $\subseteq$ *set M* **using** *L M' S undef* **by** *auto*
moreover have *no-strange-atm ?S'*
using *alien dec M* **by** (*meson* $cdcl_W$*-restart-no-strange-atm-inv decide other*)
have $cdcl_W$*-M-level-inv ?S'*
using *M dec rtranclp-mono*[*of decide* $cdcl_W$*-restart*] **by** (*meson* $cdcl_W$*-restart-consistent-inv*
decide other)
then have *lev''*: $cdcl_W$*-M-level-inv ?S'*
using *S rtranclp-*$cdcl_W$*-restart-consistent-inv rtranclp-propagate-is-rtranclp-*$cdcl_W$*-restart*
by *blast*
then have *n-d''*: *no-dup* (*trail ?S'*)
unfolding $cdcl_W$*-M-level-inv-def* **by** *auto*
have *length* (*trail S*) $\leq$ *length* (*trail ?S'*) $\wedge$ *lits-of-l* (*trail ?S'*) $\subseteq$ *set M*
using *S L M' S undef* **by** *simp*
then have *Suc n* $\leq$ *length* (*trail ?S'*) $\wedge$ *lits-of-l* (*trail ?S'*) $\subseteq$ *set M*
using *l-M' S undef* **by** *auto*
moreover have *S''*: *state-butlast ?S'* = (*trail ?S'*, *N*, {#}, *None*)
using *S undef n-d'' lev''* **by** *auto*
moreover have $cdcl_W$*-stgy*** (*init-state N*) *?S'*
using *S'' no-step no-confl st dec* **by** (*auto dest*: *decide* $cdcl_W$*-stgy.intros*)
ultimately show *?thesis* **using** *n-d''* **by** *blast*
qed
}
ultimately show *?case* **by** *blast*
qed

lemma $cdcl_W$*-stgy-strong-completeness'*:
assumes
MN: *set M* $\models$*s set-mset N* **and**
cons: *consistent-interp* (*set M*) **and**
tot: *total-over-m* (*set M*) (*set-mset N*) **and**
atm-incl: *atm-of* ' (*set M*) $\subseteq$ *atms-of-mm N* **and**
distM: *distinct M*
shows
$\exists$ *M' S*. *lits-of-l M'* = *set M* $\wedge$
state-butlast S = (*M'*, *N*, {#}, *None*) $\wedge$
$cdcl_W$*-stgy*** (*init-state N*) *S*
proof –
have ‹$\exists$ *M' S*. *lits-of-l M'* $\subseteq$ *set M* $\wedge$
no-dup M' $\wedge$ *length M'* = *n* $\wedge$
state-butlast S = (*M'*, *N*, {#}, *None*) $\wedge$
$cdcl_W$*-stgy*** (*init-state N*) *S*›
if ‹*n* $\leq$ *length M*› **for** *n* :: *nat*
using *that*
proof (*induction n*)
case *0*
then show *?case* **by** (*auto intro*!: *exI*[*of* - ‹*init-state N*›])
next
case (*Suc n*) **note** *IH* = *this*(*1*) **and** *n-le-M* = *this*(*2*)
then obtain *M' S* **where**
M': *lits-of-l M'* $\subseteq$ *set M* **and**
n-d[*simp*]: *no-dup M'* **and**

S: *state-butlast S* = (*M'*, *N*, {#}, *None*) **and**
st: $cdcl_W$*-stgy*** (*init-state N*) *S* **and**
l-M': ‹*length M'* = *n*›
by *auto*
have
M: $cdcl_W$*-M-level-inv S* **and**
alien: *no-strange-atm S*
using $cdcl_W$*-M-level-inv-S0-*$cdcl_W$*-restart rtranclp-*$cdcl_W$*-stgy-consistent-inv st* **apply** *blast*
using $cdcl_W$*-M-level-inv-S0-*$cdcl_W$*-restart no-strange-atm-S0 rtranclp-*$cdcl_W$*-restart-no-strange-atm-inv*
rtranclp-$cdcl_W$*-stgy-rtranclp-*$cdcl_W$*-restart st* **by** *blast*

{ assume *no-step*: ¬*no-step propagate S*
then obtain *S'* **where** *S'*: *propagate S S'*
by *auto*
have *lev*: $cdcl_W$*-M-level-inv S'*
using *M S' rtranclp-*$cdcl_W$*-restart-consistent-inv rtranclp-propagate-is-rtranclp-*$cdcl_W$*-restart* **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'* $cdcl_W$*-propagate-conflict-completeness*[*OF assms*(*1*−*3*), *of S*] *M' S*
by (*auto simp*: *comp-def*)
moreover have $cdcl_W$*-stgy S S'* **using** *S'* **by** (*simp add*: $cdcl_W$*-stgy.propagate'*)
moreover {
have *trail S* = *M'*
using *S* **by** (*auto simp*: *comp-def rev-map*)
then have *length* (*trail S'*) = *Suc n*
using *S' l-M'* **by** (*auto elim*: *propagateE*) **}**
moreover {
have *stS'*: $cdcl_W$*-stgy*** (*init-state N*) *S'*
using *st* $cdcl_W$*-stgy.propagate'*[*OF S'*] **by** (*auto simp*: *r-into-rtranclp*)
then have *init-clss S'* = *N*
using *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 *S S'* **by** (*auto elim*: *propagateE*)
have *state-butlast S'* = (*trail S'*, *N*, {#}, *None*)
using *S* **by** *auto* **}**
moreover
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 - S'*])
by *auto*
}
moreover { assume *no-step*: *no-step propagate S*
have *no-confl*: *no-step conflict S*
proof −
{ fix *D*
assume *D* ∈# *N* **and** *M'* ⊨*as CNot D*

then have *set M* $\models$ *D* **using** *MN* **unfolding** *true-clss-def* **by** *auto*
moreover have *set M* $\models$*s CNot D*
using ‹*M'* $\models$*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 defined-lit-map*)
then have *lits-of-l M'* $\subset$ *set M*
using *M' l-M' lenM n-le-M* **by** *auto*
then obtain *L* **where** *L*: *L* $\in$ *set M* **and** *undef-m*: *L* $\notin$ *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* $\in$ *atms-of-mm* (*init-clss S*)
using *atm-incl calculation S* **by** *auto*
ultimately have *dec*: *decide S* (*cons-trail* (*Decided L*) *S*)
using *decide-rule*[*of S - cons-trail* (*Decided L*) *S*] *S* **by** *auto*
let *?S'* = *cons-trail* (*Decided L*) *S*
have *lits-of-l* (*trail ?S'*) $\subseteq$ *set M* **using** *L M' S undef* **by** *auto*
moreover have *no-strange-atm ?S'*
using *alien dec M* **by** (*meson* $cdcl_W$*-restart-no-strange-atm-inv decide other*)
have $cdcl_W$*-M-level-inv ?S'*
using *M dec rtranclp-mono*[*of decide* $cdcl_W$*-restart*] **by** (*meson* $cdcl_W$*-restart-consistent-inv*
decide other)
then have *lev''*: $cdcl_W$*-M-level-inv ?S'*
using *S rtranclp-*$cdcl_W$*-restart-consistent-inv rtranclp-propagate-is-rtranclp-*$cdcl_W$*-restart*
by *blast*
then have *n-d''*: *no-dup* (*trail ?S'*)
unfolding $cdcl_W$*-M-level-inv-def* **by** *auto*
have *Suc* (*length* (*trail S*)) = *length* (*trail ?S'*) $\wedge$ *lits-of-l* (*trail ?S'*) $\subseteq$ *set M*
using *S L M' S undef* **by** *simp*
then have *Suc n* = *length* (*trail ?S'*) $\wedge$ *lits-of-l* (*trail ?S'*) $\subseteq$ *set M*
using *l-M' S undef* **by** *auto*
moreover have *S''*: *state-butlast ?S'* = (*trail ?S'*, *N*, {#}, *None*)
using *S undef n-d'' lev''* **by** *auto*
moreover have $cdcl_W$*-stgy*** (*init-state N*) *?S'*
using *S'' no-step no-confl st dec* **by** (*auto dest*: *decide* $cdcl_W$*-stgy.intros*)
ultimately have *?case* **using** *n-d'' L M'* **by** (*auto intro*!: *exI*[*of -* ‹*Decided L # trail S*›] *exI*[*of -*
‹*?S'*›])
}
ultimately show *?case* **by** *blast*
qed
from *this*[*of* ‹*length M*›] **obtain** *M' S* **where**
M'-M: ‹*lits-of-l M'* $\subseteq$ *set M*› **and**
n-d: ‹*no-dup M'*› **and**
‹*length M'* = *length M*› **and**
‹*state-butlast S* = (*M'*, *N*, {#}, *None*) $\wedge$ $cdcl_W$*-stgy*** (*init-state N*) *S*›
by *auto*
moreover have ‹*lits-of-l M'* = *set M*›

apply (*rule card-subset-eq*)
subgoal by *auto*
subgoal using *M'-M* **.**
subgoal using *M'-M n-d no-dup-length-eq-card-atm-of-lits-of-l*[*OF n-d*] *M'-M* ‹*finite* (*set M*)› *distinct-card*[*OF distM*] *calculation*(*3*)
card-image-le[*of* ‹ *lits-of-l M'*› *atm-of*] *card-seteq*[*OF* ‹*finite* (*set M*)›, *of* ‹*lits-of-l M'*›]
by *auto*
done
ultimately show *?thesis*
by (*auto intro*!: *exI*[*of - S*])
qed

theorem 2.9.11 page 98 of Weidenbach's book (with strategy)

lemma *cdcl*$_W$*-stgy-strong-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**
atm-incl: *atm-of* ' (*set M*) ⊆ *atms-of-mm N* **and**
distM: *distinct M*
shows
∃ *M' k S*.
lits-of-l M' = *set M* ∧
state-butlast S = (*M'*, *N*, {#}, *None*) ∧
cdcl$_W$*-stgy*** (*init-state N*) *S* ∧
final-cdcl$_W$*-restart-state S*
proof −
from *cdcl*$_W$*-stgy-strong-completeness-n*[*OF assms*, *of length M*]
obtain *M' T* **where**
l: *length M* ≤ *length M'* **and**
M'-M: *lits-of-l M'* ⊆ *set M* **and**
no-dup: *no-dup M'* **and**
T: *state-butlast T* = (*M'*, *N*, {#}, *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 simp*: *lits-of-def image-image no-dup-def*) **}**
moreover have *card* (*lits-of-l M'*) = *card* (*set* ((*map* (λ*l*. *atm-of* (*lit-of l*)) *M'*)))
using *no-dup* **by** (*induction M'*) (*auto simp add*: *defined-lit-map card-insert-if lits-of-def*)
ultimately have *card* (*set M*) ≤ *card* (*lits-of-l M'*) **using** *l* **unfolding** *lits-of-def* **by** *auto*
then have *s*: *set M* = *lits-of-l M'*
using *M'-M card-seteq* **by** *blast*
moreover {
have *M'* ⊨*asm N*
using *MN s* **unfolding** *true-annots-def Ball-def true-annot-def true-clss-def* **by** *auto*
then have *final-cdcl*$_W$*-restart-state T*
using *T no-dup* **unfolding** *final-cdcl*$_W$*-restart-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) ≡
  (∀ M K M' D. trail S = M' @ Decided K # M ⟶ D ∈# clauses S ⟶ ¬M ⊨as CNot D)

lemma no-smaller-confl-init-sate[simp]:
  no-smaller-confl (init-state N) unfolding no-smaller-confl-def by auto

lemma cdcl_W-o-no-smaller-confl-inv:
  fixes S S' :: 'st
  assumes
    cdcl_W-o S S' and
    n-s: no-step conflict S and
    lev: cdcl_W-M-level-inv S and
    max-lev: conflict-is-false-with-level S and
    smaller: no-smaller-confl S
  shows no-smaller-confl S'
  using assms(1,2) unfolding no-smaller-confl-def
proof (induct rule: 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 trail T = M'' @ Decided K # M' and D: Da ∈# local.clauses T
    then have trail S = tl M'' @ Decided K # M'
        ∨ (M'' = [] ∧ Decided K # M' = Decided L # trail S)
      using T undef by (cases M'') auto
    moreover {
      assume trail S = tl M'' @ Decided K # M'
      then have ¬M' ⊨as CNot Da
        using D T undef confl smaller unfolding no-smaller-confl-def smaller by fastforce
    }
    moreover {
      assume Decided K # M' = Decided L # trail S
      then have ¬M' ⊨as CNot Da using smaller D confl T n-s by (auto simp: conflict.simps)
    }
    ultimately show ¬M' ⊨as CNot Da by fast
  qed
next
  case resolve
  then show ?case using smaller max-lev unfolding no-smaller-confl-def by auto
next
  case skip
  then show ?case using smaller max-lev unfolding no-smaller-confl-def by auto
next
  case (backtrack L D K i M1 M2 T D') note confl = this(1) and decomp = this(2) and
    T = this(9)
  obtain c where M: trail S = c @ M2 @ Decided K # M1
    using decomp by auto

  show ?case
```

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

lemma 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'*: *trail S' = M'' @ Decided K # M'*
and *D* ∈# *clauses S'*
obtain *M N U C L* **where**
S: *state-butlast S* = (*M*, *N*, *U*, *None*) **and**
S': *state-butlast S'* = (*Propagated L* (*C* + {#*L*#}) # *M*, *N*, *U*, *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*) *annotated-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$_W$-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*
shows *no-smaller-confl S'*
using *assms*
proof (*induct rule*: *cdcl$_W$-stgy.induct*)
case (*conflict' S'*)
then show *?case* **using** *conflict-no-smaller-confl-inv*[*of S S'*] **by** *blast*
next
case (*propagate' S'*)
then show *?case* **using** *propagate-no-smaller-confl-inv*[*of S S'*] **by** *blast*
next
case (*other' S'*)
then show *?case*
using *cdcl$_W$-o-no-smaller-confl-inv*[*of S*] **by** *auto*
qed

lemma *conflict-conflict-is-false-with-level*:
assumes
conflict: *conflict S T* **and**
smaller: *no-smaller-confl S* **and**
M-lev: *cdcl$_W$-M-level-inv S*
shows *conflict-is-false-with-level T*
using *conflict*
proof (*cases rule*: *conflict.cases*)
case (*conflict-rule D*) **note** *confl* = *this*(*1*) **and** *D* = *this*(*2*) **and** *not-D* = *this*(*3*) **and** *T* = *this*(*4*)
then have [*simp*]: *conflicting T* = *Some D*
by *auto*
have *M-lev-T*: *cdcl$_W$-M-level-inv T*
using *conflict M-lev* **by** (*auto simp*: *cdcl$_W$-restart-consistent-inv*
dest: *cdcl$_W$-restart.intros*)
then have *bt*: *backtrack-lvl T* = *count-decided* (*trail T*)
unfolding *cdcl$_W$-M-level-inv-def* **by** *auto*
have *n-d*: *no-dup* (*trail T*)
using *M-lev-T* **unfolding** *cdcl$_W$-M-level-inv-def* **by** *auto*
show *?thesis*
proof (*rule ccontr*, *clarsimp*)

```
    assume
      empty: D ≠ {#} and
      lev: ∀ L∈#D. get-level (trail T) L ≠ backtrack-lvl T
    moreover {
      have get-level (trail T) L ≤ backtrack-lvl T if L∈#D for L
        using that count-decided-ge-get-level[of trail T L] M-lev-T
        unfolding cdcl_W-M-level-inv-def by auto
      then have get-level (trail T) L < backtrack-lvl T if L∈#D for L
        using lev that by fastforce } note lev' = this
    ultimately have count-decided (trail T) > 0
      using M-lev-T unfolding cdcl_W-M-level-inv-def by (cases D) fastforce+
    then have ex: ‹∃ x∈set (trail T). is-decided x›
      unfolding no-dup-def count-decided-def by cases auto
    have ‹∃ M2 L M1. trail T = M2 @ Decided L # M1 ∧ (∀ m∈set M2. ¬ is-decided m)›
      by (rule split-list-first-propE[of trail T is-decided, OF ex])
        (force elim!: is-decided-ex-Decided)
    then obtain M2 L M1 where
      tr-T: trail T = M2 @ Decided L # M1 and nm: ∀ m ∈ set M2. ¬ is-decided m
      by blast
    moreover {
      have get-level (trail T) La = backtrack-lvl T if − La ∈ lits-of-l M2 for La
        unfolding tr-T bt
        apply (subst get-level-skip-end)
        using that apply (simp add: atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set
            Decided-Propagated-in-iff-in-lits-of-l; fail)
        using nm bt tr-T by (simp add: count-decided-0-iff) }
    moreover {
      have tr: M2 @ Decided L # M1 = (M2 @ [Decided L]) @ M1
        by auto
      have get-level (trail T) L = backtrack-lvl T
        using n-d nm unfolding tr-T tr bt
        by (auto simp: image-image atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set
            atm-lit-of-set-lits-of-l count-decided-0-iff[symmetric]) }
    moreover have trail S = trail T
      using T by auto
    ultimately have M1 ⊨as CNot D
      using lev' not-D unfolding true-annots-true-cls-def-iff-negation-in-model
      by (force simp: count-decided-0-iff[symmetric] get-level-def)
    then show False
      using smaller T tr-T D by (auto simp: no-smaller-confl-def)
  qed
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
    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')
  then have no-smaller-confl S'
```

    using *conflict'.hyps conflict-no-smaller-confl-inv n-l* **by** *blast*
  moreover have *conflict-is-false-with-level S'*
    using *conflict-conflict-is-false-with-level assms(4) conflict'.hyps n-l* **by** *blast*
  then show *?case* **by** *blast*
next
  case (*propagate' S'*)
  then show *?case* **by** (*auto elim*: *propagateE*)
next
  case (*other' S'*) **note** *n-s* = *this*(*1*,*2*) **and** *o* = *this*(*3*) **and** *lev* = *this*(*6*)
  show *?case*
    using $cdcl_W$*-o-conflict-is-false-with-level-inv*[*OF o*] *other'.prems* **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**
    dist: *distinct-*$cdcl_W$*-state S* **and**
    conflicting: $cdcl_W$*-conflicting S* **and**
    decomp: *all-decomposition-implies-m* (*clauses 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-*$cdcl_W$*-restart*
    by (*blast intro*: *rtranclp-*$cdcl_W$*-restart-consistent-inv*)+
  moreover have *distinct-*$cdcl_W$*-state S'*
    using *rtanclp-distinct-*$cdcl_W$*-state-inv*[*of S S'*] *lev rtranclp-*$cdcl_W$*-stgy-rtranclp-*$cdcl_W$*-restart*[*OF st*]
    dist **by** *auto*
  moreover have $cdcl_W$*-conflicting S'*
    using *rtranclp-*$cdcl_W$*-restart-all-inv*(*6*)[*of S S'*] *st alien conflicting decomp dist learned lev*
    rtranclp-$cdcl_W$*-stgy-rtranclp-*$cdcl_W$*-restart* **by** *blast*
  ultimately show *?case*
    using $cdcl_W$*-stgy-no-smaller-confl*[*OF cdcl*] $cdcl_W$*-stgy-ex-lit-of-max-level*[*OF cdcl*] *cdcl*
    by (*auto simp del*: *simp add*: $cdcl_W$*-stgy.simps elim!*: *propagateE*)
qed

Final States are Conclusive

theorem 2.9.9 page 97 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* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss S'*)))

$\vee$ (*conflicting* S' = *None* $\wedge$ *trail* S' $\models$*asm init-clss* S')
proof −
let *?S* = *init-state N*
have
termi: $\forall S''$. $\neg$*cdcl*$_W$*-stgy* S' S'' **and**
step: *cdcl*$_W$*-stgy*** *?S* S' **using** *full* **unfolding** *full-def* **by** *auto*
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* (*clauses* S') (*get-all-ann-decomposition* (*trail* S'))
using *no-d tranclp-cdcl*$_W$*-stgy-tranclp-cdcl*$_W$*-restart*[*of ?S* S'] *step*
rtranclp-cdcl$_W$*-restart-all-inv*(*1*−*6*)[*of ?S* S']
unfolding *rtranclp-unfold* **by** *auto*
have *confl-k*: *conflict-is-false-with-level* S'
using *rtranclp-cdcl*$_W$*-stgy-no-smaller-confl-inv*[*OF step*] *no-d* **by** *auto*
have *learned-entailed*: ‹*cdcl*$_W$*-learned-clauses-entailed-by-init* S'›
using *rtranclp-cdcl*$_W$*-learned-clauses-entailed*[*of* ‹*?S*› ‹S'›] *step*
by (*simp add*: *rtranclp-cdcl*$_W$*-stgy-rtranclp-cdcl*$_W$*-restart*)

show *?thesis*
using *cdcl*$_W$*-stgy-final-state-conclusive*[*OF termi decomp learned level-inv alien no-dup confl*
confl-k learned-entailed] **.**
qed

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*)
apply (*auto elim*: *conflictE*; *fail*)[]
apply (*auto elim*: *propagateE*; *fail*)[]
using *cdcl*$_W$*-o-fst-empty-conflicting-false* **by** *blast*

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*)
apply (*auto elim*: *conflictE*; *fail*)[]
apply (*auto elim*: *propagateE*; *fail*)[]
by (*metis conflict-with-false-implies-terminated other*)

lemma *rtranclp-cdcl$_W$-stgy-conflicting-is-false*:
  cdcl$_W$-stgy** S S′ ⟹ conflicting S = Some {#} ⟹ S′ = S
  apply (*induction rule*: *rtranclp-induct*)
    apply *simp*
  using *cdcl$_W$-stgy-conflicting-is-false* **by** *blast*

definition *conflict-or-propagate* :: *′st ⇒ ′st ⇒ bool* **where**
conflict-or-propagate S T ⟷ conflict S T ∨ propagate S T

declare *conflict-or-propagate-def*[*simp*]

lemma *conflict-or-propagate-intros*:
  conflict S T ⟹ conflict-or-propagate S T
  propagate S T ⟹ conflict-or-propagate S T
  by *auto*

theorem 2.9.9 page 97 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 {#}* ∧ *unsatisfiable* (*set-mset N*))
   ∨ (*conflicting S′ = None* ∧ *trail S′ ⊨asm N* ∧ *satisfiable* (*set-mset N*))
proof −
  have *N*: *init-clss S′ = N*
    using *full* **unfolding** *full-def* **by** (*auto dest*: *rtranclp-cdcl$_W$-stgy-no-more-init-clss*)
  consider
      (*confl*) *conflicting S′ = Some {#}* **and** *unsatisfiable* (*set-mset* (*init-clss S′*))
    | (*sat*) *conflicting S′ = None* **and** *trail S′ ⊨asm init-clss S′*
    using *full-cdcl$_W$-stgy-final-state-conclusive*[*OF assms*] **by** *auto*
  then show *?thesis*
  proof *cases*
    case *confl*
    then show *?thesis* **by** (*auto simp*: *N*)
  next
    case *sat*
    have *cdcl$_W$-M-level-inv* (*init-state N*) **by** *auto*
    then have *cdcl$_W$-M-level-inv S′*
      using *full rtranclp-cdcl$_W$-stgy-consistent-inv* **unfolding** *full-def* **by** *blast*
    then have *consistent-interp* (*lits-of-l* (*trail S′*))
      unfolding *cdcl$_W$-M-level-inv-def* **by** *blast*
    moreover have *lits-of-l* (*trail S′*) *⊨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

1.1.6 Structural Invariant

The condition that no learned clause is a tautology is overkill for the termination (in the sense that the no-duplicate condition is enough), but it allows to reuse *simple-clss*.

The invariant contains all the structural invariants that holds,

definition *cdcl$_W$-all-struct-inv* **where**

```
  cdcl_W-all-struct-inv S ⟷
    no-strange-atm S ∧
    cdcl_W-M-level-inv S ∧
    (∀ s ∈# learned-clss S. ¬tautology s) ∧
    distinct-cdcl_W-state S ∧
    cdcl_W-conflicting S ∧
    all-decomposition-implies-m (clauses S) (get-all-ann-decomposition (trail S)) ∧
    cdcl_W-learned-clause S

lemma cdcl_W-all-struct-inv-inv:
  assumes cdcl_W-restart 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-restart-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-restart-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-restart-all-inv[OF assms(1)] assms(2) unfolding cdcl_W-all-struct-inv-def by fast
  show cdcl_W-conflicting S'
     using cdcl_W-restart-all-inv[OF assms(1)] assms(2) unfolding cdcl_W-all-struct-inv-def by fast
  show all-decomposition-implies-m (clauses S') (get-all-ann-decomposition (trail S'))
     using cdcl_W-restart-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-restart-all-inv[OF assms(1)] assms(2) unfolding cdcl_W-all-struct-inv-def by fast

  show ∀ s∈#learned-clss S'. ¬ tautology s
    using assms(1)[THEN learned-clss-are-not-tautologies] assms(2)
    unfolding cdcl_W-all-struct-inv-def by fast
qed

lemma rtranclp-cdcl_W-all-struct-inv-inv:
  assumes cdcl_W-restart** 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 ⟹ cdcl_W-all-struct-inv S ⟹ cdcl_W-all-struct-inv T
  by (meson cdcl_W-stgy-tranclp-cdcl_W-restart rtranclp-cdcl_W-all-struct-inv-inv rtranclp-unfold)

lemma rtranclp-cdcl_W-stgy-cdcl_W-all-struct-inv:
  cdcl_W-stgy** S T ⟹ cdcl_W-all-struct-inv S ⟹ cdcl_W-all-struct-inv T
  by (induction rule: rtranclp-induct) (auto intro: cdcl_W-stgy-cdcl_W-all-struct-inv)

lemma beginning-not-decided-invert:
  assumes A: M @ A = M' @ Decided K # H and
  nm: ∀ m∈set M. ¬is-decided m
  shows ∃ 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 annotated-lit.disc(1) not-gr0
      nth-append nth-append-length nth-mem zero-less-diff)
  finally show ?thesis by fast
```

qed

1.1.7 Strategy-Specific Invariant

definition *cdcl$_W$-stgy-invariant* **where**
cdcl$_W$-stgy-invariant S $\longleftrightarrow$
conflict-is-false-with-level S
$\wedge$ *no-smaller-confl S*

lemma *cdcl$_W$-stgy-cdcl$_W$-stgy-invariant*:
assumes
cdcl$_W$-restart: *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$_W$-stgy-invariant-def cdcl$_W$-stgy-no-smaller-confl inv-s* **by** *blast*

lemma *rtranclp-cdcl$_W$-stgy-cdcl$_W$-stgy-invariant*:
assumes
cdcl$_W$-restart: *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*; *fail*)
using *cdcl$_W$-stgy-cdcl$_W$-stgy-invariant rtranclp-cdcl$_W$-all-struct-inv-inv*
rtranclp-cdcl$_W$-stgy-rtranclp-cdcl$_W$-restart **by** *blast*

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* **and**
learned-entailed: ‹*cdcl$_W$-learned-clauses-entailed-by-init S*›
shows *conflicting T = Some* {#} $\wedge$ *unsatisfiable* (*set-mset* (*init-clss S*))
$\vee$ *conflicting T = None* $\wedge$ *trail T* $\models$*asm init-clss S* $\wedge$ *satisfiable* (*set-mset* (*init-clss S*))
proof −
have *no-step cdcl$_W$-stgy T* **and** *st*: *cdcl$_W$-stgy** S 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$_W$-restart 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*)
moreover have ‹*cdcl$_W$-learned-clauses-entailed-by-init T*›
using *inv learned-entailed* **unfolding** *cdcl$_W$-all-struct-inv-def*
using *rtranclp-cdcl$_W$-learned-clauses-entailed rtranclp-cdcl$_W$-stgy-rtranclp-cdcl$_W$-restart*[*OF st*]
by *blast*
ultimately have *conflicting T = Some* {#} $\wedge$ *unsatisfiable* (*set-mset* (*init-clss T*))
$\vee$ *conflicting T = None* $\wedge$ *trail T* $\models$*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 *full-*$cdcl_W$*-stgy-inv-normal-form2*:
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* (*clauses* T))
∨ *conflicting* T = *None* ∧ *satisfiable* (*set-mset* (*clauses* T))
proof −
have *no-step* $cdcl_W$*-stgy* T **and** *st*: $cdcl_W$*-stgy*** S 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-*$cdcl_W$*-restart 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* (*clauses* T))
∨ *conflicting* T = *None* ∧ *trail* T ⊨*asm clauses* T
using $cdcl_W$*-stgy-final-state-conclusive2*[*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*
ultimately show *?thesis*
by (*metis satisfiable-carac′ true-annot-def true-annots-def true-clss-def*)
qed

1.1.8 Additional Invariant: No Smaller Propagation

definition *no-smaller-propa* :: ‹′*st* ⇒ *bool*› **where**
no-smaller-propa (S ::′*st*) ≡
(∀ *M K M′ D L*. *trail* S = M' @ *Decided K* # *M* ⟶ *D* + {#*L*#} ∈# *clauses* S ⟶ *undefined-lit M*
L
⟶ ¬*M* ⊨*as CNot D*)

lemma *propagated-cons-eq-append-decide-cons*:
Propagated L E # *Ms* = M' @ *Decided K* # *M* ⟷
M' ≠ [] ∧ *Ms* = *tl* M' @ *Decided K* # *M* ∧ *hd* M' = *Propagated L E*
by (*metis* (*no-types*, *lifting*) *annotated-lit.disc*(*1*) *annotated-lit.disc*(*2*) *append-is-Nil-conv hd-append*
list.exhaust-sel list.sel(*1*) *list.sel*(*3*) *tl-append2*)

lemma *in-get-all-mark-of-propagated-in-trail*:
‹*C* ∈ *set* (*get-all-mark-of-propagated M*) ⟷ (∃ *L*. *Propagated L C* ∈ *set M*)›
by (*induction M rule*: *ann-lit-list-induct*) *auto*

lemma *no-smaller-propa-tl*:
assumes

⟨*no-smaller-propa S*⟩ **and**
⟨*trail S* ≠ []⟩ **and**
⟨¬*is-decided*(*hd-trail S*)⟩ **and**
⟨*trail U* = *tl* (*trail S*)⟩ **and**
⟨*clauses U* = *clauses S*⟩
shows
⟨*no-smaller-propa U*⟩
using *assms* **by** (*cases* ⟨*trail S*⟩) (*auto simp*: *no-smaller-propa-def*)

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

lemma *decide-no-smaller-step*:
assumes *dec*: ⟨*decide S T*⟩ **and** *smaller-propa*: ⟨*no-smaller-propa S*⟩ **and**
n-s: ⟨*no-step propagate S*⟩
shows ⟨*no-smaller-propa T*⟩
unfolding *no-smaller-propa-def*
proof *clarify*
fix *M K M′ D L*
assume
tr: ⟨*trail T* = *M′* @ *Decided K* # *M*⟩ **and**
D: ⟨*D*+{#*L*#} ∈# *clauses T*⟩ **and**
undef: ⟨*undefined-lit M L*⟩ **and**
M: ⟨*M* ⊨*as CNot D*⟩
then have *Ex* (*propagate S*)
apply (*cases M′*)
using *propagate-rule*[*of S D*+{#*L*#} *L cons-trail* (*Propagated L* (*D* + {#*L*#})) *S*] *dec*
smaller-propa
by (*auto simp*: *no-smaller-propa-def elim*!: *rulesE*)
then show *False*
using *n-s* **by** *blast*
qed

lemma *no-smaller-propa-reduce-trail-to*:
⟨*no-smaller-propa S* ⟹ *no-smaller-propa* (*reduce-trail-to M1 S*)⟩
unfolding *no-smaller-propa-def*
by (*subst* (*asm*) *append-take-drop-id*[*symmetric*, *of* - ⟨*length* (*trail S*) − *length M1*⟩])
(*auto simp*: *trail-reduce-trail-to-drop*
simp del: *append-take-drop-id*)

lemma *backtrackg-no-smaller-propa*:
assumes *o*: ⟨*backtrackg S T*⟩ **and** *smaller-propa*: ⟨*no-smaller-propa S*⟩ **and**
n-d: ⟨*no-dup* (*trail S*)⟩ **and**
n-s: ⟨*no-step propagate S*⟩ **and**
tr-CNot: ⟨*trail S* ⊨*as CNot* (*the* (*conflicting S*))⟩
shows ⟨*no-smaller-propa T*⟩
proof −
obtain *D D′* :: ′*v clause* **and** *K L* :: ′*v literal* **and**
M1 M2 :: (′*v*, ′*v clause*) *ann-lit list* **and** *i* :: *nat* **where**
confl: *conflicting S* = *Some* (*add-mset L D*) **and**
decomp: (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*)) **and**
bt: *get-level* (*trail S*) *L* = *backtrack-lvl S* **and**
lev-L: *get-level* (*trail S*) *L* = *get-maximum-level* (*trail S*) (*add-mset L D′*) **and**
i: *get-maximum-level* (*trail S*) *D′* ≡ *i* **and**
lev-K: *get-level* (*trail S*) *K* = *i* + *1* **and**
D-D′: ⟨*D′* ⊆# *D*⟩ **and**

T: *T* ∼ *cons-trail* (*Propagated L* (*add-mset L D′*))
(*reduce-trail-to M1*
(*add-learned-cls* (*add-mset L D′*)
(*update-conflicting None S*)))
using *o* **by** (*auto elim*!: *rulesE*)
let *?D′* = ‹*add-mset L D′*›
have [*simp*]: *trail* (*reduce-trail-to M1 S*) = *M1*
using *decomp* **by** *auto*
obtain *M′′ c* **where** *M′′*: *trail S* = *M′′* @ *tl* (*trail T*) **and** *c*: ‹*M′′* = *c* @ *M2* @ [*Decided K*]›
using *decomp T* **by** *auto*
have *M1*: *M1* = *tl* (*trail T*) **and** *tr-T*: *trail T* = *Propagated L ?D′* # *M1*
using *decomp T* **by** *auto*

have *i-lvl*: ‹*i* = *backtrack-lvl T*›
using *no-dup-append-in-atm-notin*[*of* ‹*c* @ *M2*› ‹*Decided K* # *tl* (*trail T*)› *K*]
n-d lev-K **unfolding** *c M′′* **by** (*auto simp*: *image-Un tr-T*)

from *o* **show** *?thesis*
unfolding *no-smaller-propa-def*
proof *clarify*
fix *M K′ M′ E′ L′*
assume
tr: ‹*trail T* = *M′* @ *Decided K′* # *M*› **and**
E: ‹*E′*+{#*L′*#} ∈# *clauses T*› **and**
undef: ‹*undefined-lit M L′*› **and**
M: ‹*M* ⊨*as CNot E′*›
have *n-d-T*: ‹*no-dup* (*trail T*)› **and** *M1-D′*: *M1* ⊨*as CNot D′*
using *backtrack-M1-CNot-D′*[*OF n-d i decomp - confl - T*] *lev-K bt lev-L tr-CNot confl D-D′*
by (*auto dest*: *subset-mset-trans-add-mset*)
have *False* **if** *D*: ‹*add-mset L D′* = *add-mset L′ E′*› **and** *M-D*: ‹*M* ⊨*as CNot E′*›
proof −
have ‹*i* ≠ *0*›
using *i-lvl tr T* **by** *auto*
moreover
have *get-maximum-level M1 D′* = *i*
using *T i n-d D-D′ M1-D′* **unfolding** *M′′ tr-T*
by (*subst* (*asm*) *get-maximum-level-skip-beginning*)
(*auto dest*: *defined-lit-no-dupD dest*!: *true-annots-CNot-definedD*)
ultimately obtain *L-max* **where**
L-max-in: *L-max* ∈# *D′* **and**
lev-L-max: *get-level M1 L-max* = *i*
using *i get-maximum-level-exists-lit-of-max-level*[*of D′ M1*]
by (*cases D′*) *auto*
have *count-dec-M*: *count-decided M* < *i*
using *T i-lvl* **unfolding** *tr* **by** *auto*
have − *L-max* ∉ *lits-of-l M*
proof (*rule ccontr*)
assume ‹¬ *?thesis*›
then have ‹*undefined-lit* (*M′* @ [*Decided K′*]) *L-max*›
using *n-d-T* **unfolding** *tr*
by (*auto dest*: *in-lits-of-l-defined-litD dest*: *defined-lit-no-dupD simp*: *atm-of-eq-atm-of*)
then have *get-level* (*tl M′* @ *Decided K′* # *M*) *L-max* < *i*
apply (*subst get-level-skip*)
apply (*cases M′*; *auto simp add*: *atm-of-eq-atm-of lits-of-def*; *fail*)
using *count-dec-M count-decided-ge-get-level*[*of M L-max*] **by** *auto*

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

lemmas backtrack-no-smaller-propa = backtrackg-no-smaller-propa[OF backtrack-backtrackg]

lemma cdcl_W-stgy-no-smaller-propa:
  assumes
    cdcl: ‹cdcl_W-stgy S T› and
    smaller-propa: ‹no-smaller-propa S› and
    inv: ‹cdcl_W-all-struct-inv S›
  shows ‹no-smaller-propa T›
  using cdcl
proof (cases rule: cdcl_W-stgy-cases)
  case conflict
  then show ?thesis
    using smaller-propa by (auto simp: no-smaller-propa-def elim!: rulesE)
next
  case propagate
  then show ?thesis
    using smaller-propa by (auto simp: no-smaller-propa-def propagated-cons-eq-append-decide-cons
      elim!: rulesE)
next
  case skip
  then show ?thesis
    using smaller-propa by (auto intro: no-smaller-propa-tl elim!: rulesE)
next
  case resolve
  then show ?thesis
    using smaller-propa by (auto intro: no-smaller-propa-tl elim!: rulesE)
next
  case decide note n-s = this(1,2) and dec = this(3)
```

show *?thesis*
  using *n-s dec decide-no-smaller-step*[*of S T*] *smaller-propa*
  by *auto*
next
 case *backtrack* **note** *n-s* = *this*(*1*,*2*) **and** *o* = *this*(*3*)
 have *inv-T*: $cdcl_W$*-all-struct-inv T*
  using *cdcl* $cdcl_W$*-stgy-*$cdcl_W$*-all-struct-inv inv* **by** *blast*
 have ‹*trail S* ⊨*as CNot* (*the* (*conflicting S*))› **and** ‹*no-dup* (*trail S*)›
  using *inv o* **unfolding** $cdcl_W$*-all-struct-inv-def*
  by (*auto simp*: $cdcl_W$*-M-level-inv-def* $cdcl_W$*-conflicting-def*
   elim: *rulesE*)
 then show *?thesis*
  using *backtrack-no-smaller-propa*[*of S T*] *n-s o smaller-propa*
  by *auto*
qed

lemma *rtranclp-*$cdcl_W$*-stgy-no-smaller-propa*:
 assumes
  cdcl: ‹$cdcl_W$*-stgy*** *S T*› **and**
  smaller-propa: ‹*no-smaller-propa S*› **and**
  inv: ‹$cdcl_W$*-all-struct-inv S*›
 shows ‹*no-smaller-propa T*›
 using *cdcl* **apply** (*induction rule*: *rtranclp-induct*)
 subgoal using *smaller-propa* **by** *simp*
 subgoal using *inv* **by** (*auto intro*: *rtranclp-*$cdcl_W$*-stgy-*$cdcl_W$*-all-struct-inv*
   $cdcl_W$*-stgy-no-smaller-propa*)
 done

lemma *hd-trail-level-ge-1-length-gt-1*:
 fixes *S* :: *'st*
 defines *M*[*symmetric*, *simp*]: ‹*M* ≡ *trail S*›
 defines *L*[*symmetric*, *simp*]: ‹*L* ≡ *hd M*›
 assumes
  smaller: ‹*no-smaller-propa S*› **and**
  struct: ‹$cdcl_W$*-all-struct-inv S*› **and**
  dec: ‹*count-decided M* ≥ *1*› **and**
  proped: ‹*is-proped L*›
 shows ‹*size* (*mark-of L*) > *1*›
proof (*rule ccontr*)
 assume *size-C*: ‹¬ *?thesis*›
 have *nd*: ‹*no-dup M*›
  using *struct* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def M*[*symmetric*]
  by *blast*

 obtain *M'* **where** *M'*: ‹*M* = *L* # *M'*›
  using *dec L* **by** (*cases M*) (*auto simp del*: *L*)
 obtain *K C* **where** *K*: ‹*L* = *Propagated K C*›
  using *proped* **by** (*cases L*) *auto*

 obtain *K' M1 M2* **where** *decomp*: ‹*M* = *M2* @ *Decided K'* # *M1*›
  using *dec le-count-decided-decomp*[*of M 0*] *nd* **by** *auto*
 then have *decomp'*: ‹*M'* = *tl M2* @ *Decided K'* # *M1*›
  unfolding *M' K* **by** (*cases M2*) *auto*

 have ‹*K* ∈# *C*›
  using *struct* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-conflicting-def*

M M' K **by** *blast*
then have C: ‹$C = \{\#\} + \{\#K\#\}$›
using *size-C K* **by** (*cases C*) *auto*
have ‹*undefined-lit M1 K*›
using *nd* **unfolding** M' *K* $decomp'$ **by** *simp*
moreover have ‹$\{\#\} + \{\#K\#\} \in\#$ *clauses S*›
using *struct* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-learned-clause-alt-def M* M' *K C reasons-in-clauses-def*
by *auto*
moreover have ‹*M1* $\models$*as CNot* $\{\#\}$›
by *auto*
ultimately show *False*
using *smaller* **unfolding** *no-smaller-propa-def M decomp*
by *blast*
qed

1.1.9 More Invariants: Conflict is False if no decision

If the level is higher than 0, then the conflict is not empty.

definition *conflict-non-zero-unless-level-0* :: ‹$'st \Rightarrow bool$› **where**
‹*conflict-non-zero-unless-level-0 S* $\longleftrightarrow$
(*conflicting S* = *Some* $\{\#\}$ $\longrightarrow$ *count-decided* (*trail S*) = *0*)›

definition *no-false-clause*:: ‹$'st \Rightarrow bool$› **where**
‹*no-false-clause S* $\longleftrightarrow$ ($\forall$ $C \in\#$ *clauses S*. $C \neq \{\#\}$)›

lemma $cdcl_W$*-restart-no-false-clause*:
assumes
‹$cdcl_W$*-restart S T*›
‹*no-false-clause S*›
shows ‹*no-false-clause T*›
using *assms* **unfolding** *no-false-clause-def*
by (*induction rule*: $cdcl_W$*-restart-all-induct*) (*auto simp add*: *clauses-def*)

The proofs work smoothly thanks to the side-conditions about levels of the rule *resolve*.

lemma $cdcl_W$*-restart-conflict-non-zero-unless-level-0*:
assumes
‹$cdcl_W$*-restart S T*›
‹*no-false-clause S*› **and**
‹*conflict-non-zero-unless-level-0 S*›
shows ‹*conflict-non-zero-unless-level-0 T*›
using *assms* **by** (*induction rule*: $cdcl_W$*-restart-all-induct*)
(*auto simp add*: *conflict-non-zero-unless-level-0-def no-false-clause-def*)

lemma *rtranclp-*$cdcl_W$*-restart-no-false-clause*:
assumes
‹$cdcl_W$*-restart*** *S T*›
‹*no-false-clause S*›
shows ‹*no-false-clause T*›
using *assms* **by** (*induction rule*: *rtranclp-induct*) (*auto intro*: $cdcl_W$*-restart-no-false-clause*)

lemma *rtranclp-*$cdcl_W$*-restart-conflict-non-zero-unless-level-0*:
assumes
‹$cdcl_W$*-restart*** *S T*›

⟨*no-false-clause S*⟩ **and**
⟨*conflict-non-zero-unless-level-0 S*⟩
shows ⟨*conflict-non-zero-unless-level-0 T*⟩
using *assms* **by** (*induction rule*: *rtranclp-induct*)
(*auto intro*: *rtranclp-cdcl*$_W$*-restart-no-false-clause cdcl*$_W$*-restart-conflict-non-zero-unless-level-0*)

definition *propagated-clauses-clauses* :: $'st \Rightarrow bool$ **where**
⟨*propagated-clauses-clauses* $S \equiv \forall L\ K.$ *Propagated* $L\ K \in$ *set* (*trail* S) $\longrightarrow K \in\#$ *clauses* S⟩

lemma *propagate-single-literal-clause-get-level-is-0*:
assumes
smaller: ⟨*no-smaller-propa S*⟩ **and**
propa-tr: ⟨*Propagated* L {#*L*#} $\in$ *set* (*trail* S)⟩ **and**
n-d: ⟨*no-dup* (*trail* S)⟩ **and**
propa: ⟨*propagated-clauses-clauses S*⟩
shows ⟨*get-level* (*trail* S) $L = 0$⟩
proof (*rule ccontr*)
assume H: ⟨$\neg$ *?thesis*⟩
then obtain $M\ M'\ K$ **where**
tr: ⟨*trail* $S = M'$ @ *Decided* K # M⟩ **and**
nm: ⟨$\forall m \in$ *set* M. $\neg$*is-decided m*⟩
using *split-list-last-prop*[*of trail S is-decided*]
by (*auto simp*: *filter-empty-conv is-decided-def get-level-def dest*!: *List.set-dropWhileD*)
have *uL*: ⟨$-L \notin$ *lits-of-l* (*trail* S)⟩
using *n-d propa-tr* **unfolding** *lits-of-def* **by** (*fastforce simp*: *no-dup-cannot-not-lit-and-uminus*)
then have [*iff*]: ⟨*defined-lit* M' $L \longleftrightarrow L \in$ *lits-of-l* M'⟩
by (*auto simp add*: *tr Decided-Propagated-in-iff-in-lits-of-l*)
have ⟨*get-level* M $L = 0$⟩ **for** L
using *nm* **by** *auto*
have [*simp*]: ⟨$L \neq -K$⟩
using *tr propa-tr n-d* **unfolding** *lits-of-def* **by** (*fastforce simp*: *no-dup-cannot-not-lit-and-uminus in-set-conv-decomp*)
have ⟨$L \in$ *lits-of-l* (M' @ [*Decided* K])⟩
apply (*rule ccontr*)
using H **unfolding** *tr*
apply (*subst* (*asm*) *get-level-skip*)
using *uL tr* **apply** (*auto simp*: *atm-of-eq-atm-of Decided-Propagated-in-iff-in-lits-of-l*; *fail*)[]
apply (*subst* (*asm*) *get-level-skip-beginning*)
using ⟨*get-level* M $L = 0$⟩ **by** (*auto simp*: *atm-of-eq-atm-of uminus-lit-swap lits-of-def*)
then have ⟨*undefined-lit M L*⟩
using *n-d* **unfolding** *tr* **by** (*auto simp*: *defined-lit-map lits-of-def image-Un no-dup-def*)
moreover have {#} + {#*L*#} $\in\#$ *clauses* S
using *propa propa-tr* **unfolding** *propagated-clauses-clauses-def* **by** *auto*
moreover have $M \models$*as CNot* {#}
by *auto*
ultimately show *False*
using *smaller tr* **unfolding** *no-smaller-propa-def* **by** *blast*
qed

Conflict Minimisation

Remove Literals of Level 0 **lemma** *conflict-minimisation-level-0*:
fixes S :: $'st$
defines D[*simp*]: ⟨$D \equiv$ *the* (*conflicting* S)⟩
defines [*simp*]: ⟨$M \equiv$ *trail* S⟩
defines ⟨$D' \equiv$ *filter-mset* (λL. *get-level* M $L > 0$) D⟩

assumes
ns-s: ⟨*no-step skip S*⟩ **and**
ns-r: ⟨*no-step resolve S*⟩ **and**
inv-s: $cdcl_W$*-stgy-invariant S* **and**
inv: $cdcl_W$*-all-struct-inv S* **and**
conf: ⟨*conflicting S* ≠ *None*⟩ ⟨*conflicting S* ≠ *Some* {#}⟩ **and**
M-nempty: ⟨*M* ~= []⟩
shows
clauses S ⊨*pm D′* **and**
⟨− *lit-of* (*hd M*) ∈# *D′*⟩
proof −
define *D0* **where** *D0*: ⟨*D0* = *filter-mset* (λ*L*. *get-level M L* = *0*) *D*⟩
have *D-D0-D′*: ⟨*D* = *D0* + *D′*⟩
using *multiset-partition*[*of D* ⟨(λ*L*. *get-level M L* = *0*)⟩]
unfolding *D0 D′-def* **by** *auto*
have
confl: ⟨$cdcl_W$*-conflicting S*⟩ **and**
decomp: ⟨*all-decomposition-implies-m* (*clauses S*) (*get-all-ann-decomposition* (*trail S*))⟩ **and**
learned: ⟨$cdcl_W$*-learned-clause S*⟩ **and**
M-lev: ⟨$cdcl_W$*-M-level-inv S*⟩ **and**
alien: ⟨*no-strange-atm S*⟩
using *inv* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *fast+*
have *clss-D*: ⟨*clauses S* ⊨*pm D*⟩
using *learned conf* **unfolding** $cdcl_W$*-learned-clause-alt-def* **by** *auto*
have *M-CNot-D*: ⟨*trail S* ⊨*as CNot D*⟩ **and** *m-confl*: ⟨*every-mark-is-a-conflict S*⟩
using *conf confl* **unfolding** $cdcl_W$*-conflicting-def* **by** *auto*
have *n-d*: ⟨*no-dup M*⟩
using *M-lev* **unfolding** $cdcl_W$*-M-level-inv-def* **by** *auto*
have *uhd-D*: ⟨− *lit-of* (*hd M*) ∈# *D*⟩
using *ns-s ns-r conf M-nempty inv-s M-CNot-D n-d*
unfolding $cdcl_W$*-stgy-invariant-def conflict-is-false-with-level-def*
by (*cases* ⟨*trail S*⟩; *cases* ⟨*hd* (*trail S*)⟩) (*auto simp*: *skip.simps resolve.simps*
get-level-cons-if atm-of-eq-atm-of true-annots-true-cls-def-iff-negation-in-model
uminus-lit-swap Decided-Propagated-in-iff-in-lits-of-l split: *if-splits*)

have *count-dec-ge-0*: ⟨*count-decided M* > *0*⟩
proof (*rule ccontr*)
assume *H*: ⟨~ *?thesis*⟩
then have ⟨*get-maximum-level M D* = *0*⟩ **for** *D*
by (*metis* (*full-types*) *count-decided-ge-get-maximum-level gr0I le-0-eq*)
then show *False*
using *ns-s ns-r conf M-nempty m-confl uhd-D H*
by (*cases* ⟨*trail S*⟩; *cases* ⟨*hd* (*trail S*)⟩)
(*auto 5 5 simp*: *skip.simps resolve.simps intro*!: *state-eq-ref*)
qed
then obtain *M0 K M1* **where**
M: ⟨*M* = *M1* @ *Decided K* # *M0*⟩ **and**
lev-K: ⟨*get-level* (*trail S*) *K* = *Suc 0*⟩
using *backtrack-ex-decomp*[*of S 0*, *OF*] *M-lev*
by (*auto dest*!: *get-all-ann-decomposition-exists-prepend*
simp: $cdcl_W$*-M-level-inv-def simp flip*: *append.assoc*
simp del: *append-assoc*)

have *count-M0*: ⟨*count-decided M0* = *0*⟩
using *n-d lev-K* **unfolding** *M-def*[*symmetric*] *M* **by** *auto*
have [*simp*]: ⟨*get-all-ann-decomposition M0* = [([], *M0*)]⟩

using *count-M0* **by** (*induction M0 rule*: *ann-lit-list-induct*) *auto*
have [*simp*]: ‹*get-all-ann-decomposition* (*M1* @ *Decided K* # *M0*) ≠ [([], *M0*)]› **for** *M1 K M0*
using *length-get-all-ann-decomposition*[*of* ‹*M1* @ *Decided K* # *M0*›]
unfolding *M* **by** *auto*
have ‹*last* (*get-all-ann-decomposition* (*M1* @ *Decided K* # *M0*)) = ([], *M0*)›
apply (*induction M1 rule*: *ann-lit-list-induct*)
subgoal by *auto*
subgoal by *auto*
subgoal for *L m M1*
by (*cases* ‹*get-all-ann-decomposition* (*M1* @ *Decided K* # *M0*)›) *auto*
done
then have *clss-S-M0*: ‹*set-mset* (*clauses S*) ⊨*ps unmark-l M0*›
using *decomp* **unfolding** *M-def*[*symmetric*] *M*
by (*cases* ‹*get-all-ann-decomposition* (*M1* @ *Decided K* # *M0*)› *rule*: *rev-cases*)
(*auto simp*: *all-decomposition-implies-def*)
have *H*: ‹*total-over-m I* (*set-mset* (*clauses S*) ∪ *unmark-l M0*) = *total-over-m I* (*set-mset* (*clauses S*))›
for *I*
using *alien* **unfolding** *no-strange-atm-def total-over-m-def total-over-set-def*
M-def[*symmetric*] *M*
by (*auto simp*: *clauses-def*)
have *uL-M0-D0*: ‹−*L* ∈ *lits-of-l M0*› **if** ‹*L* ∈# *D0*› **for** *L*
proof (*rule ccontr*)
assume *L-M0*: ‹~ *?thesis*›
have ‹*L* ∈# *D*› **and** *lev-L*: ‹*get-level M L* = *0*›
using *that* **unfolding** *D-D0-D′* **unfolding** *D0* **by** *auto*
then have ‹−*L* ∈ *lits-of-l M*›
using *M-CNot-D that* **by** (*auto simp*: *true-annots-true-cls-def-iff-negation-in-model*)
then have ‹−*L* ∈ *lits-of-l* (*M1* @ [*Decided K*])›
using *L-M0* **unfolding** *M* **by** *auto*
then have ‹*0* < *get-level* (*M1* @ [*Decided K*]) *L*› **and** ‹*defined-lit* (*M1* @ [*Decided K*]) *L*›
using *get-level-last-decided-ge*[*of M1 K L*] **unfolding** *Decided-Propagated-in-iff-in-lits-of-l*
by *fast+*
then show *False*
using *n-d lev-L get-level-skip-end*[*of* ‹*M1* @ [*Decided K*]› *L M0*]
unfolding *M* **by** *auto*
qed
have *clss-D0*: ‹*clauses S* ⊨*pm* {#− *L*#}› **if** ‹*L* ∈# *D0*› **for** *L*
using *that clss-S-M0 uL-M0-D0*[*of L*] **unfolding** *true-clss-clss-def H true-clss-cls-def*
true-clss-def lits-of-def
by *auto*
have *lD0D′*: ‹*l* ∈ *atms-of D0* ⟹ *l* ∈ *atms-of D*› ‹*l* ∈ *atms-of D′* ⟹ *l* ∈ *atms-of D*› **for** *l*
unfolding *D-D0-D′* **by** *auto*
have
H1: ‹*total-over-m I* (*set-mset* (*clauses S*) ∪ {{#−*L*#}}) = *total-over-m I* (*set-mset* (*clauses S*))›
if ‹*L* ∈# *D0*› **for** *L*
using *alien conf atm-of-lit-in-atms-of*[*OF that*]
unfolding *no-strange-atm-def total-over-m-def total-over-set-def*
M-def[*symmetric*] *M that* **by** (*auto 5 5 simp*: *clauses-def dest*!: *lD0D′*)
then have *I-D0*: ‹*total-over-m I* (*set-mset* (*clauses S*)) ⟶
consistent-interp I ⟶
Multiset.Ball (*clauses S*) ((⊨) *I*) ⟶ ~*I* ⊨ *D0*› **for** *I*
using *clss-D0* **unfolding** *true-clss-cls-def true-cls-def consistent-interp-def*
true-cls-def true-cls-mset-def — TODO tune proof
apply *auto*
by (*metis atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set literal.sel*(*1*)

true-cls-def true-cls-mset-def true-lit-def uminus-Pos)

have
H1: ‹*total-over-m I* (*set-mset* (*clauses S*) ∪ {*D0* + *D′*}) = *total-over-m I* (*set-mset* (*clauses S*))›
and
H2: ‹*total-over-m I* (*set-mset* (*clauses S*) ∪ {*D′*}) = *total-over-m I* (*set-mset* (*clauses S*))› **for** *I*
using *alien conf* **unfolding** *no-strange-atm-def total-over-m-def total-over-set-def*
M-def[*symmetric*] *M* **by** (*auto 5 5 simp*: *clauses-def dest*!: *lD0D′*)
show ‹*clauses S* ⊨*pm D′*›
using *clss-D clss-D0 I-D0* **unfolding** *D-D0-D′ true-clss-cls-def true-clss-def H1 H2*
by *auto*
have ‹*0* < *get-level* (*trail S*) (*lit-of* (*hd-trail S*))›
apply (*cases* ‹*trail S*›)
using *M-nempty count-dec-ge-0* **by** *auto*
then show ‹− *lit-of* (*hd M*) ∈# *D′*›
using *uhd-D* **unfolding** *D′-def* **by** *auto*
qed

lemma *literals-of-level0-entailed*:
assumes
struct-invs: ‹$cdcl_W$*-all-struct-inv S*› **and**
in-trail: ‹*L* ∈ *lits-of-l* (*trail S*)› **and**
lev: ‹*get-level* (*trail S*) *L* = *0*›
shows
‹*clauses S* ⊨*pm* {#*L*#}›
proof −
have *decomp*: ‹*all-decomposition-implies-m* (*clauses S*) (*get-all-ann-decomposition* (*trail S*))›
using *struct-invs* **unfolding** $cdcl_W$*-all-struct-inv-def*
by *fast*
have *L-trail*: ‹{#*L*#} ∈ *unmark-l* (*trail S*)›
using *in-trail* **by** (*auto simp*: *in-unmark-l-in-lits-of-l-iff*)
have *n-d*: ‹*no-dup* (*trail S*)›
using *struct-invs* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def*
by *fast*

show *?thesis*
proof (*cases* ‹*count-decided* (*trail S*) = *0*›)
case *True*
have ‹*get-all-ann-decomposition* (*trail S*) = [([], *trail S*)]›
apply (*rule no-decision-get-all-ann-decomposition*)
using *True* **by** (*auto simp*: *count-decided-0-iff*)
then show *?thesis*
using *decomp L-trail*
unfolding *all-decomposition-implies-def*
by (*auto intro*: *true-clss-clss-in-imp-true-clss-cls*)
next
case *False*
then obtain *K M1 M2 M3* **where**
decomp′: ‹(*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*))› **and**
lev-K: ‹*get-level* (*trail S*) *K* = *Suc 0*› **and**
M3: ‹*trail S* = *M3* @ *M2* @ *Decided K* # *M1*›
using *struct-invs backtrack-ex-decomp*[*of S 0*] *n-d* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *blast*
then have *dec-M1*: ‹*count-decided M1* = *0*›
using *n-d* **by** *auto*
define *M2′* **where** ‹*M2′* = *M3* @ *M2*›
then have *M3*: ‹*trail S* = *M2′* @ *Decided K* # *M1*› **using** *M3* **by** *auto*

have ⟨*get-all-ann-decomposition M1* = [([], *M1*)]⟩
apply (*rule no-decision-get-all-ann-decomposition*)
using *dec-M1* **by** (*auto simp*: *count-decided-0-iff*)
then have ⟨([], *M1*) ∈ *set* (*get-all-ann-decomposition* (*trail S*))⟩
using *hd-get-all-ann-decomposition-skip-some*[*of Nil M1 M1* ⟨- @ -⟩] *decomp′*
by *auto*
then have ⟨*set-mset* (*clauses S*) ⊨*ps unmark-l M1*⟩
using *decomp*
unfolding *all-decomposition-implies-def* **by** *auto*
moreover {
have ⟨*L* ∈ *lits-of-l M1*⟩
using *n-d lev M3 in-trail*
by (*cases* ⟨*undefined-lit* (*M2′* @ *Decided K* # []) *L*⟩) (*auto dest*: *in-lits-of-l-defined-litD*)
then have ⟨{#*L*#} ∈ *unmark-l M1*⟩
using *in-trail* **by** (*auto simp*: *in-unmark-l-in-lits-of-l-iff*)
}
ultimately show *?thesis*
unfolding *all-decomposition-implies-def*
by (*auto intro*: *true-clss-clss-in-imp-true-clss-cls*)
qed
qed

1.1.10 Some higher level use on the invariants

In later refinement we mostly us the group invariants and don't try to be as specific as above. The corresponding theorems are collected here.

lemma *conflict-conflict-is-false-with-level-all-inv*:
⟨*conflict S T* ⟹
no-smaller-confl S ⟹
$cdcl_W$*-all-struct-inv S* ⟹
conflict-is-false-with-level T⟩
by (*rule conflict-conflict-is-false-with-level*) (*auto simp*: $cdcl_W$*-all-struct-inv-def*)

lemma $cdcl_W$*-stgy-ex-lit-of-max-level-all-inv*:
assumes
$cdcl_W$*-stgy S S′* **and**
n-l: *no-smaller-confl S* **and**
conflict-is-false-with-level S **and**
$cdcl_W$*-all-struct-inv S*
shows *conflict-is-false-with-level S′*
by (*rule* $cdcl_W$*-stgy-ex-lit-of-max-level*) (*use assms* **in** ⟨*auto simp*: $cdcl_W$*-all-struct-inv-def*⟩)

lemma $cdcl_W$*-o-conflict-is-false-with-level-inv-all-inv*:
assumes
⟨$cdcl_W$*-o S T*⟩
⟨$cdcl_W$*-all-struct-inv S*⟩
⟨*conflict-is-false-with-level S*⟩
shows ⟨*conflict-is-false-with-level T*⟩
by (*rule* $cdcl_W$*-o-conflict-is-false-with-level-inv*)
(*use assms* **in** ⟨*auto simp*: $cdcl_W$*-all-struct-inv-def*⟩)

lemma *no-step-*$cdcl_W$*-total*:
assumes

⟨*no-step cdcl*$_W$ *S*⟩
⟨*conflicting S* = *None*⟩
⟨*no-strange-atm S*⟩
shows ⟨*total-over-m* (*lits-of-l* (*trail S*)) (*set-mset* (*clauses S*))⟩
proof (*rule ccontr*)
assume ⟨¬ *?thesis*⟩
then obtain *L* **where** ⟨*L* ∈ *atms-of-mm* (*clauses S*)⟩ **and** ⟨*undefined-lit* (*trail S*) (*Pos L*)⟩
by (*auto simp*: *total-over-m-def total-over-set-def*
Decided-Propagated-in-iff-in-lits-of-l)
then have ⟨*Ex* (*decide S*)⟩
using *decide-rule*[*of S* ⟨*Pos L*⟩ ⟨*cons-trail* (*Decided* (*Pos L*)) *S*⟩] *assms*
unfolding *no-strange-atm-def clauses-def*
by *force*
then show *False*
using *assms* **by** (*auto simp*: *cdcl*$_W$*.simps cdcl*$_W$*-o.simps*)
qed

lemma *cdcl*$_W$*-Ex-cdcl*$_W$*-stgy*:
assumes
⟨*cdcl*$_W$ *S T*⟩
shows ⟨*Ex*(*cdcl*$_W$*-stgy S*)⟩
using *assms* **by** (*meson assms cdcl*$_W$*.simps cdcl*$_W$*-stgy.simps*)

lemma *no-step-skip-hd-in-conflicting*:
assumes
inv-s: ⟨*cdcl*$_W$*-stgy-invariant S*⟩ **and**
inv: ⟨*cdcl*$_W$*-all-struct-inv S*⟩ **and**
ns: ⟨*no-step skip S*⟩ **and**
confl: ⟨*conflicting S* ≠ *None*⟩ ⟨*conflicting S* ≠ *Some* {#}⟩
shows ⟨−*lit-of* (*hd* (*trail S*)) ∈# *the* (*conflicting S*)⟩
proof −
let
?M = ⟨*trail S*⟩ **and**
?N = ⟨*init-clss S*⟩ **and**
?U = ⟨*learned-clss S*⟩ **and**
?k = ⟨*backtrack-lvl S*⟩ **and**
?D = ⟨*conflicting S*⟩
obtain *D* **where** *D*: ⟨*?D* = *Some D*⟩
using *confl* **by** (*cases ?D*) *auto*
have *M-D*: ⟨*?M* ⊨*as CNot D*⟩
using *inv D* **unfolding** *cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-conflicting-def* **by** *auto*
then have *tr*: ⟨*trail S* ≠ []⟩
using *confl D* **by** *auto*
obtain *L M* **where** *M*: ⟨*?M* = *L* # *M*⟩
using *tr* **by** (*cases* ⟨*?M*⟩) *auto*
have *conlf-k*: ⟨*conflict-is-false-with-level S*⟩
using *inv-s* **unfolding** *cdcl*$_W$*-stgy-invariant-def* **by** *simp*
then obtain *L-k* **where**
L-k: ⟨*L-k* ∈# *D*⟩ **and** *lev-L-k*: ⟨*get-level ?M L-k* = *?k*⟩
using *confl D* **by** *auto*
have *dec*: ⟨*?k* = *count-decided ?M*⟩
using *inv* **unfolding** *cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-M-level-inv-def* **by** *auto*
moreover {
have ⟨*no-dup ?M*⟩
using *inv* **unfolding** *cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-M-level-inv-def* **by** *auto*

then have ⟨−*lit-of L* ∉ *lits-of-l M*⟩
unfolding *M* **by** (*auto simp*: *defined-lit-map lits-of-def uminus-lit-swap*)
}
ultimately have *L-D*: ⟨*lit-of L* ∉# *D*⟩
using *M-D* **unfolding** *M* **by** (*auto simp add*: *true-annots-true-cls-def-iff-negation-in-model uminus-lit-swap*)
show *?thesis*
proof (*cases L*)
case (*Decided L′*) **note** *L′* = *this*(*1*)
moreover have ⟨*atm-of L′* = *atm-of L-k*⟩
using *lev-L-k count-decided-ge-get-level*[*of M L-k*] **unfolding** *M dec L′*
by (*auto simp*: *get-level-cons-if split*: *if-splits*)
then have ⟨*L′* = −*L-k*⟩
using *L-k L-D L′* **by** (*auto simp*: *atm-of-eq-atm-of*)
then show *?thesis* **using** *L-k* **unfolding** *D M L′* **by** *simp*
next
case (*Propagated L′ C*)
then show *?thesis*
using *ns confl* **by** (*auto simp*: *skip.simps M D*)
qed
qed

lemma
fixes *S*
assumes
nss: ⟨*no-step skip S*⟩ **and**
nsr: ⟨*no-step resolve S*⟩ **and**
invs: ⟨*cdcl$_W$-all-struct-inv S*⟩ **and**
stgy: ⟨*cdcl$_W$-stgy-invariant S*⟩ **and**
confl: ⟨*conflicting S* ≠ *None*⟩ **and**
confl′: ⟨*conflicting S* ≠ *Some* {#}⟩
shows *no-skip-no-resolve-single-highest-level*:
⟨*the* (*conflicting S*) =
add-mset (−(*lit-of* (*hd* (*trail S*)))) {#*L* ∈# *the* (*conflicting S*).
get-level (*trail S*) *L* < *local.backtrack-lvl S*#}⟩ (**is** *?A*) **and**
no-skip-no-resolve-level-lvl-nonzero:
⟨*0* < *backtrack-lvl S*⟩ (**is** *?B*) **and**
no-skip-no-resolve-level-get-maximum-lvl-le:
⟨*get-maximum-level* (*trail S*) (*remove1-mset* (−(*lit-of* (*hd* (*trail S*)))) (*the* (*conflicting S*)))
< *backtrack-lvl S*⟩ (**is** *?C*)
proof −
define *K* **where** ⟨*K* ≡ *lit-of* (*hd* (*trail S*))⟩
have *K*: ⟨−*K* ∈# *the* (*conflicting S*)⟩
using *no-step-skip-hd-in-conflicting*[*OF stgy invs nss confl confl′*]
unfolding *K-def* **.**
have
⟨*no-strange-atm S*⟩ **and**
lev: ⟨*cdcl$_W$-M-level-inv S*⟩ **and**
⟨∀ *s*∈#*learned-clss S*. ¬ *tautology s*⟩ **and**
dist: ⟨*distinct-cdcl$_W$-state S*⟩ **and**
conf: ⟨*cdcl$_W$-conflicting S*⟩ **and**
⟨*all-decomposition-implies-m* (*local.clauses S*)
(*get-all-ann-decomposition* (*trail S*))⟩ **and**
learned: ⟨*cdcl$_W$-learned-clause S*⟩
using *invs* **unfolding** *cdcl$_W$-all-struct-inv-def*
by *auto*

obtain *D* **where**
D[*simp*]: ‹*conflicting S* = *Some* (*add-mset* (−*K*) *D*)›
using *confl K* **by** (*auto dest*: *multi-member-split*)

have *dist*: ‹*distinct-mset* (*the* (*conflicting S*))›
using *dist confl* **unfolding** *distinct-cdcl*$_W$*-state-def* **by** *auto*
then have [*iff*]: ‹*L* ∉# *remove1-mset L* (*the* (*conflicting S*))› **for** *L*
by (*meson distinct-mem-diff-mset union-single-eq-member*)
from *this*[*of K*] **have** [*simp*]: ‹−*K* ∉# *D*› **using** *dist* **by** *auto*

have *nd*: ‹*no-dup* (*trail S*)›
using *lev* **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *auto*
have *CNot*: ‹*trail S* ⊨*as CNot* (*add-mset* (−*K*) *D*)›
using *conf* **unfolding** *cdcl*$_W$*-conflicting-def*
by *fastforce*
then have *tr*: ‹*trail S* ≠ []›
by *auto*
have [*simp*]: ‹*K* ∉# *D*›
using *nd K-def tr CNot* **unfolding** *true-annots-true-cls-def-iff-negation-in-model*
by (*cases* ‹*trail S*›)
(*auto simp*: *uminus-lit-swap Decided-Propagated-in-iff-in-lits-of-l dest*!: *multi-member-split*)
have *H1*:
‹*0* < *backtrack-lvl S*›
proof (*cases* ‹*is-proped* (*hd* (*trail S*))›)
case *proped*: *True*
obtain *C M* **where**
[*simp*]: ‹*trail S* = *Propagated K C* # *M*›
using *tr proped K-def*
by (*cases* ‹*trail S*›; *cases* ‹*hd* (*trail S*)›)
(*auto simp*: *K-def*)
have ‹*a* @ *Propagated L mark* # *b* = *Propagated K C* # *M* ⟶
b ⊨*as CNot* (*remove1-mset L mark*) ∧ *L* ∈# *mark*› **for** *L mark a b*
using *conf* **unfolding** *cdcl*$_W$*-conflicting-def*
by *fastforce*
from *this*[*of* ‹[]›] **have** [*simp*]: ‹*K* ∈# *C*› ‹*M* ⊨*as CNot* (*remove1-mset K C*)›
by *auto*
have [*simp*]: ‹*get-maximum-level* (*Propagated K C* # *M*) *D* = *get-maximum-level M D*›
by (*rule get-maximum-level-skip-first*)
(*auto simp*: *atms-of-def atm-of-eq-atm-of uminus-lit-swap*[*symmetric*])

have ‹*get-maximum-level M D* < *count-decided M*›
using *nsr tr confl K proped count-decided-ge-get-maximum-level*[*of M D*]
by (*auto simp*: *resolve.simps get-level-cons-if atm-of-eq-atm-of*)
then show *?thesis* **by** *simp*
next
case *proped*: *False*
have ‹*get-maximum-level* (*tl* (*trail S*)) *D* < *count-decided* (*trail S*)›
using *tr confl K proped count-decided-ge-get-maximum-level*[*of* ‹*tl* (*trail S*)› *D*]
by (*cases* ‹*trail S*›; *cases* ‹*hd* (*trail S*)›)
(*auto simp*: *resolve.simps get-level-cons-if atm-of-eq-atm-of*)
then show *?thesis*
by *simp*
qed
show *H2*: *?C*
proof (*cases* ‹*is-proped* (*hd* (*trail S*))›)

```
  case proped: True
  obtain C M where
    [simp]: ‹trail S = Propagated K C # M›
    using tr proped K-def
    by (cases ‹trail S›; cases ‹hd (trail S)›)
      (auto simp: K-def)
  have ‹a @ Propagated L mark # b = Propagated K C # M ⟶
     b ⊨as CNot (remove1-mset L mark) ∧ L ∈# mark› for L mark a b
    using conf unfolding cdcl_W-conflicting-def
    by fastforce
  from this[of ‹[]›] have [simp]: ‹K ∈# C› ‹M ⊨as CNot (remove1-mset K C)›
    by auto
  have [simp]: ‹get-maximum-level (Propagated K C # M) D = get-maximum-level M D›
    by (rule get-maximum-level-skip-first)
      (auto simp: atms-of-def atm-of-eq-atm-of uminus-lit-swap[symmetric])

  have ‹get-maximum-level M D < count-decided M›
    using nsr tr confl K proped count-decided-ge-get-maximum-level[of M D]
    by (auto simp: resolve.simps get-level-cons-if atm-of-eq-atm-of)
  then show ?thesis by simp
next
  case proped: False
  have ‹get-maximum-level (tl (trail S)) D = get-maximum-level (trail S) D›
    apply (rule get-maximum-level-cong)
    using K-def ‹- K ∉# D› ‹K ∉# D›
    apply (cases ‹trail S›)
    by (auto simp: get-level-cons-if atm-of-eq-atm-of)
  moreover have ‹get-maximum-level (tl (trail S)) D < count-decided (trail S)›
    using tr confl K proped count-decided-ge-get-maximum-level[of ‹tl (trail S)› D]
    by (cases ‹trail S›; cases ‹hd (trail S)›)
       (auto simp: resolve.simps get-level-cons-if atm-of-eq-atm-of)
  ultimately show ?thesis
    by (simp add: K-def)
qed

have H:
  ‹get-level (trail S) L < local.backtrack-lvl S›
  if ‹L ∈# remove1-mset (-K) (the (conflicting S))›
  for L
proof (cases ‹is-proped (hd (trail S))›)
  case proped: True
  obtain C M where
    [simp]: ‹trail S = Propagated K C # M›
    using tr proped K-def
    by (cases ‹trail S›; cases ‹hd (trail S)›)
      (auto simp: K-def)
  have ‹a @ Propagated L mark # b = Propagated K C # M ⟶
     b ⊨as CNot (remove1-mset L mark) ∧ L ∈# mark› for L mark a b
    using conf unfolding cdcl_W-conflicting-def
    by fastforce
  from this[of ‹[]›] have [simp]: ‹K ∈# C› ‹M ⊨as CNot (remove1-mset K C)›
    by auto
  have [simp]: ‹get-maximum-level (Propagated K C # M) D = get-maximum-level M D›
    by (rule get-maximum-level-skip-first)
      (auto simp: atms-of-def atm-of-eq-atm-of uminus-lit-swap[symmetric])
```

```
    have ‹get-maximum-level M D < count-decided M›
      using nsr tr confl K that proped count-decided-ge-get-maximum-level[of M D]
      by (auto simp: resolve.simps get-level-cons-if atm-of-eq-atm-of)
    then show ?thesis
      using get-maximum-level-ge-get-level[of L D M] that
      by (auto simp: resolve.simps get-level-cons-if atm-of-eq-atm-of)
  next
    case proped: False
    have L-K: ‹L ≠ − K› ‹−L ≠ K› ‹L ≠ −lit-of (hd (trail S))›
      using that by (auto simp: uminus-lit-swap K-def[symmetric])
    have ‹L ≠ lit-of (hd (trail S))›
      using tr that K-def ‹K ∉# D›
      by (cases ‹trail S›; cases ‹hd (trail S)›)
         (auto simp: resolve.simps get-level-cons-if atm-of-eq-atm-of)

    have ‹get-maximum-level (tl (trail S)) D < count-decided (trail S)›
      using tr confl K that proped count-decided-ge-get-maximum-level[of ‹tl (trail S)› D]
      by (cases ‹trail S›; cases ‹hd (trail S)›)
         (auto simp: resolve.simps get-level-cons-if atm-of-eq-atm-of)
    then show ?thesis
      using get-maximum-level-ge-get-level[of L D ‹(trail S)›] that tr L-K ‹L ≠ lit-of (hd (trail S))›
        count-decided-ge-get-level[of ‹tl (trail S)› L] proped
      by (cases ‹trail S›; cases ‹hd (trail S)›)
        (auto simp: resolve.simps get-level-cons-if atm-of-eq-atm-of)
  qed
  have [simp]: ‹get-level (trail S) K = local.backtrack-lvl S›
    using tr K-def
    by (cases ‹trail S›; cases ‹hd (trail S)›)
      (auto simp: resolve.simps get-level-cons-if atm-of-eq-atm-of)
  show ?A
    apply (rule distinct-set-mset-eq)
    subgoal using dist by auto
    subgoal using dist by (auto simp: distinct-mset-filter K-def[symmetric])
    subgoal using H by (auto simp: K-def[symmetric])
    done
  show ?B
    using H1 .
qed

end

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

context conflict-driven-clause-learning_W
begin
```

1.1.11 Termination

No Relearning of a clause

Because of the conflict minimisation, this version is less clear than the version without: instead of extracting the clause from the conflicting clause, we must take it from the clause used to backjump; i.e., the annotation of the first literal of the trail.

We also prove below that no learned clause is subsumed by a (smaller) clause in the clause set.

lemma *$cdcl_W$-stgy-no-relearned-clause*:
  assumes
    cdcl: ‹*backtrack S T*› **and**
    inv: ‹*$cdcl_W$-all-struct-inv S*› **and**
    smaller: ‹*no-smaller-propa S*›
  shows
    ‹*mark-of* (*hd-trail T*) ∉# *clauses S*›
proof (*rule ccontr*)
  assume *n-dist*: ‹¬ *?thesis*›
  obtain *K L* :: *'v literal* **and**
    M1 M2 :: (*'v*, *'v clause*) *ann-lit list* **and** *i* :: *nat* **and** *D D'* **where**
    confl-S: *conflicting S* = *Some* (*add-mset 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**
    max-D-L: *get-level* (*trail S*) *L* = *get-maximum-level* (*trail S*) (*add-mset L D'*) **and**
    i: *get-maximum-level* (*trail S*) *D'* ≡ *i* **and**
    lev-K: *get-level* (*trail S*) *K* = *i* + *1* **and**
    T: *T* ∼ *cons-trail* (*Propagated L* (*add-mset L D'*))
        (*reduce-trail-to M1*
          (*add-learned-cls* (*add-mset L D'*)
            (*update-conflicting None S*))) **and**
    D-D': ‹*D'* ⊆# *D*› **and**
    ‹*clauses S* ⊨*pm add-mset L D'*›
    using *cdcl* **by** (*auto elim*!: *rulesE*)

  obtain *M2'* **where** *M2'*: ‹*trail S* = (*M2'* @ *M2*) @ *Decided K* # *M1*›
    using *decomp* **by** *auto*
  have *inv-T*: ‹*$cdcl_W$-all-struct-inv T*›
    using *cdcl $cdcl_W$-stgy-$cdcl_W$-all-struct-inv inv W-other backtrack bj*
      $cdcl_W$-all-struct-inv-inv $cdcl_W$-$cdcl_W$-restart **by** *blast*

  have *M1-D'*: ‹*M1* ⊨*as CNot D'*›
    using *backtrack-M1-CNot-D'*[*of S D'* ‹*i*› *K M1 M2 L* ‹*add-mset L D*› *T*
        ‹*Propagated L* (*add-mset L D'*)›] *inv confl-S decomp i T D-D' lev-K lev-L max-D-L*
    unfolding *$cdcl_W$-all-struct-inv-def $cdcl_W$-conflicting-def $cdcl_W$-M-level-inv-def*
    by (*auto simp*: *subset-mset-trans-add-mset*)
  have ‹*undefined-lit M1 L*›
    using *inv-T T decomp* **unfolding** *$cdcl_W$-all-struct-inv-def $cdcl_W$-M-level-inv-def*
    by (*auto simp*: *defined-lit-map*)
  moreover have ‹*D'* + {#*L*#} ∈# *clauses S*›
    using *n-dist T* **by** (*auto simp*: *clauses-def*)
  ultimately show *False*
    using *smaller M1-D'* **unfolding** *no-smaller-propa-def M2'* **by** *blast*
qed

lemma *$cdcl_W$-stgy-no-relearned-larger-clause*:
  assumes
    cdcl: ‹*backtrack S T*› **and**
    inv: ‹*$cdcl_W$-all-struct-inv S*› **and**
    smaller: ‹*no-smaller-propa S*› **and**
    smaller-conf: ‹*no-smaller-confl S*› **and**
    E-subset: ‹*E* ⊂# *mark-of* (*hd-trail T*)›
  shows ‹*E* ∉# *clauses S*›
proof (*rule ccontr*)

assume *n-dist*: ‹¬ *?thesis*›
obtain *K L* :: *'v literal* **and**
M1 M2 :: (*'v*, *'v clause*) *ann-lit list* **and** *i* :: *nat* **and** *D D'* **where**
confl-S: *conflicting S* = *Some* (*add-mset 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**
max-D-L: *get-level* (*trail S*) *L* = *get-maximum-level* (*trail S*) (*add-mset L D'*) **and**
i: *get-maximum-level* (*trail S*) *D'* ≡ *i* **and**
lev-K: *get-level* (*trail S*) *K* = *i* + *1* **and**
T: *T* ∼ *cons-trail* (*Propagated L* (*add-mset L D'*))
(*reduce-trail-to M1*
(*add-learned-cls* (*add-mset L D'*)
(*update-conflicting None S*))) **and**
D-D': ‹*D'* ⊆# *D*› **and**
‹*clauses S* ⊨*pm add-mset L D'*›
using *cdcl* **by** (*auto elim*!: *rulesE*)

obtain *M2'* **where** *M2'*: ‹*trail S* = (*M2'* @ *M2*) @ *Decided K* # *M1*›
using *decomp* **by** *auto*
have *inv-T*: ‹$cdcl_W$*-all-struct-inv T*›
using *cdcl* $cdcl_W$*-stgy-*$cdcl_W$*-all-struct-inv inv W-other backtrack bj*
$cdcl_W$*-all-struct-inv-inv* $cdcl_W$*-*$cdcl_W$*-restart* **by** *blast*
have ‹*distinct-mset* (*add-mset L D'*)›
using *inv-T T* **unfolding** $cdcl_W$*-all-struct-inv-def distinct-*$cdcl_W$*-state-def*
by *auto*
then have *dist-E*: ‹*distinct-mset E*›
using *distinct-mset-mono-strict*[*OF E-subset*] *T* **by** *auto*

have *M1-D'*: ‹*M1* ⊨*as CNot D'*›
using *backtrack-M1-CNot-D'*[*of S D'* ‹*i*› *K M1 M2 L* ‹*add-mset L D*› *T*
‹*Propagated L* (*add-mset L D'*)›] *inv confl-S decomp i T D-D' lev-K lev-L max-D-L*
unfolding $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-conflicting-def* $cdcl_W$*-M-level-inv-def*
by (*auto simp*: *subset-mset-trans-add-mset*)
have *undef-L*: ‹*undefined-lit M1 L*›
using *inv-T T decomp* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def*
by (*auto simp*: *defined-lit-map*)

show *False*
proof (*cases* ‹*L* ∈# *E*›)
case *True*
then obtain *E'* **where**
E: ‹*E* = *add-mset L E'*›
by (*auto dest*: *multi-member-split*)
then have ‹*distinct-mset E'*› **and** ‹*L* ∉# *E'*› **and** *E'-E'*: ‹*E'* ⊆# *D'*›
using *dist-E T E-subset* **by** *auto*
then have *M1-E'*: ‹*M1* ⊨*as CNot E'*›
using *M1-D' T* **unfolding** *true-annots-true-cls-def-iff-negation-in-model*
by (*auto dest*: *multi-member-split*[*of - E*] *mset-subset-eq-insertD*)
have *propa*: ‹⋀*M' K M L D*. *trail S* = *M'* @ *Decided K* # *M* ⟹
D + {#*L*#} ∈# *clauses S* ⟹ *undefined-lit M L* ⟹ ¬ *M* ⊨*as CNot D*›
using *smaller* **unfolding** *no-smaller-propa-def* **by** *blast*
show *False*
using *M1-E' propa*[*of* ‹*M2'* @ *M2*› *K M1 E'*, *OF M2'* - *undef-L*] *n-dist* **unfolding** *E*
by *auto*
next
case *False*

then have ‹$E \subseteq\#\ D'$›
  using *E-subset T* **by** (*auto simp*: *subset-add-mset-notin-subset*)
  then have *M1-E*: ‹*M1* $\models$*as CNot E*›
  using *M1-D′ T dist-E E-subset* **unfolding** *true-annots-true-cls-def-iff-negation-in-model*
  by (*auto dest*: *multi-member-split*[*of - E*] *mset-subset-eq-insertD*)
  have *confl*: ‹$\bigwedge M'$ *K M L D*. *trail S* = *M′* @ *Decided K* # *M* $\Longrightarrow$
  D $\in\#$ *clauses S* $\Longrightarrow$ ¬ *M* $\models$*as CNot D*›
  using *smaller-conf* **unfolding** *no-smaller-confl-def* **by** *blast*
  show *False*
  using *confl*[*of* ‹*M2′* @ *M2*› *K M1 E*, *OF M2′*] *n-dist M1-E*
  by *auto*
qed
qed

lemma *cdcl*$_W$*-stgy-no-relearned-highest-subres-clause*:
assumes
cdcl: ‹*backtrack S T*› **and**
inv: ‹*cdcl*$_W$*-all-struct-inv S*› **and**
smaller: ‹*no-smaller-propa S*› **and**
smaller-conf: ‹*no-smaller-confl S*› **and**
E-subset: ‹*mark-of* (*hd-trail T*) = *add-mset* (*lit-of* (*hd-trail T*)) *E*›
shows ‹*add-mset* (− *lit-of* (*hd-trail T*)) *E* $\notin\#$ *clauses S*›
proof (*rule ccontr*)
assume *n-dist*: ‹¬ *?thesis*›
obtain *K L* :: *′v literal* **and**
M1 M2 :: (*′v*, *′v clause*) *ann-lit list* **and** *i* :: *nat* **and** *D D′* **where**
confl-S: *conflicting S* = *Some* (*add-mset 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**
max-D-L: *get-level* (*trail S*) *L* = *get-maximum-level* (*trail S*) (*add-mset L D′*) **and**
i: *get-maximum-level* (*trail S*) *D′* ≡ *i* **and**
lev-K: *get-level* (*trail S*) *K* = *i* + *1* **and**
T: *T* ∼ *cons-trail* (*Propagated L* (*add-mset L D′*))
(*reduce-trail-to M1*
(*add-learned-cls* (*add-mset L D′*)
(*update-conflicting None S*))) **and**
D-D′: ‹$D' \subseteq\#\ D$› **and**
‹*clauses S* $\models$*pm add-mset L D′*›
using *cdcl* **by** (*auto elim*!: *rulesE*)

obtain *M2′* **where** *M2′*: ‹*trail S* = (*M2′* @ *M2*) @ *Decided K* # *M1*›
using *decomp* **by** *auto*
have *inv-T*: ‹*cdcl*$_W$*-all-struct-inv T*›
using *cdcl cdcl*$_W$*-stgy-cdcl*$_W$*-all-struct-inv inv W-other backtrack bj*
cdcl$_W$*-all-struct-inv-inv cdcl*$_W$*-cdcl*$_W$*-restart* **by** *blast*
have ‹*distinct-mset* (*add-mset L D′*)›
using *inv-T T* **unfolding** *cdcl*$_W$*-all-struct-inv-def distinct-cdcl*$_W$*-state-def*
by *auto*

have *M1-D′*: ‹*M1* $\models$*as CNot D′*›
using *backtrack-M1-CNot-D′*[*of S D′* ‹*i*› *K M1 M2 L* ‹*add-mset L D*› *T*
‹*Propagated L* (*add-mset L D′*)›] *inv confl-S decomp i T D-D′ lev-K lev-L max-D-L*
unfolding *cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-conflicting-def cdcl*$_W$*-M-level-inv-def*
by (*auto simp*: *subset-mset-trans-add-mset*)
have *undef-L*: ‹*undefined-lit M1 L*›
using *inv-T T decomp* **unfolding** *cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-M-level-inv-def*

by (*auto simp*: *defined-lit-map*)
then have *undef-uL*: ‹*undefined-lit M1* (−*L*)›
by *auto*
have *propa*: ‹$\bigwedge M'$ *K M L D*. *trail S* = *M'* @ *Decided K* # *M* ⟹
D + {#*L*#} ∈# *clauses S* ⟹ *undefined-lit M L* ⟹ ¬ *M* ⊨*as CNot D*›
using *smaller* unfolding *no-smaller-propa-def* by *blast*
have *E*[*simp*]: ‹*E* = *D'*›
using *E-subset T* by (*auto dest*: *multi-member-split*)
have *propa*: ‹$\bigwedge M'$ *K M L D*. *trail S* = *M'* @ *Decided K* # *M* ⟹
D + {#*L*#} ∈# *clauses S* ⟹ *undefined-lit M L* ⟹ ¬ *M* ⊨*as CNot D*›
using *smaller* unfolding *no-smaller-propa-def* by *blast*
show *False*
using *T M1-D' propa*[*of* ‹*M2'* @ *M2*› *K M1 D'*, *OF M2'* - *undef-uL*] *n-dist* unfolding *E*
by *auto*
qed

lemma $cdcl_W$*-stgy-distinct-mset*:
assumes
cdcl: ‹$cdcl_W$*-stgy S T*› and
inv: $cdcl_W$*-all-struct-inv S* and
smaller: ‹*no-smaller-propa S*› and
dist: ‹*distinct-mset* (*clauses S*)›
shows
‹*distinct-mset* (*clauses T*)›
proof (*rule ccontr*)
assume *n-dist*: ‹¬ *distinct-mset* (*clauses T*)›
then have ‹*backtrack S T*›
using *cdcl dist* by (*auto simp*: $cdcl_W$*-stgy.simps* $cdcl_W$*-o.simps* $cdcl_W$*-bj.simps*
elim: *propagateE conflictE decideE skipE resolveE*)
then show *False*
using *n-dist* $cdcl_W$*-stgy-no-relearned-clause*[*of S T*] *dist*
by (*auto simp*: *inv smaller elim*!: *rulesE*)
qed

This is a more restrictive version of the previous theorem, but is a better bound for an implementation that does not do duplication removal (esp. as part of preprocessing).

lemma $cdcl_W$*-stgy-learned-distinct-mset*:
assumes
cdcl: ‹$cdcl_W$*-stgy S T*› and
inv: $cdcl_W$*-all-struct-inv S* and
smaller: ‹*no-smaller-propa S*› and
dist: ‹*distinct-mset* (*learned-clss S* + *remdups-mset* (*init-clss S*))›
shows
‹*distinct-mset* (*learned-clss T* + *remdups-mset* (*init-clss T*))›
proof (*rule ccontr*)
assume *n-dist*: ‹¬ *?thesis*›
then have ‹*backtrack S T*›
using *cdcl dist* by (*auto simp*: $cdcl_W$*-stgy.simps* $cdcl_W$*-o.simps* $cdcl_W$*-bj.simps*
elim: *propagateE conflictE decideE skipE resolveE*)
then show *False*
using *n-dist* $cdcl_W$*-stgy-no-relearned-clause*[*of S T*] *dist*
by (*auto simp*: *inv smaller clauses-def elim*!: *rulesE*)
qed

lemma *rtranclp-*$cdcl_W$*-stgy-distinct-mset-clauses*:

assumes
st: $cdcl_W$*-stgy*** *R S* **and**
invR: $cdcl_W$*-all-struct-inv R* **and**
dist: *distinct-mset* (*clauses R*) **and**
no-smaller: ‹*no-smaller-propa R*›
shows *distinct-mset* (*clauses S*)
using *assms* **by** (*induction rule*: *rtranclp-induct*)
(*auto simp*: $cdcl_W$*-stgy-distinct-mset rtranclp-*$cdcl_W$*-stgy-no-smaller-propa*
rtranclp-$cdcl_W$*-stgy-*$cdcl_W$*-all-struct-inv*)

lemma *rtranclp-*$cdcl_W$*-stgy-distinct-mset-learned-clauses*:
assumes
st: $cdcl_W$*-stgy*** *R S* **and**
invR: $cdcl_W$*-all-struct-inv R* **and**
dist: *distinct-mset* (*learned-clss R* + *remdups-mset* (*init-clss R*)) **and**
no-smaller: ‹*no-smaller-propa R*›
shows *distinct-mset* (*learned-clss S* + *remdups-mset* (*init-clss S*))
using *assms* **by** (*induction rule*: *rtranclp-induct*)
(*auto simp*: $cdcl_W$*-stgy-learned-distinct-mset rtranclp-*$cdcl_W$*-stgy-no-smaller-propa*
rtranclp-$cdcl_W$*-stgy-*$cdcl_W$*-all-struct-inv*)

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 no-smaller-propa-def*)

lemma $cdcl_W$*-stgy-learned-distinct-mset-new*:
assumes
cdcl: ‹$cdcl_W$*-stgy S T*› **and**
inv: $cdcl_W$*-all-struct-inv S* **and**
smaller: ‹*no-smaller-propa S*› **and**
dist: ‹*distinct-mset* (*learned-clss S* − *A*)›
shows ‹*distinct-mset* (*learned-clss T* − *A*)›
proof (*rule ccontr*)
have [*iff*]: ‹*distinct-mset* (*add-mset C* (*learned-clss S*) − *A*) ⟷
$C \notin\#$ (*learned-clss S*) − *A*› **for** *C*
using *dist distinct-mset-add-mset*[*of C* ‹*learned-clss S* − *A*›]
proof −
have *f1*: *learned-clss S* − *A* = *remove1-mset C* (*add-mset C* (*learned-clss S*) − *A*)
by (*metis Multiset.diff-right-commute add-mset-remove-trivial*)
have *remove1-mset C* (*add-mset C* (*learned-clss S*) − *A*) = *add-mset C* (*learned-clss S*) − *A* ⟶
distinct-mset (*add-mset C* (*learned-clss S*) − *A*)
by (*metis* (*no-types*) *Multiset.diff-right-commute add-mset-remove-trivial dist*)
then have ¬ *distinct-mset* (*add-mset C* (*learned-clss S* − *A*)) ∨
distinct-mset (*add-mset C* (*learned-clss S*) − *A*) ≠ ($C \in\#$ *learned-clss S* − *A*)
by (*metis* (*full-types*) *Multiset.diff-right-commute*
distinct-mset-add-mset[*of C* ‹*learned-clss S* − *A*›] *add-mset-remove-trivial*
diff-single-trivial insert-DiffM)
then show *?thesis*
using *f1* **by** (*metis* (*full-types*) *distinct-mset-add-mset*[*of C* ‹*learned-clss S* − *A*›]
diff-single-trivial dist insert-DiffM)

qed

assume *n-dist*: ⟨¬ *?thesis*⟩
then have ⟨*backtrack S T*⟩
using *cdcl dist* **by** (*auto simp*: $cdcl_W$*-stgy.simps* $cdcl_W$*-o.simps* $cdcl_W$*-bj.simps*
elim: *propagateE conflictE decideE skipE resolveE*)
then show *False*
using *n-dist* $cdcl_W$*-stgy-no-relearned-clause*[*of S T*]
by (*auto simp*: *inv smaller clauses-def elim*!: *rulesE*
dest!: *in-diffD*)
qed

lemma *rtranclp-*$cdcl_W$*-stgy-distinct-mset-clauses-new-abs*:
assumes
st: $cdcl_W$*-stgy*** *R S* **and**
invR: $cdcl_W$*-all-struct-inv R* **and**
no-smaller: ⟨*no-smaller-propa R*⟩ **and**
⟨*distinct-mset* (*learned-clss R* − *A*)⟩
shows *distinct-mset* (*learned-clss S* − *A*)
using *assms* **by** (*induction rule*: *rtranclp-induct*)
(*auto simp*: $cdcl_W$*-stgy-distinct-mset rtranclp-*$cdcl_W$*-stgy-no-smaller-propa*
rtranclp-$cdcl_W$*-stgy-*$cdcl_W$*-all-struct-inv*
$cdcl_W$*-stgy-learned-distinct-mset-new*)

lemma *rtranclp-*$cdcl_W$*-stgy-distinct-mset-clauses-new*:
assumes
st: $cdcl_W$*-stgy*** *R S* **and**
invR: $cdcl_W$*-all-struct-inv R* **and**
no-smaller: ⟨*no-smaller-propa R*⟩
shows *distinct-mset* (*learned-clss S* − *learned-clss R*)
using *assms* **by** (*rule rtranclp-*$cdcl_W$*-stgy-distinct-mset-clauses-new-abs*) *auto*

Decrease of a Measure

fun $cdcl_W$*-restart-measure* **where**
$cdcl_W$*-restart-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*:
assumes
no-strange-atm S **and**
no-d: *no-dup* (*trail S*) **and**
finite (*atms-of-mm* (*init-clss S*))
shows *length* (*trail S*) ≤ *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 *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*)

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 $cdcl_W$*-restart-measure-decreasing*:
  fixes *S* :: *'st*
  assumes
    $cdcl_W$*-restart 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'* ⟹ *mark-of* (*hd-trail S'*) ∉# *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 ($cdcl_W$*-restart-measure S'*, $cdcl_W$*-restart-measure S*) ∈ *lexn less-than 3*
  using *assms*(*1*) *assms*(*2*,*3*)
proof (*induct rule*: $cdcl_W$*-restart-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* $cdcl_W$*-restart.propagate* $cdcl_W$*-restart-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*))
$\leq$ *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*) $\leq$ *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*) *S*)
using *decide-rule decide.hyps* **by** *force*
then have $cdcl_W$*-restart S* (*cons-trail* (*Decided L*) *S*)
using $cdcl_W$*-restart.simps* $cdcl_W$*-o.intros* **by** *blast* **}** **note** $cdcl_W$*-restart* = *this*
moreover **{**
have *lev*: $cdcl_W$*-M-level-inv* (*cons-trail* (*Decided L*) *S*)
using $cdcl_W$*-restart M-level* $cdcl_W$*-restart-consistent-inv*[*OF* $cdcl_W$*-restart*] **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*) *S*)
using *M-level alien calculation*(*4*) $cdcl_W$*-restart-no-strange-atm-inv* **by** *blast*
then have *length* (*Decided L* # (*trail S*))
$\leq$ *card* (*atms-of-mm* (*init-clss S*))
using *no-dup undef*
length-model-le-vars[*of cons-trail* (*Decided L*) *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 D'*) **note** *conf* = *this*(*1*) **and** *decomp* = *this*(*3*) **and** *D-D'* = *this*(*7*)
and *T* = *this*(*9*)
let *?D'* = ‹*add-mset L D'*›
have *bt*: *backtrack S T*
using *backtrack-rule*[*OF backtrack.hyps*] **by** *auto*
have *?D'* ∉# *learned-clss S*
using *no-relearn*[*OF bt*] *conf T* **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 T*
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 T*. ¬ *tautology s*
using *learned-clss-are-not-tautologies*[*OF* $cdcl_W$*-restart.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*))
≤ *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*) ⊆ *atms-of-mm* (*init-clss S*)
using *alien conf atms-of-subset-mset-mono*[*OF D-D*′] **unfolding** *no-strange-atm-def*
by *auto*
then have *card-f*: *card* (*atms-of-mm* ({#?*D*′#} + *learned-clss S*))
≤ *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*))
≤ *3* ^ *card* (*atms-of-mm* (*init-clss S*)) **by** *simp*
ultimately have (*3*::*nat*) ^ *card* (*atms-of-mm* (*init-clss S*))
≥ *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*
next
case (*forget C T*) **note** *no-forget* = *this*(*9*)
then have *C* ∈# *learned-clss S* **and** *C* ∉# *learned-clss T*
using *forget.hyps* **by** *auto*
then have ¬ *learned-clss S* ⊆# *learned-clss T*
by (*auto simp add*: *mset-subset-eqD*)
then show *?case* **using** *no-forget* **by** *blast*
qed

lemma $cdcl_W$*-stgy-step-decreasing*:
fixes *S T* :: ′*st*
assumes
cdcl: ‹$cdcl_W$*-stgy S T*› **and**
struct-inv: ‹$cdcl_W$*-all-struct-inv S*› **and**
smaller: ‹*no-smaller-propa S*›
shows ($cdcl_W$*-restart-measure T*, $cdcl_W$*-restart-measure S*) ∈ *lexn less-than 3*
proof (*rule* $cdcl_W$*-restart-measure-decreasing*)
show ‹$cdcl_W$*-restart S T*›
using *cdcl* $cdcl_W$-$cdcl_W$*-restart* $cdcl_W$*-stgy*-$cdcl_W$ **by** *blast*
show ‹¬ (*learned-clss S* ⊆# *learned-clss T* ∧ [] = *trail T* ∧ *conflicting T* = *None*)›
using *cdcl* **by** (*cases rule*: $cdcl_W$*-stgy-cases*) (*auto elim*!: *rulesE*)
show ‹*learned-clss S* ⊆# *learned-clss T*›
using *cdcl* **by** (*cases rule*: $cdcl_W$*-stgy-cases*) (*auto elim*!: *rulesE*)
show ‹*mark-of* (*hd-trail S*′) ∉# *learned-clss S*› **if** ‹*backtrack S S*′› **for** *S*′
using $cdcl_W$*-stgy-no-relearned-clause*[*of S S*′] $cdcl_W$*-stgy-no-smaller-propa*[*of S S*′]
cdcl struct-inv smaller that **unfolding** *clauses-def*
by (*auto elim*!: *rulesE*)
show ‹*no-strange-atm S*› **and** ‹$cdcl_W$*-M-level-inv S*› **and** ‹*distinct*-$cdcl_W$*-state S*› **and**
‹$cdcl_W$*-conflicting S*› **and** ‹∀ *s*∈#*learned-clss S*. ¬ *tautology s*›
using *struct-inv* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *blast*+
qed

lemma *empty-trail-no-smaller-propa*: ‹*trail R* = [] ⟹ *no-smaller-propa R*›
by (*simp add*: *no-smaller-propa-def*)

Roughly corresponds to theorem 2.9.15 page 100 of Weidenbach's book but using a different bound (the bound is below)

lemma *tranclp-cdcl$_W$-stgy-decreasing*:
fixes *R S T* :: *'st*
assumes *cdcl$_W$-stgy^{++} R S* **and**
tr: *trail R* = [] **and**
cdcl$_W$-all-struct-inv R
shows (*cdcl$_W$-restart-measure S*, *cdcl$_W$-restart-measure R*) ∈ *lexn less-than 3*
using *assms*
apply *induction*
using *empty-trail-no-smaller-propa cdcl$_W$-stgy-no-relearned-clause cdcl$_W$-stgy-step-decreasing*
apply *blast*
using *tranclp-into-rtranclp*[*of cdcl$_W$-stgy R*] *lexn-transI*[*OF trans-less-than*, *of 3*]
rtranclp-cdcl$_W$-stgy-no-smaller-propa **unfolding** *trans-def*
by (*meson cdcl$_W$-stgy-step-decreasing empty-trail-no-smaller-propa*
rtranclp-cdcl$_W$-stgy-cdcl$_W$-all-struct-inv)

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

The following theorems is deeply linked with the strategy: It shows that a decision alone cannot lead to a conflict. This is obvious but I expect this to be a major part of the proof that the number of learnt clause cannot be larger that $(2::'a)^n$.

lemma *no-conflict-after-decide*:
assumes
dec: ‹*decide S T*› **and**
inv: ‹*cdcl$_W$-all-struct-inv T*› **and**
smaller: ‹*no-smaller-propa T*› **and**
smaller-confl: ‹*no-smaller-confl T*›
shows ‹¬*conflict T U*›
proof (*rule ccontr*)
assume ‹¬ *?thesis*›
then obtain *D* **where**
D: ‹*D* ∈# *clauses T*› **and**
confl: ‹*trail T* ⊨*as CNot D*›
by (*auto simp*: *conflict.simps*)
obtain *L* **where**
‹*conflicting S* = *None*› **and**
undef: ‹*undefined-lit* (*trail S*) *L*› **and**
‹*atm-of L* ∈ *atms-of-mm* (*init-clss S*)› **and**

```
      T: ‹T ∼ cons-trail (Decided L) S›
      using dec by (auto simp: decide.simps)
    have dist: ‹distinct-mset D›
      using inv D unfolding cdcl_W-all-struct-inv-def distinct-cdcl_W-state-def
      by (auto dest!: multi-member-split simp: clauses-def)
    have L-D: ‹L ∉# D›
      using confl undef T
      by (auto dest!: multi-member-split simp: Decided-Propagated-in-iff-in-lits-of-l)

    show False
    proof (cases ‹-L ∈# D›)
      case True
      have H: ‹trail T = M' @ Decided K # M ⟹
        D + {#L#} ∈# clauses T ⟹ undefined-lit M L ⟹ ¬ M ⊨as CNot D›
        for M K M' D L
        using smaller unfolding no-smaller-propa-def
        by auto
      have ‹trail S ⊨as CNot (remove1-mset (-L) D)›
        using true-annots-CNot-lit-of-notin-skip[of ‹Decided L› ‹trail S› ‹remove1-mset (-L) D›] T True
          dist confl L-D
        by (auto dest: multi-member-split)
      then show False
        using True H[of ‹Nil› L ‹trail S› ‹remove1-mset (-L) D› ‹-L›] T D confl undef
        by auto
    next
      case False
      have H: ‹trail T = M' @ Decided K # M ⟹
        D ∈# clauses T ⟹ ¬ M ⊨as CNot D›
        for M K M' D
        using smaller-confl unfolding no-smaller-confl-def
        by auto
      have ‹trail S ⊨as CNot D›
        using true-annots-CNot-lit-of-notin-skip[of ‹Decided L› ‹trail S› D] T False
          dist confl L-D
        by (auto dest: multi-member-split)
      then show False
        using False H[of ‹Nil› L ‹trail S› D] T D confl undef
        by auto
    qed
  qed
```

```
abbreviation list-weight-propa-trail :: ‹ ('v literal, 'v literal, 'v literal multiset) annotated-lit list ⇒ bool
list› where
‹list-weight-propa-trail M ≡ map is-proped M›
```

```
definition comp-list-weight-propa-trail :: ‹nat ⇒ ('v literal, 'v literal, 'v literal multiset) annotated-lit
list ⇒ bool list› where
‹comp-list-weight-propa-trail b M ≡ replicate (b - length M) False @ list-weight-propa-trail M›
```

```
lemma comp-list-weight-propa-trail-append[simp]:
  ‹comp-list-weight-propa-trail b (M @ M') =
     comp-list-weight-propa-trail (b - length M') M @ list-weight-propa-trail M'›
  by (auto simp: comp-list-weight-propa-trail-def)
```

```
lemma comp-list-weight-propa-trail-append-single[simp]:
  ‹comp-list-weight-propa-trail b (M @ [K]) =
```

comp-list-weight-propa-trail (*b* − *1*) *M* @ [*is-proped K*]›
by (*auto simp*: *comp-list-weight-propa-trail-def*)

lemma *comp-list-weight-propa-trail-cons*[*simp*]:
‹*comp-list-weight-propa-trail b* (*K* # *M'*) =
comp-list-weight-propa-trail (*b* − *Suc* (*length M'*)) [] @ *is-proped K* # *list-weight-propa-trail M'*›
by (*auto simp*: *comp-list-weight-propa-trail-def*)

fun *of-list-weight* :: ‹*bool list* ⇒ *nat*› **where**
‹*of-list-weight* [] = *0*›
| ‹*of-list-weight* (*b* # *xs*) = (*if b then 1 else 0*) + *2* ∗ *of-list-weight xs*›

lemma *of-list-weight-append*[*simp*]:
‹*of-list-weight* (*a* @ *b*) = *of-list-weight a* + *2*^(*length a*) ∗ *of-list-weight b*›
by (*induction a*) *auto*

lemma *of-list-weight-append-single*[*simp*]:
‹*of-list-weight* (*a* @ [*b*]) = *of-list-weight a* + *2*^(*length a*) ∗ (*if b then 1 else 0*)›
using *of-list-weight-append*[*of* ‹*a*› ‹[*b*]›]
by (*auto simp del*: *of-list-weight-append*)

lemma *of-list-weight-replicate-False*[*simp*]: ‹*of-list-weight* (*replicate n False*) = *0*›
by (*induction n*) *auto*

lemma *of-list-weight-replicate-True*[*simp*]: ‹*of-list-weight* (*replicate n True*) = *2*^*n* − *1*›
apply (*induction n*)
subgoal by *auto*
subgoal for *m*
using *power-gt1-lemma*[*of* ‹*2* :: *nat*›]
by (*auto simp add*: *algebra-simps Suc-diff-Suc*)
done

lemma *of-list-weight-le*: ‹*of-list-weight xs* ≤ *2*^(*length xs*) − *1*›
proof −
have ‹*of-list-weight xs* ≤ *of-list-weight* (*replicate* (*length xs*) *True*)›
by (*induction xs*) *auto*
then show ‹*?thesis*›
by *auto*
qed

lemma *of-list-weight-lt*: ‹*of-list-weight xs* < *2*^(*length xs*)›
using *of-list-weight-le*[*of xs*] **by** (*metis One-nat-def Suc-le-lessD*
Suc-le-mono Suc-pred of-list-weight-le zero-less-numeral zero-less-power)

lemma [*simp*]: ‹*of-list-weight* (*comp-list-weight-propa-trail n* []) = *0*›
by (*auto simp*: *comp-list-weight-propa-trail-def*)

abbreviation *propa-weight*
:: ‹*nat* ⇒ (*'v literal*, *'v literal*, *'v literal multiset*) *annotated-lit list* ⇒ *nat*›
where
‹*propa-weight n M* ≡ *of-list-weight* (*comp-list-weight-propa-trail n M*)›

lemma *length-comp-list-weight-propa-trail*[*simp*]: ‹*length* (*comp-list-weight-propa-trail a M*) = *max* (*length M*) *a*›
by (*auto simp*: *comp-list-weight-propa-trail-def*)

lemma (**in** $-$)*pow2-times-n*:
⟨$Suc\ a \leq n \Longrightarrow 2 * 2\hat{}(n - Suc\ a) = (2::nat)\hat{}\ (n - a)$⟩
⟨$Suc\ a \leq n \Longrightarrow 2\hat{}(n - Suc\ a) * 2 = (2::nat)\hat{}\ (n - a)$⟩
⟨$Suc\ a \leq n \Longrightarrow 2\hat{}(n - Suc\ a) * b * 2 = (2::nat)\hat{}\ (n - a) * b$⟩
⟨$Suc\ a \leq n \Longrightarrow 2\hat{}(n - Suc\ a) * (b * 2) = (2::nat)\hat{}\ (n - a) * b$⟩
⟨$Suc\ a \leq n \Longrightarrow 2\hat{}(n - Suc\ a) * (2 * b) = (2::nat)\hat{}\ (n - a) * b$⟩
⟨$Suc\ a \leq n \Longrightarrow 2 * b * 2\hat{}(n - Suc\ a) = (2::nat)\hat{}\ (n - a) * b$⟩
⟨$Suc\ a \leq n \Longrightarrow 2 * (b * 2\hat{}(n - Suc\ a)) = (2::nat)\hat{}\ (n - a) * b$⟩
apply (*simp-all add*: *Suc-diff-Suc semiring-normalization-rules*(*27*))
using *Suc-diff-le* **by** *fastforce*+

lemma *decide-propa-weight*:
⟨*decide S T* $\Longrightarrow$ *n* $\geq$ *length* (*trail T*) $\Longrightarrow$ *propa-weight n* (*trail S*) $\leq$ *propa-weight n* (*trail T*)⟩
by (*auto elim*!: *decideE simp*: *comp-list-weight-propa-trail-def*
algebra-simps pow2-times-n)

lemma *propagate-propa-weight*:
⟨*propagate S T* $\Longrightarrow$ *n* $\geq$ *length* (*trail T*) $\Longrightarrow$ *propa-weight n* (*trail S*) $<$ *propa-weight n* (*trail T*)⟩
by (*auto elim*!: *propagateE simp*: *comp-list-weight-propa-trail-def*
algebra-simps pow2-times-n)

The theorem below corresponds the bound of theorem 2.9.15 page 100 of Weidenbach's book. In the current version there is no proof of the bound.

The following proof contains an immense amount of stupid bookkeeping. The proof itself is rather easy and Isabelle makes it extra-complicated.

Let's consider the sequence $S \to ... \to T$. The bookkeping part:

1. We decompose it into its components $f\ 0 \to f\ 1 \to ... \to f\ n$.
2. Then we extract the backjumps out of it, which are at position *nth-nj 0*, *nth-nj 1*, ...
3. Then we extract the conflicts out of it, which are at position *nth-confl 0*, *nth-confl 1*, ...

Then the simple part:

1. each backtrack increases *propa-weight*
2. but *propa-weight* is bounded by $(2::'a)^{card\ (atms\text{-}of\text{-}mm\ (init\text{-}clss\ S))}$ Therefore, we get the bound.

Comments on the proof:

- The main problem of the proof is the number of inductions in the bookkeeping part.
- The proof is actually by contradiction to make sure that enough backtrack step exists. This could probably be avoided, but without change in the proof.

  Comments on the bound:
- The proof is very very crude: Any propagation also decreases the bound. The lemma $[\![$*decide ?S ?T*; $cdcl_W$*-all-struct-inv ?T*; *no-smaller-propa ?T*; *no-smaller-confl ?T*$]\!] \Longrightarrow$ $\neg$ *conflict ?T ?U* above shows that a decision cannot lead immediately to a conflict.
- TODO: can a backtrack could be immediately followed by another conflict (if there are several conflicts for the initial backtrack)? If not the bound can be divided by two.

lemma *cdcl-pow2-n-learned-clauses*:
  assumes
    cdcl: ‹$cdcl_W{}^{**}$ *S T*› **and**
    confl: ‹*conflicting S* = *None*› **and**
    inv: ‹$cdcl_W$*-all-struct-inv S*›
  shows ‹*size* (*learned-clss T*) ≤ *size* (*learned-clss S*) + *2* ^ (*card* (*atms-of-mm* (*init-clss S*)))›
    (**is** ‹- ≤ - + *?b*›)
proof (*rule ccontr*)
  assume *ge*: ‹¬ *?thesis*›
  let *?m* = ‹*card* (*atms-of-mm* (*init-clss S*))›
  obtain *n* :: *nat* **where**
    n: ‹($cdcl_W$ ^^*n*) *S T*›
    using *cdcl* **unfolding** *rtranclp-power* **by** *fast*
  then obtain *f* :: ‹*nat* ⇒ *'st*› **where**
    f: ‹⋀*i*. *i* < *n* ⟹ $cdcl_W$ (*f i*) (*f* (*Suc i*))› **and**
    [*simp*]: ‹*f 0* = *S*› **and**
    [*simp*]: ‹*f n* = *T*›
    using *power-ex-decomp*[*OF n*]
    by *auto*

  have *cdcl-st-k*: ‹$cdcl_W{}^{**}$ *S* (*f k*)› **if** ‹*k* ≤ *n*› **for** *k*
    using *that*
    apply (*induction k*)
    subgoal by *auto*
    subgoal for *k* **using** *f*[*of k*] **by** (*auto*)
    done
  let *?g* = ‹λ*i*. *size* (*learned-clss* (*f i*))›
  have ‹*?g 0* = *size* (*learned-clss S*)›
    by *auto*
  have *g-n*: ‹*?g n* > *?g 0* + *2* ^ (*card* (*atms-of-mm* (*init-clss S*)))›
    using *ge* **by** *auto*
  have *g*: ‹*?g* (*Suc i*) = *?g i* ∨ (*?g* (*Suc i*) = *Suc* (*?g i*) ∧ *backtrack* (*f i*) (*f* (*Suc i*)))› **if** ‹*i* < *n*›
    for *i*
    using *f*[*OF that*]
    by (*cases rule*: $cdcl_W$*.cases*)
      (*auto elim*: *propagateE conflictE decideE backtrackE skipE resolveE*
        simp: $cdcl_W$*-o.simps* $cdcl_W$*-bj.simps*)
  have *g-le*: ‹*?g i* ≤ *i* + *?g 0*› **if** ‹*i* ≤ *n*› **for** *i*
    using *that*
    apply (*induction i*)
    subgoal by *auto*
    subgoal for *i*
      using *g*[*of i*]
      by *auto*
    done
  from *this*[*of n*] **have** *n-ge-m*: ‹*n* > *?b*›
    using *g-n ge* **by** *auto*
  then have *n0*: ‹*n* > *0*›
    using *not-add-less1* **by** *fastforce*
  define *nth-bj* **where**
    ‹*nth-bj* = *rec-nat 0* (λ- *j*. (*LEAST i*. *i* > *j* ∧ *i* < *n* ∧ *backtrack* (*f i*) (*f* (*Suc i*))))›
  have [*simp*]: ‹*nth-bj 0* = *0*›
    by (*auto simp*: *nth-bj-def*)
  have *nth-bj-Suc*: ‹*nth-bj* (*Suc i*) = (*LEAST x*. *nth-bj i* < *x* ∧ *x* < *n* ∧ *backtrack* (*f x*) (*f* (*Suc x*)))›
    for *i*
    by (*auto simp*: *nth-bj-def*)

have *between-nth-bj-not-bt*:
‹¬*backtrack* (*f k*) (*f* (*Suc k*))›
if ‹*k* < *n*› ‹*k* > *nth-bj i*› ‹*k* < *nth-bj* (*Suc i*) › **for** *k i*
using *not-less-Least*[*of k* ‹λ*x*. *nth-bj i* < *x* ∧ *x* < *n* ∧ *backtrack* (*f x*) (*f* (*Suc x*))›] *that*
unfolding *nth-bj-Suc*[*symmetric*]
by *auto*

have *g-nth-bj-eq*:
‹*?g* (*Suc k*) = *?g k*›
if ‹*k* < *n*› ‹*k* > *nth-bj i*› ‹*k* < *nth-bj* (*Suc i*)› **for** *k i*
using *between-nth-bj-not-bt*[*OF that*(*1*−*3*)] *f*[*of k*, *OF that*(*1*)]
by (*auto elim*: *propagateE conflictE decideE backtrackE skipE resolveE*
simp: *cdcl*$_W$*-o.simps cdcl*$_W$*-bj.simps cdcl*$_W$*.simps*)
have *g-nth-bj-eq2*:
‹*?g* (*Suc k*) = *?g* (*Suc* (*nth-bj i*))›
if ‹*k* < *n*› ‹*k* > *nth-bj i*› ‹*k* < *nth-bj* (*Suc i*)› **for** *k i*
using *that*
apply (*induction k*)
subgoal by *blast*
subgoal for *k*
using *g-nth-bj-eq less-antisym* **by** *fastforce*
done
have [*simp*]: ‹*?g* (*Suc 0*) = *?g 0*›
using *confl f*[*of 0*] *n0*
by (*auto elim*: *propagateE conflictE decideE backtrackE skipE resolveE*
simp: *cdcl*$_W$*-o.simps cdcl*$_W$*-bj.simps cdcl*$_W$*.simps*)
have ‹(*?g* (*nth-bj i*) = *size* (*learned-clss S*) + (*i* − *1*)) ∧
nth-bj i < *n* ∧
nth-bj i ≥ *i* ∧
(*i* > *0* ⟶ *backtrack* (*f* (*nth-bj i*)) (*f* (*Suc* (*nth-bj i*)))) ∧
(*i* > *0* ⟶ *?g* (*Suc* (*nth-bj i*)) = *size* (*learned-clss S*) + *i*) ∧
(*i* > *0* ⟶ *nth-bj i* > *nth-bj* (*i*−*1*))›
if ‹*i* ≤ *?b*+*1*›
for *i*
using *that*
proof (*induction i*)
case *0*
then show *?case* **using** *n0* **by** *auto*
next
case (*Suc i*)
then have *IH*: ‹*?g* (*nth-bj i*) = *size* (*learned-clss S*) + (*i* − *1*)›
‹*0* < *i* ⟹ *backtrack* (*f* (*nth-bj i*)) (*f* (*Suc* (*nth-bj i*)))›
‹*0* < *i* ⟹ *?g* (*Suc* (*nth-bj i*)) = *size* (*learned-clss S*) + *i*› **and**
i-le-m: ‹*Suc i* ≤ *?b*+*1*› **and**
le-n: ‹*nth-bj i* < *n*› **and**
gei: ‹*nth-bj i* ≥ *i*›
by *auto*
have *ex-larger*: ‹∃ *x*>*nth-bj i*. *x* < *n* ∧ *backtrack* (*f x*) (*f* (*Suc x*))›
proof (*rule ccontr*)
assume ‹¬ *?thesis*›
then have [*simp*]: ‹*n*>*x* ⟹ *x*>*nth-bj i* ⟹ *?g* (*Suc x*) = *?g x*› **for** *x*
using *g*[*of x*] *n-ge-m*
by *auto*
have *eq1*: ‹*nth-bj i* < *Suc x* ⟹ ¬ *nth-bj i* < *x* ⟹ *x* = *nth-bj i*› **and**
eq2: ‹*nth-bj i* < *x* ⟹ ¬ *nth-bj i* < *x* − *Suc 0* ⟹ *nth-bj i* = *x* − *Suc 0*›

for *x*
by *simp-all*
have *ex-larger*: ‹*n*>*x* ⟹ *x*>*nth-bj i* ⟹ *?g* (*Suc x*) = *?g* (*Suc* (*nth-bj i*))› **for** *x*
apply (*induction x*)
subgoal by *auto*
subgoal for *x*
by (*cases* ‹*nth-bj i* < *x*›) (*auto dest*: *eq1*)
done
from *this*[*of* ‹*n*−*1*›] **have** *g-n-nth-bj*: ‹*?g n* = *?g* (*Suc* (*nth-bj i*))›
using *n-ge-m i-le-m le-n*
by (*cases* ‹*nth-bj i* < *n* − *Suc 0*›)
(*auto dest*: *eq2*)
then have ‹*size* (*learned-clss* (*f* (*Suc* (*nth-bj i*)))) < *size* (*learned-clss T*)›
using *g-n i-le-m n-ge-m g-le*[*of* ‹*Suc* (*nth-bj i*)›] *le-n ge*
‹*?g* (*nth-bj i*) = *size* (*learned-clss S*) + (*i* − *1*)›
using *Suc.IH* **by** *auto*
then show *False*
using *g-n i-le-m n-ge-m g-le*[*of* ‹*Suc* (*nth-bj i*)›] *g-n-nth-bj* **by** *auto*
qed

from *LeastI-ex*[*OF ex-larger*]
have *bt*: ‹*backtrack* (*f* (*nth-bj* (*Suc i*))) (*f* (*Suc* (*nth-bj* (*Suc i*))))› **and**
le: ‹*nth-bj* (*Suc i*) < *n*› **and**
nth-mono: ‹*nth-bj i* < *nth-bj* (*Suc i*)›
unfolding *nth-bj-Suc*[*symmetric*]
by *auto*

have *g-nth-Suc-g-Suc-nth*: ‹*?g* (*nth-bj* (*Suc i*)) = *?g* (*Suc* (*nth-bj i*))›
using *g-nth-bj-eq2*[*of* ‹*nth-bj* (*Suc i*) − *1*› *i*] *le nth-mono*
apply *auto*
by (*metis Suc-pred gr0I less-Suc0 less-Suc-eq less-imp-diff-less*)
have *H1*: ‹*size* (*learned-clss* (*f* (*Suc* (*nth-bj* (*Suc i*))))) =
1 + *size* (*learned-clss* (*f* (*nth-bj* (*Suc i*))))› **if** ‹*i* = *0*›
using *bt* **unfolding** *that*
by (*auto simp*: *that elim*: *backtrackE*)
have *?case* **if** ‹*i* > *0*›
using *IH that nth-mono le bt gei*
by (*auto elim*: *backtrackE simp*: *g-nth-Suc-g-Suc-nth*)
moreover have *?case* **if** ‹*i* = *0*›
using *le bt gei nth-mono IH g-nth-bj-eq2*[*of* ‹*nth-bj* (*Suc i*) − *1*› *i*]
g-nth-Suc-g-Suc-nth
apply (*intro conjI*)
subgoal by (*simp add*: *that*)
subgoal by (*auto simp*: *that elim*: *backtrackE*)
subgoal by (*auto simp*: *that elim*: *backtrackE*)
subgoal *Hk* **by** (*auto simp*: *that elim*: *backtrackE*)
subgoal using *H1* **by** (*auto simp*: *that elim*: *backtrackE*)
subgoal using *nth-mono* **by** *auto*
done
ultimately show *?case* **by** *blast*
qed
then have
‹(*?g* (*nth-bj i*) = *size* (*learned-clss S*) + (*i* − *1*))› **and**
nth-bj-le: ‹*nth-bj i* < *n*› **and**
nth-bj-ge: ‹*nth-bj i* ≥ *i*› **and**
bt-nth-bj: ‹*i* > *0* ⟹ *backtrack* (*f* (*nth-bj i*)) (*f* (*Suc* (*nth-bj i*)))› **and**

⟨*i* > *0* ⟹ *?g* (*Suc* (*nth-bj i*)) = *size* (*learned-clss S*) + *i*⟩ **and**
nth-bj-mono: ⟨*i* > *0* ⟹ *nth-bj* (*i* − *1*) < *nth-bj i*⟩
if ⟨*i* ≤ *?b+1*⟩
for *i*
using *that* **by** *blast+*
have
confl-None: ⟨*conflicting* (*f* (*Suc* (*nth-bj i*))) = *None*⟩ **and**
confl-nth-bj: ⟨*conflicting* (*f* (*nth-bj i*)) ≠ *None*⟩
if ⟨*i* ≤ *?b+1*⟩ ⟨*i* > *0*⟩
for *i*
using *bt-nth-bj*[*OF that*] **by** (*auto simp*: *backtrack.simps*)

have *conflicting-still-conflicting*:
⟨*conflicting* (*f k*) ≠ *None* ⟶ *conflicting* (*f* (*Suc k*)) ≠ *None*⟩
if ⟨*k* < *n*⟩ ⟨*k* > *nth-bj i*⟩ ⟨*k* < *nth-bj* (*Suc i*)⟩ **for** *k i*
using *between-nth-bj-not-bt*[*OF that*] *f*[*OF that*(*1*)]
by (*auto elim*: *propagateE conflictE decideE backtrackE skipE resolveE*
simp: $cdcl_W$*-o.simps* $cdcl_W$*-bj.simps* $cdcl_W$*.simps*)

define *nth-confl* **where**
⟨*nth-confl n* ≡ *LEAST i*. *i* > *nth-bj n* ∧ *i* < *nth-bj* (*Suc n*) ∧ *conflict* (*f i*) (*f* (*Suc i*))⟩ **for** *n*
have ⟨∃ *i*>*nth-bj a*. *i* < *nth-bj* (*Suc a*) ∧ *conflict* (*f i*) (*f* (*Suc i*))⟩
if *a-n*: ⟨*a* ≤ *?b*⟩ ⟨*a* > *0*⟩
for *a*
proof (*rule ccontr*)
assume *H*: ⟨¬ *?thesis*⟩
have ⟨*conflicting* (*f* (*nth-bj a* + *Suc i*)) = *None*⟩
if ⟨*nth-bj a* + *Suc i* ≤ *nth-bj* (*Suc a*)⟩ **for** *i* :: *nat*
using *that*
apply (*induction i*)
subgoal
using *confl-None*[*of a*] *a-n n-ge-m* **by** *auto*
subgoal for *i*
apply (*cases* ⟨*Suc* (*nth-bj a* + *i*) < *n*⟩)
using *f*[*of* ⟨*nth-bj a* + *Suc i*⟩] *H*
apply (*auto elim*: *propagateE conflictE decideE backtrackE skipE resolveE*
simp: $cdcl_W$*-o.simps* $cdcl_W$*-bj.simps* $cdcl_W$*.simps*)[]
using *nth-bj-le*[*of* ⟨*Suc a*⟩] *a-n*(*1*) **by** *auto*
done
from *this*[*of* ⟨*nth-bj* (*Suc a*) − *1* − *nth-bj a*⟩] *a-n*
show *False*
using *nth-bj-mono*[*of* ⟨*Suc a*⟩] *a-n nth-bj-le*[*of* ⟨*Suc a*⟩] *confl-nth-bj*[*of* ⟨*Suc a*⟩]
by *auto*
qed
from *LeastI-ex*[*OF this*] **have** *nth-bj-le-nth-confl*: ⟨*nth-bj a* < *nth-confl a*⟩ **and**
nth-confl: ⟨*conflict* (*f* (*nth-confl a*)) (*f* (*Suc* (*nth-confl a*)))⟩ **and**
nth-confl-le-nth-bj-Suc: ⟨*nth-confl a* < *nth-bj* (*Suc a*)⟩
if *a-n*: ⟨*a* ≤ *?b*⟩ ⟨*a* > *0*⟩
for *a*
using *that* **unfolding** *nth-confl-def*[*symmetric*]
by *blast+*
have *nth-confl-conflicting*: ⟨*conflicting* (*f* (*Suc* (*nth-confl a*))) ≠ *None*⟩
if *a-n*: ⟨*a* ≤ *?b*⟩ ⟨*a* > *0*⟩
for *a*
using *nth-confl*[*OF a-n*]
by (*auto simp*: *conflict.simps*)

have *no-conflict-before-nth-confl*: ‹¬*conflict* (*f k*) (*f* (*Suc k*))›
  if ‹$k >$ *nth-bj a*› **and**
    ‹$k <$ *nth-confl a*› **and**
    a-n: ‹$a \leq$ *?b*› ‹$a > 0$›
  for *k a*
  using *not-less-Least*[*of k* ‹λ*i*. *i* > *nth-bj a* ∧ *i* < *nth-bj* (*Suc a*) ∧ *conflict* (*f i*) (*f* (*Suc i*))›] *that*
  nth-confl-le-nth-bj-Suc[*of a*]
  unfolding *nth-confl-def*[*symmetric*]
  by *auto*
have *conflicting-after-nth-confl*: ‹*conflicting* (*f* (*Suc* (*nth-confl a*) + *k*)) ≠ *None*›
  if *a-n*: ‹$a \leq$ *?b*› ‹$a > 0$› **and**
    k: ‹*Suc* (*nth-confl a*) + *k* < *nth-bj* (*Suc a*)›
  for *a k*
  using *k*
  apply (*induction k*)
  subgoal using *nth-confl-conflicting*[*OF a-n*] **by** *simp*
  subgoal for *k*
    using *conflicting-still-conflicting*[*of* ‹*Suc* (*nth-confl a* + *k*)› *a*] *a-n*
      nth-bj-le[*of a*] *nth-bj-le-nth-confl*[*of a*]
    apply (*cases* ‹*Suc* (*nth-confl a* + *k*) < *n*›)
    apply *auto*
     by (*metis* (*no-types*, *lifting*) *Suc-le-lessD add.commute le-less less-trans-Suc nth-bj-le*
       plus-1-eq-Suc)
  done
have *conflicting-before-nth-confl*: ‹*conflicting* (*f* (*Suc* (*nth-bj a*) + *k*)) = *None*›
  if *a-n*: ‹$a \leq$ *?b*› ‹$a > 0$› **and**
    k: ‹*Suc* (*nth-bj a*) + *k* < *nth-confl a*›
  for *a k*
  using *k*
  apply (*induction k*)
  subgoal using *confl-None*[*of a*] *a-n* **by** *simp*
  subgoal for *k*
    using *f*[*of* ‹*Suc* (*nth-bj a*) + *k*›] *no-conflict-before-nth-confl*[*of a* ‹*Suc* (*nth-bj a*) + *k*›] *a-n*
      nth-confl-le-nth-bj-Suc[*of a*] *nth-bj-le*[*of* ‹*Suc a*›]
    apply (*cases* ‹*Suc* (*nth-bj a* + *k*) < *n*›)
    apply (*auto elim*!: *propagateE conflictE decideE backtrackE skipE resolveE*
        simp: $cdcl_W$*-o.simps* $cdcl_W$*-bj.simps* $cdcl_W$*.simps*)[]
    by *linarith*
  done
have
  ex-trail-decomp: ‹∃*M*. *trail* (*f* (*Suc* (*nth-confl a*))) = *M* @ *trail* (*f* (*Suc* (*nth-confl a* + *k*)))›
  if *a-n*: ‹$a \leq$ *?b*› ‹$a > 0$› **and**
    k: ‹*Suc* (*nth-confl a*) + *k* ≤ *nth-bj* (*Suc a*)›
  for *a k*
  using *k*
proof (*induction k*)
  case *0*
  then show ‹*?case*› **by** *auto*
next
  case (*Suc k*)
  moreover have ‹*nth-confl a* + *k* < *n*›
    proof −
have *nth-bj* (*Suc a*) < *n*
  by (*rule nth-bj-le*) (*use a-n*(*1*) **in** *simp*)
then show *?thesis*
  using *Suc.prems* **by** *linarith*

qed
moreover have ‹∃ *Ma*. *M* @ *trail* (*f* (*Suc* (*nth-confl a* + *k*))) =
Ma @ *tl* (*trail* (*f* (*Suc* (*nth-confl a* + *k*))))› **for** *M*
by (*cases* ‹*trail* (*f* (*Suc* (*nth-confl a* + *k*)))›) *auto*
ultimately show *?case*
using *f*[*of* ‹*Suc* (*nth-confl a* + *k*)›] *conflicting-after-nth-confl*[*of a* ‹*k*›, *OF a-n*] *Suc*
between-nth-bj-not-bt[*of* ‹*Suc* (*nth-confl a* + *k*)› ‹*a*›]
nth-bj-le-nth-confl[*of a*, *OF a-n*]
apply (*cases* ‹*Suc* (*nth-confl a* + *k*) < *n*›)
subgoal
by (*auto elim*!: *propagateE conflictE decideE skipE resolveE*
simp: $cdcl_W$*-o.simps* $cdcl_W$*-bj.simps* $cdcl_W$*.simps*)[]
subgoal
by (*metis* (*no-types*, *lifting*) *Suc-leD Suc-lessI a-n*(*1*) *add.commute add-Suc*
add-mono-thms-linordered-semiring(*1*) *le-numeral-extra*(*4*) *not-le nth-bj-le plus-1-eq-Suc*)
done
qed
have *propa-weight-decreasing-confl*:
‹*propa-weight n* (*trail* (*f* (*Suc* (*nth-bj* (*Suc a*))))) > *propa-weight n* (*trail* (*f* (*nth-confl a*)))›
if *a-n*: ‹*a* ≤ *?b*› ‹*a* > *0*› **and**
n: ‹*n* ≥ *length* (*trail* (*f* (*nth-confl a*)))›
for *a n*
proof −
have *pw0*: ‹*propa-weight n* (*trail* (*f* (*Suc* (*nth-confl a*)))) =
propa-weight n (*trail* (*f* (*nth-confl a*)))› **and**
[*simp*]: ‹*trail* (*f* (*Suc* (*nth-confl a*))) = *trail* (*f* (*nth-confl a*))›
using *nth-confl*[*OF a-n*] **by** (*auto elim*!: *conflictE*)
have *H*: ‹*nth-bj* (*Suc a*) = *Suc* (*nth-confl a*) ∨ *nth-bj* (*Suc a*) ≥ *Suc* (*Suc* (*nth-confl a*))›
using *nth-bj-le-nth-confl*[*of a*, *OF a-n*]
using *a-n*(*1*) *nth-confl-le-nth-bj-Suc that*(*2*) **by** *force*

from *ex-trail-decomp*[*of a* ‹*nth-bj* (*Suc a*) − (*1*+*nth-confl a*)›, *OF a-n*]
obtain *M* **where**
M: ‹*trail* (*f* (*Suc* (*nth-confl a*))) = *M* @ *trail* (*f* (*nth-bj* (*Suc a*)))›
apply −
apply (*rule disjE*[*OF H*])
subgoal
by *auto*
subgoal
using *nth-bj-le-nth-confl*[*of a*, *OF a-n*] *nth-bj-ge*[*of* ‹*Suc a*›] *a-n*
by (*auto simp add*: *numeral-2-eq-2*)
done
obtain *K M1 M2 D D′ L* **where**
decomp: ‹(*Decided K* # *M1*, *M2*)
∈ *set* (*get-all-ann-decomposition* (*trail* (*f* (*nth-bj* (*Suc a*)))))› **and**
‹*get-maximum-level* (*trail* (*f* (*nth-bj* (*Suc a*)))) (*add-mset L D′*) =
backtrack-lvl (*f* (*nth-bj* (*Suc a*)))› **and**
‹*get-level* (*trail* (*f* (*nth-bj* (*Suc a*)))) *L* = *backtrack-lvl* (*f* (*nth-bj* (*Suc a*)))› **and**
‹*get-level* (*trail* (*f* (*nth-bj* (*Suc a*)))) *K* =
Suc (*get-maximum-level* (*trail* (*f* (*nth-bj* (*Suc a*)))) *D′*)› **and**
‹*D′* ⊆# *D*› **and**
‹*clauses* (*f* (*nth-bj* (*Suc a*))) ⊨*pm add-mset L D′*› **and**
st-Suc: ‹*f* (*Suc* (*nth-bj* (*Suc a*))) ∼
cons-trail (*Propagated L* (*add-mset L D′*))
(*reduce-trail-to M1*
(*add-learned-cls* (*add-mset L D′*)

```
        (update-conflicting None (f (nth-bj (Suc a))))))›
    using bt-nth-bj[of ‹Suc a›] a-n
    by (auto elim!: backtrackE)
  obtain M3 where
    tr: ‹trail (f (nth-bj (Suc a))) = M3 @ M2 @ Decided K # M1›
    using decomp by auto
  define M2' where
    ‹M2' = M3 @ M2›
  then have
    tr: ‹trail (f (nth-bj (Suc a))) = M2' @ Decided K # M1›
    using tr by auto
  define M' where
    ‹M' = M @ M2'›
  then have tr2: ‹trail (f (nth-confl a)) = M' @ Decided K # M1›
    using tr M n
    by auto
  have [simp]: ‹max (length M) (n − Suc (length M1 + (length M2')))
    = (n − Suc (length M1 + (length M2')))›
    using tr M st-Suc n by auto
  have [simp]: ‹2 ∗
    (of-list-weight (list-weight-propa-trail M1) ∗
     (2 ^ length M2' ∗
      (2 ^ (n − Suc (length M1 + length M2'))))) =
  of-list-weight (list-weight-propa-trail M1) ∗ 2 ^ (n − length M1)›
 using tr M n by (auto simp: algebra-simps field-simps pow2-times-n
   comm-semiring-1-class.semiring-normalization-rules(26))
  have n-ge: ‹Suc (length M + (length M2' + length M1)) ≤ n›
    using n st-Suc tr M by auto
  have WTF: ‹a < b ⟹ b ≤ c ⟹ a < c› and
     WTF': ‹a ≤ b ⟹ b < c ⟹ a < c› for a b c :: nat
    by auto

  have 3: ‹propa-weight (n − Suc (length M1 + (length M2'))) M
    ≤ 2^(n − Suc (length M1 + length M2')) − 1›
    using of-list-weight-le
    apply auto
     by (metis ‹max (length M) (n − Suc (length M1 + (length M2'))) = n − Suc (length M1 + (length
M2'))›
       length-comp-list-weight-propa-trail)
  have 1: ‹of-list-weight (list-weight-propa-trail M2') ∗
    2 ^ (n − Suc (length M1 + length M2')) < Suc (if M2' = [] then 0
      else 2 ^ (n − Suc (length M1)) − 2 ^ (n − Suc (length M1 + length M2')))›
    apply (cases ‹M2' = []›)
    subgoal by auto
    subgoal
 apply (rule WTF')
   apply (rule Nat.mult-le-mono1[of ‹of-list-weight (list-weight-propa-trail M2')›,
   OF of-list-weight-le[of ‹(list-weight-propa-trail M2')›]])
 using of-list-weight-le[of ‹(list-weight-propa-trail M2')›] n M tr
 by (auto simp add: comm-semiring-1-class.semiring-normalization-rules(26)
   algebra-simps)
    done
  have WTF2:
    ‹a ≤ a' ⟹ b < b' ⟹ a + b < a' + b'› for a b c a' b' c' :: nat
    by auto
```

have ‹*propa-weight* (*n* − *Suc* (*length M1* + *length M2'*)) *M* +
of-list-weight (*list-weight-propa-trail M2'*) ∗
2 ^ (*n* − *Suc* (*length M1* + *length M2'*))
< *2* ^ (*n* − *Suc* (*length M1*))›
apply (*rule WTF*)
apply (*rule WTF2*[*OF 3 1*])
using *n-ge*[*unfolded nat-le-iff-add*] **by** (*auto simp*: *ac-simps algebra-simps*)
then have ‹*propa-weight n* (*trail* (*f* (*nth-confl a*))) < *propa-weight n* (*trail* (*f* (*Suc* (*nth-bj* (*Suc a*)))))›
using *tr2 M st-Suc n tr*
by (*auto simp*: *pow2-times-n algebra-simps*
comm-semiring-1-class.semiring-normalization-rules(*26*))
then show ‹*?thesis*›
using *pw0* **by** *auto*
qed
have *length-trail-le-m*: ‹*length* (*trail* (*f k*)) < *?m* + *1*›
if ‹$k \leq n$›
for *k*
proof −
have ‹$cdcl_W$*-all-struct-inv* (*f k*)›
using *rtranclp-*$cdcl_W$*-*$cdcl_W$*-restart*[*OF cdcl-st-k*[*OF that*]] *inv*
rtranclp-$cdcl_W$*-all-struct-inv-inv* **by** *blast*
then have ‹$cdcl_W$*-M-level-inv* (*f k*)› **and** ‹*no-strange-atm* (*f k*)›
unfolding $cdcl_W$*-all-struct-inv-def* **by** *blast+*
then have ‹*no-dup* (*trail* (*f k*))› **and**
incl: ‹*atm-of* ‘ *lits-of-l* (*trail* (*f k*)) ⊆ *atms-of-mm* (*init-clss* (*f k*))›
unfolding $cdcl_W$*-M-level-inv-def no-strange-atm-def*
by *auto*
have *eq*: ‹(*atms-of-mm* (*init-clss* (*f k*))) = (*atms-of-mm* (*init-clss S*))›
using *rtranclp-*$cdcl_W$*-restart-init-clss*[*OF rtranclp-*$cdcl_W$*-*$cdcl_W$*-restart*[*OF cdcl-st-k*[*OF that*]]]
by *auto*
have ‹*length* (*trail* (*f k*)) = *card* (*atm-of* ‘ *lits-of-l* (*trail* (*f k*)))›
using ‹*no-dup* (*trail* (*f k*))› *no-dup-length-eq-card-atm-of-lits-of-l* **by** *blast*
also have ‹*card* (*atm-of* ‘ *lits-of-l* (*trail* (*f k*))) ≤ *?m*›
using *card-mono*[*OF - incl*] *eq* **by** *auto*
finally show *?thesis*
by *linarith*
qed
have *propa-weight-decreasing-propa*:
‹*propa-weight ?m* (*trail* (*f* (*nth-confl a*))) ≥ *propa-weight ?m* (*trail* (*f* (*Suc* (*nth-bj a*))))›
if *a-n*: ‹*a* ≤ *?b*› ‹*a* > *0*›
for *a*
proof −
have *ppa*: ‹*propa-weight ?m* (*trail* (*f* (*Suc* (*nth-bj a*) + *Suc k*)))
≥ *propa-weight ?m* (*trail* (*f* (*Suc* (*nth-bj a*) + *k*)))›
if ‹*k* < *nth-confl a* − *Suc* (*nth-bj a*)›
for *k*
proof −
have ‹*Suc* (*nth-bj a* + *k*) < *n*› **and** ‹*Suc* (*nth-bj a* + *k*) < *nth-confl a*›
using *that nth-bj-le-nth-confl*[*OF a-n*] *nth-confl-le-nth-bj-Suc*[*OF a-n*]
nth-bj-le[*of* ‹*Suc a*›] *a-n*
by *auto*
then show *?thesis*
using *f*[*of* ‹(*Suc* (*nth-bj a*) + *k*)›] *conflicting-before-nth-confl*[*OF a-n*, *of* ‹*k*›]
no-conflict-before-nth-confl[*OF - - a-n*, *of* ‹*Suc* (*nth-bj a*) + *k*›] *that*
length-trail-le-m[*of* ‹*Suc* (*Suc* (*nth-bj a*) + *k*)›]

```
        by (auto elim!: skipE resolveE backtrackE
            simp: cdcl_W-o.simps cdcl_W-bj.simps cdcl_W.simps
      dest!: propagate-propa-weight[of - - ?m]
        decide-propa-weight[of - - ?m])
    qed
    have WTF3: ‹(Suc (nth-bj a + (nth-confl a − Suc (nth-bj a)))) = nth-confl a›
      using a-n(1) nth-bj-le-nth-confl that(2) by fastforce
    have ‹propa-weight ?m (trail (f (Suc (nth-bj a) + k)))
      ≥ propa-weight ?m (trail (f (Suc (nth-bj a))))›
      if ‹k ≤ nth-confl a − Suc (nth-bj a)›
      for k
      using that
      apply (induction k)
      subgoal by auto
      subgoal for k using ppa[of k]
        apply (cases ‹k < nth-confl a − Suc (nth-bj a)›)
subgoal by auto
subgoal by linarith
      done
      done
    from this[of ‹nth-confl a − (Suc (nth-bj a))›]
    show ?thesis
      by (auto simp: WTF3)
  qed
  have propa-weight-decreasing-confl:
    ‹propa-weight ?m (trail (f (Suc (nth-bj a))))
      < propa-weight ?m (trail (f (Suc (nth-bj (Suc a)))))›
    if a-n: ‹a ≤ ?b› ‹a > 0›
    for a
  proof −
    have WTF: ‹b < c ⟹ a ≤ b ⟹ a < c› for a b c :: nat by linarith
    have ‹nth-confl a < n›
      by (metis Suc-le-mono a-n(1) add.commute add-lessD1 less-imp-le nat-le-iff-add
        nth-bj-le nth-confl-le-nth-bj-Suc plus-1-eq-Suc that(2))

    show ?thesis
      apply (rule WTF)
        apply (rule propa-weight-decreasing-confl[OF a-n, of ?m])
subgoal using length-trail-le-m[of ‹nth-confl a›] ‹nth-confl a < n› by auto
       apply (rule propa-weight-decreasing-propa[OF a-n])
      done
  qed
  have weight1: ‹propa-weight ?m (trail (f (Suc (nth-bj 1)))) ≥ 1›
    using bt-nth-bj[of 1]
    by (auto simp: elim!: backtrackE intro!: trans-le-add1)
  have ‹propa-weight ?m (trail (f (Suc (nth-bj (Suc a))))) ≥
       propa-weight ?m (trail (f (Suc (nth-bj 1)))) + a›
    if a-n: ‹a ≤ ?b›
    for a :: nat
    using that
    apply (induction a)
    subgoal by auto
    subgoal for a
      using propa-weight-decreasing-confl[of ‹Suc a›]
      by auto
    done
```

from *this*[*of* ‹*?b*›] **have** ‹*propa-weight ?m* (*trail* (*f* (*Suc* (*nth-bj* (*Suc* (*?b*)))))) $\geq$ *1* + *?b*›
  using *weight1* **by** *auto*
moreover have
  ‹*max* (*length* (*trail* (*f* (*Suc* (*nth-bj* (*Suc ?b*)))))) *?m* = *?m*›
  using *length-trail-le-m*[*of* ‹(*Suc* (*nth-bj* (*Suc ?b*)))›] *Suc-leI nth-bj-le*
  nth-bj-le[*of* ‹*Suc* (*?b*)›] **by** (*auto simp*: *max-def*)
ultimately show ‹*False*›
  using *of-list-weight-le*[*of* ‹*comp-list-weight-propa-trail ?m* (*trail* (*f* (*Suc* (*nth-bj* (*Suc ?b*)))))›]
  by (*simp del*: *state-eq-init-clss state-eq-trail*)
qed

Application of the previous theorem to an initial state:

corollary *cdcl-pow2-n-learned-clauses2*:
  assumes
    cdcl: ‹$cdcl_W{}^{**}$ (*init-state N*) *T*› **and**
    inv: ‹$cdcl_W$*-all-struct-inv* (*init-state N*)›
  shows ‹*size* (*learned-clss T*) $\leq$ *2* ^ (*card* (*atms-of-mm N*))›
  using *assms cdcl-pow2-n-learned-clauses*[*of* ‹*init-state N*› *T*]
  by *auto*

end

end

1.2 Merging backjump rules

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

Before showing that Weidenbach's CDCL is included in NOT's CDCL, we need to work on a variant of Weidenbach's calculus: NOT's backjump assumes the existence of a clause that is suitable to backjump. This clause is obtained in W's CDCL by applying:

1. *conflict-driven-clause-learning*$_W$*.conflict* to find the conflict
2. the conflict is analysed by repetitive application of *conflict-driven-clause-learning*$_W$*.resolve* and *conflict-driven-clause-learning*$_W$*.skip*,
3. finally *conflict-driven-clause-learning*$_W$*.backtrack* is used to backtrack.

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 states.

1.2.1 Inclusion of the States

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

declare $cdcl_W$*-restart.intros*[*intro*] $cdcl_W$*-bj.intros*[*intro*] $cdcl_W$*-o.intros*[*intro*]
state-prop [*simp del*]

lemma *backtrack-no-*$cdcl_W$*-bj*:

assumes *cdcl*: $cdcl_W$*-bj* T U
shows $\neg$*backtrack* V T
using *cdcl*
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 add*: $cdcl_W$*-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
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*)
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 $\Longrightarrow$ $cdcl_W$*-restart*** S T
using *mono-rtranclp*[*of skip-or-resolve* $cdcl_W$*-restart*]
by (*blast intro*: *skip-or-resolve.cases*)
then have *skip-or-resolve*** S U
using *bj IH backtrack-no*-$cdcl_W$*-bj* **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*-$cdcl_W$*-restart*:
skip-or-resolve** S T $\Longrightarrow$ $cdcl_W$*-restart*** S T
by (*induction rule*: *rtranclp-induct*)
(*auto dest*!: $cdcl_W$*-bj.intros* $cdcl_W$*-restart.intros* $cdcl_W$*-o.intros simp*: *skip-or-resolve.simps*)

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

lemma *wf-skip-or-resolve*:
wf {(T, S). *skip-or-resolve* S T}

proof −
  have *skip-or-resolve x y* $\Longrightarrow$ *length* (*trail y*) < *length* (*trail x*) **for** *x y*
    by (*auto simp*: *skip-or-resolve.simps elim*!: *skipE resolveE*)
  then show *?thesis*
    using *wfP-if-measure*[*of* λ-. *True skip-or-resolve* λ*S*. *length* (*trail S*)]
    by *meson*
qed

definition inv_{NOT} :: *'st* ⇒ *bool* **where**
inv_{NOT} ≡ λ*S*. *no-dup* (*trail S*)

declare inv_{NOT}*-def*[*simp*]
end

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

1.2.2 More lemmas about Conflict, Propagate and Backjumping

Termination

lemma $cdcl_W$*-bj-measure*:
  assumes $cdcl_W$*-bj S T*
  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 elim*!: *backtrackE skipE resolveE*)+

lemma *wf-*$cdcl_W$*-bj*:
  wf {(*b*,*a*). $cdcl_W$*-bj a b*}
  apply (*rule wfP-if-measure*[*of* λ-. *True*
    - λ*T*. *length* (*trail T*) + (*if conflicting T* = *None then 0 else 1*), *simplified*])
  using $cdcl_W$*-bj-measure* **by** *simp*

lemma $cdcl_W$*-bj-exists-normal-form*:
  shows ∃ *T*. *full* $cdcl_W$*-bj S T*
  using *wf-exists-normal-form-full*[*OF wf-*$cdcl_W$*-bj*] **.**

lemma *rtranclp-skip-state-decomp*:
  assumes $skip^{**}$ *S T*
  shows
    ∃ *M*. *trail S* = *M* @ *trail T* ∧ (∀ *m*∈*set M*. ¬*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 elim*!: *skipE*)

Analysing is confluent

lemma *backtrack-reduce-trail-to-state-eq*:
  assumes
    V-T: ‹*V* ∼ *tl-trail T*› **and**
    decomp: ‹(*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail V*))›
  shows ‹*reduce-trail-to M1* (*add-learned-cls E* (*update-conflicting None V*))
    ∼ *reduce-trail-to M1* (*add-learned-cls E* (*update-conflicting None T*))›
proof −

let *?f* = ‹λT. *add-learned-cls E* (*update-conflicting None T*)›
have [*simp*]: ‹*length* (*trail T*) ≠ *length M1*› ‹*trail T* ≠ []›
using *decomp V-T* **by** (*cases* ‹*trail T*›; *auto*)+
have ‹*reduce-trail-to M1* (*?f V*) ∼ *reduce-trail-to M1* (*?f* (*tl-trail T*))›
apply (*rule reduce-trail-to-state-eq*)
using *V-T* **by** (*simp-all add*: *add-learned-cls-state-eq update-conflicting-state-eq*)
moreover {
have ‹*add-learned-cls E* (*update-conflicting None* (*tl-trail T*)) ∼
tl-trail (*add-learned-cls E* (*update-conflicting None T*))›
apply (*rule state-eq-trans*[*OF state-eq-sym*[*THEN iffD1*], *of*
‹*add-learned-cls E* (*tl-trail* (*update-conflicting None T*))›])
apply (*auto simp*: *tl-trail-update-conflicting tl-trail-add-learned-cls-commute*
update-conflicting-state-eq add-learned-cls-state-eq tl-trail-state-eq; *fail*)[]
apply (*rule state-eq-trans*[*OF state-eq-sym*[*THEN iffD1*], *of*
‹*add-learned-cls E* (*tl-trail* (*update-conflicting None T*))›])
apply (*auto simp*: *tl-trail-update-conflicting tl-trail-add-learned-cls-commute*
update-conflicting-state-eq add-learned-cls-state-eq tl-trail-state-eq; *fail*)[]
apply (*rule state-eq-trans*[*OF state-eq-sym*[*THEN iffD1*], *of*
‹*tl-trail* (*add-learned-cls E* (*update-conflicting None T*))›])
apply (*auto simp*: *tl-trail-update-conflicting tl-trail-add-learned-cls-commute*
update-conflicting-state-eq add-learned-cls-state-eq tl-trail-state-eq)
done
note - = *reduce-trail-to-state-eq*[*OF this*, *of M1 M1*]**}**
ultimately show ‹*reduce-trail-to M1* (*?f V*) ∼ *reduce-trail-to M1* (*?f T*)›
by (*subst* (*2*) *reduce-trail-to.simps*)
(*auto simp*: *tl-trail-update-conflicting tl-trail-add-learned-cls-commute intro*: *state-eq-trans*)
qed

lemma *rtranclp-skip-backtrack-reduce-trail-to-state-eq*:
assumes
V-T: ‹$skip^{**}$ *T V*› **and**
decomp: ‹(*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail V*))›
shows ‹*reduce-trail-to M1* (*add-learned-cls E* (*update-conflicting None T*))
∼ *reduce-trail-to M1* (*add-learned-cls E* (*update-conflicting None V*))›
using *V-T decomp*
proof (*induction arbitrary*: *M2 rule*: *rtranclp-induct*)
case *base*
then show *?case* **by** *auto*
next
case (*step U V*) **note** *st* = *this*(*1*) **and** *skip* = *this*(*2*) **and** *IH* = *this*(*3*) **and** *decomp* = *this*(*4*)
obtain *M2'* **where**
decomp': ‹(*Decided K* # *M1*, *M2'*) ∈ *set* (*get-all-ann-decomposition* (*trail U*))›
using *get-all-ann-decomposition-exists-prepend*[*OF decomp*] *skip*
by *atomize* (*auto elim*!: *rulesE simp del*: *get-all-ann-decomposition.simps*
simp: *Decided-cons-in-get-all-ann-decomposition-append-Decided-cons*
append-Cons[*symmetric*] *append-assoc*[*symmetric*]
simp del: *append-Cons append-assoc*)
show *?case*
using *backtrack-reduce-trail-to-state-eq*[*OF - decomp*, *of U E*] *skip IH*[*OF decomp'*]
by (*auto elim*!: *skipE simp del*: *get-all-ann-decomposition.simps intro*: *state-eq-trans'*)
qed

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-restart] 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 D' where
    conf: conflicting V = Some (add-mset L D) and
    decomp: (Decided K # M1, M2) ∈ 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) (add-mset L D') and
    max-D: get-maximum-level (trail V) D' ≡ i and
    lev-k: get-level (trail V) K = Suc i and
    W: W ∼ cons-trail (Propagated L (add-mset L D'))
              (reduce-trail-to M1
                (add-learned-cls (add-mset L D')
                  (update-conflicting None V))) and
    D-D': ‹D' ⊆# D› and
    NU-D': ‹clauses V ⊨pm add-mset L D'›
    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' ∉# E and
    E: E ≠ {#} and
    V: V ∼ tl-trail T
    using skip by (elim skipE) metis
  let ?M = Propagated L' C' # M
  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]: E = add-mset L D
    using V conf raw by auto
  have cdcl_W-restart** S T
    using bj cdcl_W-bj.skip mono-rtranclp[of skip cdcl_W-restart 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 tr-M 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 trail V L*] **by** *auto*
then have *L-L'*: *atm-of L* ≠ *atm-of L'*
using *n-d'* **unfolding** *lits-of-def MV* **by** (*auto simp*: *defined-lit-map*)
have *L'-M*: *undefined-lit M L'*
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'* ∉# *D*
using *L-L' L'-M* **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 defined-lit-map lits-of-def 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*
have *V-T*: ‹*V* ∼ *tl-trail T*›
using *skip* **by** (*auto elim*: *rulesE*)
have
W-S: *W* ∼ *cons-trail* (*Propagated L* (*add-mset L D'*)) (*reduce-trail-to M1* (*add-learned-cls* (*add-mset L D'*) (*update-conflicting None T*)))
apply (*rule state-eq-trans*[*OF W*])
unfolding *DE*
apply (*rule cons-trail-state-eq*)
apply (*rule backtrack-reduce-trail-to-state-eq*)
using *V decomp* **by** *auto*
have *atm-of-L'-D'*: *atm-of L'* ∉ *atms-of D'*
by (*metis DE LE* ‹*D'* ⊆# *D*› ‹*L'* ∉# *D*› *atm-of-in-atm-of-set-in-uminus atms-of-def insert-iff mset-subset-eqD set-mset-add-mset-insert*)

obtain *M2'* **where**
decomp': (*Decided K* # *M1*, *M2'*) ∈ *set* (*get-all-ann-decomposition* (*trail T*))
using *decomp V* **unfolding** *tr-M MV* **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 tr-M* **by** (*auto split*: *if-split-asm*)
moreover have *get-level ?M L* = *get-maximum-level ?M* (*add-mset L D'*)
using *count-decided-ge-get-maximum-level*[*of* ‹*trail V*› *D'*] *calculation*(*2*) *lev-L max MV atm-of-L'-D'*
unfolding *get-maximum-level-add-mset*
by *auto*
moreover have *i* = *get-maximum-level ?M D'*
using *max-D MV atm-of-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 simp*: *defined-lit-map*)
ultimately have *backtrack T W*
apply −
apply (*rule backtrack-rule*[*of T L D K M1 M2' D' i*])
unfolding *tr-M*[*symmetric*]
subgoal using *raw* **by** (*simp*; *fail*)
subgoal by (*simp*; *fail*)

subgoal by (*simp*; *fail*)
subgoal by (*simp*; *fail*)
subgoal by (*simp*; *fail*)
subgoal using *lev-k tr* **unfolding** *MV*[*symmetric*] **by** (*auto*; *fail*)[]
subgoal using *D-D′* **by** (*simp*; *fail*)
subgoal using *NU-D′ V-T* **by** (*simp*; *fail*)
subgoal using *W-S lev-k* **by** (*auto*; *fail*)[]
done
then show *?thesis* **using** *IH inv* **by** *blast*
qed

See also theorem *rtranclp-skip-backtrack-backtrack*

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 D′* **where**
S: *conflicting S* = *Some* (*add-mset 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*) (*add-mset L D′*) **and**
i: *get-maximum-level* (*trail S*) *D′* ≡ *i* **and**
lev-K: *get-level* (*trail S*) *K* = *Suc i* **and**
W: *W* ∼ *cons-trail* (*Propagated L* (*add-mset L D′*))
(*reduce-trail-to M1*
(*add-learned-cls* (*add-mset L D′*)
(*update-conflicting None S*))) **and**
D-D′: ‹*D′* ⊆# *D*› **and**
NU-D′: ‹*clauses S* ⊨*pm add-mset L D′*›
using *bt* **by** (*elim backtrackE*) *metis*
let *?D* = *add-mset L D*
let *?D′* = *add-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* $cdcl_W$*-restart*] **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* **by** *auto*
have *T*: *state-butlast T* = (M_T, *init-clss S*, *learned-clss S*, *Some* (*add-mset L D*)) **and**
bt-S-T: *backtrack-lvl S* = *backtrack-lvl T* **and**
clss-S-T: ‹*clauses S* = *clauses T*›
using M_T *rtranclp-skip-state-decomp*[*of S T*] *skip S* **by** (*auto simp*: *clauses-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*$_W$*-restart*] **by** *blast*
then have $M_T \models as$ *CNot ?D*
unfolding *cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-conflicting-def* **using** *T* **by** *auto*
then have $\forall L' \in\# ?D.$ *defined-lit* M_T L'
using *Decided-Propagated-in-iff-in-lits-of-l*
by (*auto dest*: *true-annots-CNot-definedD*)
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 *undef-D*: $\forall L' \in\# ?D.$ *undefined-lit MS* L'
unfolding *M* **by** (*auto dest*: *defined-lit-no-dupD*)
then have *H*: $\bigwedge L'.\ L' \in\# D \Longrightarrow$ *get-level* (*trail S*) L' = *get-level* M_T L'
get-level (*trail S*) L = *get-level* M_T L
unfolding *M* **by** (*auto simp*: *lits-of-def*)
have [*simp*]: *get-maximum-level* (*trail S*) D = *get-maximum-level* M_T D
using ‹$M_T \models as$ *CNot* (*add-mset L D*)› *M nm undef-D* **by** (*auto simp*: *get-maximum-level-skip-beginning*)

have *lev-l'*: *get-level* M_T L = *backtrack-lvl S*
using *lev-l nm* **by** (*auto simp*: *H*)
have [*simp*]: *trail* (*reduce-trail-to M1 T*) = *M1*
by (*metis* (*no-types*) *M* M_T *append-assoc get-all-ann-decomposition-exists-prepend*[*OF decomp*] *nm reduce-trail-to-trail-tl-trail-decomp beginning-not-decided-invert*)
obtain *c* **where** *c*: ‹M_T = *c* @ *Decided K* # *M1*›
using *nm decomp* **by** (*auto dest*!: *get-all-ann-decomposition-exists-prepend*
simp: M_T[*symmetric*] *M append-assoc*[*symmetric*]
simp del: *append-assoc*
dest!: *beginning-not-decided-invert*)
obtain c'' **where**
c'': ‹(*Decided K* # *M1*, c'') ∈ *set* (*get-all-ann-decomposition* (*c* @ *Decided K* # *M1*))›
using *Decided-cons-in-get-all-ann-decomposition-append-Decided-cons*[*of K M1*] **by** *blast*
have *W*: *W* ∼ *cons-trail* (*Propagated L* (*add-mset L D'*)) (*reduce-trail-to M1*
(*add-learned-cls* (*add-mset L D'*) (*update-conflicting None T*)))
apply (*rule state-eq-trans*[*OF W*])
apply (*rule cons-trail-state-eq*)
apply (*rule rtranclp-skip-backtrack-reduce-trail-to-state-eq*[*of - - K M1*])
using *skip* **apply** (*simp*; *fail*)
using c'' **by** (*auto simp*: M_T[*symmetric*] *M c*)
have *max-trail-S-MT-L-D'*: ‹*get-maximum-level* (*trail S*) *?D'* = *get-maximum-level* M_T *?D'*›
by (*rule get-maximum-level-cong*) (*use H D-D'* **in** *auto*)
then have *lev-l-D'*: *get-level* M_T L = *get-maximum-level* M_T *?D'*
using *lev-l-D H* **by** *auto*
have *i'*: *i* = *get-maximum-level* M_T *D'*
unfolding *i*[*symmetric*]
by (*rule get-maximum-level-cong*) (*use H D-D'* **in** *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 *undefined-lit MS K*
using ‹*no-dup* (*trail S*)› *decomp'* **unfolding** *M* M_T
by (*auto simp*: *lits-of-def defined-lit-map no-dup-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 L D K M1 M2′ D′ i*])
  subgoal using *T* **by** *auto*
  subgoal using *T* **by** *auto*
  subgoal using *T lev-l′ lev-l-D′ bt-S-T* **by** *auto*
  subgoal using *T lev-l-D′ bt-S-T* **by** *auto*
  subgoal using *i′ W lev-K* **unfolding** M_T[*symmetric*] *clss-S-T* **by** *auto*
  subgoal using *lev-K* **unfolding** M_T[*symmetric*] *clss-S-T* **by** *auto*
  subgoal using *D-D′* **.**
  subgoal using *NU-D′* **unfolding** *clss-S-T* **.**
  subgoal using *W* **unfolding** *i′*[*symmetric*] **by** *auto*
  done
qed

lemma $cdcl_W$*-bj-decomp-resolve-skip-and-bj*:
  assumes $cdcl_W$*-bj*** *S T*
  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*
      then have $cdcl_W$*-restart*** *S V*
        using *rtranclp-skip-or-resolve-rtranclp-*$cdcl_W$*-restart* **by** *blast*
      with *bj bt* **have** *False* **using** *backtrack-no-*$cdcl_W$*-bj* **by** *simp*
    }
    then show *?thesis* **using** *IH* **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

1.2.3 CDCL with Merging

inductive $cdcl_W$*-merge-restart* :: *′st* ⇒ *′st* ⇒ *bool* **where**
fw-r-propagate: *propagate S S′* ⟹ $cdcl_W$*-merge-restart S S′* |
fw-r-conflict: *conflict S T* ⟹ *full* $cdcl_W$*-bj T U* ⟹ $cdcl_W$*-merge-restart S U* |
fw-r-decide: *decide S S′* ⟹ $cdcl_W$*-merge-restart S S′*|
fw-r-rf: $cdcl_W$*-rf S S′* ⟹ $cdcl_W$*-merge-restart S S′*

lemma *rtranclp-*$cdcl_W$*-bj-rtranclp-*$cdcl_W$*-restart*:
  $cdcl_W$*-bj*** *S T* ⟹ $cdcl_W$*-restart*** *S T*

using *mono-rtranclp*[*of* $cdcl_W$*-bj* $cdcl_W$*-restart*] **by** *blast*

lemma $cdcl_W$*-merge-restart-*$cdcl_W$*-restart*:
assumes $cdcl_W$*-merge-restart S T*
shows $cdcl_W$*-restart*** *S T*
using *assms*
proof *induction*
case (*fw-r-conflict S T U*) **note** *confl* = *this*(*1*) **and** *bj* = *this*(*2*)
have $cdcl_W$*-restart S T* **using** *confl* **by** (*simp add*: $cdcl_W$*-restart.intros r-into-rtranclp*)
moreover
have $cdcl_W$*-bj*** *T U* **using** *bj* **unfolding** *full-def* **by** *auto*
then have $cdcl_W$*-restart*** *T U* **using** *rtranclp-*$cdcl_W$*-bj-rtranclp-*$cdcl_W$*-restart* **by** *blast*
ultimately show *?case* **by** *auto*
qed (*simp-all add*: $cdcl_W$*-o.intros* $cdcl_W$*-restart.intros r-into-rtranclp*)

lemma $cdcl_W$*-merge-restart-conflicting-true-or-no-step*:
assumes $cdcl_W$*-merge-restart S T*
shows *conflicting T* = *None* ∨ *no-step* $cdcl_W$*-restart 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 $cdcl_W$*-restart U V* **and** *conflicting U* = *Some D*
then have *False*
using *n-s* **unfolding** *full-def*
by (*induction rule*: $cdcl_W$*-restart-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* ⇒ *'st* ⇒ *bool* **where**
fw-propagate: *propagate S S'* ⟹ $cdcl_W$*-merge S S'* |
fw-conflict: *conflict S T* ⟹ *full* $cdcl_W$*-bj T U* ⟹ $cdcl_W$*-merge S U* |
fw-decide: *decide S S'* ⟹ $cdcl_W$*-merge S S'*|
fw-forget: *forget S S'* ⟹ $cdcl_W$*-merge S S'*

lemma $cdcl_W$*-merge-*$cdcl_W$*-merge-restart*:
$cdcl_W$*-merge S T* ⟹ $cdcl_W$*-merge-restart S T*
by (*meson* $cdcl_W$*-merge.cases* $cdcl_W$*-merge-restart.simps forget*)

lemma *rtranclp-*$cdcl_W$*-merge-tranclp-*$cdcl_W$*-merge-restart*:
$cdcl_W$*-merge*** *S T* ⟹ $cdcl_W$*-merge-restart*** *S T*
using *rtranclp-mono*[*of* $cdcl_W$*-merge* $cdcl_W$*-merge-restart*] $cdcl_W$*-merge-*$cdcl_W$*-merge-restart* **by** *blast*

lemma $cdcl_W$*-merge-rtranclp-*$cdcl_W$*-restart*:
$cdcl_W$*-merge S T* ⟹ $cdcl_W$*-restart*** *S T*
using $cdcl_W$*-merge-*$cdcl_W$*-merge-restart* $cdcl_W$*-merge-restart-*$cdcl_W$*-restart* **by** *blast*

lemma *rtranclp-*$cdcl_W$*-merge-rtranclp-*$cdcl_W$*-restart*:
$cdcl_W$*-merge*** *S T* ⟹ $cdcl_W$*-restart*** *S T*
using *rtranclp-mono*[*of* $cdcl_W$*-merge* $cdcl_W$*-restart***] $cdcl_W$*-merge-rtranclp-*$cdcl_W$*-restart* **by** *auto*

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 $(\lambda S\ T.$ $cdcl_W$*-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-*$cdcl_W$*-restart assms*(*1*)
rtranclp-$cdcl_W$*-all-struct-inv-inv rtranclp-mono*[*of* $cdcl_W$*-merge* $cdcl_W$*-restart***] **by** *fastforce*
then have $(\lambda S\ T.$ $cdcl_W$*-all-struct-inv S* $\wedge$ $cdcl_W$*-merge S T*$)^{++}$ *c d*
using *fw* **by** *auto*
then show *?case* **using** *IH* **by** *auto*
qed

lemma *backtrack-is-full1-*$cdcl_W$*-bj*:
assumes *bt*: *backtrack S T*
shows *full1* $cdcl_W$*-bj S T*
using *bt backtrack-no-*$cdcl_W$*-bj* **unfolding** *full1-def* **by** *blast*

lemma *rtrancl-*$cdcl_W$*-conflicting-true-*$cdcl_W$*-merge-restart*:
assumes $cdcl_W$*-restart*** *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** $cdcl_W$*-restart* = *this*(*2*) **and** *IH* = *this*(*3*)[*OF this*(*4*−)] **and**
confl[*simp*] = *this*(*5*) **and** *inv* = *this*(*4*)
from $cdcl_W$*-restart*
show *?case*
proof *cases*
case *propagate*
moreover have *conflicting U* = *None* **and** *conflicting V* = *None*
using *propagate propagate* **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 have *conflicting U* = *None* **and** *conflicting V* $\neq$ *None*
using *conflict* **by** (*auto elim*!: *conflictE*)
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*
then consider
(*s-or-r*) *skip-or-resolve U V* |
(*bt*) *backtrack U V*
by (*auto simp*: $cdcl_W$*-bj.simps*)

then show *?thesis*
proof *cases*
case *s-or-r*
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 elim*!: *skipE resolveE*)
then show *?thesis*
by (*metis* (*full-types*) *IH f1 rtranclp-trans tranclp-into-rtranclp*)
next
case *bt*
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-*$cdcl_W$*-restart-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 show *?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*
qed
qed
next
case *rf*
moreover have *conflicting U* = *None* **and** *conflicting V* = *None*
using *rf* **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-*$cdcl_W$*-restart-no-step-*$cdcl_W$*-merge-restart*:
no-step $cdcl_W$*-restart S* ⟹ *no-step* $cdcl_W$*-merge-restart S*
by (*auto simp*: $cdcl_W$*-restart.simps* $cdcl_W$*-merge-restart.simps* $cdcl_W$*-o.simps* $cdcl_W$*-bj.simps*)

lemma *no-step-*$cdcl_W$*-merge-restart-no-step-*$cdcl_W$*-restart*:
assumes
conflicting S = *None* **and**
$cdcl_W$*-M-level-inv S* **and**
no-step $cdcl_W$*-merge-restart S*
shows *no-step* $cdcl_W$*-restart S*

proof −
  { **fix** S'
    assume *conflict S S'*
    then have $cdcl_W$*-restart S S'* **using** $cdcl_W$*-restart.conflict* **by** *auto*
    then have $cdcl_W$*-M-level-inv S'*
      using *assms(2)* $cdcl_W$*-restart-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** $cdcl_W$*-restart.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-*$cdcl_W$*-restart-iff-full-*$cdcl_W$*-merge*:
  assumes *confl*: *conflicting S* = *None* **and** *lev*: $cdcl_W$*-M-level-inv S*
  shows *full* $cdcl_W$*-restart S V* $\longleftrightarrow$ *full* $cdcl_W$*-merge-restart S V*
proof
  assume *full*: *full* $cdcl_W$*-merge-restart S V*
  then have *st*: $cdcl_W$*-restart*** *S V*
    using *rtranclp-mono[of* $cdcl_W$*-merge-restart* $cdcl_W$*-restart****]* $cdcl_W$*-merge-restart-*$cdcl_W$*-restart*
    unfolding *full-def* **by** *auto*

  have *n-s*: *no-step* $cdcl_W$*-merge-restart V*
    using *full* **unfolding** *full-def* **by** *auto*
  have *n-s-bj*: *no-step* $cdcl_W$*-bj V*
    using *rtranclp-*$cdcl_W$*-merge-restart-no-step-*$cdcl_W$*-bj confl full* **unfolding** *full-def* **by** *auto*
  have $\bigwedge S'$. *conflict V S'* $\Longrightarrow$ $cdcl_W$*-M-level-inv S'*
    using $cdcl_W$*-restart.conflict* $cdcl_W$*-restart-consistent-inv lev rtranclp-*$cdcl_W$*-restart-consistent-inv st*
by *blast*
  then have $\bigwedge S'$. *conflict V S'* $\Longrightarrow$ *False*

using *n-s n-s-bj* $cdcl_W$*-bj-exists-normal-form* $cdcl_W$*-merge-restart.simps* **by** *meson*
then have *n-s-*$cdcl_W$*-restart*: *no-step* $cdcl_W$*-restart V*
using *n-s n-s-bj* **by** (*auto simp*: $cdcl_W$*-restart.simps* $cdcl_W$*-o.simps* $cdcl_W$*-merge-restart.simps*)
then show *full* $cdcl_W$*-restart S V* **using** *st* **unfolding** *full-def* **by** *auto*
next
assume *full*: *full* $cdcl_W$*-restart S V*
have *no-step* $cdcl_W$*-merge-restart V*
using *full no-step-*$cdcl_W$*-restart-no-step-*$cdcl_W$*-merge-restart* **unfolding** *full-def* **by** *blast*
moreover {
consider
(*fw*) $cdcl_W$*-merge-restart*** *S V* **and** *conflicting V = None* |
(*bj*) *T U* **where**
$cdcl_W$*-merge-restart*** *S T* **and**
conflicting V $\neq$ *None* **and**
conflict T U **and**
$cdcl_W$*-bj*** *U V*
using *full rtrancl-*$cdcl_W$*-conflicting-true-*$cdcl_W$*-merge-restart confl lev* **unfolding** *full-def*
by *meson*
then have $cdcl_W$*-merge-restart*** *S V*
proof *cases*
case *fw*
then show *?thesis* **by** *fast*
next
case (*bj T U*)
have *no-step* $cdcl_W$*-bj V*
using *full* **unfolding** *full-def* **by** (*meson* $cdcl_W$*-o.bj other*)
then have *full* $cdcl_W$*-bj U V*
using ‹ $cdcl_W$*-bj*** *U V*› **unfolding** *full-def* **by** *auto*
then have $cdcl_W$*-merge-restart T V*
using ‹*conflict T U*› $cdcl_W$*-merge-restart.fw-r-conflict* **by** *blast*
then show *?thesis* **using** ‹$cdcl_W$*-merge-restart*** *S T*› **by** *auto*
qed }
ultimately show *full* $cdcl_W$*-merge-restart S V* **unfolding** *full-def* **by** *fast*
qed

lemma *init-state-true-full-*$cdcl_W$*-restart-iff-full-*$cdcl_W$*-merge*:
shows *full* $cdcl_W$*-restart* (*init-state N*) *V* $\longleftrightarrow$ *full* $cdcl_W$*-merge-restart* (*init-state N*) *V*
by (*rule conflicting-true-full-*$cdcl_W$*-restart-iff-full-*$cdcl_W$*-merge*) *auto*

1.2.4 CDCL with Merge and Strategy

The intermediate step

inductive $cdcl_W$*-s'* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **for** *S* :: *'st* **where**
conflict': *conflict S S'* $\Longrightarrow$ $cdcl_W$*-s' S S'* |
propagate': *propagate S S'* $\Longrightarrow$ $cdcl_W$*-s' S S'* |
decide': *no-step conflict S* $\Longrightarrow$ *no-step propagate S* $\Longrightarrow$ *decide S S'* $\Longrightarrow$ $cdcl_W$*-s' S S'* |
bj': *full1* $cdcl_W$*-bj S S'* $\Longrightarrow$ $cdcl_W$*-s' S S'*

inductive-cases $cdcl_W$*-s'E*: $cdcl_W$*-s' S T*

lemma *rtranclp-*$cdcl_W$*-bj-full1-cdclp-*$cdcl_W$*-stgy*:
$cdcl_W$*-bj*** *S S'* $\Longrightarrow$ $cdcl_W$*-stgy*** *S S'*
proof (*induction rule*: *converse-rtranclp-induct*)
case *base*
then show *?case* **by** *simp*

next
  case $(step\ T\ U)$ **note** $st = this(2)$ **and** $bj = this(1)$ **and** $IH = this(3)$
  have *n-s*: *no-step conflict T no-step propagate T*
    using *bj* **by** (*auto simp add*: $cdcl_W$*-bj.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 *n-s* $cdcl_W$*-o.bj local.bj other*$'$ **by** (*meson r-into-rtranclp*)
  next
    case U' **note** $U' = this(1)$
    have *no-step conflict U no-step propagate U*
      using U' **by** (*fastforce simp*: $cdcl_W$*-bj.simps elim*!: *rulesE*)+
    then have $cdcl_W$*-stgy T U*
      using *n-s* $cdcl_W$*-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*$'$ $S\ T \Longrightarrow$ $cdcl_W$*-stgy*** $S\ T$
  by (*induction rule*: $cdcl_W$*-s*$'$*.induct*)
    (*auto simp*: *full1-def*
     dest: *tranclp-into-rtranclp rtranclp-*$cdcl_W$*-bj-full1-cdclp-*$cdcl_W$*-stgy* $cdcl_W$*-stgy.intros*)

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 *assms* **apply** (*cases rule*: $cdcl_W$*-stgy.cases*)
  using *bj*$'$[*of S U*] **by** (*auto intro*: $cdcl_W$*-s*$'$*.intros simp*: $cdcl_W$*-o.simps full1-def*)

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*$'^{**}$ $S\ U \lor (\exists\, T.$ $cdcl_W$*-s*$'^{**}$ $S\ T \land$ $cdcl_W$*-bj*$^{++}$ $T\ U \land$ *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 $cdcl_W$*-s*$'^{**}$ $S\ T$
      using *IH* **by** (*auto elim*: *conflictE*)
    moreover have *f2*: $cdcl_W$*-s*$'^{**}$ $T\ V$
      using $cdcl_W$*-s*$'$*.conflict*$'$ *conflict*$'$ **by** *blast*
    ultimately show *?thesis* **by** *auto*
  next
    case *propagate*$'$
    then have $cdcl_W$*-s*$'^{**}$ $S\ T$
      using *IH* **by** (*auto elim*: *propagateE*)

moreover have *f2*: $cdcl_W$-s'^{**} T V
  using $cdcl_W$-s'.*propagate'* *propagate'* **by** *blast*
ultimately show *?thesis* **by** *auto*
next
case *other'* **note** $o = this(3)$ **and** $n\text{-}s = this(1,2)$ **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 V
        using *backtrack-is-full1*-$cdcl_W$-*bj backtrack* **by** *blast*
      then have $cdcl_W$-s' T V
        using *full bj' n-s* **by** *blast* **}**
    ultimately show *?thesis* **by** *auto*
  next
    case (*bj* S') **note** $S\text{-}S' = this(1)$ **and** $bj\text{-}T = this(2)$
    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 V
        using *backtrack-is-full1*-$cdcl_W$-*bj backtrack* **by** *blast*
      then have *full1* $cdcl_W$-*bj* S' V
        using *bj-T* **unfolding** *full1-def* **by** *fastforce* **}**
    ultimately have $cdcl_W$-s' S' V **by** (*simp add*: $cdcl_W$-s'.*bj'*)
    then show *?thesis* **using** $S\text{-}S'$ **by** *auto*
  qed
next
  case *skip*
  then have *confl-V*: *conflicting* $V \neq None$
    using *skip* **by** (*auto elim*: *rulesE*)
  have $cdcl_W$-*bj* T V
    using *local.skip* **by** *blast*
  then show *?thesis*
    using *confl-V step.IH* **by** *auto*
next
  case *resolve*
  have *confl-V*: *conflicting* $V \neq None$
    using *resolve* **by** (*auto elim!*: *rulesE*)
  have $cdcl_W$-*bj* T V
    using *local.resolve* **by** *blast*

then show *?thesis*
using *confl-V step.IH* **by** *auto*
qed
qed
qed

lemma *n-step-cdcl$_W$-stgy-iff-no-step-cdcl$_W$-restart-cl-cdcl$_W$-o*:
assumes *inv*: *cdcl$_W$-all-struct-inv S*
shows *no-step cdcl$_W$-s$'$ S* $\longleftrightarrow$ *no-step cdcl$_W$-stgy S* (**is** *?S$'$ S* $\longleftrightarrow$ *?C S*)
proof
assume *?C S*
then show *?S$'$ S*
by (*auto simp*: *cdcl$_W$-s$'$.simps full1-def tranclp-unfold-begin cdcl$_W$-stgy.simps*)
next
assume *n-s*: *?S$'$ S*
then show *?C S*
by (*metis bj$'$ cdcl$_W$-bj-exists-normal-form cdcl$_W$-o.cases cdcl$_W$-s$'$.intros cdcl$_W$-stgy.cases decide$'$ full-unfold*)
qed

lemma *cdcl$_W$-s$'$-tranclp-cdcl$_W$-restart*:
assumes *cdcl$_W$-s$'$ S S$'$* **shows** *cdcl$_W$-restart^{++} S S$'$*
using *assms*
proof (*cases rule*: *cdcl$_W$-s$'$.cases*)
case *conflict$'$*
then show *?thesis* **by** *blast*
next
case *propagate$'$*
then show *?thesis* **by** *blast*
next
case *decide$'$*
then show *?thesis*
using *cdcl$_W$-stgy.simps cdcl$_W$-stgy-tranclp-cdcl$_W$-restart* **by** (*meson cdcl$_W$-o.simps*)
next
case *bj$'$*
then show *?thesis*
by (*metis cdcl$_W$-s$'$.bj$'$ cdcl$_W$-s$'$-is-rtranclp-cdcl$_W$-stgy full1-def rtranclp-cdcl$_W$-stgy-rtranclp-cdcl$_W$-restart rtranclp-unfold tranclp-unfold-begin*)
qed

lemma *tranclp-cdcl$_W$-s$'$-tranclp-cdcl$_W$-restart*:
cdcl$_W$-s$'^{++}$ S S$'$ $\Longrightarrow$ *cdcl$_W$-restart^{++} S S$'$*
apply (*induct rule*: *tranclp.induct*)
using *cdcl$_W$-s$'$-tranclp-cdcl$_W$-restart* **apply** *blast*
by (*meson cdcl$_W$-s$'$-tranclp-cdcl$_W$-restart tranclp-trans*)

lemma *rtranclp-cdcl$_W$-s$'$-rtranclp-cdcl$_W$-restart*:
cdcl$_W$-s$'^{**}$ S S$'$ $\Longrightarrow$ *cdcl$_W$-restart** S S$'$*
using *rtranclp-unfold*[*of cdcl$_W$-s$'$ S S$'$*] *tranclp-cdcl$_W$-s$'$-tranclp-cdcl$_W$-restart*[*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* $\longleftrightarrow$ *full cdcl$_W$-s$'$ S T* (**is** *?S* $\longleftrightarrow$ *?S$'$*)
proof
assume *?S$'$*
then have *cdcl$_W$-restart** S T*

using *rtranclp-cdcl*$_W$*-s′-rtranclp-cdcl*$_W$*-restart*[*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′ n-step-cdcl*$_W$*-stgy-iff-no-step-cdcl*$_W$*-restart-cl-cdcl*$_W$*-o* **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*$_W$*-restart*)
consider
(*s′*) *cdcl*$_W$*-s′*** *S T* |
(*st*) *S′* **where** *cdcl*$_W$*-s′*** *S S′* **and** *cdcl*$_W$*-bj*$^{++}$ *S′ T* **and** *conflicting T* $\neq$ *None*
using *rtranclp-cdcl*$_W$*-stgy-connected-to-rtranclp-cdcl*$_W$*-s′*[*of S T*] *inv* ‹*?S*›
unfolding *full-def cdcl*$_W$*-all-struct-inv-def*
by *blast*
then show *?S′*
proof *cases*
case *s′*
then show *?thesis*
using ‹*full cdcl*$_W$*-stgy S T*› **unfolding** *full-def*
by (*metis inv-T n-step-cdcl*$_W$*-stgy-iff-no-step-cdcl*$_W$*-restart-cl-cdcl*$_W$*-o*)
next
case (*st S′*) **note** *st* = *this*(*1*) **and** *bj* = *this*(*2*) **and** *confl* = *this*(*3*)
have *no-step cdcl*$_W$*-bj T*
using ‹*?S*› *cdcl*$_W$*-stgy.conflict′ cdcl*$_W$*-stgy.intros*(*2*) *other′* **unfolding** *full-def* **by** *blast*
then have *full1 cdcl*$_W$*-bj S′ T*
using *bj* **unfolding** *full1-def* **by** *blast*
then have *cdcl*$_W$*-s′ S′ T*
using *cdcl*$_W$*-s′.bj′*[*of S′ 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*$_W$*-stgy S T*› *n-step-cdcl*$_W$*-stgy-iff-no-step-cdcl*$_W$*-restart-cl-cdcl*$_W$*-o*
unfolding *full-def* **by** *blast*
ultimately show *?thesis*
unfolding *full-def* **by** *blast*
qed
qed

end

end

Chapter 2

NOT's CDCL and DPLL

theory *CDCL-WNOT-Measure*
imports *Weidenbach-Book-Base.WB-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.

2.1 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 :: $nat \Rightarrow nat \Rightarrow nat\ list \Rightarrow nat$ **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*:
$(\sum i{=}1..{<}Suc\ j.\ f\ i) = (\sum i{=}0..{<}j.\ f\ (Suc\ i))$
using *set-sum-atLeastLessThan-add*[*of* - *1 j*] **by** *force*

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

lemma μ_C*-append*:
assumes $s \geq length\ (M@M')$
shows $\mu_C\ s\ b\ (M@M') = \mu_C\ (s - length\ M')\ b\ M + \mu_C\ s\ b\ M'$
proof −
have $\mu_C\ s\ b\ (M@M') = (\sum i{=}0..{<}length\ (M@M').\ (M@M')!i * b\hat{}\ (s + i - length\ (M@M')))$
unfolding μ_C*-def* **by** *blast*
moreover then have $\ldots = (\sum i{=}0..{<}\ length\ M.\ (M@M')!i * b\hat{}\ (s + i - length\ (M@M')))$
$+ (\sum i{=}length\ M..{<}length\ (M@M').\ (M@M')!i * b\hat{}\ (s + i - length\ (M@M')))$
by (*auto intro*!: *sum.atLeastLessThan-concat*[*symmetric*])
moreover
have $\forall i{\in}\{0..{<}\ length\ M\}.\ (M@M')!i * b\hat{}\ (s + i - length\ (M@M')) = M\ !\ i * b\ \hat{}\ (s - length\ M' + i - length\ M)$
using ‹$s \geq length\ (M@M')$› **by** (*auto simp add*: *nth-append ac-simps*)
then have $\mu_C\ (s - length\ M')\ b\ M = (\sum i{=}0..{<}\ length\ M.\ (M@M')!i * b\hat{}\ (s + i - length\ (M@M')))$
unfolding μ_C*-def* **by** *auto*
ultimately have $\mu_C\ s\ b\ (M@M') = \mu_C\ (s - length\ M')\ b\ M$
$+ (\sum i{=}length\ M..{<}length\ (M@M').\ (M@M')!i * b\hat{}\ (s + i - length\ (M@M')))$
by *auto*
moreover {
have $(\sum i{=}length\ M..{<}length\ (M@M').\ (M@M')!i * b\hat{}\ (s + i - length\ (M@M'))) =$
$(\sum i{=}0..{<}length\ M'.\ M'!i * b\hat{}\ (s + i - length\ M'))$
unfolding *length-append set-sum-atLeastLessThan-add* **by** *auto*
then have $(\sum i{=}length\ M..{<}length\ (M@M').\ (M@M')!i * b\hat{}\ (s + i - length\ (M@M'))) = \mu_C\ s\ b\ M'$
unfolding μ_C*-def* **.**
}
ultimately show *?thesis* **by** *presburger*

qed

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

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*: *sum-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*: *sum-distrib-right sum-distrib-left 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 ^ s
      by (metis assms(4) le-add-diff-inverse power-add)
    finally show ?thesis unfolding μC-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 μC-bounded:
  fixes b :: nat
  assumes
    M-le: ∀ i < length M. M!i < b and
    s ≥ length M
    b > 0
  shows μC s b M < b ^ s
proof −
  consider (M0) M = [] | (M) b > 0 and M ≠ []
    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 μC-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 μC-base-0:
  assumes length M ≤ s
  shows μC s 0 M ≤ M!0
proof −
  {
    assume s = length M
    moreover {
      fix n
      have (∑ i=0..<n. M ! i * (0::nat) ^ i) ≤ M ! 0
        apply (induction n rule: nat-induct)
        by simp (rename-tac n, case-tac n, auto)
    }
    ultimately have ?thesis unfolding μC-def by auto
  }
  moreover
  {
    assume length M < s
    then have μC s 0 M = 0 unfolding μC-def by auto}
  ultimately show ?thesis using assms unfolding μC-def by linarith
qed

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

lemma *finite-*νNOT[*simp*]:
  finite (νNOT *s base*)
proof −
  have νNOT *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
  Weidenbach-Book-Base.WB-List-More
  Weidenbach-Book-Base.Wellfounded-More
  Entailment-Definition.Partial-Annotated-Herbrand-Interpretation
  CDCL-WNOT-Measure
begin

2.2 NOT's CDCL

2.2.1 Auxiliary Lemmas and Measure

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

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

lemma *atms-of-ms-single-image-atm-of-lit-of*:
‹*atms-of-ms* (*unmark-s A*) = *atm-of* ‘ (*lit-of* ‘ *A*)›
unfolding *atms-of-ms-def* **by** *auto*

2.2.2 Initial Definitions

The State

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

locale *dpll-state-ops* =
fixes
trail :: ‹*'st* ⇒ (*'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
abbreviation *state*$_{NOT}$ :: ‹*'st* ⇒ (*'v*, *unit*) *ann-lit list* × *'v clauses*› **where**
‹*state*$_{NOT}$ *S* ≡ (*trail S*, *clauses*$_{NOT}$ *S*)›
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-cls*$_{NOT}$ *remove-cls*$_{NOT}$ — related to the state
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*› +
assumes
prepend-trail$_{NOT}$:
‹*state*$_{NOT}$ (*prepend-trail L st*) = (*L* # *trail st*, *clauses*$_{NOT}$ *st*)› **and**
tl-trail$_{NOT}$:
‹*state*$_{NOT}$ (*tl-trail st*) = (*tl* (*trail st*), *clauses*$_{NOT}$ *st*)› **and**
add-cls$_{NOT}$:
‹*state*$_{NOT}$ (*add-cls*$_{NOT}$ *C st*) = (*trail st*, *add-mset C* (*clauses*$_{NOT}$ *st*))› **and**
remove-cls$_{NOT}$:
‹*state*$_{NOT}$ (*remove-cls*$_{NOT}$ *C st*) = (*trail st*, *removeAll-mset C* (*clauses*$_{NOT}$ *st*))›
begin
lemma
trail-prepend-trail[*simp*]:
‹*trail* (*prepend-trail L st*) = *L* # *trail st*›
and
trail-tl-trail$_{NOT}$[*simp*]: ‹*trail* (*tl-trail st*) = *tl* (*trail st*)› **and**
trail-add-cls$_{NOT}$[*simp*]: ‹*trail* (*add-cls*$_{NOT}$ *C st*) = *trail st*› **and**
trail-remove-cls$_{NOT}$[*simp*]: ‹*trail* (*remove-cls*$_{NOT}$ *C st*) = *trail st*› **and**

clauses-prepend-trail[*simp*]:
‹*clauses*$_{NOT}$ (*prepend-trail L st*) = *clauses*$_{NOT}$ *st*›

and
clauses-tl-trail[*simp*]: ‹$clauses_{NOT}$ (*tl-trail st*) = $clauses_{NOT}$ *st*› **and**
clauses-add-cls$_{NOT}$[*simp*]:
‹$clauses_{NOT}$ (*add-cls*$_{NOT}$ *C st*) = *add-mset C* ($clauses_{NOT}$ *st*)› **and**
clauses-remove-cls$_{NOT}$[*simp*]:
‹$clauses_{NOT}$ (*remove-cls*$_{NOT}$ *C st*) = *removeAll-mset C* ($clauses_{NOT}$ *st*)›
using *prepend-trail*$_{NOT}$[*of L st*] *tl-trail*$_{NOT}$[*of st*] *add-cls*$_{NOT}$[*of C st*] *remove-cls*$_{NOT}$[*of C st*]
by (*cases* ‹$state_{NOT}$ *st*›; *auto*)+

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* ⟹ *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 trail-tl-trail*$_{NOT}$ *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*) ⟹
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* ⟹
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'* ⟹ *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* ⇒ *'st* ⇒ *bool*› (**infix** ∼ *50*) **where**
‹*S* ∼ *T* ⟷ *trail S* = *trail T* ∧ *clauses*$_{NOT}$ *S* = *clauses*$_{NOT}$ *T*›

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

lemma *state-eq*$_{NOT}$*-sym*:
‹*S* ∼ *T* ⟷ *T* ∼ *S*›
unfolding *state-eq*$_{NOT}$*-def* **by** *auto*

lemma *state-eq*$_{NOT}$*-trans*:

‹$S \sim T \Longrightarrow T \sim U \Longrightarrow S \sim U$›
unfolding *state-eq$_{NOT}$-def* **by** *auto*

lemma
shows
state-eq$_{NOT}$-trail: ‹$S \sim T \Longrightarrow$ *trail* S = *trail* T› **and**
state-eq$_{NOT}$-clauses: ‹$S \sim T \Longrightarrow$ *clauses$_{NOT}$* 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*: ‹$S \sim T$›
shows ‹*reduce-trail-to$_{NOT}$* F $S \sim$ *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 — End on locale *dpll-state*.

Definition of the Transitions

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-conds :: ‹('*v*, *unit*) *ann-lit* ⇒ '*st* ⇒ '*st* ⇒ *bool*›
begin
inductive *propagate$_{NOT}$* :: ‹'*st* ⇒ '*st* ⇒ *bool*› **where**
propagate$_{NOT}$[*intro*]: ‹*add-mset* L C ∈# *clauses$_{NOT}$* $S \Longrightarrow$ *trail* S ⊨*as CNot* C
⟹ *undefined-lit* (*trail* S) L
⟹ *propagate-conds* (*Propagated* L ()) S T
⟹ $T \sim$ *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*⟩ +
fixes
decide-conds :: ⟨*'st* ⇒ *'st* ⇒ *bool*⟩
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-conds S T ⟹
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-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
backjump-conds :: ⟨*'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool*⟩
begin

inductive *backjump* **where**
⟨*trail S* = *F'* @ *Decided K* # *F*
⟹ *T* ∼ *prepend-trail* (*Propagated L* ()) (*reduce-trail-to$_{NOT}$ F 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 add-mset L C'*
⟹ *F* ⊨*as CNot C'*
⟹ *backjump-conds C C' L S T*
⟹ *backjump S T*⟩
inductive-cases *backjumpE*: ⟨*backjump S T*⟩

The condition *atm-of L* ∈ *atms-of-mm* (*clauses$_{NOT}$ S*) ∪ *atm-of* ‘ *lits-of-l* (*trail S*) is not implied by the the condition *clauses$_{NOT}$ S* ⊨*pm add-mset L C'* (no negation).

end

2.2.3 DPLL with Backjumping

locale *dpll-with-backjumping-ops* =
propagate-ops trail clauses$_{NOT}$ prepend-trail tl-trail add-cls$_{NOT}$ remove-cls$_{NOT}$ propagate-conds +
decide-ops trail clauses$_{NOT}$ prepend-trail tl-trail add-cls$_{NOT}$ remove-cls$_{NOT}$ decide-conds +
backjumping-ops trail clauses$_{NOT}$ prepend-trail tl-trail add-cls$_{NOT}$ remove-cls$_{NOT}$ backjump-conds
for
trail :: ⟨*'st* ⇒ (*'v*, *unit*) *ann-lits*⟩ **and**
clauses$_{NOT}$ :: ⟨*'st* ⇒ *'v clauses*⟩ **and**

$prepend\text{-}trail :: \langle('v,\ unit)\ ann\text{-}lit \Rightarrow 'st \Rightarrow 'st\rangle$ **and**
$tl\text{-}trail :: \langle'st \Rightarrow 'st\rangle$ **and**
$add\text{-}cls_{NOT} :: \langle'v\ clause \Rightarrow 'st \Rightarrow 'st\rangle$ **and**
$remove\text{-}cls_{NOT} :: \langle'v\ clause \Rightarrow 'st \Rightarrow 'st\rangle$ **and**
$inv :: \langle'st \Rightarrow bool\rangle$ **and**
$decide\text{-}conds :: \langle'st \Rightarrow 'st \Rightarrow bool\rangle$ **and**
$backjump\text{-}conds :: \langle'v\ clause \Rightarrow 'v\ clause \Rightarrow 'v\ literal \Rightarrow 'st \Rightarrow 'st \Rightarrow bool\rangle$ **and**
$propagate\text{-}conds :: \langle('v,\ unit)\ ann\text{-}lit \Rightarrow 'st \Rightarrow 'st \Rightarrow bool\rangle$ +
assumes
bj-can-jump:
$\langle\bigwedge S\ C\ F'\ K\ F\ L.$
$inv\ S \Longrightarrow$
$trail\ S = F'\ @\ Decided\ K\ \#\ F \Longrightarrow$
$C \in\#\ clauses_{NOT}\ S \Longrightarrow$
$trail\ S \models as\ CNot\ C \Longrightarrow$
$undefined\text{-}lit\ F\ L \Longrightarrow$
$atm\text{-}of\ L \in atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \cup atm\text{-}of\ `\ (lits\text{-}of\text{-}l\ (F'\ @\ Decided\ K\ \#\ F)) \Longrightarrow$
$clauses_{NOT}\ S \models pm\ add\text{-}mset\ L\ C' \Longrightarrow$
$F \models as\ CNot\ C' \Longrightarrow$
$\neg no\text{-}step\ backjump\ S\rangle$ **and**
can-propagate-or-decide-or-backjump:
$\langle atm\text{-}of\ L \in atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \Longrightarrow$
$undefined\text{-}lit\ (trail\ S)\ L \Longrightarrow$
$satisfiable\ (set\text{-}mset\ (clauses_{NOT}\ S)) \Longrightarrow$
$inv\ S \Longrightarrow$
$no\text{-}dup\ (trail\ S) \Longrightarrow$
$\exists\ T.\ decide_{NOT}\ S\ T \vee propagate_{NOT}\ S\ T \vee backjump\ S\ T\rangle$
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 $dpll\text{-}bj :: \langle'st \Rightarrow 'st \Rightarrow bool\rangle$ **for** $S :: 'st$ **where**
$bj\text{-}decide_{NOT}$: $\langle decide_{NOT}\ S\ S' \Longrightarrow dpll\text{-}bj\ S\ S'\rangle$ |
$bj\text{-}propagate_{NOT}$: $\langle propagate_{NOT}\ S\ S' \Longrightarrow dpll\text{-}bj\ S\ S'\rangle$ |
bj-backjump: $\langle backjump\ S\ S' \Longrightarrow dpll\text{-}bj\ S\ S'\rangle$

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 $S\ T :: \langle'st\rangle$
assumes
$\langle dpll\text{-}bj\ S\ T\rangle$ **and**
$\langle inv\ S\rangle$
$\langle\bigwedge L\ T.\ undefined\text{-}lit\ (trail\ S)\ L \Longrightarrow atm\text{-}of\ L \in atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S)$
$\Longrightarrow T \sim prepend\text{-}trail\ (Decided\ L)\ S$
$\Longrightarrow P\ S\ T\rangle$ **and**
$\langle\bigwedge C\ L\ T.\ add\text{-}mset\ L\ C \in\#\ clauses_{NOT}\ S \Longrightarrow trail\ S \models as\ CNot\ C \Longrightarrow undefined\text{-}lit\ (trail\ S)\ L$
$\Longrightarrow T \sim prepend\text{-}trail\ (Propagated\ L\ ())\ S$

$\Longrightarrow$ *P S T*⟩ **and**
⟨$\bigwedge$*C F' K F L C' T. C* ∈# *clauses*$_{NOT}$ *S* $\Longrightarrow$ *F'* @ *Decided K* # *F* ⊨*as CNot C*
$\Longrightarrow$ *trail S* = *F'* @ *Decided K* # *F*
$\Longrightarrow$ *undefined-lit F L*
$\Longrightarrow$ *atm-of L* ∈ *atms-of-mm* (*clauses*$_{NOT}$ *S*) ∪ *atm-of* ‘ (*lits-of-l* (*F'* @ *Decided K* # *F*))
$\Longrightarrow$ *clauses*$_{NOT}$ *S* ⊨*pm add-mset L C'*
$\Longrightarrow$ *F* ⊨*as CNot C'*
$\Longrightarrow$ *T* ∼ *prepend-trail* (*Propagated L* ()) (*reduce-trail-to*$_{NOT}$ *F S*)
$\Longrightarrow$ *P S T*⟩
shows ⟨*P S T*⟩
apply (*induct T rule*: *dpll-bj-induct*[*OF local.dpll-with-backjumping-ops-axioms*])
apply (*rule assms*(*1*))
using *assms*(*3*) **apply** *blast*
apply (*elim propagate*$_{NOT}$*E*) **using** *assms*(*4*) **apply** *blast*
apply (*elim backjumpE*) **using** *assms*(*5*) ⟨*inv S*⟩ **by** *simp*

Basic properties

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

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

Valuations **lemma** *dpll-bj-sat-iff*:
assumes ⟨*dpll-bj S T*⟩ **and** ⟨*inv S*⟩
shows ⟨*I* ⊨*sm clauses*$_{NOT}$ *S* ⟷ *I* ⊨*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 add-mset L C*⟩
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 add-mset L D*⟩
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*) $\leq$ *1* + *length* *M*›
by (*induction M rule*: *ann-lit-list-induct*) *auto*

lemma *length-in-get-all-ann-decomposition-bounded*:
assumes *i*:‹*i* $\in$ *set* (*trail-weight S*)›
shows ‹*i* $\leq$ *Suc* (*length* (*trail S*))›
proof −
obtain *a b* **where**
‹(*a*, *b*) $\in$ *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 clause set* $\Rightarrow$ *'a list* $\Rightarrow$ *nat*› **where**
‹*unassigned-lit N M* $\equiv$ *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*) $\subseteq$ *atms-of-ms A*› **and**
MA: ‹*atm-of* ‘ *lits-of-l* (*trail S*) $\subseteq$ *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 T*) **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*) $\subseteq$ *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* () # *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-*$decide_{NOT}$ $decide_{NOT}$.$decide_{NOT}$[*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*)) ≤ *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*) ⊆ *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 dest*: *no-dup-appendD*)
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 *fin-atms-A*: ‹*finite* (*atms-of-ms A*)› **using** *finite* **by** *simp*

then have *F-le-A*: ‹*length* (*Propagated L* () # *F*) ≤ *card* (*atms-of-ms A*)›
  using *incl finite* **unfolding** *no-dup-length-eq-card-atm-of-lits-of-l*[*OF no-dup*]
  by (*simp add*: *card-mono*)
have *tr-S-le-A*: ‹*length* (*trail S*) ≤ *card* (*atms-of-ms A*)›
  using *n-d MA* **by** (*metis fin-atms-A card-mono no-dup-length-eq-card-atm-of-lits-of-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) ≤ 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) ≤ 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) ≤ ?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) ⊆ atms-of-ms A ∧ atm-of ‘ lits-of-l (trail S) ⊆ atms-of-ms A
    ∧ no-dup (trail S) ∧ inv S}›
  (is ‹wf ?A›)
proof (rule wf-bounded-measure[of -
        ‹λ-. (2 + card (atms-of-ms A))^(1 + card (atms-of-ms A))›
        ‹λS. 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 = ‹μ_C ?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 (clauses_NOT 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 ‹μ_C *?s ?b* (*trail-weight a*) < μ_C *?s ?b* (*trail-weight b*)› **by** *simp*
moreover from μ_C*-bounded*[*OF bounded-M l-trail-weight-M*]
have ‹μ_C *?s ?b* (*trail-weight b*) ≤ *?b* ^ *?s*› **by** *auto*
ultimately show ‹*?b* ^ *?s* ≤ *?b* ^ *?s* ∧
μ_C *?s ?b* (*trail-weight b*) ≤ *?b* ^ *?s* ∧
μ_C *?s ?b* (*trail-weight a*) < μ_C *?s ?b* (*trail-weight b*)›
by *blast*
qed

Alternative termination proof **abbreviation** *DPLL-mes*$_W$ **where**
‹*DPLL-mes*$_W$ *A M* ≡
map (λ*L*. *if is-decided L then 2*::*nat else 1*) (*rev M*) @ *replicate* (*card A* − *length M*) *3*›

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 no-dup-def*)

lemma *dpll-bj-trail-mes-decreasing-less-than*:
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**
nd: ‹*no-dup* (*trail S*)› **and**
fin-A: ‹*finite A*›
shows ‹(*DPLL-mes*$_W$ (*atms-of-ms A*) (*trail T*), *DPLL-mes*$_W$ (*atms-of-ms A*) (*trail S*)) ∈
lexn less-than (*card* ((*atms-of-ms A*)))›
using *assms*(*1*,*2*)
proof (*induction rule*: *dpll-bj-all-induct*)
case (*decide*$_{NOT}$ *L T*)
define *n* **where**
‹*n* = *card* (*atms-of-ms A*) − *card* (*atm-of* ‘ *lits-of-l* (*trail S*))›

have [*simp*]: ‹*length* (*trail S*) = *card* (*atm-of* ‘ *lits-of-l* (*trail S*))›
using *nd* **by** (*auto simp*: *no-dup-def lits-of-def image-image dest*: *distinct-card*)
have ‹*atm-of L* ∉ *atm-of* ‘ *lits-of-l* (*trail S*)›
by (*metis decide*$_{NOT}$*.hyps*(*1*) *defined-lit-map imageE in-lits-of-l-defined-litD*)

have ‹*card* (*atms-of-ms A*) > *card* (*atm-of* ‘ *lits-of-l* (*trail S*))›
by (*metis N-A* ‹*atm-of L* ∉ *atm-of* ‘ *lits-of-l* (*trail S*)› *atms-of-ms-finite card-seteq decide*$_{NOT}$*.hyps*(*2*) *M-A fin-A not-le subsetCE*)
then have
n-0: ‹*n* > *0*› **and**
n-Suc: ‹*card* (*atms-of-ms A*) − *Suc* (*card* (*atm-of* ‘ *lits-of-l* (*trail S*))) = *n* − *1*›
unfolding *n-def* **by** *auto*

show *?case*
using *fin-A decide*$_{NOT}$ *n-0* **unfolding** *state-eq*$_{NOT}$*-trail*[*OF decide*$_{NOT}$(*3*)]
by (*cases n*) (*auto simp*: *prepend-same-lexn n-def*[*symmetric*] *n-Suc lexn-Suc simp del*: *state-simp*$_{NOT}$ *lexn.simps*)

next
case (*propagate*$_{NOT}$ *C L T*) **note** *C* = *this*(*1*) **and** *undef* = *this*(*3*) **and** *T* = *this*(*3*)
then have ‹*card* (*atms-of-ms A*) > *length* (*trail S*)›
proof −
have *f7*: *atm-of L* ∈ *atms-of-ms A*
using *N-A C in-m-in-literals* **by** *blast*
have *undefined-lit* (*trail S*) (− *L*)
using *undef* **by** *auto*
then show *?thesis*
using *f7 nd fin-A M-A undef* **by** (*metis atm-of-in-atm-of-set-in-uminus atms-of-ms-finite card-seteq in-lits-of-l-defined-litD leI no-dup-length-eq-card-atm-of-lits-of-l*)
qed
then show *?case*
using *fin-A* **unfolding** *state-eq*$_{NOT}$*-trail*[*OF propagate*$_{NOT}$(*4*)]
by (*cases* ‹*card* (*atms-of-ms A*) − *length* (*trail S*)›)
(*auto simp*: *prepend-same-lexn lexn-Suc*
simp del: *state-simp*$_{NOT}$ *lexn.simps*)
next
case (*backjump C F′ K F L C′ T*) **note** *tr-S* = *this*(*3*)
have ‹*trail* (*reduce-trail-to*$_{NOT}$ *F S*) = *F*›
by (*simp add*: *tr-S*)
have ‹*no-dup F*›
using *nd tr-S* **by** (*auto dest*: *no-dup-appendD*)
then have *card-A-F*: ‹*card* (*atms-of-ms A*) > *length F*›
using *distinctcard-atm-of-lit-of-eq-length*[*of* ‹*trail S*›] *card-mono*[*OF - M-A*] *fin-A nd tr-S*
by *auto*
have ‹*no-dup* (*F′* @ *F*)›
using *nd tr-S* **by** (*auto dest*: *no-dup-appendD*)
then have ‹*no-dup F′*›
apply (*subst* (*asm*) *no-dup-rev*[*symmetric*])
using *nd tr-S* **by** (*auto dest*: *no-dup-appendD*)
then have *card-A-F′*: ‹*card* (*atms-of-ms A*) > *length F′* + *length F*›
using *distinctcard-atm-of-lit-of-eq-length*[*of* ‹*trail S*›] *card-mono*[*OF - M-A*] *fin-A nd tr-S*
by *auto*
show *?case*
using *card-A-F card-A-F′*
unfolding *state-eq*$_{NOT}$*-trail*[*OF backjump*(*8*)]
by (*cases* ‹*card* (*atms-of-ms A*) − *length F*›)
(*auto simp*: *tr-S prepend-same-lexn lexn-Suc simp del*: *state-simp*$_{NOT}$ *lexn.simps*)
qed

lemma
assumes *fin*[*simp*]: ‹*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*}›
(**is** ‹*wf ?A*›)
unfolding *conj-commute*[*of* ‹*dpll-bj* - -›]
apply (*rule wf-wf-if-measure′*[*of* - - - ‹λ*S*. *DPLL-mes*$_{W}$ ((*atms-of-ms A*)) (*trail S*)›])
apply (*rule wf-lexn*)
apply (*rule wf-less-than*)
by (*rule dpll-bj-trail-mes-decreasing-less-than*; *use fin* **in** *simp*)

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 $\forall S'$. $\neg$ *dpll-bj* S S'

theorem *dpll-backjump-final-state*:
  fixes A :: ‹'*v clause set*› **and** S T :: ‹'*st*›
  assumes
    ‹*atms-of-mm* (*clauses*$_{NOT}$ S) ⊆ *atms-of-ms* A› **and**
    ‹*atm-of* ‘ *lits-of-l* (*trail* S) ⊆ *atms-of-ms* A› **and**
    ‹*no-dup* (*trail* S)› **and**
    ‹*finite* A› **and**
    inv: ‹*inv* S› **and**
    n-d: ‹*no-dup* (*trail* 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))
    ∨ (*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 atm-I-N* **unfolding** *total-over-m-def total-over-set-def*
by (*fastforce simp*: *image-iff lits-of-def*)
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*))
then show *False*
using *l-N n-s can-propagate-or-decide-or-backjump*[*of* ‹*Pos l*› *S*] *inv n-d sat*
by (*auto dest*: *dpll-bj.intros*)
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*›
by (*metis* (*full-types*) *is-decided-ex-Decided 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-decided L* ∧ *L* ∈ *set ?M*}›
unfolding *M-K* **by** *standard force*+
ultimately have *N-C-M*: ‹*?N* ∪ *?C'* ⊨*ps unmark-l ?M*›
by *auto*
have *N-M-False*: ‹*?N* ∪ (λ*L*. *unmark L*) ‘ (*set ?M*) ⊨*ps* {{#}}›
unfolding *true-clss-clss-def true-annots-def Ball-def true-annot-def*
proof (*intro allI impI*)
fix *LL* :: '*v literal set*
assume
tot: ‹*total-over-m LL* (*set-mset* ($clauses_{NOT}$ *S*) ∪ *unmark-l* (*trail S*) ∪ {{#}})› **and**
cons: ‹*consistent-interp LL*› **and**
LL: ‹*LL* ⊨*s set-mset* ($clauses_{NOT}$ *S*) ∪ *unmark-l* (*trail S*)›
have ‹*total-over-m LL* (*CNot C*)›
by (*metis* ‹*C* ∈# $clauses_{NOT}$ *S*› *insert-absorb tot total-over-m-CNot-toal-over-m total-over-m-insert total-over-m-union*)
then have *total-over-m LL* (*unmark-l* (*trail S*) ∪ *CNot C*)
using *tot* **by** *force*
then show *LL* ⊨*s* {{#}}
using *tot cons LL*
by (*metis* (*no-types*) ‹*C* ∈# $clauses_{NOT}$ *S*› ‹*trail S* ⊨*as CNot C*› *consistent-CNot-not true-annots-true-clss-clss true-clss-clss-def true-clss-def true-clss-union*)
qed
have ‹*undefined-lit F K*› **using** ‹*no-dup ?M*› **unfolding** *M-K* **by** (*auto simp*: *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* **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-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*)› *sat*
    unfolding *M-K* **by** *auto*
  then show *?thesis* **by** *fast*
qed *auto*
qed

end — End of the locale *dpll-with-backjumping-ops*.

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

lemma *rtranclp-dpll-bj-inv*:

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

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

lemma *rtranclp-dpll-bj-atms-of-ms-clauses-inv*:
assumes
⟨*dpll-bj*** *S T*⟩ **and** ⟨*inv S*⟩
shows ⟨*atms-of-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-bj*** *S T*⟩ **and**
⟨*inv S*⟩ **and**
⟨*atm-of* ' (*lits-of-l* (*trail S*)) $\subseteq$ *atms-of-mm* (*clauses*$_{NOT}$ *S*)⟩
shows ⟨*atm-of* ' (*lits-of-l* (*trail T*)) $\subseteq$ *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-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-bj*** *S T*⟩ **and**
⟨*inv S*⟩
⟨*atms-of-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-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-bj*$^{++}$ *S T*

$\wedge$ *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) $\wedge$ *inv* S}
$\subseteq$ {(T, S). *dpll-bj* S T $\wedge$ *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) $\wedge$ *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-bj*$^{++}$ S T› **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**
‹*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)) $\subseteq$ *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) $\in$ *?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
$\wedge$ *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) $\wedge$ *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 $\Longrightarrow$ *inv* S $\Longrightarrow$ $I$$\models$*sextm clauses*$_{NOT}$ S $\longleftrightarrow$ $I$$\models$*sextm clauses*$_{NOT}$ T›
by (*simp add*: *dpll-bj-clauses*)

lemma *rtranclp-dpll-bj-sat-ext-iff*:
‹*dpll-bj*** S T $\Longrightarrow$ *inv* S $\Longrightarrow$ $I$$\models$*sextm clauses*$_{NOT}$ S $\longleftrightarrow$ $I$$\models$*sextm clauses*$_{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 the locale *dpll-with-backjumping*.

2.2.4 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-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
    learn-conds :: ‹*'v clause* ⇒ *'st* ⇒ *bool*›
begin

inductive *learn* :: ‹*'st* ⇒ *'st* ⇒ *bool*› **where**
learn$_{NOT}$*-rule*: ‹*clauses*$_{NOT}$ *S* ⊨*pm C* ⟹
  atms-of C ⊆ *atms-of-mm* (*clauses*$_{NOT}$ *S*) ∪ *atm-of* ‘ (*lits-of-l* (*trail S*)) ⟹
  learn-conds C S ⟹
  T ∼ *add-cls*$_{NOT}$ *C S* ⟹
  learn S T›
inductive-cases *learn*$_{NOT}$*E*: ‹*learn S T*›

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

end

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* ⇒ ('*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*› +
fixes
forget-conds :: ‹'*v clause* ⇒ '*st* ⇒ *bool*›
begin

inductive $forget_{NOT}$ :: ‹'*st* ⇒ '*st* ⇒ *bool*› **where**
$forget_{NOT}$:
‹*removeAll-mset* *C*($clauses_{NOT}$ *S*) ⊨*pm* *C* ⟹
forget-conds *C* *S* ⟹
C ∈# $clauses_{NOT}$ *S* ⟹
T ∼ $remove\text{-}cls_{NOT}$ *C* *S* ⟹

$forget_{NOT}$ *S* *T*›
inductive-cases $forget_{NOT}E$: ‹$forget_{NOT}$ *S* *T*›

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

locale $learn\text{-}and\text{-}forget_{NOT}$ =
learn-ops trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ *learn-conds* +
forget-ops trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ *forget-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**
learn-conds forget-conds :: ‹'*v clause* ⇒ '*st* ⇒ *bool*›
begin
inductive $learn\text{-}and\text{-}forget_{NOT}$ :: ‹'*st* ⇒ '*st* ⇒ *bool*›
where
lf-learn: ‹*learn* *S* *T* ⟹ $learn\text{-}and\text{-}forget_{NOT}$ *S* *T*› |
lf-forget: ‹$forget_{NOT}$ *S* *T* ⟹ $learn\text{-}and\text{-}forget_{NOT}$ *S* *T*›
end

Definition of CDCL

locale *conflict-driven-clause-learning-ops* =
dpll-with-backjumping-ops trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
inv decide-conds backjump-conds propagate-conds +

learn-and-forget$_{NOT}$ *trail clauses*$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$ *learn-conds forget-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**
inv :: ⟨*'st* ⇒ *bool*⟩ **and**
decide-conds :: ⟨*'st* ⇒ *'st* ⇒ *bool*⟩ **and**
backjump-conds :: ⟨*'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool*⟩ **and**
propagate-conds :: ⟨(*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *'st* ⇒ *bool*⟩ **and**
learn-conds forget-conds :: ⟨*'v clause* ⇒ *'st* ⇒ *bool*⟩
begin

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

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

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

Consistency of the trail **lemma** *cdcl*$_{NOT}$*-consistent*:
assumes
⟨*cdcl*$_{NOT}$ S T⟩ **and**
⟨*inv* S⟩ **and**
⟨*no-dup* (*trail* S)⟩
shows ⟨*consistent-interp* (*lits-of-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*›
shows ‹*atms-of-mm* (*clauses*$_{NOT}$ *T*) ⊆ *atms-of-mm* (*clauses*$_{NOT}$ *S*) ∪ *atm-of* ‘ (*lits-of-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 ‹*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*: *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**
‹*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*: *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**
‹*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*,*3*)
proof (*induction rule*: *cdcl*$_{NOT}$*-all-induct*)
case *dpll-bj*
then show *?case*
using *dpll-bj-all-decomposition-implies-inv* **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** *inv* = *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-cls*$_{NOT}$ *C S*)›
using *T state-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-remove-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*›
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* (*add-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* ⊨*sm* {#*C*#} + *clauses*$_{NOT}$ *S*› **by** *auto*
}
then have *H*: ‹*I* ⊨*sextm* (*clauses*$_{NOT}$ *S*) ⟹ *I* ⊨*sext insert C* (*set-mset* (*clauses*$_{NOT}$ *S*))›
unfolding *true-clss-ext-def* **by** *auto*
show *?case*
apply *standard*
using *T* **apply** (*auto simp add*: *H*)[]
using *T* **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* ⊨*sext set-mset* (*clauses*$_{NOT}$ *S*) − {*C*}› **and**
‹*I* ⊆ *J*› **and**
tot: ‹*total-over-m J* (*set-mset* (*clauses*$_{NOT}$ *S*))› **and**
cons: ‹*consistent-interp J*›
then have ‹*J* ⊨*s set-mset* (*clauses*$_{NOT}$ *S*) − {*C*}›
unfolding *true-clss-ext-def* **by** (*meson Diff-subset total-over-m-subset*)

moreover
with *cls-C* **have** ‹*J* ⊨ *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* ⊨*sm* (*clauses*$_{NOT}$ *S*)› **by** (*metis insert-Diff-single true-clss-insert*)
}
then have *H*: ‹*I* ⊨*sext set-mset* (*clauses*$_{NOT}$ *S*) − {*C*} ⟹ *I* ⊨*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 the locale *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**
atms-clauses-S: ‹*atms-of-mm* ($clauses_{NOT}\ S$) ⊆ *A*› **and**
atms-trail-S: ‹*atm-of* ‘(*lits-of-l* (*trail S*)) ⊆ *A*›
shows ‹*atm-of* ‘ (*lits-of-l* (*trail T*)) ⊆ *A* ∧ *atms-of-mm* ($clauses_{NOT}\ T$) ⊆ *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 ‹*atms-of-mm* ($clauses_{NOT}\ U$) ⊆ *A*›
using $cdcl_{NOT}$-*atms-of-ms-clauses-decreasing*[*OF* $cdcl_{NOT}$] *IH* ‹*inv T*› **by** *fast*
moreover
have ‹*atm-of* ‘(*lits-of-l* (*trail U*)) ⊆ *A*›
using $cdcl_{NOT}$-*atms-in-trail-in-set*[*OF* $cdcl_{NOT}$, *of A*]
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*›
  shows ‹*I*⊨*sextm clauses*$_{NOT}$ *S* ⟷ *I*⊨*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* ⟷ (*finite A* ∧ *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*))›

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* ≡ *learn S T* ∨ *forget*$_{NOT}$ *S T*›

lemma *rtranclp-learn-or-forget-cdcl*$_{NOT}$:
  ‹*learn-or-forget*** *S T* ⟹ *cdcl*$_{NOT}$** *S T*›
  using *rtranclp-mono*[*of learn-or-forget cdcl*$_{NOT}$] **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**
i-learn-forget: ‹¬*learn* (*f i*) (*f* (*Suc i*)) ∧ ¬*forget*$_{NOT}$ (*f i*) (*f* (*Suc i*))› **and**
‹∀*k*<*i*. *learn-or-forget* (*f k*) (*f* (*Suc k*))›
proof −
obtain i_0 **where** ‹¬ *learn* (*f* i_0) (*f* (*Suc* i_0)) ∧ ¬*forget*$_{NOT}$ (*f* i_0) (*f* (*Suc* i_0))›
using *j* **by** *auto*
then have ‹{*i*. *i*≤i_0 ∧ ¬ *learn* (*f i*) (*f* (*Suc i*)) ∧ ¬*forget*$_{NOT}$ (*f i*) (*f* (*Suc i*))} ≠ {}›
by *auto*
let *?I* = ‹{*i*. *i*≤i_0 ∧ ¬ *learn* (*f i*) (*f* (*Suc i*)) ∧ ¬*forget*$_{NOT}$ (*f i*) (*f* (*Suc i*))}›
let *?i* = ‹*Min ?I*›
have ‹*finite ?I*›
by *auto*
have ‹¬ *learn* (*f ?i*) (*f* (*Suc ?i*)) ∧ ¬*forget*$_{NOT}$ (*f ?i*) (*f* (*Suc ?i*))›
using *Min-in*[*OF* ‹*finite ?I*› ‹*?I* ≠ {}›] **by** *auto*
moreover have ‹∀*k*<*?i*. *learn-or-forget* (*f k*) (*f* (*Suc k*))›
using *Min.coboundedI*[*of* ‹{*i*. *i* ≤ i_0 ∧ ¬ *learn* (*f i*) (*f* (*Suc i*)) ∧ ¬ *forget*$_{NOT}$ (*f i*)
(*f* (*Suc i*))}›, *simplified*]
by (*meson* ‹¬ *learn* (*f* i_0) (*f* (*Suc* i_0)) ∧ ¬ *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
define *g* **where** ‹*g* = (λ*n*. *f* (*n* + *Suc i*))›
have ‹*dpll-bj* (*f i*) (*g 0*)›
using *i-learn-forget cdcl*$_{NOT}$ *cdcl*$_{NOT}$*.cases* **unfolding** *g-def* **by** *auto*
{
fix *j*
assume ‹*j* ≤ *i*›
then have ‹*learn-or-forget*** (*f 0*) (*f j*)›
apply (*induction j*)
apply *simp*
by (*metis* (*no-types*, *lifting*) *Suc-leD Suc-le-lessD rtranclp.simps*
‹∀*k*<*i*. *learn* (*f k*) (*f* (*Suc k*)) ∨ *forget*$_{NOT}$ (*f k*) (*f* (*Suc k*))›)
}
then have ‹*learn-or-forget*** (*f 0*) (*f i*)› **by** *blast*

then have ‹(*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*))
  − μ_C (*1* + *card* (*atms-of-ms A*)) (*2* + *card* (*atms-of-ms A*)) (*trail-weight* (*g 0*))
  < (*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*))
  − μ_C (*1* + *card* (*atms-of-ms A*)) (*2* + *card* (*atms-of-ms A*)) (*trail-weight* (*f 0*))›
  using *learn-or-forget-dpll-*μ_C[*of* ‹*f 0*› ‹*f i*› ‹*g 0*› *A*] *inv* ‹*dpll-bj* (*f i*) (*g 0*)›
  unfolding *cdcl*$_{NOT}$*-NOT-all-inv-def* **by** *linarith*

moreover have *cdcl*$_{NOT}$*-i*: ‹*cdcl*$_{NOT}$** (*f 0*) (*g 0*)›
  using *rtranclp-learn-or-forget-cdcl*$_{NOT}$[*of* ‹*f 0*› ‹*f i*›] ‹*learn-or-forget*** (*f 0*) (*f i*)›
    cdcl$_{NOT}$[*of i*] **unfolding** *g-def* **by** *auto*
moreover have ‹$\bigwedge$*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*≥*j* ⟹ *learn-or-forget* (*g i*) (*g* (*Suc i*))›
  using *IH* **unfolding** μ[*symmetric*] **by** *presburger*
show *?thesis*
proof
  {
    fix *k*
    assume ‹*k* ≥ *j* + *Suc i*›
    then have ‹*learn-or-forget* (*f k*) (*f* (*Suc k*))›
      using *j*[*of* ‹*k*−*Suc i*›] **unfolding** *g-def* **by** *auto*
  }
  then show ‹∀ *k*≥*j*+*Suc i*. *learn-or-forget* (*f k*) (*f* (*Suc k*))›
    by *auto*
qed
qed
next
case *0* **note** *H* = *this*(*1*) **and** *cdcl*$_{NOT}$ = *this*(*2*) **and** *inv* = *this*(*3*)
show *?case*
proof (*rule ccontr*)
  assume ‹¬ *?case*›
  then have *j*: ‹∃ *i*. ¬ *learn* (*f i*) (*f* (*Suc i*)) ∧ ¬*forget*$_{NOT}$ (*f i*) (*f* (*Suc i*))›
    by *blast*
  obtain *i* **where**
    ‹¬*learn* (*f i*) (*f* (*Suc i*)) ∧ ¬*forget*$_{NOT}$ (*f i*) (*f* (*Suc i*))› **and**
    ‹∀ *k*<*i*. *learn-or-forget* (*f k*) (*f* (*Suc k*))›
  proof −
    obtain i_0 **where** ‹¬ *learn* (*f* i_0) (*f* (*Suc* i_0)) ∧ ¬*forget*$_{NOT}$ (*f* i_0) (*f* (*Suc* i_0))›
      using *j* **by** *auto*
    then have ‹{*i*. *i*≤i_0 ∧ ¬ *learn* (*f i*) (*f* (*Suc i*)) ∧ ¬*forget*$_{NOT}$ (*f i*) (*f* (*Suc i*))} ≠ {}›
      by *auto*
    let *?I* = ‹{*i*. *i*≤i_0 ∧ ¬ *learn* (*f i*) (*f* (*Suc i*)) ∧ ¬*forget*$_{NOT}$ (*f i*) (*f* (*Suc i*))}›
    let *?i* = ‹*Min ?I*›
    have ‹*finite ?I*›
      by *auto*
    have ‹¬ *learn* (*f ?i*) (*f* (*Suc ?i*)) ∧ ¬*forget*$_{NOT}$ (*f ?i*) (*f* (*Suc ?i*))›
      using *Min-in*[*OF* ‹*finite ?I*› ‹*?I* ≠ {}›] **by** *auto*
    moreover have ‹∀ *k*<*?i*. *learn-or-forget* (*f k*) (*f* (*Suc k*))›
      using *Min.coboundedI*[*of* ‹{*i*. *i* ≤ i_0 ∧ ¬ *learn* (*f i*) (*f* (*Suc i*)) ∧ ¬ *forget*$_{NOT}$ (*f i*)
        (*f* (*Suc i*))}›, *simplified*]
      by (*meson* ‹¬ *learn* (*f* i_0) (*f* (*Suc* i_0)) ∧ ¬ *forget*$_{NOT}$ (*f* i_0) (*f* (*Suc* i_0))› *less-imp-le*
        dual-order.trans not-le)
    ultimately show *?thesis* **using** *that* **by** *blast*
  qed

have ⟨*dpll-bj* (*f i*) (*f* (*Suc i*))⟩
using ⟨¬ *learn* (*f i*) (*f* (*Suc i*)) ∧ ¬ $forget_{NOT}$ (*f i*) (*f* (*Suc i*))⟩ $cdcl_{NOT}$ $cdcl_{NOT}$*.cases*
by *blast*
{
fix *j*
assume ⟨$j \leq i$⟩
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-*μ_C[*of* ⟨*f 0*⟩ ⟨*f i*⟩ ⟨*f* (*Suc i*)⟩ *A*] *inv 0*
⟨*dpll-bj* (*f i*) (*f* (*Suc i*))⟩ **unfolding** $cdcl_{NOT}$*-NOT-all-inv-def* **by** *linarith*
qed
qed

lemma *wf-*$cdcl_{NOT}$*-no-learn-and-forget-infinite-chain*:
assumes
no-infinite-lf: ⟨⋀*f j*. ¬ (∀ *i*≥*j*. *learn-or-forget* (*f i*) (*f* (*Suc i*)))⟩
shows ⟨*wf* {(*T*, *S*). $cdcl_{NOT}$ *S T* ∧ $cdcl_{NOT}$*-NOT-all-inv A S*}⟩
(**is** ⟨*wf* {(*T*, *S*). $cdcl_{NOT}$ *S T* ∧ *?inv S*}⟩)
unfolding *wf-iff-no-infinite-down-chain*
proof (*rule ccontr*)
assume ⟨¬ ¬ (∃ *f*. ∀ *i*. (*f* (*Suc i*), *f i*) ∈ {(*T*, *S*). $cdcl_{NOT}$ *S T* ∧ *?inv S*})⟩
then obtain *f* **where**
⟨∀ *i*. $cdcl_{NOT}$ (*f i*) (*f* (*Suc i*)) ∧ *?inv* (*f i*)⟩
by *fast*
then have ⟨∃ *j*. ∀ *i*≥*j*. *learn-or-forget* (*f i*) (*f* (*Suc i*))⟩
using *infinite-*$cdcl_{NOT}$*-exists-learn-and-forget-infinite-chain*[*of f*] **by** *meson*
then show *False* **using** *no-infinite-lf* **by** *blast*
qed

lemma *inv-and-tranclp-cdcl-*$_{NOT}$*-tranclp-*$cdcl_{NOT}$*-and-inv*:
⟨$cdcl_{NOT}{}^{++}$ *S T* ∧ $cdcl_{NOT}$*-NOT-all-inv A S* ⟷ (λ*S T*. $cdcl_{NOT}$ *S T* ∧ $cdcl_{NOT}$*-NOT-all-inv A S*)$^{++}$ *S T*⟩
(**is** ⟨*?A* ∧ *?I* ⟷ *?B*⟩)
proof
assume ⟨*?A* ∧ *?I*⟩
then have *?A* **and** *?I* **by** *blast*+
then show *?B*
apply *induction*
apply (*simp add*: *tranclp.r-into-trancl*)
by (*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 ⟨*?A* ∧ *?I*⟩ **by** *blast*
qed

lemma *wf-tranclp-*$cdcl_{NOT}$*-no-learn-and-forget-infinite-chain*:
assumes

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

lemma $cdcl_{NOT}$*-final-state*:
assumes
n-s: ‹*no-step* $cdcl_{NOT}$ *S*› **and**
inv: ‹$cdcl_{NOT}$*-NOT-all-inv A S*› **and**
decomp: ‹*all-decomposition-implies-m* ($clauses_{NOT}$ *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 −
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 the locale *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 decide-conds backjump-conds propagate-conds
⟨$\lambda C\ S.$ *distinct-mset* $C \wedge \neg$*tautology* $C \wedge$ *learn-restrictions* $C\ S \wedge$
($\exists F\ K\ d\ F'\ C'\ L.$ *trail* $S = F'$ @ *Decided* K # $F \wedge C =$ *add-mset* $L\ C' \wedge F \models$*as CNot* C'
$\wedge$ *add-mset* $L\ C' \notin\#$ *clauses*$_{NOT}$ S)⟩
⟨$\lambda C\ S.$ $\neg(\exists F'\ F\ K\ d\ L.$ *trail* $S = F'$ @ *Decided* K # $F \wedge F \models$*as CNot* (*remove1-mset* $L\ C$))
$\wedge$ *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**
decide-conds :: ⟨'*st* ⇒ '*st* ⇒ *bool*⟩ **and**
backjump-conds :: ⟨'*v clause* ⇒ '*v clause* ⇒ '*v literal* ⇒ '*st* ⇒ '*st* ⇒ *bool*⟩ **and**
propagate-conds :: ⟨('*v*, *unit*) *ann-lit* ⇒ '*st* ⇒ '*st* ⇒ *bool*⟩ **and**
learn-restrictions forget-restrictions :: ⟨'*v clause* ⇒ '*st* ⇒ *bool*⟩
begin

lemma *cdcl*$_{NOT}$*-learn-all-induct*[*consumes 1*, *case-names dpll-bj learn forget*$_{NOT}$]:
fixes $S\ T$ :: ⟨'*st*⟩
assumes ⟨*cdcl*$_{NOT}$ $S\ T$⟩ **and**
dpll: ⟨$\bigwedge T.$ *dpll-bj* $S\ T \Longrightarrow P\ S\ T$⟩ **and**
learning:
⟨$\bigwedge C\ F\ K\ F'\ C'\ L\ T.$ *clauses*$_{NOT}$ $S \models$*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 # $F \Longrightarrow$
$C =$ *add-mset* $L\ C' \Longrightarrow$
$F \models$*as CNot* $C' \Longrightarrow$
add-mset $L\ C' \notin\#$ *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*)
    ⊆ *simple-clss* (*atms-of-mm* ($clauses_{NOT}$ *S*) ∪ *atm-of* ‘ *lits-of-l* (*trail S*))›
proof
  fix *C* **assume** *C*: ‹*C* ∈ *set-mset* ($clauses_{NOT}$ *T* − $clauses_{NOT}$ *S*)›
  have ‹*distinct-mset C*› ‹¬*tautology C*› **using** *learn C n-d* **by** (*elim* $learn_{NOT}E$; *auto*)+
  then have ‹*C* ∈ *simple-clss* (*atms-of C*)›
    using *distinct-mset-not-tautology-implies-in-simple-clss* **by** *blast*
  moreover have ‹*atms-of C* ⊆ *atms-of-mm* ($clauses_{NOT}$ *S*) ∪ *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*) ∪ *atm-of* ‘ *lits-of-l* (*trail S*))›
    by *auto*
  ultimately show ‹*C* ∈ *simple-clss* (*atms-of-mm* ($clauses_{NOT}$ *S*) ∪ *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* ≡
   {*C*+{#*L*#} |*C L*. *C*+{#*L*#} ∈# $clauses_{NOT}$ *S* ∧ *distinct-mset* (*C*+{#*L*#})
   ∧ ¬*tautology* (*C*+{#*L*#})
     ∧ (∃ *F' K F*. *trail S* = *F'* @ *Decided K* # *F* ∧ *F* ⊨*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* ∼ *remove-cls*$_{NOT}$ *C S* ⟹ *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* ∼ *add-cls*$_{NOT}$ *C' S*› **and**
    n-d: ‹*no-dup* (*trail S*)›
  shows ‹*conflicting-bj-clss T*
    = *conflicting-bj-clss S*
     ∪ (*if* ∃ *C L*. *C'* = *add-mset L C* ∧ *distinct-mset* (*add-mset L C*) ∧ ¬*tautology* (*add-mset L C*)
    ∧ (∃ *F' K d F*. *trail S* = *F'* @ *Decided K* # *F* ∧ *F* ⊨*as CNot C*)
     then {*C'*} *else* {})›
proof −
  define *P* **where** ‹*P* = (λ*C L T*. *distinct-mset* (*add-mset L C*) ∧ ¬ *tautology* (*add-mset L C*) ∧
    (∃ *F' K F*. *trail T* = *F'* @ *Decided K* # *F* ∧ *F* ⊨*as CNot C*))›
  have *conf*: ‹⋀*T*. *conflicting-bj-clss T* = {*add-mset L C* |*C L*. *add-mset L C* ∈# $clauses_{NOT}$ *T* ∧ *P C L T*}›
    unfolding *conflicting-bj-clss-def P-def* **by** *auto*
  have *P-S-T*: ‹⋀*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* = {*add-mset L C* |*C L*. *add-mset L C* ∈# $clauses_{NOT}$ *S* ∧ *P C L T*} ∪
     {*add-mset L C* |*C L*. *add-mset L C* ∈# {#*C'*#} ∧ *P C L T*}›
    using *T n-d* **unfolding** *conf* **by** *auto*
  moreover have ‹{*add-mset L C* |*C L*. *add-mset L C* ∈# $clauses_{NOT}$ *S* ∧ *P C L T*} = *conflicting-bj-clss S*›

using T *n-d* **unfolding** *P-def conflicting-bj-clss-def* **by** *auto*
moreover have ⟨$\{add\text{-}mset\ L\ C\ |C\ L.\ add\text{-}mset\ L\ C \in\#\ \{\#C'\#\} \wedge P\ C\ L\ T\} =$
$(if\ \exists\,C\ L.\ C' = add\text{-}mset\ L\ C \wedge P\ C\ L\ S\ then\ \{C'\}\ else\ \{\})$⟩
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-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 *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-add-mset-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*) ∪ *atm-of* ‘ *lits-of-l* (*trail S*) ⊆ *A*› **and**
  fin-A: ‹*finite A*›
  shows ‹(μ_L (*card A*) *T*, μ_L (*card A*) *S*) ∈ *less-than* <∗*lex*∗> *less-than*›
proof −
  have [*simp*]: ‹(*atms-of-mm* ($clauses_{NOT}$ *T*) ∪ *atm-of* ‘ *lits-of-l* (*trail T*))
    = (*atms-of-mm* ($clauses_{NOT}$ *S*) ∪ *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* = *add-mset L C′*› **and**
        F: ‹*F* ⊨*as CNot C′*› **and**
        C-S:‹*add-mset L C′* ∉# $clauses_{NOT}$ *S*›
        by *blast*
      moreover have ‹*distinct-mset C*› ‹¬ *tautology C*› **using** *inv* **by** *blast*+
      ultimately have ‹*add-mset L C′* ∈ *conflicting-bj-clss T*›
        using *T n-d* **unfolding** *conflicting-bj-clss-def* **by** *fastforce*
      moreover have ‹*add-mset L C′* ∉ *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-bj-clss-add-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*: ‹$\bigwedge$*x*. *x*∈ *conflicting-bj-clss T* ⟹ ¬ *tautology x* ∧ *distinct-mset x*›

unfolding *conflicting-bj-clss-def* **by** *auto*
have T: ‹*conflicting-bj-clss* T
$\subseteq$ *simple-clss* (*atms-of-mm* ($clauses_{NOT}$ T) $\cup$ *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) $\cup$ *atm-of* ‘ *lits-of-l* (*trail* T))
$\geq$ *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) $\cup$ *atm-of* ‘ *lits-of-l* (*trail* T) $\subseteq$ A›
using $learn_{NOT}E$[*OF learnST*] A **by** *simp*
then have ‹*3* ^ (*card* A) $\geq$ *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 $\neq$ *conflicting-bj-clss* T›
‹*conflicting-bj-clss* S $\subseteq$ *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) $\subseteq$ *atms-of-ms* A and in the clauses *atms-of-mm* ($clauses_{NOT}$ S) $\subseteq$ *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 $\equiv$ ((*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) $\subseteq$ *atms-of-ms* A› **and**
atm-lits: ‹*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 ‹(μ_{CDCL} A T, μ_{CDCL} A S)
$\in$ *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*)
∧ *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*)) − μ_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*)) − μ_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*)) − μ_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*)) − μ_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* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) − μ_C*'* *A T*)
∗ (*1* + *3* ^ *card* (*atms-of-ms A*)) ∗ *2*
+ *conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) *T* ∗ *2*
< ((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) − μ_C*'* *A S*)
∗ (*1* + *3* ^ *card* (*atms-of-ms A*)) ∗ *2*
+ *conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) *S* ∗ *2*›
by *linarith*
then show *?case* **unfolding** μ_{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* ∈# $clauses_{NOT}$ *S* ∧ *distinct-mset x* ∧ ¬ *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*]: ‹*conflicting-bj-clss* (*add-cls*$_{NOT}$ *C S*) = *conflicting-bj-clss S* ∪ {*C*}›
 using *dist tauto F-C* **by** (*subst conflicting-bj-clss-add-cls*$_{NOT}$[*OF n-d*]) (*force simp*: C' *tr-S n-d*)
 ultimately have [*simp*]: ‹*conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) *S*
 = *Suc* (*conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) (*add-cls*$_{NOT}$ *C S*))›
 by *simp*
 have *1*: ‹*clauses*$_{NOT}$ *T* = *clauses*$_{NOT}$ (*add-cls*$_{NOT}$ *C S*)› **using** *T* **by** *auto*
 have *2*: ‹*conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) *T*
 = *conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) (*add-cls*$_{NOT}$ *C S*)›
 using *T* **unfolding** *conflicting-bj-clss-def* **by** *auto*
 have *3*: ‹$\mu_C{}'$ *A T* = $\mu_C{}'$ *A* (*add-cls*$_{NOT}$ *C S*)›
 using *T* **unfolding** $\mu_C{}'$*-def* **by** *auto*
 have ‹((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) − $\mu_C{}'$ *A* (*add-cls*$_{NOT}$ *C S*))
 ∗ (*1* + *3* ^ *card* (*atms-of-ms A*)) ∗ *2*
 = ((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) − $\mu_C{}'$ *A S*)
 ∗ (*1* + *3* ^ *card* (*atms-of-ms A*)) ∗ *2*›
 using *n-d* **unfolding** $\mu_C{}'$*-def* **by** *auto*
 moreover
 have ‹*conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) (*add-cls*$_{NOT}$ *C S*)
 ∗ *2*
 + *card* (*set-mset* (*clauses*$_{NOT}$ (*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** $\mu_{CDCL}{}'$*-def 1 2 3* **by** *presburger*
next
 case (*forget*$_{NOT}$ *C T*) **note** *T* = *this*(*4*)
 have [*simp*]: ‹$\mu_C{}'$ *A* (*remove-cls*$_{NOT}$ *C S*) = $\mu_C{}'$ *A S*›
 unfolding $\mu_C{}'$*-def* **by** *auto*
 have ‹*forget*$_{NOT}$ *S T*›
 apply (*rule forget*$_{NOT}$*.intros*) **using** *forget*$_{NOT}$ **by** *auto*
 then have ‹*conflicting-bj-clss T* = *conflicting-bj-clss S*›
 using *do-not-forget-before-backtrack-rule-clause-learned-clause-untouched* **by** *blast*
 moreover have ‹*card* (*set-mset* (*clauses*$_{NOT}$ *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** $\mu_{CDCL}{}'$*-def*
 using *T* ‹$\mu_C{}'$ *A* (*remove-cls*$_{NOT}$ *C S*) = $\mu_C{}'$ *A S*› **by** (*metis* (*no-types*) *add-le-cancel-left* $\mu_C{}'$*-def not-le state-eq*$_{NOT}$*-trail*)
qed

lemma *cdcl*$_{NOT}$*-clauses-bound*:
 assumes
 ‹*cdcl*$_{NOT}$ *S T*› **and**
 ‹*inv S*› **and**
 ‹*atms-of-mm* (*clauses*$_{NOT}$ *S*) ⊆ *A*› **and**
 ‹*atm-of* ‘(*lits-of-l* (*trail S*)) ⊆ *A*› **and**
 n-d: ‹*no-dup* (*trail S*)› **and**
 fin-A[*simp*]: ‹*finite A*›
 shows ‹*set-mset* (*clauses*$_{NOT}$ *T*) ⊆ *set-mset* (*clauses*$_{NOT}$ *S*) ∪ *simple-clss A*›
 using *assms*
proof (*induction rule*: *cdcl*$_{NOT}$*-learn-all-induct*)
 case *dpll-bj*
 then show *?case* **using** *dpll-bj-clauses* **by** *simp*

next
  case *forget*$_{NOT}$
  then show *?case* **using** *clauses-remove-cls*$_{NOT}$ **unfolding** *state-eq*$_{NOT}$*-def* **by** *auto*
next
  case (*learn C F K d F′ C′ L*) **note** *atms-C* = *this*(*2*) **and** *dist* = *this*(*3*) **and** *tauto* = *this*(*4*) **and** *T* = *this*(*10*) **and** *atms-clss-S* = *this*(*12*) **and** *atms-trail-S* = *this*(*13*)
  have ‹*atms-of C* ⊆ *A*›
    using *atms-C atms-clss-S atms-trail-S* **by** *fast*
  then have ‹*simple-clss* (*atms-of C*) ⊆ *simple-clss A*›
    by (*simp add*: *simple-clss-mono*)
  then have ‹*C* ∈ *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*) ⊆ *A*› **and**
    MA: ‹*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* {*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 ‹⋀*x*. *x* ∈# *clauses*$_{NOT}$ *T* ⟹ ¬ *tautology x* ⟹ *distinct-mset x* ⟹ *x* ∈ *simple-clss A*›
    using *rtranclp-cdcl*$_{NOT}$*-clauses-bound*[*OF assms*] **by** (*metis* (*no-types*, *hide-lams*) *Un-iff 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*) ⊆ *?S* ∪ *simple-clss A*›
    using *rtranclp-cdcl*$_{NOT}$*-clauses-bound*[*OF assms*] **by** *auto*
  then have ‹*card*(*set-mset* (*clauses*$_{NOT}$ *T*)) ≤ *card* (*?S* ∪ *simple-clss A*)›
    using *finite* **by** (*simp add*: *assms*(*5*) *simple-clss-finite card-mono*)
  then show *?thesis*
    by (*meson le-trans simple-clss-card card-Un-le local.finite nat-add-left-cancel-le*)
qed

definition $\mu_{CDCL}{}'$*-bound* :: ‹*'v clause set* ⇒ *'st* ⇒ *nat*› **where**
‹$\mu_{CDCL}{}'$*-bound A S* =
  ((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*))) ∗ (*1* + *3* ^ *card* (*atms-of-ms A*)) ∗ *2*
    + *2*∗*3* ^ (*card* (*atms-of-ms A*))
    + *card* {*C*. *C* ∈# *clauses*$_{NOT}$ *S* ∧ (*tautology C* ∨ ¬*distinct-mset C*)} + *3* ^ (*card* (*atms-of-ms A*))›

lemma $\mu_{CDCL}{}'$*-bound-reduce-trail-to*$_{NOT}$[*simp*]:
  ‹$\mu_{CDCL}{}'$*-bound A* (*reduce-trail-to*$_{NOT}$ *M S*) = $\mu_{CDCL}{}'$*-bound A S*›
  unfolding $\mu_{CDCL}{}'$*-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*) ⊆ *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* (*atms-of-ms A*)› **and**
    U: ‹*U* ∼ *reduce-trail-to*$_{NOT}$ *M T*›
  shows ‹$\mu_{CDCL}{}'$ *A U* ≤ μ_{CDCL}*'-bound A S*›
proof −
  have ‹ ((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) − $\mu_C{}'$ *A U*)
    ≤ (*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*))›
    by *auto*
  then have ‹((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) − $\mu_C{}'$ *A U*)
        ∗ (*1* + *3* ^ *card* (*atms-of-ms A*)) ∗ *2*
    ≤ (*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) ∗ (*1* + *3* ^ *card* (*atms-of-ms A*)) ∗ *2*›
    using *mult-le-mono1* **by** *blast*
  moreover
    have ‹*conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) *T* ∗ *2* ≤ *2* ∗ *3* ^ *card* (*atms-of-ms A*)›
      by *linarith*
  moreover have ‹*card* (*set-mset* (*clauses*$_{NOT}$ *U*))
      ≤ *card* {*C*. *C* ∈# *clauses*$_{NOT}$ *S* ∧ (*tautology C* ∨ ¬*distinct-mset C*)} + *3* ^ *card* (*atms-of-ms A*)›
    using *rtranclp-cdcl*$_{NOT}$*-card-simple-clauses-bound*[*OF assms*(*1*−*6*)] *U* **by** *auto*
  ultimately show *?thesis*
    unfolding 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*) ⊆ *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* (*atms-of-ms A*)›
  shows ‹$\mu_{CDCL}{}'$ *A T* ≤ μ_{CDCL}*'-bound A S*›
proof −
  have ‹$\mu_{CDCL}{}'$ *A* (*reduce-trail-to*$_{NOT}$ (*trail T*) *T*) = $\mu_{CDCL}{}'$ *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*) ⊆ *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[*simp*]: ‹*finite* (*atms-of-ms A*)›
  shows ‹μ_{CDCL}*'-bound A T* ≤ μ_{CDCL}*'-bound A S*›
proof −
  have ‹{*C*. *C* ∈# *clauses*$_{NOT}$ *T* ∧ (*tautology C* ∨ ¬ *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 the locale *conflict-driven-clause-learning-learning-before-backjump-only-distinct-learnt.*

2.2.5 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-cls*$_{NOT}$ *remove-cls*$_{NOT}$
inv decide-conds backjump-conds propagate-conds learn-conds forget-conds
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**
inv :: ‹*'st* $\Rightarrow$ *bool*› **and**
decide-conds :: ‹*'st* $\Rightarrow$ *'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* $\Rightarrow$ *'st* $\Rightarrow$ *bool*› **and**
learn-conds forget-conds :: ‹*'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

Definition We define our increasing restart very abstractly: the predicate (called *cdcl*$_{NOT}$) does not have to be a CDCL calculus. We just need some assuptions to prove termination:

- 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* ⇒ *'st* ⇒ *bool*› **and**
cdcl$_{NOT}$ :: ‹*'st* ⇒ *'st* ⇒ *bool*› +
fixes
f :: ‹*nat* ⇒ *nat*› **and**
bound-inv :: ‹*'bound* ⇒ *'st* ⇒ *bool*› **and**
μ :: ‹*'bound* ⇒ *'st* ⇒ *nat*› **and**
cdcl$_{NOT}$*-inv* :: ‹*'st* ⇒ *bool*› **and**
μ-bound :: ‹*'bound* ⇒ *'st* ⇒ *nat*›
assumes
f: ‹*unbounded f*› **and**
f-ge-1: ‹$\bigwedge n.\ n \geq 1 \Longrightarrow f\ n \neq 0$› **and**
bound-inv: ‹$\bigwedge A\ S\ T.$ *cdcl*$_{NOT}$*-inv* $S \Longrightarrow$ *bound-inv* $A\ S \Longrightarrow$ *cdcl*$_{NOT}$ $S\ T \Longrightarrow$ *bound-inv* $A\ T$› **and**
cdcl$_{NOT}$*-measure*: ‹$\bigwedge A\ S\ T.$ *cdcl*$_{NOT}$*-inv* $S \Longrightarrow$ *bound-inv* $A\ S \Longrightarrow$ *cdcl*$_{NOT}$ $S\ T \Longrightarrow \mu\ A\ T < \mu$ $A\ S$› **and**

measure-bound2: ‹$\bigwedge A\ T\ U.$ $cdcl_{NOT}$*-inv* $T \Longrightarrow$ *bound-inv* $A\ T \Longrightarrow cdcl_{NOT}{}^{**}\ T\ U$
$\Longrightarrow \mu\ A\ U \leq \mu$*-bound* $A\ T$› **and**
measure-bound4: ‹$\bigwedge A\ T\ U.$ $cdcl_{NOT}$*-inv* $T \Longrightarrow$ *bound-inv* $A\ T \Longrightarrow cdcl_{NOT}{}^{**}\ T\ U$
$\Longrightarrow \mu$*-bound* $A\ U \leq \mu$*-bound* $A\ T$› **and**
$cdcl_{NOT}$*-restart-inv*: ‹$\bigwedge A\ U\ V.$ $cdcl_{NOT}$*-inv* $U \Longrightarrow$ *restart* $U\ V \Longrightarrow$ *bound-inv* $A\ U \Longrightarrow$ *bound-inv* $A\ V$›
and
exists-bound: ‹$\bigwedge R\ S.$ $cdcl_{NOT}$*-inv* $R \Longrightarrow$ *restart* $R\ S \Longrightarrow \exists A.$ *bound-inv* $A\ S$› **and**
$cdcl_{NOT}$*-inv*: ‹$\bigwedge S\ T.$ $cdcl_{NOT}$*-inv* $S \Longrightarrow cdcl_{NOT}\ S\ T \Longrightarrow cdcl_{NOT}$*-inv* T› **and**
$cdcl_{NOT}$*-inv-restart*: ‹$\bigwedge S\ T.$ $cdcl_{NOT}$*-inv* $S \Longrightarrow$ *restart* $S\ T \Longrightarrow cdcl_{NOT}$*-inv* T›
begin

lemma $cdcl_{NOT}$-$cdcl_{NOT}$*-inv*:
assumes
‹$(cdcl_{NOT}{}^{\frown\frown}n)\ S\ T$› **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
‹$(cdcl_{NOT}{}^{\frown\frown}n)\ S\ T$› **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
‹$(cdcl_{NOT}{}^{\frown\frown}(Suc\ n))\ S\ T$› **and**
‹*bound-inv* $A\ S$›
‹$cdcl_{NOT}$*-inv* S›
shows ‹$\mu\ A\ T < \mu\ A\ S - n$›
using *assms*
proof (*induction n arbitrary*: T)
case *0*
then show *?case* **using** $cdcl_{NOT}$*-measure* **by** *auto*
next
case (*Suc n*) **note** $IH = this(1)[OF\ \text{-}\ this(3)\ this(4)]$ **and** $S\text{-}T = this(2)$ **and** $b\text{-}inv = this(3)$ **and** $c\text{-}inv = this(4)$
obtain $U :: 'st$ **where** $S\text{-}U$: ‹$(cdcl_{NOT}{}^{\frown\frown}(Suc\ n))\ S\ U$› **and** $U\text{-}T$: ‹$cdcl_{NOT}\ U\ T$› **using** $S\text{-}T$ **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* ∧ $cdcl_{NOT}$*-inv S* ∧ *bound-inv A S*}› (**is** ‹*wf ?A*›)
apply (*rule wfP-if-measure2*[*of - -* ‹μ *A*›])
using $cdcl_{NOT}$*-comp-n-le*[*of 0 - - A*] **by** *auto*

lemma *rtranclp-*$cdcl_{NOT}$*-measure*:
assumes
‹$cdcl_{NOT}{}^{**}$ *S T*› **and**
‹*bound-inv A S*› **and**
‹$cdcl_{NOT}$*-inv S*›
shows ‹μ *A T* ≤ 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 ‹μ *A U* < μ *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* ≥ *1*+μ *A S*›
shows ‹¬($cdcl_{NOT}$ ^^ *m*) *S T*›
using *assms* $cdcl_{NOT}$*-comp-n-le*[*of* ‹*m*−*1*› *S T A*] **by** *fastforce*

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

inductive $cdcl_{NOT}$*-restart* **where**
restart-step: ‹($cdcl_{NOT}$^^*m*) *S T* ⟹ *m* ≥ *f n* ⟹ *restart T U*
⟹ $cdcl_{NOT}$*-restart* (*S*, *n*) (*U*, *Suc n*)› |
restart-full: ‹*full1* $cdcl_{NOT}$ *S T* ⟹ $cdcl_{NOT}$*-restart* (*S*, *n*) (*T*, *Suc n*)›

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

lemma $cdcl_{NOT}$*-restart-*$cdcl_{NOT}$*-raw-restart*:
‹$cdcl_{NOT}$*-restart S T* ⟹ $cdcl_{NOT}$*-raw-restart*** (*fst S*) (*fst T*)›

proof (*induction rule*: *cdcl$_{NOT}$-restart.induct*)
  case (*restart-step m S T n U*)
  then have ‹*cdcl$_{NOT}$** S T*› **by** (*meson relpowp-imp-rtranclp*)
  then have ‹*cdcl$_{NOT}$-raw-restart** S T*› **using** *cdcl$_{NOT}$-raw-restart.intros*(*1*)
    rtranclp-mono[*of cdcl$_{NOT}$ cdcl$_{NOT}$-raw-restart*] **by** *blast*
  moreover have ‹*cdcl$_{NOT}$-raw-restart T U*›
    using ‹*restart T U*› *cdcl$_{NOT}$-raw-restart.intros*(*2*) **by** *blast*
  ultimately show *?case* **by** *auto*
next
  case (*restart-full S T*)
  then have ‹*cdcl$_{NOT}$** S T*› **unfolding** *full1-def* **by** *auto*
  then show *?case* **using** *cdcl$_{NOT}$-raw-restart.intros*(*1*)
    rtranclp-mono[*of cdcl$_{NOT}$ cdcl$_{NOT}$-raw-restart*] **by** *auto*
qed

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

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

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

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

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

locale *cdcl*$_{NOT}$*-increasing-restarts* =
cdcl$_{NOT}$*-increasing-restarts-ops restart cdcl*$_{NOT}$ *f bound-inv* μ *cdcl*$_{NOT}$*-inv* μ*-bound* +
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*› **and**
f :: ‹*nat* ⇒ *nat*› **and**
restart :: ‹*'st* ⇒ *'st* ⇒ *bool*› **and**
bound-inv :: ‹*'bound* ⇒ *'st* ⇒ *bool*› **and**
μ :: ‹*'bound* ⇒ *'st* ⇒ *nat*› **and**
cdcl$_{NOT}$ :: ‹*'st* ⇒ *'st* ⇒ *bool*› **and**
cdcl$_{NOT}$*-inv* :: ‹*'st* ⇒ *bool*› **and**
μ*-bound* :: ‹*'bound* ⇒ *'st* ⇒ *nat*› +
assumes
measure-bound: ‹$\bigwedge$*A T V n*. *cdcl*$_{NOT}$*-inv T* ⟹ *bound-inv A T*
⟹ *cdcl*$_{NOT}$*-restart* (*T*, *n*) (*V*, *Suc n*) ⟹ μ *A V* ≤ μ*-bound A T*› **and**
cdcl$_{NOT}$*-raw-restart-*μ*-bound*:
‹*cdcl*$_{NOT}$*-restart* (*T*, *a*) (*V*, *b*) ⟹ *cdcl*$_{NOT}$*-inv T* ⟹ *bound-inv A T*
⟹ μ*-bound A V* ≤ μ*-bound A T*›
begin

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

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

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

lemma *wf-cdcl*$_{NOT}$*-restart*:
‹*wf* {(*T*, *S*). *cdcl*$_{NOT}$*-restart S T* ∧ *cdcl*$_{NOT}$*-inv* (*fst S*)}› (**is** ‹*wf ?A*›)
proof (*rule ccontr*)
assume ‹¬ *?thesis*›
then obtain *g* **where**

g: ‹$\bigwedge$*i*. *cdcl*$_{NOT}$*-restart* (*g* *i*) (*g* (*Suc* *i*))› **and**
cdcl$_{NOT}$*-inv-g*: ‹$\bigwedge$*i*. *cdcl*$_{NOT}$*-inv* (*fst* (*g* *i*))›
unfolding *wf-iff-no-infinite-down-chain* **by** *fast*

have *snd-g*: ‹$\bigwedge$*i*. *snd* (*g* *i*) = *i* + *snd* (*g* *0*)›
apply (*induct-tac* *i*)
apply *simp*
by (*metis Suc-eq-plus1-left add.commute add.left-commute*
cdcl$_{NOT}$*-with-restart-increasing-number* *g*)
then have *snd-g-0*: ‹$\bigwedge$*i*. *i* > *0* $\Longrightarrow$ *snd* (*g* *i*) = *i* + *snd* (*g* *0*)›
by *blast*
have *unbounded-f-g*: ‹*unbounded* (λ*i*. *f* (*snd* (*g* *i*)))›
using *f* **unfolding** *bounded-def* **by** (*metis add.commute f less-or-eq-imp-le snd-g*
not-bounded-nat-exists-larger not-le le-iff-add)

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

⟨$cdcl_{NOT}$-restart** (fst (g 1), snd (g 1)) (fst (g j), snd (g j))⟩
rtranclp-$cdcl_{NOT}$-with-restart-bound-inv **by** auto
obtain T m **where**
$cdcl_{NOT}$-m: ⟨($cdcl_{NOT}$ ^^ m) (fst (g j)) T⟩ **and**
f-m: ⟨f (snd (g j)) ≤ 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)) ≤ μ-bound A (fst (g 1))⟩
$cdcl_{NOT}$-comp-bounded[OF inv $cdcl_{NOT}$-inv-g, of] $cdcl_{NOT}$-inv-g $cdcl_{NOT}$-m
⟨?j < m⟩ **by** auto
qed

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

lemma rtranclp-$cdcl_{NOT}$-with-inv-inv-rtranclp-$cdcl_{NOT}$:
assumes
inv: ⟨$cdcl_{NOT}$-inv S⟩ **and**
binv: ⟨bound-inv A S⟩
shows ⟨(λS T. $cdcl_{NOT}$ S T ∧ $cdcl_{NOT}$-inv S ∧ bound-inv A S)** S T ⟷ $cdcl_{NOT}$** S T⟩
(**is** ⟨?A** S T ⟷ ?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* (λS T. *cdcl*$_{NOT}$ S T ∧ *cdcl*$_{NOT}$*-inv* S ∧ *bound-inv* A S) T U›
using *wf-exists-normal-form-full*[*OF wf-cdcl*$_{NOT}$, *of* A T] **by** *auto*
moreover have *inv-T*: ‹*cdcl*$_{NOT}$*-inv* T›
using ‹*cdcl*$_{NOT}$ (*fst* S) T› *cdcl*$_{NOT}$*-inv inv* **by** *blast*
moreover have *b-inv-T*: ‹*bound-inv* A T›
using ‹*cdcl*$_{NOT}$ (*fst* S) T› *binv bound-inv inv* **by** *blast*
ultimately have ‹*full cdcl*$_{NOT}$ T U›
using *rtranclp-cdcl*$_{NOT}$*-with-inv-inv-rtranclp-cdcl*$_{NOT}$ *rtranclp-cdcl*$_{NOT}$*-bound-inv*
rtranclp-cdcl$_{NOT}$*-cdcl*$_{NOT}$*-inv* **unfolding** *full-def* **by** *blast*
then have ‹*full1 cdcl*$_{NOT}$ (*fst* S) U›
using T *full-fullI* **by** *metis*
then show *False* **by** (*metis n-s prod.collapse restart-full*)
qed

end

2.2.6 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*$_{NOT}$ *decide-conds* +
forget-ops trail clauses$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$ *forget-conds* +
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**
decide-conds :: ‹'*st* ⇒ '*st* ⇒ *bool*› **and**
propagate-conds :: ‹('*v*, *unit*) *ann-lit* ⇒ '*st* ⇒ '*st* ⇒ *bool*› **and**
forget-conds :: ‹'*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
⟹ 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 add-mset* L C'
⟹ C'' = *add-mset* L C'
⟹ 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* :: ⟨$'st \Rightarrow 'st \Rightarrow bool$⟩ **for** S :: $'st$ **where**
cdcl$_{NOT}$*-merged-bj-learn-decide*$_{NOT}$: ⟨*decide*$_{NOT}$ S S' $\Longrightarrow$ *cdcl*$_{NOT}$*-merged-bj-learn* S S'⟩ |
cdcl$_{NOT}$*-merged-bj-learn-propagate*$_{NOT}$: ⟨*propagate*$_{NOT}$ S S' $\Longrightarrow$ *cdcl*$_{NOT}$*-merged-bj-learn* S S'⟩ |
cdcl$_{NOT}$*-merged-bj-learn-backjump-l*: ⟨*backjump-l* S S' $\Longrightarrow$ *cdcl*$_{NOT}$*-merged-bj-learn* S S'⟩ |
cdcl$_{NOT}$*-merged-bj-learn-forget*$_{NOT}$: ⟨*forget*$_{NOT}$ S S' $\Longrightarrow$ *cdcl*$_{NOT}$*-merged-bj-learn* S S'⟩

lemma *cdcl*$_{NOT}$*-merged-bj-learn-no-dup-inv*:
  ⟨*cdcl*$_{NOT}$*-merged-bj-learn* S T $\Longrightarrow$ *no-dup* (*trail* S) $\Longrightarrow$ *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 dest*: *no-dup-appendD*)[]
  using *forget*$_{NOT}$*.simps* **apply** (*auto*; *fail*)
  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*$_{NOT}$
    decide-conds propagate-conds forget-conds
    ⟨$\lambda C\ C'\ L'\ S\ T.$ *backjump-l-cond* C C' L' S T
    $\wedge$ *distinct-mset* C' $\wedge$ $L' \notin\#$ C' $\wedge$ $\neg$*tautology* (*add-mset* L' C')⟩
  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**
    decide-conds :: ⟨$'st \Rightarrow 'st \Rightarrow bool$⟩ **and**
    propagate-conds :: ⟨$('v, unit)$ *ann-lit* $\Rightarrow 'st \Rightarrow 'st \Rightarrow bool$⟩ **and**
    forget-conds :: ⟨$'v$ *clause* $\Rightarrow 'st \Rightarrow bool$⟩ **and**
    backjump-l-cond :: ⟨$'v$ *clause* $\Rightarrow 'v$ *clause* $\Rightarrow 'v$ *literal* $\Rightarrow 'st \Rightarrow 'st \Rightarrow bool$⟩ +
  fixes
    inv :: ⟨$'st \Rightarrow bool$⟩
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' $\wedge$ $L' \notin\#$ C' $\wedge$ $\neg$*tautology* (*add-mset* L' C')⟩

sublocale *backjumping-ops trail clauses*$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$
  backjump-conds
  by *standard*

end

locale *cdcl*$_{NOT}$*-merge-bj-learn* =
  cdcl$_{NOT}$*-merge-bj-learn-proxy trail clauses*$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$
    decide-conds propagate-conds forget-conds backjump-l-cond inv
  for
    trail :: ⟨$'st \Rightarrow ('v, unit)$ *ann-lits*⟩ **and**
    clauses$_{NOT}$ :: ⟨$'st \Rightarrow 'v$ *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**
decide-conds :: ‹'st ⇒ 'st ⇒ bool› **and**
propagate-conds :: ‹('v, unit) ann-lit ⇒ 'st ⇒ 'st ⇒ bool› **and**
forget-conds :: ‹'v clause ⇒ 'st ⇒ bool› **and**
backjump-l-cond :: ‹'v clause ⇒ 'v clause ⇒ 'v literal ⇒ 'st ⇒ 'st ⇒ bool› **and**
inv :: ‹'st ⇒ bool› +
assumes
bj-merge-can-jump:
‹⋀S C F' K F L.
inv S
⟹ *trail S* = *F'* @ *Decided K* # *F*
⟹ *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* (*F'* @ *Decided K* # *F*))
⟹ *clauses*$_{NOT}$ *S* ⊨*pm add-mset L C'*
⟹ *F* ⊨*as CNot C'*
⟹ ¬*no-step backjump-l S*› **and**
cdcl-merged-inv: ‹⋀S T. *cdcl*$_{NOT}$*-merged-bj-learn S T* ⟹ *inv S* ⟹ *inv T*› **and**
can-propagate-or-decide-or-backjump-l:
‹*atm-of L* ∈ *atms-of-mm* (*clauses*$_{NOT}$ *S*) ⟹
undefined-lit (*trail S*) *L* ⟹
inv S ⟹
satisfiable (*set-mset* (*clauses*$_{NOT}$ *S*)) ⟹
∃ *T*. *decide*$_{NOT}$ *S T* ∨ *propagate*$_{NOT}$ *S T* ∨ *backjump-l S T*›
begin

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 *tautology-single-add*:
‹*tautology* (*L* + {#*a*#}) ⟷ *tautology L* ∨ −*a* ∈# *L*›
unfolding *tautology-decomp* **by** (*cases a*) *auto*

lemma *backjump-l-implies-exists-backjump*:
assumes *bj*: ‹*backjump-l S T*› **and** ‹*inv S*› **and** *n-d*: ‹*no-dup* (*trail S*)›
shows ‹∃ *U*. *backjump S U*›
proof −
obtain *C F' K F L C'* **where**
tr: ‹*trail S* = *F'* @ *Decided K* # *F*› **and**
C: ‹*C* ∈# *clauses*$_{NOT}$ *S*› **and**
T: ‹*T* ∼ *prepend-trail* (*Propagated L* ()) (*reduce-trail-to*$_{NOT}$ *F* (*add-cls*$_{NOT}$ (*add-mset L C'*) *S*))›
and
tr-C: ‹*trail S* ⊨*as CNot C*› **and**
undef: ‹*undefined-lit F L*› **and**
L: ‹*atm-of L* ∈ *atms-of-mm* (*clauses*$_{NOT}$ *S*) ∪ *atm-of* ‘ (*lits-of-l* (*trail S*))› **and**
S-C-L: ‹*clauses*$_{NOT}$ *S* ⊨*pm add-mset L C'*› **and**
F-C': ‹*F* ⊨*as CNot C'*› **and**
cond: ‹*backjump-l-cond C C' L S T*› **and**

dist: ‹*distinct-mset* (*add-mset* L C')› **and**
taut: ‹¬ *tautology* (*add-mset* L C')›
using *bj* **by** (*elim backjump-lE*) *force*
have ‹$L \notin\#$ C'›
using *dist* **by** *auto*
show *?thesis*
using *backjump.intros*[*OF tr - C tr-C undef L S-C-L F-C'*] *cond dist taut*
by *auto*
qed

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-cls*$_{NOT}$ *remove-cls*$_{NOT}$ *inv decide-conds backjump-conds propagate-conds*
proof (*unfold-locales*, *goal-cases*)
case *1*
{ fix S S'
assume *bj*: ‹*backjump-l* S S'›
then obtain F' K F L C' C D **where**
S': ‹$S' \sim$ *prepend-trail* (*Propagated* L ()) (*reduce-trail-to*$_{NOT}$ F (*add-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 ⊨*as CNot* C› **and**
undef-L: ‹*undefined-lit* F L› **and**
atm-L:
‹*atm-of* L ∈ *insert* (*atm-of* K) (*atms-of-mm* (*clauses*$_{NOT}$ S) ∪ *atm-of* ‘ (*lits-of-l* F' ∪ *lits-of-l* F))›
and
cls-S-C': ‹*clauses*$_{NOT}$ S ⊨*pm add-mset* L C'› **and**
F-C': ‹F ⊨*as CNot* C'› **and**
dist: ‹*distinct-mset* (*add-mset* L C')› **and**
not-tauto: ‹¬ *tautology* (*add-mset* L C')› **and**
cond: ‹*backjump-l-cond* C C' L S S'›
‹D = *add-mset* L C'›
by (*elim backjump-lE*) *simp*
interpret *backjumping-ops trail clauses*$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$ *backjump-conds*
by *unfold-locales*
have ‹∃ 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* **by** *simp*
}
then show *?case* **using** *1 bj-merge-can-jump* **by** *meson*
next
case *2*
then show *?case*
using *can-propagate-or-decide-or-backjump-l backjump-l-implies-exists-backjump* **by** *blast*
qed

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

lemma *backjump-l-learn-backjump*:
assumes *bt*: ⟨*backjump-l S T*⟩ **and** *inv*: ⟨*inv S*⟩
shows ⟨∃ *C′ L D*. *learn S* (*add-cls*$_{NOT}$ *D S*)
∧ *D* = *add-mset L C′*
∧ *backjump* (*add-cls*$_{NOT}$ *D S*) *T*
∧ *atms-of* (*add-mset L C′*) ⊆ *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-trail-to*$_{NOT}$ *F* (*add-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* = *add-mset L C′*⟩
⟨*F* ⊨*as CNot C′*⟩ **and**
distinct: ⟨*distinct-mset D*⟩ **and**
not-tauto: ⟨¬ *tautology D*⟩ **and**
cond: ⟨*backjump-l-cond C C′ L S T*⟩
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* (*add-mset L C′*) ⊆ *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-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-cls*$_{NOT}$ *D S*) *T*⟩
apply (*rule backjump.intros*[*of - - - - - L C C′*])
using ⟨*F* ⊨*as CNot C′*⟩ *C-cls-S tr-S-CNot-C undef T distinct not-tauto D cond*
by (*auto simp*: *tr-S state-eq*$_{NOT}$*-def simp del*: *state-simp*$_{NOT}$)
ultimately show *?thesis* **using** *D* **by** *blast*
qed

lemma *backjump-l-backjump-learn*:
assumes *bt*: ⟨*backjump-l S T*⟩ **and** *inv*: ⟨*inv S*⟩
shows ⟨∃ *C′ L D S′*. *backjump S S′*
∧ *learn S′ T*
∧ *D* = (*add-mset L C′*)
∧ *T* ∼ *add-cls*$_{NOT}$ *D S′*
∧ *atms-of* (*add-mset L C′*) ⊆ *atms-of-mm* (*clauses*$_{NOT}$ *S*) ∪ *atm-of* ‘ (*lits-of-l* (*trail S*))
∧ *clauses*$_{NOT}$ *S* ⊨*pm D*⟩

```
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-trail-to_NOT F (add-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 = add-mset L C'›
    ‹F ⊨as CNot C'› and
    distinct: ‹distinct-mset D› and
    not-tauto: ‹¬ tautology D› and
    cond: ‹backjump-l-cond C C' L S T›
    using bt inv by (elim backjump-lE) simp
  let ?S' = ‹prepend-trail (Propagated L ()) (reduce-trail-to_NOT F S)›
  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 (add-mset L C') ⊆ 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' T›
    apply (rule learn.intros)
        using clss-C apply auto[]
      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)
     using T apply (auto simp: tr-S state-eq_NOT-def simp del: state-simp_NOT)
    done
  moreover have bj: ‹backjump S (prepend-trail (Propagated L ()) (reduce-trail-to_NOT F S))›
    apply (rule backjump.intros[of S F' K F - L])
    using ‹F ⊨as CNot C'› C-cls-S tr-S-CNot-C undef T distinct not-tauto D cond clss-C atm-L
    by (auto simp: tr-S)
  moreover have ‹T ∼ (add-cls_NOT D ?S')›
    using T by (auto simp: tr-S state-eq_NOT-def simp del: state-simp_NOT)
  ultimately show ?thesis
    using D clss-C by blast
qed

lemma cdcl_NOT-merged-bj-learn-is-tranclp-cdcl_NOT:
  ‹cdcl_NOT-merged-bj-learn S T ⟹ inv S ⟹ 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*)
  obtain C' :: ⟨*'v clause*⟩ **and** *L* :: ⟨*'v literal*⟩ **and** *D* :: ⟨*'v clause*⟩ **where**
    f3: ⟨*learn S* (*add-cls*$_{NOT}$ *D S*) ∧
      backjump (*add-cls*$_{NOT}$ *D S*) *T* ∧
      atms-of (*add-mset L C'*) ⊆ *atms-of-mm* (*clauses*$_{NOT}$ *S*) ∪ *atm-of* ' *lits-of-l* (*trail S*)⟩ **and**
    D: ⟨*D* = *add-mset L C'*⟩
    using *backjump-l-learn-backjump*[*OF bt inv*] **by** *blast*
  then have *f4*: ⟨$cdcl_{NOT}$ *S* (*add-cls*$_{NOT}$ *D S*)⟩
    using *c-learn* **by** *blast*
  have ⟨$cdcl_{NOT}$ (*add-cls*$_{NOT}$ *D S*) *T*⟩
    using *f3 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* ⟹ *inv S* ⟹ $cdcl_{NOT}{}^{**}$ *S T* ∧ *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*)
  have ⟨$cdcl_{NOT}{}^{**}$ *T U*⟩
    using $cdcl_{NOT}$*-merged-bj-learn-is-tranclp-*$cdcl_{NOT}$[*OF* $cdcl_{NOT}$] *IH*
     inv **by** *auto*
  then have ⟨$cdcl_{NOT}{}^{**}$ *S U*⟩ **using** *IH* **by** *fastforce*
  moreover have ⟨*inv U*⟩ **using** *IH* $cdcl_{NOT}$ *cdcl-merged-inv 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* ⟹ *inv S* ⟹ $cdcl_{NOT}{}^{**}$ *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* ⟹ *inv S* ⟹ *inv T*⟩
  using *rtranclp-*$cdcl_{NOT}$*-merged-bj-learn-is-rtranclp-*$cdcl_{NOT}$*-and-inv* **by** *blast*

lemma *rtranclp-*$cdcl_{NOT}$*-merged-bj-learn-no-dup-inv*:
  ⟨$cdcl_{NOT}$*-merged-bj-learn*** *S T* ⟹ *no-dup* (*trail S*) ⟹ *no-dup* (*trail T*)⟩
  by (*induction rule*: *rtranclp-induct*) (*auto simp*: $cdcl_{NOT}$*-merged-bj-learn-no-dup-inv*)

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 $\mu_{CDCL}{}'$*-merged* :: ⟨*'v clause set* ⇒ *'st* ⇒ *nat*⟩ **where**
⟨$\mu_{CDCL}{}'$*-merged A T* ≡
  ((*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*) ⊆ *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: ‹*finite A*›
shows ‹μ_{CDCL}'*-merged A T* < μ_{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 μ_{CDCL}'*-merged-def* μ_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 μ_{CDCL}'*-merged-def* μ_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 μ_{CDCL}'*-merged-def* μ_C'*-def* **by** *simp*
next
case (*cdcl*$_{NOT}$*-merged-bj-learn-backjump-l T*) **note** *bj-l* = *this*(*1*)
obtain *C' L D S'* **where**
learn: ‹*learn S' T*› **and**
bj: ‹*backjump S S'*› **and**

atms-C: ‹*atms-of* (*add-mset L C'*) ⊆ *atms-of-mm* (*clauses*$_{NOT}$ *S*) ∪ *atm-of* ‘ (*lits-of-l* (*trail S*))›
and
D: ‹*D* = *add-mset L C'*› **and**
T: ‹*T* ∼ *add-cls*$_{NOT}$ *D S'*›
using *bj-l inv backjump-l-backjump-learn* [*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 *tr-S-T*: ‹*trail-weight S'* = *trail-weight T*›
using *T* **by** *auto*
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 S'*))
< ((*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 *bj bj-backjump* **apply** *blast*
using *inv* **apply** *blast*
using *atms-C atm-clss atm-trail D* **apply** (*simp add*: *n-d*; *fail*)
using *atm-trail n-d* **apply** (*simp*; *fail*)
apply (*simp add*: *n-d*; *fail*)
using *fin-A* **apply** (*simp*; *fail*)
done
then show *?case*
using *card-T-S* **unfolding** μ_{CDCL}*'-merged-def* μ_C*'-def tr-S-T* **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 - -* ‹μ_{CDCL}*'-merged A*›])
using *cdcl*$_{NOT}$*-decreasing-measure' fin-A* **by** *simp*

lemma *in-atms-neg-defined*: ‹*x* ∈ *atms-of C'* ⟹ *F* ⊨*as CNot C'* ⟹ *x* ∈ *atm-of* ‘ *lits-of-l F*›
by (*metis* (*no-types*, *lifting*) *atms-of-def imageE true-annots-CNot-all-atms-defined*)

lemma *cdcl*$_{NOT}$*-merged-bj-learn-atms-of-ms-clauses-decreasing*:
assumes ‹*cdcl*$_{NOT}$*-merged-bj-learn S T*›**and** ‹*inv S*›
shows ‹*atms-of-mm* (*clauses*$_{NOT}$ *T*) ⊆ *atms-of-mm* (*clauses*$_{NOT}$ *S*) ∪ *atm-of* ‘ (*lits-of-l* (*trail S*))›
using *assms*
apply (*induction rule*: *cdcl*$_{NOT}$*-merged-bj-learn.induct*)
prefer *4* **apply** (*auto dest*!: *dpll-bj-atms-of-ms-clauses-inv set-mp*
simp add: *atms-of-ms-def Union-eq*
elim!: *decide*$_{NOT}$*E propagate*$_{NOT}$*E forget*$_{NOT}$*E*)[*3*]
apply (*elim backjump-lE*)
by (*auto dest*!: *in-atms-neg-defined simp del*:)

lemma *cdcl*$_{NOT}$*-merged-bj-learn-atms-in-trail-in-set*:
assumes
‹*cdcl*$_{NOT}$*-merged-bj-learn 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*
apply (*induction rule*: $cdcl_{NOT}$*-merged-bj-learn.induct*)
apply (*meson bj-decide*$_{NOT}$ *dpll-bj-atms-in-trail-in-set*)
apply (*meson bj-propagate*$_{NOT}$ *dpll-bj-atms-in-trail-in-set*)
defer
apply (*metis forget*$_{NOT}$*E state-eq*$_{NOT}$*-trail trail-remove-cls*$_{NOT}$)
by (*metis* (*no-types*, *lifting*) *backjump-l-backjump-learn bj-backjump dpll-bj-atms-in-trail-in-set state-eq*$_{NOT}$*-trail trail-add-cls*$_{NOT}$)

lemma *rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-trail-clauses-bound*:
assumes
cdcl: ‹$cdcl_{NOT}$*-merged-bj-learn*** *S T*› **and**
inv: ‹*inv S*› **and**
atms-clauses-S: ‹*atms-of-mm* (*clauses*$_{NOT}$ *S*) ⊆ *A*› **and**
atms-trail-S: ‹*atm-of* ‘(*lits-of-l* (*trail S*)) ⊆ *A*›
shows ‹*atm-of* ‘ (*lits-of-l* (*trail T*)) ⊆ *A* ∧ *atms-of-mm* (*clauses*$_{NOT}$ *T*) ⊆ *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*$_{NOT}$*-merged-bj-learn-is-rtranclp-cdcl*$_{NOT}$*-and-inv* **by** *blast*
then have ‹*atms-of-mm* (*clauses*$_{NOT}$ *U*) ⊆ *A*›
using *cdcl*$_{NOT}$*-merged-bj-learn-atms-of-ms-clauses-decreasing cdcl*$_{NOT}$ *IH* ‹*inv T*› **by** *fast*
moreover
have ‹*atm-of* ‘(*lits-of-l* (*trail U*)) ⊆ *A*›
using *cdcl*$_{NOT}$*-merged-bj-learn-atms-in-trail-in-set*[*of - - A*] ‹*inv T*› *cdcl*$_{NOT}$ *step.IH* **by** *auto*
ultimately show *?case* **by** *fast*
qed

lemma *cdcl*$_{NOT}$*-merged-bj-learn-trail-clauses-bound*:
assumes
cdcl: ‹*cdcl*$_{NOT}$*-merged-bj-learn S T*› **and**
inv: ‹*inv S*› **and**
atms-clauses-S: ‹*atms-of-mm* (*clauses*$_{NOT}$ *S*) ⊆ *A*› **and**
atms-trail-S: ‹*atm-of* ‘(*lits-of-l* (*trail S*)) ⊆ *A*›
shows ‹*atm-of* ‘ (*lits-of-l* (*trail T*)) ⊆ *A* ∧ *atms-of-mm* (*clauses*$_{NOT}$ *T*) ⊆ *A*›
using *rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-trail-clauses-bound*[*of S T*] *assms* **by** *auto*

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 *st*: ‹$cdcl_{NOT}$*-merged-bj-learn*** *S T*›
using [[*simp-trace*]]
by (*simp add*: *rtranclp-unfold st*)
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-*$cdcl_{NOT}$*-merged-bj-learn-trail-clauses-bound*[*OF st inv atm-clss atm-trail*]
by *fast*
moreover have ‹*atm-of* ‘ (*lits-of-l* (*trail T*))⊆ *atms-of-ms A*›
using *rtranclp-*$cdcl_{NOT}$*-merged-bj-learn-trail-clauses-bound*[*OF st inv atm-clss atm-trail*]
by *fast*
moreover have ‹*no-dup* (*trail T*)›
using *rtranclp-*$cdcl_{NOT}$*-merged-bj-learn-no-dup-inv*[*OF st 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 $cdcl_{NOT}$*-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))
        then show False
          using can-propagate-or-decide-or-backjump-l[of ‹Pos l› S] l-N
          cdcl_NOT-merged-bj-learn-decide_NOT n-s inv sat
          by (auto dest!: cdcl_NOT-merged-bj-learn.intros)
      qed

    have ‹?M ⊨as CNot C›
    apply (rule all-variables-defined-not-imply-cnot)
      using atms-N-M ‹C ∈ ?N› ‹¬ ?M ⊨a C› atms-of-atms-of-ms-mono[OF ‹C ∈ ?N›]
      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** *d* :: *unit* **and**
F F′ :: ‹(′v, unit) ann-lits› **where**
M-K: ‹?M = F′ @ Decided K # F› **and**
nm: ‹∀f∈set F′. ¬is-decided f›
by (*metis* (*full-types*) *is-decided-ex-Decided 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 (λL::′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-decided L ∧ L ∈ set ?M}›
unfolding *M-K* **apply** *standard*
apply *force*
by *auto*
ultimately have *N-C-M*: ‹?N ∪ ?C′ ⊨ps unmark-l ?M›
by *auto*
have *N-M-False*: ‹?N ∪ (λL. unmark L) ‘ (set ?M) ⊨ps {{#}}›
unfolding *true-clss-clss-def true-annots-def Ball-def true-annot-def*
proof (*intro allI impI*)
fix *LL* :: *′v literal set*
assume
tot: ‹total-over-m LL (set-mset ($clauses_{NOT}$ S) ∪ unmark-l (trail S) ∪ {{#}})› **and**
cons: ‹consistent-interp LL› **and**
LL: ‹LL ⊨s set-mset ($clauses_{NOT}$ S) ∪ unmark-l (trail S)›
have ‹total-over-m LL (CNot C)›
by (*metis* ‹C ∈# $clauses_{NOT}$ S› *insert-absorb tot total-over-m-CNot-toal-over-m total-over-m-insert total-over-m-union*)
then have *total-over-m LL* (*unmark-l* (*trail S*) ∪ *CNot C*)
using *tot* **by** *force*
then show *LL* ⊨s {{#}}
using *tot cons LL*
by (*metis* (*no-types*) ‹C ∈# $clauses_{NOT}$ S› ‹trail S ⊨as CNot C› *consistent-CNot-not true-annots-true-clss-clss true-clss-clss-def true-clss-def true-clss-union*)
qed

have ‹undefined-lit F K› **using** ‹no-dup ?M› **unfolding** *M-K* **by** (*auto simp*: *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* $\models$*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* $\models$*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* $\models$*s ?C′*›
using ‹*?N* ∪ *?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-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* $\models$*as CNot* (*image-mset uminus ?C*)›
using *nm* **unfolding** *true-annots-def CNot-def M-K* **by** (*auto simp add*: *lits-of-def*)
ultimately have *False*
using *bj-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* $\models$*as CNot C*› *bj-backjump inv sat* **unfolding** *M-K*
by (*auto simp*: $cdcl_{NOT}$*-merged-bj-learn.simps*)
then show *?thesis* **by** *fast*
qed *auto*
qed

lemma $cdcl_{NOT}$*-merged-bj-learn-all-decomposition-implies*:

assumes ⟨$cdcl_{NOT}$-*merged-bj-learn S T*⟩ **and** *inv*: ⟨*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*
proof (*induction rule*: $cdcl_{NOT}$-*merged-bj-learn.induct*)
case ($cdcl_{NOT}$-*merged-bj-learn-backjump-l T*) **note** *bj-l* = *this*(*1*)
obtain *C′ L D S′* **where**
learn: ⟨*learn S′ T*⟩ **and**
bj: ⟨*backjump S S′*⟩ **and**
atms-C: ⟨*atms-of* (*add-mset L C′*) ⊆ *atms-of-mm* ($clauses_{NOT}$ *S*) ∪ *atm-of* ‘ (*lits-of-l* (*trail S*))⟩
and
D: ⟨*D* = *add-mset L C′*⟩ **and**
T: ⟨*T* ∼ $add\text{-}cls_{NOT}$ *D S′*⟩
using *bj-l inv backjump-l-backjump-learn* [*of S*] **by** *blast*
have ⟨*all-decomposition-implies-m* ($clauses_{NOT}$ *S′*) (*get-all-ann-decomposition* (*trail S′*))⟩
using *bj bj-backjump dpll-bj-clauses inv*(*1*) *inv*(*2*)
by (*fastforce simp*: *dpll-bj-all-decomposition-implies-inv*)
then show *?case*
using *T* **by** (*auto simp*: *all-decomposition-implies-insert-single*)
qed (*auto simp*: *dpll-bj-all-decomposition-implies-inv* $cdcl_{NOT}$-*all-decomposition-implies*
dest!: *dpll-bj.intros* $cdcl_{NOT}$.*intros*)

lemma *rtranclp*-$cdcl_{NOT}$-*merged-bj-learn-all-decomposition-implies*:
assumes ⟨$cdcl_{NOT}$-*merged-bj-learn*** *S T*⟩ **and** *inv*: ⟨*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*
apply (*induction rule*: *rtranclp-induct*)
apply *simp*
using $cdcl_{NOT}$-*merged-bj-learn-all-decomposition-implies*
rtranclp-$cdcl_{NOT}$-*merged-bj-learn-is-rtranclp*-$cdcl_{NOT}$-*and-inv* **by** *blast*

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*) ⊆ *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* ($clauses_{NOT}$ *T*))
∨ (*trail T* ⊨*asm* $clauses_{NOT}$ *T* ∧ *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*) ⊆ *atms-of-ms A*⟩ **and** ⟨*atm-of* ‘ *lits-of-l* (*trail T*) ⊆ *atms-of-ms A*⟩
using *rtranclp*-$cdcl_{NOT}$-*merged-bj-learn-trail-clauses-bound*[*OF st inv atms-S atms-trail*] **by** *blast+*
moreover have ⟨*no-dup* (*trail T*)⟩
using *rtranclp*-$cdcl_{NOT}$-*merged-bj-learn-no-dup-inv inv n-d st* **by** *blast*
moreover have ⟨*inv T*⟩

using *rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-inv inv st* **by** *blast*
moreover have ‹*all-decomposition-implies-m* (*clauses*$_{NOT}$ *T*) (*get-all-ann-decomposition* (*trail T*))›
using *rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-all-decomposition-implies inv st decomp n-d* **by** *blast*
ultimately show *?thesis*
using *cdcl*$_{NOT}$*-merged-bj-learn-final-state*[*of T A*] ‹*finite A*› *n-s* **by** *fast*
qed

end

2.2.7 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 decide-conds 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**
decide-conds :: ‹*'st* ⇒ *'st* ⇒ *bool*› **and**
backjump-conds :: ‹*'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool*› **and**
propagate-conds :: ‹(*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *'st* ⇒ *bool*› **and**
learn-restrictions forget-restrictions :: ‹*'v clause* ⇒ *'st* ⇒ *bool*›
+
fixes *f* :: ‹*nat* ⇒ *nat*›
assumes
unbounded: ‹*unbounded f*› **and** *f-ge-1*: ‹⋀*n*. *n* ≥ *1* ⟹ *f n* ≥ *1*› **and**
inv-restart:‹⋀*S T*. *inv S* ⟹ *T* ∼ *reduce-trail-to*$_{NOT}$ ([]::*'a list*) *S* ⟹ *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* ⟨λ*S T*. *T* ∼ *reduce-trail-to*$_{NOT}$ ([]::′*a list*) *S*⟩ *cdcl*$_{NOT}$ *f*
⟨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*⟩
$\mu_{CDCL}{}'$ ⟨λ*S*. *inv S* ∧ *no-dup* (*trail S*)⟩
$\mu_{CDCL}{}'$*-bound*
apply *unfold-locales*
apply (*simp add*: *unbounded*)
using *f-ge-1* **apply** *force*
using *bound-inv-inv* **apply** *meson*
apply (*rule cdcl*$_{NOT}$*-decreasing-measure*′; *simp*)
apply (*rule rtranclp-cdcl*$_{NOT}$-$\mu_{CDCL}{}'$*-bound*; *simp*)
apply (*rule rtranclp-*$\mu_{CDCL}{}'$*-bound-decreasing*; *simp*)
apply *auto*[]
apply *auto*[]
using *cdcl*$_{NOT}$*-inv cdcl*$_{NOT}$*-no-dup* **apply** *blast*
using *inv-restart* **apply** *auto*[]
done

lemma *cdcl*$_{NOT}$*-with-restart-*$\mu_{CDCL}{}'$*-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 ⟨$\mu_{CDCL}{}'$ *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*) $\subseteq$ *atms-of-ms A*⟩
⟨*atm-of* ‘ *lits-of-l* (*trail T*) $\subseteq$ *atms-of-ms A*⟩
⟨*finite A*⟩
shows ⟨$\mu_{CDCL}{}'$*-bound A V* $\leq$ $\mu_{CDCL}{}'$*-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* $\leq$ $\mu_{CDCL}{}'$*-bound A S*⟩
apply (*rule rtranclp-*$\mu_{CDCL}{}'$*-bound-decreasing*)
using ⟨(*cdcl*$_{NOT}$ $\widehat{\ }\widehat{\ }$ *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* $\sim$ *reduce-trail-to*$_{NOT}$ ([]::'*a list*) *S*⟩
⟨λ*A 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*⟩
$\mu_{CDCL}{}'$ *cdcl*$_{NOT}$
⟨λ*S*. *inv S* $\wedge$ *no-dup* (*trail S*)⟩
$\mu_{CDCL}{}'$*-bound*
apply *unfold-locales*
using *cdcl*$_{NOT}$*-with-restart-*$\mu_{CDCL}{}'$*-le-*$\mu_{CDCL}{}'$*-bound* **apply** *simp*
using *cdcl*$_{NOT}$*-with-restart-*$\mu_{CDCL}{}'$*-bound-le-*$\mu_{CDCL}{}'$*-bound* **apply** *simp*
done

lemma *cdcl*$_{NOT}$*-restart-all-decomposition-implies*:
assumes ⟨*cdcl*$_{NOT}$*-restart S T*⟩ **and**
⟨*inv* (*fst S*)⟩ **and**
⟨*no-dup* (*trail* (*fst S*))⟩
⟨*all-decomposition-implies-m* (*clauses*$_{NOT}$ (*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* × *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*) ⊆ *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[*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))
∨ (*lits-of-l* (*trail* T) ⊨*sextm* $clauses_{NOT}$ S ∧ *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) ⊆ *atms-of-ms* A›
‹*atm-of* ' *lits-of-l* (*trail* T) ⊆ *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))
∨ (*trail* T ⊨*asm* $clauses_{NOT}$ T ∧ *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 ⊨*sextm* $clauses_{NOT}$ S ⟷ I ⊨*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 ⊨*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) ⊨*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 the locale $cdcl_{NOT}$*-with-backtrack-and-restarts*.

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* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$

decide-conds propagate-conds forget-conds
‹$\lambda C\ C'\ L'\ S\ T$. *distinct-mset* $C' \wedge L' \notin\#\ C' \wedge$ *backjump-l-cond* $C\ C'\ L'\ S\ T$› *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**
decide-conds :: ‹$'st \Rightarrow\ 'st \Rightarrow$ *bool*› **and**
propagate-conds :: ‹$('v,\ unit)$ *ann-lit* $\Rightarrow\ 'st \Rightarrow\ 'st \Rightarrow$ *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* :: ‹$'b$ *clause multiset* $\Rightarrow\ 'b$ *clauses*›
where
‹*not-simplified-cls* $A \equiv \{\#C \in\#\ A.\ C \notin$ *simple-clss* (*atms-of-mm* A)$\#\}$›

lemma *not-simplified-cls-tautology-distinct-mset*:
‹*not-simplified-cls* $A = \{\#C \in\#\ A.$ *tautology* $C \vee \neg$*distinct-mset* $C\#\}$›
unfolding *not-simplified-cls-def* **by** (*rule filter-mset-cong*) (*auto simp*: *simple-clss-def*)

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-tautology-distinct-mset* **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) ⊆ *atms-of-ms* A⟩ **and**
atms-trail: ⟨*atm-of* ‘(*lits-of-l* (*trail* S)) ⊆ *atms-of-ms* A⟩ **and**
fin-A[*simp*]: ⟨*finite* A⟩
shows ⟨*set-mset* ($clauses_{NOT}$ T) ⊆ *set-mset* (*not-simplified-cls* ($clauses_{NOT}$ S))
∪ *simple-clss* (*atms-of-ms* A)⟩
using *assms*(*1*−*4*)
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*)

have *st*: ⟨$cdcl_{NOT}$-*merged-bj-learn*** S T⟩
using *bj inv* $cdcl_{NOT}$-*merged-bj-learn.simps* **by** *blast*+
have ⟨*atm-of* ‘(*lits-of-l* (*trail* T)) ⊆ *atms-of-ms* A⟩ **and** ⟨*atms-of-mm* ($clauses_{NOT}$ T) ⊆ *atms-of-ms* A⟩
using *rtranclp-cdcl*$_{NOT}$-*merged-bj-learn-trail-clauses-bound*[*OF st*] *inv atms-trail atms-clss*
by *auto*

obtain F' K F L l C' C D **where**
tr-S: ⟨*trail* S = F' @ *Decided* K # F⟩ **and**
T: ⟨T ∼ *prepend-trail* (*Propagated* L l) (*reduce-trail-to*$_{NOT}$ F (*add-cls*$_{NOT}$ D S))⟩ **and**
⟨C ∈# $clauses_{NOT}$ S⟩ **and**
⟨*trail* S ⊨*as* *CNot* C⟩ **and**
undef: ⟨*undefined-lit* F L⟩ **and**
⟨$clauses_{NOT}$ S ⊨*pm* *add-mset* L C'⟩ **and**
⟨F ⊨*as* *CNot* C'⟩ **and**
D: ⟨D = *add-mset* L C'⟩ **and**
dist: ⟨*distinct-mset* (*add-mset* L C')⟩ **and**
tauto: ⟨¬ *tautology* (*add-mset* L C')⟩ **and**
⟨*backjump-l-cond* C C' L S T⟩
using ⟨*backjump-l* S T⟩ **apply** (*elim backjump-lE*) **by** *auto*

have ⟨*atms-of* C' ⊆ *atm-of* ‘ (*lits-of-l* F)⟩
using ⟨F ⊨*as* *CNot* C'⟩ **by** (*simp add*: *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set*
atms-of-def image-subset-iff in-CNot-implies-uminus(*2*))
then have ⟨*atms-of* (C'+{#L#}) ⊆ *atms-of-ms* A⟩
using T ⟨*atm-of* ‘ *lits-of-l* (*trail* T) ⊆ *atms-of-ms* A⟩ *tr-S undef* **by** *auto*
then have ⟨*simple-clss* (*atms-of* (*add-mset* L C')) ⊆ *simple-clss* (*atms-of-ms* A)⟩
apply − **by** (*rule simple-clss-mono*) (*simp-all*)
then have ⟨*add-mset* L C' ∈ *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* 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*) ⊆# *not-simplified-cls* (*clauses*$_{NOT}$ *S*)⟩
  using *assms* **apply** *induction*
  prefer *4*
  unfolding *not-simplified-cls-tautology-distinct-mset* **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*) ⊆# *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*) ⊆ *atms-of-ms A*⟩ **and**
    ⟨*atm-of* ‘(*lits-of-l* (*trail S*)) ⊆ *atms-of-ms A*⟩ **and**
    finite[*simp*]: ⟨*finite A*⟩
  shows ⟨*set-mset* (*clauses*$_{NOT}$ *T*) ⊆ *set-mset* (*not-simplified-cls* (*clauses*$_{NOT}$ *S*))
    ∪ *simple-clss* (*atms-of-ms A*)⟩
  using *assms*(*1*−*4*)
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*−*6*)] **and**
    inv = *this*(*4*) **and** *atms-clss-S* = *this*(*5*) **and** *atms-trail-S* = *this*(*6*)
  have *st*′: ⟨$cdcl_{NOT}$** *S T*⟩
    using *inv rtranclp-*$cdcl_{NOT}$*-merged-bj-learn-is-rtranclp-*$cdcl_{NOT}$*-and-inv st* **by** *blast*
  have ⟨*inv T*⟩
    using *inv rtranclp-*$cdcl_{NOT}$*-merged-bj-learn-inv st* **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-*$cdcl_{NOT}$*-merged-bj-learn-trail-clauses-bound*[*OF st*] *inv atms-clss-S*
        atms-trail-S **by** *blast*+
  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**
    finite: ‹*finite A*›
  shows ‹μ_{CDCL}'*-merged A T* ≤ μ_{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*)) − μ_C' *A T*) ∗ *2*
    ≤ (*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) ∗ *2*›
    by *auto*
  ultimately show *?thesis* **unfolding** μ_{CDCL}'*-merged-def* **by** *auto*
qed

sublocale *cdcl*$_{NOT}$*-increasing-restarts-ops* ‹λ*S T*. *T* ∼ *reduce-trail-to*$_{NOT}$ ([]::'*a list*) *S*›
   cdcl$_{NOT}$*-merged-bj-learn f*
   ‹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*
   ‹λ*S*. *inv S* ∧ *no-dup* (*trail S*)›
   μ_{CDCL}'*-bound*
   apply *unfold-locales*
             using *unbounded* **apply** *simp*
            using *f-ge-1* **apply** *force*
           using *cdcl*$_{NOT}$*-merged-bj-learn-trail-clauses-bound* **apply** *meson*
          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*)) ⊆ *atms-of-ms A*› **and**
    ‹*atm-of* ‘ *lits-of-l* (*trail* (*fst T*)) ⊆ *atms-of-ms A*› **and**

⟨*finite A*⟩
shows ⟨μ_{CDCL}'*-merged A* (*fst V*) ≤ 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*) ⊆ *atms-of-ms A*⟩
using *rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-trail-clauses-bound*[*OF st'*] *inv atms-clss atms-trail n-d*
by *simp*
then have ⟨*atms-of-mm* (*clauses*$_{NOT}$ *U*) ⊆ *atms-of-ms A*⟩
using *U* **by** *simp*
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*

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 *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*))

$\cup$ *simple-clss* (*atms-of-ms* A))
$\leq$ *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)) $\leq$ *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))
$\leq$ *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) $\leq$ *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*
‹λS T. T ∼ *reduce-trail-to$_{NOT}$* ([]::'*a list*) S›
‹λA 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
‹λS. *inv* S ∧ *no-dup* (*trail* S)›
‹λA 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-*$\mu_{CDCL}{}'$*-merged-le-*$\mu_{CDCL}{}'$*-bound* **apply** *force*
using $cdcl_{NOT}$*-restart-*$\mu_{CDCL}{}'$*-bound-le-*$\mu_{CDCL}{}'$*-bound* **by** *fastforce*

lemma *true-clss-ext-decrease-right-insert*: ‹*I* ⊨*sext insert C* (*set-mset M*) ⟹ *I* ⊨*sextm M*›
by (*metis Diff-insert-absorb insert-absorb true-clss-ext-decrease-right-remove-r*)

lemma *true-clss-ext-decrease-add-implied*:
assumes ‹*M* ⊨*pm C*›
shows ‹*I*⊨*sext insert C* (*set-mset M*) ⟷ *I*⊨*sextm M*›
proof −
{ **fix** *J*
assume
‹*I* ⊨*sextm M*› **and**
‹*I* ⊆ *J*› **and**
tot: ‹*total-over-m J* (*set-mset* ({#*C*#} + *M*))› **and**
cons: ‹*consistent-interp J*›
then have ‹*J* ⊨*sm M*› **unfolding** *true-clss-ext-def* **by** *auto*

moreover
with ‹*M* ⊨*pm C*› **have** ‹*J* ⊨ *C*›
using *tot cons* **unfolding** *true-clss-cls-def* **by** *auto*
ultimately have ‹*J* ⊨*sm* {#*C*#} + *M*› **by** *auto*
}
then have *H*: ‹*I* ⊨*sextm M* ⟹ *I* ⊨*sext insert C* (*set-mset M*)›
unfolding *true-clss-ext-def* **by** *auto*
then show *?thesis*
by (*auto simp*: *true-clss-ext-decrease-right-insert*)
qed

lemma $cdcl_{NOT}$*-merged-bj-learn-bj-sat-ext-iff*:
assumes ‹$cdcl_{NOT}$*-merged-bj-learn S T*› **and** *inv*: ‹*inv S*›
shows ‹*I*⊨*sextm* $clauses_{NOT}$ *S* ⟷ *I*⊨*sextm* $clauses_{NOT}$ *T*›
using *assms*
proof (*induction rule*: $cdcl_{NOT}$*-merged-bj-learn.induct*)
case ($cdcl_{NOT}$*-merged-bj-learn-backjump-l T*) **note** *bj-l* = *this*(*1*)
obtain *C' L D S'* **where**
learn: ‹*learn S' T*› **and**
bj: ‹*backjump S S'*› **and**
atms-C: ‹*atms-of* (*add-mset L C'*) ⊆ *atms-of-mm* ($clauses_{NOT}$ *S*) ∪ *atm-of* ‘ (*lits-of-l* (*trail S*))›
and
D: ‹*D* = *add-mset L C'*› **and**
T: ‹*T* ∼ $add\text{-}cls_{NOT}$ *D S'*› **and**
clss-D: ‹$clauses_{NOT}$ *S* ⊨*pm D*›
using *bj-l inv backjump-l-backjump-learn* [*of S*] **by** *blast*
have [*simp*]: ‹$clauses_{NOT}$ *S'* = $clauses_{NOT}$ *S*›
using *bj* **by** (*auto elim*: *backjumpE*)
have ‹(*I* ⊨*sextm* $clauses_{NOT}$ *S*) ⟷ (*I* ⊨*sextm* $clauses_{NOT}$ *S'*)›
using *bj bj-backjump dpll-bj-clauses inv* **by** *fastforce*
then show *?case*
using *clss-D T* **by** (*auto simp*: *true-clss-ext-decrease-add-implied*)
qed (*auto simp*: $cdcl_{NOT}$*-bj-sat-ext-iff*
dest!: *dpll-bj.intros* $cdcl_{NOT}$*.intros*)

lemma *rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-bj-sat-ext-iff*:
  assumes ‹*cdcl*$_{NOT}$*-merged-bj-learn*** *S T*›**and** ‹*inv S*›
  shows ‹*I*⊨*sextm clauses*$_{NOT}$ *S* ⟷ *I*⊨*sextm clauses*$_{NOT}$ *T*›
  using *assms* **apply** (*induction rule*: *rtranclp-induct*)
    apply *simp*
  using *cdcl*$_{NOT}$*-merged-bj-learn-bj-sat-ext-iff*
    rtranclp-cdcl$_{NOT}$*-merged-bj-learn-is-rtranclp-cdcl*$_{NOT}$*-and-inv* **by** *blast*

lemma *cdcl*$_{NOT}$*-restart-eq-sat-iff*:
  assumes
    ‹*cdcl*$_{NOT}$*-restart S T*› **and**
    inv: ‹*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*$_{NOT}$*-merged-bj-learn-bj-sat-ext-iff restart-full.prems* **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*$_{NOT}$*-merged-bj-learn-bj-sat-ext-iff restart-step.prems* **by** *auto*
  moreover have ‹*I* ⊨*sextm clauses*$_{NOT}$ *T* ⟷ *I* ⊨*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*⊨*sextm clauses*$_{NOT}$ (*fst S*) ⟷ *I* ⊨*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*⊨*sextm clauses*$_{NOT}$ (*fst T*) ⟷ *I* ⊨*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*))

$(get\text{-}all\text{-}ann\text{-}decomposition\ (trail\ (fst\ T)))$›
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-*$cdcl_{NOT}$*-merged-bj-learn-all-decomposition-implies*[*OF - - 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*)) ⊨*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*)) ⊨*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*)) ⊨*sextm N* ∧ *satisfiable* (*set-mset N*)›
using *full-cdcl*$_{NOT}$*-restart-normal-form*[*of* ‹(*S*, *0*)› *T*] *assms* **by** *auto*

end — End of locale *cdcl*$_{NOT}$*-merge-bj-learn-with-backtrack-restarts*.

end
theory *CDCL-WNOT*
imports *CDCL-NOT 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 no-dup-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) = defined-lit S
  by *(auto simp: defined-lit-map image-comp intro!: ext)*

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

lemma *count-decided-conver-Trail-from-W[simp]*:
  ‹*count-decided (convert-trail-from-W M) = count-decided M*›
  unfolding *count-decided-def* **by** *(auto simp: comp-def)*

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 $state_W$ ⊆ *dpll-state-ops* **where**
trail = λ*S*. *convert-trail-from-W* (*trail S*) **and**
$clauses_{NOT}$ = *clauses* **and**
prepend-trail = λ*L S*. *cons-trail* (*convert-ann-lit-from-NOT L*) *S* **and**
tl-trail = λ*S*. *tl-trail S* **and**
$add\text{-}cls_{NOT}$ = λ*C S*. *add-learned-cls C S* **and**
$remove\text{-}cls_{NOT}$ = λ*C S*. *remove-cls C S*
by *unfold-locales*

sublocale $state_W$ ⊆ *dpll-state* **where**
trail = λ*S*. *convert-trail-from-W* (*trail S*) **and**
$clauses_{NOT}$ = *clauses* **and**
prepend-trail = λ*L S*. *cons-trail* (*convert-ann-lit-from-NOT L*) *S* **and**
tl-trail = λ*S*. *tl-trail S* **and**
$add\text{-}cls_{NOT}$ = λ*C S*. *add-learned-cls C S* **and**
$remove\text{-}cls_{NOT}$ = λ*C S*. *remove-cls C S*
by *unfold-locales* (*auto simp*: *map-tl o-def*)

context $state_W$
begin
declare $state\text{-}simp_{NOT}$[*simp del*]
end

2.3.2 Inclusion of Weidendenbch's CDCL without Strategy

sublocale *conflict-driven-clause-learning*$_W$ $\subseteq$ *cdcl*$_{NOT}$*-merge-bj-learn-ops* **where**
trail = λS. *convert-trail-from-W* (*trail* S) **and**
clauses$_{NOT}$ = *clauses* **and**
prepend-trail = $\lambda L\ S$. *cons-trail* (*convert-ann-lit-from-NOT* L) S **and**
tl-trail = λS. *tl-trail* S **and**
add-cls$_{NOT}$ = $\lambda C\ S$. *add-learned-cls* $C\ S$ **and**
remove-cls$_{NOT}$ = $\lambda C\ S$. *remove-cls* $C\ S$ **and**
decide-conds = λ- -. *True* **and**
propagate-conds = λ- - -. *True* **and**
forget-conds = λ- S. *conflicting* S = *None* **and**
backjump-l-cond = $\lambda C\ C'\ L'\ S\ T$. *backjump-l-cond* $C\ C'\ L'\ S\ T$
$\wedge$ *distinct-mset* $C' \wedge L' \notin\# C' \wedge \neg$*tautology* (*add-mset* $L'\ C'$)
by *unfold-locales*

sublocale *conflict-driven-clause-learning*$_W$ $\subseteq$ *cdcl*$_{NOT}$*-merge-bj-learn-proxy* **where**
trail = λS. *convert-trail-from-W* (*trail* S) **and**
clauses$_{NOT}$ = *clauses* **and**
prepend-trail = $\lambda L\ S$. *cons-trail* (*convert-ann-lit-from-NOT* L) S **and**
tl-trail = λS. *tl-trail* S **and**
add-cls$_{NOT}$ = $\lambda C\ S$. *add-learned-cls* $C\ S$ **and**
remove-cls$_{NOT}$ = $\lambda C\ S$. *remove-cls* $C\ S$ **and**
decide-conds = λ- -. *True* **and**
propagate-conds = λ- - -. *True* **and**
forget-conds = λ- S. *conflicting* S = *None* **and**
backjump-l-cond = *backjump-l-cond* **and**
inv = *inv*$_{NOT}$
by *unfold-locales*

sublocale *conflict-driven-clause-learning*$_W$ $\subseteq$ *cdcl*$_{NOT}$*-merge-bj-learn* **where**
trail = λS. *convert-trail-from-W* (*trail* S) **and**
clauses$_{NOT}$ = *clauses* **and**
prepend-trail = $\lambda L\ S$. *cons-trail* (*convert-ann-lit-from-NOT* L) S **and**
tl-trail = λS. *tl-trail* S **and**
add-cls$_{NOT}$ = $\lambda C\ S$. *add-learned-cls* $C\ S$ **and**
remove-cls$_{NOT}$ = $\lambda C\ S$. *remove-cls* $C\ S$ **and**
decide-conds = λ- -. *True* **and**
propagate-conds = λ- - -. *True* **and**
forget-conds = λ- S. *conflicting* S = *None* **and**
backjump-l-cond = *backjump-l-cond* **and**
inv = *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* $C'\ S\ C\ F'\ K\ F\ L$)
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 *dist*: *distinct-mset ?C'* $L \notin\# C'$
by *simp-all*

have *no-dup* F
using ⟨*inv*$_{NOT}$ S⟩ ⟨*convert-trail-from-W* (*trail* S) = F' @ *Decided* K # F⟩

unfolding inv_{NOT}*-def* **by** (*metis no-dup-appendD no-dup-cons 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* ⊨*as CNot C′*› *consistent-CNot-not-tautology true-annots-true-cls* **by** *blast*
then have *taut*: ¬ *tautology* (*add-mset L ?C′*)
using ‹*F* ⊨*as CNot C′*› ‹*undefined-lit F L*› **by** (*metis CNot-remdups-mset Decided-Propagated-in-iff-in-lits-of-l in-CNot-uminus tautology-add-mset tautology-remdups-mset true-annot-singleton true-annots-def*)

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* ⊨*pm add-mset L ?C′*
by (*metis 1*(*7*) *dist*(*2*) *remdups-mset-singleton-sum true-clss-cls-remdups-mset*)
have *F* ⊨*as CNot ?C′*
by (*simp add*: ‹*F* ⊨*as CNot C′*›)
have *Ex* (*backjump-l S*)
apply *standard*
apply (*rule backjump-l.intros*[*of - - - - - L add-mset L ?C′ - ?C′*])
using *f4 f3 f2* ‹¬ *tautology* (*add-mset L ?C′*)›
1 taut dist ‹*F* ⊨*as CNot* (*remdups-mset C′*)›
$state\text{-}eq_{NOT}$*-ref* **unfolding** *backjump-l-cond-def set-mset-remdups-mset* **by** *blast+*
then show *?case*
by *blast*
next
case (*3 L S*)
then show ∃ *T*. $decide_{NOT}$ *S T* ∨ $propagate_{NOT}$ *S T* ∨ *backjump-l S T*
using $decide_{NOT}$*.intros*[*of S L*] **by** *auto*
qed

context $conflict\text{-}driven\text{-}clause\text{-}learning_W$
begin

Notations are lost while proving locale inclusion:

notation $state\text{-}eq_{NOT}$ (**infix** $\sim_{NOT}$ *50*)

2.3.3 Additional Lemmas between NOT and W states

lemma $trail_W$*-eq-reduce-trail-to*$_{NOT}$*-eq*:
trail S = *trail T* ⟹ *trail* ($reduce\text{-}trail\text{-}to_{NOT}$ *F S*) = *trail* ($reduce\text{-}trail\text{-}to_{NOT}$ *F T*)
proof (*induction F S arbitrary*: *T rule*: $reduce\text{-}trail\text{-}to_{NOT}$*.induct*)
case (*1 F S T*) **note** *IH* = *this*(*1*) **and** *tr* = *this*(*2*)
then have [] = *convert-trail-from-W* (*trail S*)
∨ *length F* = *length* (*convert-trail-from-W* (*trail S*))
∨ *trail* ($reduce\text{-}trail\text{-}to_{NOT}$ *F* (*tl-trail S*)) = *trail* ($reduce\text{-}trail\text{-}to_{NOT}$ *F* (*tl-trail T*))
using *IH* **by** (*metis* (*no-types*) *trail-tl-trail*)
then show *trail* ($reduce\text{-}trail\text{-}to_{NOT}$ *F S*) = *trail* ($reduce\text{-}trail\text{-}to_{NOT}$ *F T*)
using *tr* **by** (*metis* (*no-types*) $reduce\text{-}trail\text{-}to_{NOT}$*.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
init-clss S = *init-clss* T
learned-clss S = *learned-clss* 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*
case *4* **show** *?case* **by** *simp*
case *5* **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*−)

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** (*cases* ‹*trail* T›) (*auto elim*!: *rulesE simp*: *skip-or-resolve.simps*)
case *1* **show** *?case*
using *s-o-r IH* **by** (*cases trail* T) (*auto elim*!: *rulesE simp*: *skip-or-resolve.simps*)
case *4* **show** *?case*
using *s-o-r IH'* **by** (*cases trail* T) (*auto elim*!: *rulesE simp*: *skip-or-resolve.simps*)
case *5* **show** *?case*
using *s-o-r IH'* **by** (*cases trail* T) (*auto elim*!: *rulesE simp*: *skip-or-resolve.simps*)
qed

2.3.4 Inclusion of Weidenbach's CDCL in NOT's CDCL

This lemma shows the inclusion of Weidenbach's CDCL *cdcl*$_W$*-merge* (with merging) in NOT's *cdcl*$_{NOT}$*-merged-bj-learn*.

lemma *cdcl*$_W$*-merge-is-cdcl*$_{NOT}$*-merged-bj-learn*:
assumes
inv: *cdcl*$_W$*-all-struct-inv* S **and**
cdcl$_W$*-restart*: *cdcl*$_W$*-merge* S T

shows *cdcl*$_{NOT}$*-merged-bj-learn S T*
∨ (*no-step cdcl*$_W$*-merge T* ∧ *conflicting T* ≠ *None*)
using *cdcl*$_W$*-restart inv*
proof *induction*
case (*fw-propagate S T*) **note** *propa* = *this*(*1*)
then obtain *M N U L C* **where**
H: *state-butlast S* = (*M*, *N*, *U*, *None*) **and**
CL: *C* + {#*L*#} ∈# *clauses S* **and**
M-C: *M* ⊨*as CNot C* **and**
undef: *undefined-lit* (*trail S*) *L* **and**
T: *state-butlast T* = (*Propagated L* (*C* + {#*L*#}) # *M*, *N*, *U*, *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 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*) *S*
by (*auto elim*: *decideE*)
have *decide*$_{NOT}$ *S T*
apply (*rule decide*$_{NOT}$*.decide*$_{NOT}$)
using *undef-L* **apply** (*simp*; *fail*)
using *atm-L inv* **apply** (*auto simp*: *cdcl*$_W$*-all-struct-inv-def no-strange-atm-def clauses-def*; *fail*)
using *T undef-L* **unfolding** *state-eq*$_{NOT}$*-def* **by** (*auto simp*: *clauses-def*)
then show *?case* **using** *cdcl*$_{NOT}$*-merged-bj-learn-decide*$_{NOT}$ **by** *blast*
next
case (*fw-forget S T*) **note** *rf* = *this*(*1*) **and** *inv* = *this*(*2*)
then obtain *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* **and**
S-C: ‹*removeAll-mset C* (*clauses S*) ⊨*pm C*›
by (*auto elim*: *forgetE*)
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*: *Un-Diff state-eq*$_{NOT}$*-def clauses-def ac-simps*)
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*
have *inv-T*: *cdcl*$_W$*-all-struct-inv T*

```
    using cdcl_W-restart.simps cdcl_W-all-struct-inv-inv confl inv by blast
  then have cdcl_W-M-level-inv T
    unfolding cdcl_W-all-struct-inv-def by auto
  then consider
    (no-bt) skip-or-resolve** T U |
    (bt) T' where skip-or-resolve** T T' and backtrack T' U
    using bj rtranclp-cdcl_W-bj-skip-or-resolve-backtrack unfolding full-def by meson
  then show ?case
  proof cases
    case no-bt
    then have conflicting U ≠ None
      using confl by (induction rule: rtranclp-induct)
      (auto simp: skip-or-resolve.simps 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-restart** T T'
      using s-or-r mono-rtranclp[of skip-or-resolve cdcl_W-restart]
        rtranclp-skip-or-resolve-rtranclp-cdcl_W-restart
      by blast
    then have cdcl_W-M-level-inv T'
      using rtranclp-cdcl_W-restart-consistent-inv ‹cdcl_W-M-level-inv T› by blast
    then obtain M1 M2 i D L K D' where
      confl-T': conflicting T' = Some (add-mset 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') (add-mset L D') and
      get-maximum-level (trail T') D' = i and
      U: U ∼ cons-trail (Propagated L (add-mset L D'))
               (reduce-trail-to M1
                    (add-learned-cls (add-mset L D')
                      (update-conflicting None T'))) and
      D-D': ‹D' ⊆# D› and
      T'-L-D': ‹clauses T' ⊨pm add-mset L D'›
      using bt by (auto elim: backtrackE)
    let ?D' = ‹add-mset L D'›
    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 cdcl_W-restart** T T'
      using rtranclp-skip-or-resolve-rtranclp-cdcl_W-restart s-or-r by blast
    have inv-T': cdcl_W-all-struct-inv T'
      using ‹cdcl_W-restart** 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-*$cdcl_W$*-restart 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 ‹$cdcl_W$*-restart*** *T T'*› $cdcl_W$*-restart-init-clss confl* $cdcl_W$*-all-struct-inv-def conflict inv* **by** (*metis rtranclp-*$cdcl_W$*-restart-init-clss*)
then have *atm-L*: *atm-of L* ∈ *atms-of-mm* (*clauses S*)
  using *inv-T' confl-T'* **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*
define *M''* **where** *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'* ‹*clauses S = clauses T*› ‹*clauses T = clauses T'*› *T'-L-D'*
  unfolding $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-learned-clause-alt-def clauses-def* **by** *auto*
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 D'*
  by (*subst tr-U*, *subst* (*asm*) *tr-U*) *fastforce*
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' - D'*])
          using *tr-T* **apply** (*simp*; *fail*)
         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*; *fail*)[]
        using C_S **apply** (*auto*; *fail*)[]
       using *tr-S-*C_S **apply** (*simp*; *fail*)

      using *undef-L* **apply** (*auto*; *fail*)[]
     using *atm-L* **apply** (*simp add*: *trail-reduce-trail-to*$_{NOT}$*-add-learned-cls*; *fail*)
    using ‹*init-clss T' + learned-clss S* ⊨*pm ?D'*› **unfolding** *clauses-def*
    apply (*simp*; *fail*)

apply (*simp*; *fail*)
apply (*metis U-D convert-trail-from-W-true-annots*)
using *inv-T′ inv-U U confl-T′ undef-L M1-M2* **unfolding** $cdcl_W$*-all-struct-inv-def*
distinct-$cdcl_W$*-state-def* **by** (*auto simp*: $cdcl_W$*-M-level-inv-decomp backjump-l-cond-def*
dest: *multi-member-split*)
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**
$cdcl_W$*-restart*:$cdcl_W$*-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 $cdcl_W$*-restart* **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-*$cdcl_W$*-restart*[*of S T*] *inv rtranclp-*$cdcl_W$*-restart-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* ∧ $cdcl_W$*-merge S T*}
apply (*rule wfP-if-measure*[*of - -* μ_{FW}])
using $cdcl_W$*-merge-*μ_{FW}*-decreasing* **by** *blast*

lemma *tranclp-*$cdcl_W$*-merge-*$cdcl_W$*-merge-trancl*:
{(*T*, *S*). $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$*-merge*$^{++}$ *S T*}
⊆ {(*T*, *S*). $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$*-merge S T*}$^+$
proof −
have (*T*, *S*) ∈ {(*T*, *S*). $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$*-merge S T*}$^+$
if *inv*: $cdcl_W$*-all-struct-inv S* **and** $cdcl_W$*-merge*$^{++}$ *S T*
for *S T* :: *'st*
using *that*(*2*)
proof (*induction rule*: *tranclp-induct*)
case *base*
then show *?case* **using** *inv* **by** *auto*
next
case (*step T U*) **note** *st* = *this*(*1*) **and** *s* = *this*(*2*) **and** *IH* = *this*(*3*)
have $cdcl_W$*-all-struct-inv T*
using *st* **by** (*meson inv rtranclp-*$cdcl_W$*-all-struct-inv-inv*
rtranclp-$cdcl_W$*-merge-rtranclp-*$cdcl_W$*-restart tranclp-into-rtranclp*)
then have (*U*, *T*) ∈ {(*T*, *S*). $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$*-merge S T*}$^+$
using *s* **by** *auto*
then show *?case* **using** *IH* **by** *auto*
qed
then show *?thesis* **by** *auto*
qed

lemma *wf-tranclp-*$cdcl_W$*-merge*: *wf* {(*T*, *S*). $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$*-merge*$^{++}$ *S T*}

apply (*rule wf-subset*)
  apply (*rule wf-trancl*)
  using *wf-cdcl*$_W$*-merge* **apply** *simp*
using *tranclp-cdcl*$_W$*-merge-cdcl*$_W$*-merge-trancl* **by** *simp*

lemma *wf-cdcl*$_W$*-bj-all-struct*: *wf* $\{(T, S).$ *cdcl*$_W$*-all-struct-inv* $S \wedge$ *cdcl*$_W$*-bj* $S\ T\}$
  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 add*: *cdcl*$_W$*-all-struct-inv-def*)

lemma *cdcl*$_W$*-conflicting-true-cdcl*$_W$*-merge-restart*:
  assumes *cdcl*$_W$ $S\ V$ **and** *confl*: *conflicting* S = *None*
  shows (*cdcl*$_W$*-merge* $S\ V \wedge$ *conflicting* V = *None*) $\vee$ (*conflicting* $V \neq$ *None* $\wedge$ *conflict* $S\ V$)
  using *assms*
proof (*induction rule*: *cdcl*$_W$*.induct*)
  case *W-propagate*
  then show *?case* **by** (*auto intro*: *cdcl*$_W$*-merge.intros elim*: *rulesE*)
next
  case (*W-conflict* S')
  then show *?case* **by** (*auto intro*: *cdcl*$_W$*-merge.intros elim*: *rulesE*)
next
  case *W-other*
  then show *?case*
  proof *cases*
    case *decide*
    then show *?thesis*
      by (*auto intro*: *cdcl*$_W$*-merge.intros elim*: *rulesE*)
  next
    case *bj*
    then show *?thesis*
      using *confl* **by** (*auto simp*: *cdcl*$_W$*-bj.simps elim*: *rulesE*)
  qed
qed

lemma *trancl-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*$^{++}$ $S\ V \wedge$ *conflicting* V = *None*)
    $\vee$ ($\exists\ T\ U$. *cdcl*$_W$*-merge*** $S\ T \wedge$ *conflicting* $V \neq$ *None* $\wedge$ *conflict* $T\ U \wedge$ *cdcl*$_W$*-bj*** $U\ V$)
  using *assms*
proof *induction*
  case *base*
  then show *?case* **using** *cdcl*$_W$*-conflicting-true-cdcl*$_W$*-merge-restart* **by** *blast*
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 *W-propagate*
    moreover have *conflicting* U = *None* **and** *conflicting* V = *None*
      using *W-propagate* **by** (*auto elim*: *propagateE*)
    ultimately show *?thesis* **using** *IH cdcl*$_W$*-merge.fw-propagate*[*of* $U\ V$] **by** *auto*
  next
    case *W-conflict*
    moreover have *confl-U*: *conflicting* U = *None* **and** *confl-V*: *conflicting* $V \neq$ *None*
      using *W-conflict* **by** (*auto elim*!: *conflictE*)

moreover have *cdcl*$_W$*-merge*** *S U*
using *IH confl-U* **by** *auto*
ultimately show *?thesis* **using** *IH* **by** *auto*
next
case *W-other*
then show *?thesis*
proof *cases*
case *decide*
then show *?thesis* **using** *IH cdcl*$_W$*-merge.fw-decide*[*of U V*] **by** (*auto elim*: *decideE*)
next
case *bj*
then consider
(*s-or-r*) *skip-or-resolve U V* |
(*bt*) *backtrack U V*
by (*auto simp*: *cdcl*$_W$*-bj.simps*)
then show *?thesis*
proof *cases*
case *s-or-r*
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*$^{++}$ *S U*
$\vee$ *cdcl*$_W$*-merge*** *S T* $\wedge$ *conflicting U* $\neq$ *None*
$\wedge$ *conflict T T*$'$ $\wedge$ *cdcl*$_W$*-bj*** *T*$'$ *U*
using *IH confl* **by** (*meson bj rtranclp.intros*(*1*)
rtranclp-cdcl$_W$*-merge-restart-no-step-cdcl*$_W$*-bj*
rtranclp-cdcl$_W$*-merge-tranclp-cdcl*$_W$*-merge-restart*)
have *conflicting V* $\neq$ *None* $\wedge$ *conflicting U* $\neq$ *None*
using ‹*skip-or-resolve U V*›
by (*auto simp*: *skip-or-resolve.simps elim*!: *skipE resolveE*)
then show *?thesis*
by (*metis* (*full-types*) *IH f1 rtranclp-trans tranclp-into-rtranclp*)
next
case *bt*
then have *conflicting U* $\neq$ *None* **by** (*auto elim*: *backtrackE*)
then obtain *T T*$'$ **where**
cdcl$_W$*-merge*** *S T* **and**
conflicting U $\neq$ *None* **and**
conflict T T$'$ **and**
cdcl$_W$*-bj*** *T*$'$ *U*
using *IH confl* **by** (*meson bj rtranclp.intros*(*1*)
rtranclp-cdcl$_W$*-merge-restart-no-step-cdcl*$_W$*-bj*
rtranclp-cdcl$_W$*-merge-tranclp-cdcl*$_W$*-merge-restart*)
have *invU*: *cdcl*$_W$*-M-level-inv U*
using *inv rtranclp-cdcl*$_W$*-restart-consistent-inv step.hyps*(*1*)
by (*meson* ‹*cdcl*$_W$*-bj*** *T*$'$ *U*› ‹*cdcl*$_W$*-merge*** *S T*› ‹*conflict T T*$'$›
cdcl$_W$*-restart-consistent-inv conflict rtranclp-cdcl*$_W$*-bj-rtranclp-cdcl*$_W$*-restart*
rtranclp-cdcl$_W$*-merge-rtranclp-cdcl*$_W$*-restart*)
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 show *?thesis*
using $cdcl_W$*-merge.fw-conflict*[*of T T′ V*] ‹*conflict T T′*›
‹$cdcl_W$*-merge*** *S T*› ‹*conflicting V = None*› **by** *auto*
qed
qed
qed
qed

lemma *wf-cdcl*$_W$: *wf* {(*T*, *S*). $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$ *S T*}
unfolding *wf-iff-no-infinite-down-chain*
proof *clarify*
fix *f* :: *nat* ⇒ ′*st*
assume ∀ *i*. (*f* (*Suc i*), *f i*) ∈ {(*T*, *S*). $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$ *S T*}
then have *f*: ⋀*i*. (*f* (*Suc i*), *f i*) ∈ {(*T*, *S*). $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$ *S T*}
by *blast*
{
fix *f* :: *nat* ⇒ ′*st*
assume
f: (*f* (*Suc i*), *f i*) ∈ {(*T*, *S*). $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$ *S T*} **and**
confl: *conflicting* (*f i*) ≠ *None* **for** *i*
have (*f* (*Suc i*), *f i*) ∈ {(*T*, *S*). $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$*-bj S T*} **for** *i*
using *f*[*of i*] *confl*[*of i*] **by** (*auto simp*: $cdcl_W$*.simps* $cdcl_W$*-o.simps* $cdcl_W$*-rf.simps*
elim!: *rulesE*)
then have *False*
using *wf-cdcl*$_W$*-bj-all-struct* **unfolding** *wf-iff-no-infinite-down-chain* **by** *blast*
} **note** *no-infinite-conflict* = *this*

have *st*: $cdcl_W{}^{++}$ (*f i*) (*f* (*Suc* (*i*+*j*))) **for** *i j* :: *nat*
proof (*induction j*)
case *0*
then show *?case* **using** *f* **by** *auto*
next
case (*Suc j*)
then show *?case* **using** *f* [*of i*+*j*+*1*] **by** *auto*
qed
have *st*: *i* < *j* ⟹ $cdcl_W{}^{++}$ (*f i*) (*f j*) **for** *i j* :: *nat*
using *st*[*of i j* − *i* − *1*] **by** *auto*

obtain i_b **where** i_b: *conflicting* (*f* i_b) = *None*
using *f no-infinite-conflict* **by** *blast*

define i_0 **where** i_0: i_0 = *Max* {i_0. ∀ *i* < i_0. *conflicting* (*f i*) ≠ *None*}
have *finite* {i_0. ∀ *i* < i_0. *conflicting* (*f i*) ≠ *None*}
proof −
have {i_0. ∀ *i* < i_0. *conflicting* (*f i*) ≠ *None*} ⊆ {*0*..i_b}
using i_b **by** (*metis* (*mono-tags*, *lifting*) *atLeast0AtMost atMost-iff mem-Collect-eq not-le*
subsetI)
then show *?thesis*
by (*simp add*: *finite-subset*)
qed
moreover have {i_0. ∀ *i* < i_0. *conflicting* (*f i*) ≠ *None*} ≠ {}
by *auto*
ultimately have i_0 ∈ {i_0. ∀ *i* < i_0. *conflicting* (*f i*) ≠ *None*}
using *Max-in*[*of* {i_0. ∀ *i*<i_0. *conflicting* (*f i*) ≠ *None*}] **unfolding** i_0 **by** *fast*
then have *confl-i*$_0$: *conflicting* (*f* i_0) = *None*
proof −

have *f1*: $\forall\, n{<}i_0.$ *conflicting* (*f n*) $\neq$ *None*
using ‹$i_0 \in \{i_0.\ \forall\, i{<}i_0.$ *conflicting* (*f i*) $\neq$ *None*}› **by** *blast*
have *Suc* $i_0 \notin \{n.\ \forall\, na{<}n.$ *conflicting* (*f na*) $\neq$ *None*}
by (*metis* (*lifting*) *Max-ge* ‹*finite* $\{i_0.\ \forall\, i{<}i_0.$ *conflicting* (*f i*) $\neq$ *None*}›
i_0 *lessI not-le*)
then have $\exists\, n{<}$*Suc* $i_0.$ *conflicting* (*f n*) = *None*
by *fastforce*
then show *conflicting* (*f* i_0) = *None*
using *f1* **by** (*metis le-less less-Suc-eq-le*)
qed
have $\forall\, i < i_0.$ *conflicting* (*f i*) $\neq$ *None*
using ‹$i_0 \in \{i_0.\ \forall\, i < i_0.$ *conflicting* (*f i*) $\neq$ *None*}› **by** *blast*

have *not-conflicting-none*: *False* **if** *confl*: $\forall\, x{>}i.$ *conflicting* (*f x*) = *None* **for** *i* :: *nat*
proof −
let *?f* = $\lambda j.$ *f* (*i* + *j*+*1*)
have $cdcl_W$*-merge* (*?f j*) (*?f* (*Suc j*)) **for** *j* :: *nat*
using *f*[*of i*+*j*+*1*] *confl that* **by** (*auto dest*!: $cdcl_W$*-conflicting-true-*$cdcl_W$*-merge-restart*)
then have (*?f* (*Suc j*), *?f j*) ∈ {(*T*, *S*). $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$*-merge S T*}
for *j* :: *nat*
using *f*[*of i*+*j*+*1*] **by** *auto*
then show *False*
using *wf-*$cdcl_W$*-merge* **unfolding** *wf-iff-no-infinite-down-chain* **by** *fast*
qed

have *not-conflicting*: *False* **if** *confl*: $\forall\, x{>}i.$ *conflicting* (*f x*) $\neq$ *None* **for** *i* :: *nat*
proof −
let *?f* = $\lambda j.$ *f* (*Suc* (*i* + *j*))
have *confl*: *conflicting* (*f x*) $\neq$ *None* **if** *x* > *i* **for** *x* :: *nat*
using *confl that* **by** *auto*
have [*iff*]: ¬*propagate* (*?f j*) *S* ¬*decide* (*?f j*) *S* ¬*conflict* (*?f j*) *S*
for *j* :: *nat* **and** *S* :: *'st*
using *confl*[*of i*+*j*+*1*] **by** (*auto elim*!: *rulesE*)
have [*iff*]: ¬ *backtrack* (*f* (*Suc* (*i* + *j*))) (*f* (*Suc* (*Suc* (*i* + *j*)))) **for** *j* :: *nat*
using *confl*[*of i*+*j*+*2*] **by** (*auto elim*!: *rulesE*)
have $cdcl_W$*-bj* (*?f j*) (*?f* (*Suc j*)) **for** *j* :: *nat*
using *f*[*of i*+*j*+*1*] *confl that* **by** (*auto simp*: $cdcl_W$*.simps* $cdcl_W$*-o.simps elim*: *rulesE*)

then have (*?f* (*Suc j*), *?f j*) ∈ {(*T*, *S*). $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$*-bj S T*}
for *j* :: *nat*
using *f*[*of i*+*j*+*1*] **by** *auto*
then show *False*
using *wf-*$cdcl_W$*-bj-all-struct* **unfolding** *wf-iff-no-infinite-down-chain* **by** *fast*
qed

then have [*simp*]: $\exists\, x{>}i.$ *conflicting* (*f x*) = *None* **for** *i* :: *nat*
by *meson*
have {*j*. *j* > *i* ∧ *conflicting* (*f j*) $\neq$ *None*} $\neq$ {} **for** *i* :: *nat*
using *not-conflicting-none* **by** (*rule ccontr*) *auto*

define *g* **where** *g*: *g* = *rec-nat* i_0 (λ- *i*. *LEAST j*. *j* > *i* ∧ *conflicting* (*f j*) = *None*)
have *g-0*: *g 0* = i_0
unfolding *g* **by** *auto*
have *g-Suc*: *g* (*Suc i*) = (*LEAST j*. *j* > *g i* ∧ *conflicting* (*f j*) = *None*) **for** *i*
unfolding *g* **by** *auto*
have *g-prop*:*g* (*Suc i*) > *g i* ∧ *conflicting* (*f* (*g* (*Suc i*))) = *None* **for** *i*

proof (*cases i*)
case *0*
then show *?thesis*
using *LeastI-ex*[*of* $\lambda j.\ j > i_0 \wedge$ *conflicting* (*f j*) = *None*]
by (*auto simp*: *g*)[]
next
case (*Suc i'*)
then show *?thesis*
apply (*subst g-Suc*, *subst g-Suc*)
using *LeastI-ex*[*of* $\lambda j.\ j > g\ (Suc\ i') \wedge$ *conflicting* (*f j*) = *None*]
by *auto*
qed
then have *g-increasing*: *g* (*Suc i*) > *g i* **for** *i* :: *nat* **by** *simp*
have *confl-f-G*[*simp*]: *conflicting* (*f* (*g i*)) = *None* **for** *i* :: *nat*
by (*cases i*) (*auto simp*: *g-prop g-0 confl-*i_0)
have [*simp*]: $cdcl_W$*-M-level-inv* (*f i*) $cdcl_W$*-all-struct-inv* (*f i*) **for** *i* :: *nat*
using *f*[*of i*] **by** (*auto simp*: $cdcl_W$*-all-struct-inv-def*)
let *?fg* = $\lambda i.$ (*f* (*g i*))
have $\forall\, i <$ *Suc j*. (*f* (*g* (*Suc i*)), *f* (*g i*)) $\in$ {(*T*, *S*). $cdcl_W$*-all-struct-inv S* $\wedge$ $cdcl_W$*-merge*$^{++}$ *S T*}
for *j* :: *nat*
proof (*induction j*)
case *0*
have $cdcl_W{}^{++}$ (*?fg 0*) (*?fg 1*)
using *g-increasing*[*of 0*] **by** (*simp add*: *st*)
then show *?case* **by** (*auto dest*!: *trancl-*$cdcl_W$*-conflicting-true-*$cdcl_W$*-merge-restart*)
next
case (*Suc j*) **note** *IH* = *this*(*1*)
let *?i* = *g* (*Suc j*)
let *?j* = *g* (*Suc* (*Suc j*))
have *conflicting* (*f ?i*) = *None*
by *auto*
moreover have $cdcl_W$*-all-struct-inv* (*f ?i*)
by *auto*
ultimately have $cdcl_W{}^{++}$ (*f ?i*) (*f ?j*)
using *g-increasing* **by** (*simp add*: *st*)
then have $cdcl_W$*-merge*$^{++}$ (*f ?i*) (*f ?j*)
by (*auto dest*!: *trancl-*$cdcl_W$*-conflicting-true-*$cdcl_W$*-merge-restart*)
then show *?case* **using** *IH not-less-less-Suc-eq* **by** *auto*
qed
then have $\forall\, i.$ (*f* (*g* (*Suc i*)), *f* (*g i*)) $\in$ {(*T*, *S*). $cdcl_W$*-all-struct-inv S* $\wedge$ $cdcl_W$*-merge*$^{++}$ *S T*}
by *blast*
then show *False*
using *wf-tranclp-*$cdcl_W$*-merge* **unfolding** *wf-iff-no-infinite-down-chain* **by** *fast*
qed

lemma *wf-*$cdcl_W$*-stgy*:
‹*wf* {(*T*, *S*). $cdcl_W$*-all-struct-inv S* $\wedge$ $cdcl_W$*-stgy S T*}›
by (*rule wf-subset*[*OF wf-*$cdcl_W$]) (*auto dest*: $cdcl_W$*-stgy-*$cdcl_W$)

end

2.3.5 Inclusion of Weidendenbch's CDCL with Strategy

context *conflict-driven-clause-learning*$_W$
begin
abbreviation *propagate-conds* **where**

propagate-conds $\equiv \lambda$-. *propagate*

abbreviation (*input*) *decide-conds* **where**
decide-conds S T $\equiv$ *decide* S T $\wedge$ *no-step conflict* S $\wedge$ *no-step propagate* S

abbreviation *backjump-l-conds-stgy* :: $'v$ *clause* $\Rightarrow$ $'v$ *clause* $\Rightarrow$ $'v$ *literal* $\Rightarrow$ $'st$ $\Rightarrow$ $'st$ $\Rightarrow$ *bool* **where**
backjump-l-conds-stgy C C' L S V $\equiv$
($\exists$ T U. *conflict* S T $\wedge$ *full skip-or-resolve* T U $\wedge$ *conflicting* T = *Some* C $\wedge$
mark-of (*hd-trail* V) = *add-mset* L C' $\wedge$ *backtrack* U V)

abbreviation inv_{NOT}*-stgy* **where**
inv_{NOT}*-stgy* S $\equiv$ *conflicting* S = *None* $\wedge$ $cdcl_W$*-all-struct-inv* S $\wedge$ *no-smaller-propa* S $\wedge$
$cdcl_W$*-stgy-invariant* S $\wedge$ *propagated-clauses-clauses* S

interpretation $cdcl_W$*-with-strategy*: $cdcl_{NOT}$*-merge-bj-learn-ops* **where**
trail = λS. *convert-trail-from-W* (*trail* S) **and**
$clauses_{NOT}$ = *clauses* **and**
prepend-trail = λL S. *cons-trail* (*convert-ann-lit-from-NOT* L) S **and**
tl-trail = λS. *tl-trail* S **and**
$add\text{-}cls_{NOT}$ = λC S. *add-learned-cls* C S **and**
$remove\text{-}cls_{NOT}$ = λC S. *remove-cls* C S **and**
decide-conds = *decide-conds* **and**
propagate-conds = *propagate-conds* **and**
forget-conds = λ- -. *False* **and**
backjump-l-cond = λC C' L' S T. *backjump-l-conds-stgy* C C' L' S T
$\wedge$ *distinct-mset* C' $\wedge$ $L' \notin\#$ C' $\wedge$ $\neg$*tautology* (*add-mset* L' C')
by *unfold-locales*

interpretation $cdcl_W$*-with-strategy*: $cdcl_{NOT}$*-merge-bj-learn-proxy* **where**
trail = λS. *convert-trail-from-W* (*trail* S) **and**
$clauses_{NOT}$ = *clauses* **and**
prepend-trail = λL S. *cons-trail* (*convert-ann-lit-from-NOT* L) S **and**
tl-trail = λS. *tl-trail* S **and**
$add\text{-}cls_{NOT}$ = λC S. *add-learned-cls* C S **and**
$remove\text{-}cls_{NOT}$ = λC S. *remove-cls* C S **and**
decide-conds = *decide-conds* **and**
propagate-conds = *propagate-conds* **and**
forget-conds = λ- -. *False* **and**
backjump-l-cond = *backjump-l-conds-stgy* **and**
inv = inv_{NOT}*-stgy*
by *unfold-locales*

lemma $cdcl_W$*-with-strategy-*$cdcl_{NOT}$*-merged-bj-learn-conflict*:
assumes
$cdcl_W$*-with-strategy*.$cdcl_{NOT}$*-merged-bj-learn* S T
conflicting S = *None*
shows
conflicting T = *None*
using *assms*
apply (*cases rule*: $cdcl_W$*-with-strategy*.$cdcl_{NOT}$*-merged-bj-learn.cases*;
elim $cdcl_W$*-with-strategy*.$forget_{NOT}E$ $cdcl_W$*-with-strategy*.$propagate_{NOT}E$
$cdcl_W$*-with-strategy*.$decide_{NOT}E$ *rulesE*
$cdcl_W$*-with-strategy.backjump-lE backjumpE*)
apply (*auto elim*!: *rulesE simp*: *comp-def*)
done

lemma *cdcl*$_W$*-with-strategy-no-forget*$_{NOT}$[*iff*]: *cdcl*$_W$*-with-strategy.forget*$_{NOT}$ *S T* $\longleftrightarrow$ *False*
  by (*auto elim*: *cdcl*$_W$*-with-strategy.forget*$_{NOT}$*E*)

lemma *cdcl*$_W$*-with-strategy-cdcl*$_{NOT}$*-merged-bj-learn-cdcl*$_W$*-stgy*:
  assumes
    cdcl$_W$*-with-strategy.cdcl*$_{NOT}$*-merged-bj-learn S V*
  shows
    cdcl$_W$*-stgy*** *S V*
  using *assms*
proof (*cases rule*: *cdcl*$_W$*-with-strategy.cdcl*$_{NOT}$*-merged-bj-learn.cases*)
  case *cdcl*$_{NOT}$*-merged-bj-learn-decide*$_{NOT}$
  then show *?thesis*
    apply (*elim cdcl*$_W$*-with-strategy.decide*$_{NOT}$*E*)
    using *cdcl*$_W$*-stgy.other'*[*of S V*] *cdcl*$_W$*-o.decide*[*of S V*] **by** *blast*
next
  case *cdcl*$_{NOT}$*-merged-bj-learn-propagate*$_{NOT}$
  then show *?thesis*
    using *cdcl*$_W$*-stgy.propagate'* **by** (*blast elim*: *cdcl*$_W$*-with-strategy.propagate*$_{NOT}$*E*)
next
  case *cdcl*$_{NOT}$*-merged-bj-learn-forget*$_{NOT}$
  then show *?thesis*
    by *blast*
next
  case *cdcl*$_{NOT}$*-merged-bj-learn-backjump-l*
  then obtain *T U* **where**
    confl: *conflict S T* **and**
    full: *full skip-or-resolve T U* **and**
    bt: *backtrack U V*
    by (*elim cdcl*$_W$*-with-strategy.backjump-lE*) *blast*
  have *cdcl*$_W$*-bj*** *T U*
    using *full mono-rtranclp*[*of skip-or-resolve cdcl*$_W$*-bj*] **unfolding** *full-def*
    by (*blast elim*: *skip-or-resolve.cases*)
  moreover have *cdcl*$_W$*-bj U V* **and** *no-step cdcl*$_W$*-bj V*
    using *bt* **by** (*auto dest*: *backtrack-no-cdcl*$_W$*-bj*)
  ultimately have *full1 cdcl*$_W$*-bj T V*
    unfolding *full1-def* **by** *auto*
  then have *cdcl*$_W$*-stgy*** *T V*
    using *cdcl*$_W$*-s'.bj'*[*of T V*] *cdcl*$_W$*-s'-is-rtranclp-cdcl*$_W$*-stgy*[*of T V*] **by** *blast*
  then show *?thesis*
    using *confl cdcl*$_W$*-stgy.conflict'*[*of S T*] **by** *auto*
qed

lemma *rtranclp-transition-function*:
  ‹R^{**} *a b* $\Longrightarrow$ $\exists$ *f j*. ($\forall$ *i*<*j*. *R* (*f i*) (*f* (*Suc i*))) $\wedge$ *f 0* = *a* $\wedge$ *f j* = *b*›
proof (*induction rule*: *rtranclp-induct*)
  case *base*
  then show *?case* **by** *auto*
next
  case (*step b c*) **note** *st* = *this*(*1*) **and** *R* = *this*(*2*) **and** *IH* = *this*(*3*)
  from *IH* **obtain** *f j* **where**
    i: ‹$\forall$ *i*<*j*. *R* (*f i*) (*f* (*Suc i*))› **and**
    a: ‹*f 0* = *a*› **and**
    b: ‹*f j* = *b*›
    by *auto*
  let *?f* = ‹*f*(*Suc j* := *c*)›

have
i: ⟨∀ *i*<*Suc j*. *R* (*?f i*) (*?f* (*Suc i*))⟩ **and**
a: ⟨*?f 0* = *a*⟩ **and**
b: ⟨*?f* (*Suc j*) = *c*⟩
using *i a b R* **by** *auto*
then show *?case* **by** *blast*
qed

lemma *cdcl*$_W$*-bj-cdcl*$_W$*-stgy*: ⟨*cdcl*$_W$*-bj S T* ⟹ *cdcl*$_W$*-stgy S T*⟩
by (*rule cdcl*$_W$*-stgy.other'*) (*auto simp*: *cdcl*$_W$*-bj.simps cdcl*$_W$*-o.simps elim*!: *rulesE*)

lemma *cdcl*$_W$*-restart-propagated-clauses-clauses*:
⟨*cdcl*$_W$*-restart S T* ⟹ *propagated-clauses-clauses S* ⟹ *propagated-clauses-clauses T*⟩
by (*induction rule*: *cdcl*$_W$*-restart-all-induct*) (*auto simp*: *propagated-clauses-clauses-def*
in-get-all-mark-of-propagated-in-trail simp: *state-prop*)

lemma *rtranclp-cdcl*$_W$*-restart-propagated-clauses-clauses*:
⟨*cdcl*$_W$*-restart*** *S T* ⟹ *propagated-clauses-clauses S* ⟹ *propagated-clauses-clauses T*⟩
by (*induction rule*: *rtranclp-induct*) (*auto simp*: *cdcl*$_W$*-restart-propagated-clauses-clauses*)

lemma *rtranclp-cdcl*$_W$*-stgy-propagated-clauses-clauses*:
⟨*cdcl*$_W$*-stgy*** *S T* ⟹ *propagated-clauses-clauses S* ⟹ *propagated-clauses-clauses T*⟩
using *rtranclp-cdcl*$_W$*-restart-propagated-clauses-clauses*[*of S T*]
rtranclp-cdcl$_W$*-stgy-rtranclp-cdcl*$_W$*-restart* **by** *blast*

lemma *conflicting-clause-bt-lvl-gt-0-backjump*:
assumes
inv: ⟨*inv*$_{NOT}$*-stgy S*⟩ **and**
C: ⟨*C* ∈# *clauses S*⟩ **and**
tr-C: ⟨*trail S* ⊨*as CNot C*⟩ **and**
bt: ⟨*backtrack-lvl S* > *0*⟩
shows ⟨∃ *T U V*. *conflict S T* ∧ *full skip-or-resolve T U* ∧ *backtrack U V*⟩
proof −
let *?T* = *update-conflicting* (*Some C*) *S*
have *confl-S-T*: *conflict S ?T*
using *C tr-C inv* **by** (*auto intro*!: *conflict-rule*)
have *count*: *count-decided* (*trail S*) > *0*
using *inv bt* **unfolding** *cdcl*$_W$*-stgy-invariant-def cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-M-level-inv-def*
by *auto*
have ⟨(∃ *K M'*. *trail S* = *M'* @ *Decided K* # *M*) ⟹ *D* ∈# *clauses S* ⟹ ¬ *M* ⊨*as CNot D*⟩ **for** *M D*
using *inv C tr-C* **unfolding** *cdcl*$_W$*-stgy-invariant-def no-smaller-confl-def*
by *auto*
from *this*[*OF* - *C*] **have** *C-ne*: ⟨*C* ≠ {#}⟩
using *tr-C bt count* **by** (*fastforce simp*: *filter-empty-conv in-set-conv-decomp count-decided-def*
elim!: *is-decided-ex-Decided*)

obtain *U* **where**
U: ⟨*full skip-or-resolve ?T U*⟩
by (*meson wf-exists-normal-form-full wf-skip-or-resolve*)
then have *s-o-r*: *skip-or-resolve*** *?T U*
unfolding *full-def* **by** *blast*
then obtain *C'* **where** *C'*: ⟨*conflicting U* = *Some C'*⟩
by (*induction rule*: *rtranclp-induct*) (*auto simp*: *skip-or-resolve.simps elim*: *rulesE*)
have ⟨*cdcl*$_W$*-stgy*** *?T U*⟩
using *s-o-r* **by** *induction*

(*auto simp*: *skip-or-resolve.simps dest*!: $cdcl_W$*-bj.intros* $cdcl_W$*-bj-*$cdcl_W$*-stgy*)
then have ‹$cdcl_W$*-stgy*** *S U*›
using *confl-S-T* **by** (*auto dest*!: $cdcl_W$*-stgy.intros*)
then have
inv-U: ‹$cdcl_W$*-all-struct-inv U*› **and**
no-smaller-U: ‹*no-smaller-propa U*› **and**
inv-stgy-U: ‹$cdcl_W$*-stgy-invariant U*›
using *inv rtranclp-*$cdcl_W$*-stgy-*$cdcl_W$*-all-struct-inv rtranclp-*$cdcl_W$*-stgy-no-smaller-propa*
rtranclp-$cdcl_W$*-stgy-*$cdcl_W$*-stgy-invariant* **by** *blast+*
show *?thesis*
proof (*cases* C')
case (*add L D*)
then obtain *V* **where** ‹$cdcl_W$*-stgy U V*›
using *conflicting-no-false-can-do-step*[*of U* C'] C' *inv-U inv-stgy-U*
unfolding $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-stgy-invariant-def*
by (*auto simp del*: *conflict-is-false-with-level-def*)
then have ‹*backtrack U V*›
using C' *U* **unfolding** *full-def*
by (*auto simp*: $cdcl_W$*-stgy.simps* $cdcl_W$*-o.simps* $cdcl_W$*-bj.simps elim*: *rulesE*)
then show *?thesis*
using *U confl-S-T* **by** *blast*
next
case [*simp*]: *empty*
obtain *f j* **where**
f-s-o-r: ‹$i<j \Longrightarrow$ *skip-or-resolve* (*f i*) (*f* (*Suc i*))› **and**
f-0: ‹*f 0* = *?T*› **and**
f-j: ‹*f j* = *U*› **for** *i*
using *rtranclp-transition-function*[*OF s-o-r*] **by** *blast*
have *j-0*: ‹$j \neq 0$›
using C' *f-j C-ne f-0* **by** (*cases j*) *auto*

have *bt-lvl-f-l*: ‹*backtrack-lvl* (*f k*) = *backtrack-lvl* (*f 0*)› **if** ‹$k \leq j$› **for** *k*
using *that*
proof (*induction k*)
case *0*
then show *?case* **by** (*simp add*: *f-0*)
next
case (*Suc k*)
then have ‹*backtrack-lvl* (*f* (*Suc k*)) = *backtrack-lvl* (*f k*)›
apply (*cases* ‹$k < j$›; *cases* ‹*trail* (*f k*)›)
using *f-s-o-r*[*of k*] **apply** (*auto simp*: *skip-or-resolve.simps elim*!: *rulesE*)[*2*]
by (*auto simp*: *skip-or-resolve.simps elim*!: *rulesE simp del*: *local.state-simp*)
then show *?case*
using *f-s-o-r*[*of k*] *Suc* **by** *simp*
qed

have *st-f*: ‹$cdcl_W$*-stgy*** *?T* (*f k*)› **if** ‹$k < j$› **for** *k*
using *that*
proof (*induction k*)
case *0*
then show *?case* **by** (*simp add*: *f-0*)
next
case (*Suc k*)
then show *?case*
apply (*cases* ‹$k < j$›)
using *f-s-o-r*[*of k*] **apply** (*auto simp*: *skip-or-resolve.simps*

dest!: $cdcl_W$*-bj.intros* $cdcl_W$*-bj-*$cdcl_W$*-stgy*)[]
using *f-s-o-r*[*of j* − *1*] *j-0* **by** (*simp del*: *local.state-simp*)
qed note *st-f-T* = *this*(*1*)
have *st-f-s-k*: ‹$cdcl_W$*-stgy*** *S* (*f k*)› **if** ‹*k* < *j*› **for** *k*
using *confl-S-T that st-f-T*[*of k*] **by** (*auto dest*!: $cdcl_W$*-stgy.intros*)
have *f-confl*: *conflicting* (*f k*) ≠ *None* **if** ‹*k* ≤ *j*› **for** *k*
using *that f-s-o-r*[*of k*] *f-j C′*
by (*auto simp*: *skip-or-resolve.simps le-eq-less-or-eq elim*!: *rulesE*)
have ‹*size* (*the* (*conflicting* (*f j*))) = *0*›
using *f-j C′* **by** *simp*
moreover have ‹*size* (*the* (*conflicting* (*f 0*))) > *0*›
using *C-ne f-0* **by** (*cases C*) *auto*
then have ‹∃ *x*∈*set* [*0*..<*Suc j*]. *0* < *size* (*the* (*conflicting* (*f x*)))›
by *force*
ultimately obtain *ys l zs* **where**
‹[*0*..<*Suc j*] = *ys* @ *l* # *zs*› **and**
‹*0* < *size* (*the* (*conflicting* (*f l*)))› **and**
‹∀ *z*∈*set zs*. ¬ *0* < *size* (*the* (*conflicting* (*f z*)))›
using *split-list-last-prop*[*of* [*0*..<*Suc j*] λ*i*. *size* (*the* (*conflicting* (*f i*))) > *0*]
by *blast*
moreover have ‹*l* < *j*›
by (*metis C′ Suc-le-lessD* ‹*C′* = {#}› *append1-eq-conv append-cons-eq-upt-length-i-end calculation*(*1*) *calculation*(*2*) *f-j le-eq-less-or-eq neq0-conv option.sel size-eq-0-iff-empty upt-Suc*)
ultimately have ‹*size* (*the* (*conflicting* (*f* (*Suc l*)))) = *0*›
by (*metis* (*no-types*, *hide-lams*) ‹*size* (*the* (*conflicting* (*f j*))) = *0*› *append1-eq-conv append-cons-eq-upt-length-i-end less-eq-nat.simps*(*1*) *list.exhaust list.set-intros*(*1*) *neq0-conv upt-Suc upt-eq-Cons-conv*)
then have *confl-Suc-l*: ‹*conflicting* (*f* (*Suc l*)) = *Some* {#}›
using *f-confl*[*of Suc l*] ‹*l* < *j*› **by** (*cases* ‹*conflicting* (*f* (*Suc l*))›) *auto*
let *?T′* = ‹*f l*›
let *?T″* = ‹*f* (*Suc l*)›
have *res*: ‹*resolve ?T′ ?T″*›
using *confl-Suc-l* ‹*0* < *size* (*the* (*conflicting* (*f l*)))› *f-s-o-r*[*of l*] ‹*l* < *j*›
by (*auto simp*: *skip-or-resolve.simps elim*: *rulesE*)
then have *confl-T′*: ‹*size* (*the* (*conflicting* (*f l*))) = *1*›
using *confl-Suc-l* **by** (*auto elim*!: *rulesE simp*: *Diff-eq-empty-iff-mset subset-eq-mset-single-iff*)

then have *size* (*mark-of* (*hd* (*trail ?T′*))) = *1* **and** *hd-t′-dec*:¬*is-decided* (*hd* (*trail ?T′*))
and *tr-T′-ne*: ‹*trail ?T′* ≠ []›
using *res C′ confl-Suc-l*
by (*auto elim*!: *resolveE simp*: *Diff-eq-empty-iff-mset subset-eq-mset-single-iff*)
then obtain *L* **where** *L*: *mark-of* (*hd* (*trail ?T′*)) = {#*L*#}
by (*cases hd* (*trail ?T′*); *cases mark-of* (*hd* (*trail ?T′*))) *auto*
have
inv-f-l: ‹$cdcl_W$*-all-struct-inv* (*f l*)› **and**
no-smaller-f-l: ‹*no-smaller-propa* (*f l*)› **and**
inv-stgy-f-l: ‹$cdcl_W$*-stgy-invariant* (*f l*)› **and**
propa-cls-f-l: ‹*propagated-clauses-clauses* (*f l*)›
using *inv st-f-s-k*[*OF* ‹*l* < *j*›] *rtranclp-*$cdcl_W$*-stgy-*$cdcl_W$*-all-struct-inv*[*of S f l*] *rtranclp-*$cdcl_W$*-stgy-no-smaller-propa*[*of S f l*] *rtranclp-*$cdcl_W$*-stgy-*$cdcl_W$*-stgy-invariant*[*of S f l*] *rtranclp-*$cdcl_W$*-stgy-propagated-clauses-clauses*
by *blast*+

have *hd-T′*: ‹*hd* (*trail ?T′*) = *Propagated L* {#*L*#}›
  using *inv-f-l L tr-T′-ne hd-t′-dec* **unfolding** *cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-conflicting-def*
  by (*cases trail ?T′*; *cases* (*hd* (*trail ?T′*))) *force+*
let *?D* = *mark-of* (*hd* (*trail ?T′*))
have ‹*get-level* (*trail* (*f l*)) *L* = *0*›
  using *propagate-single-literal-clause-get-level-is-0*[*of f l L*]
    propa-cls-f-l no-smaller-f-l hd-T′ inv-f-l
  unfolding *cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-M-level-inv-def*
  by (*cases* ‹*trail* (*f l*)›) *auto*
then have ‹*count-decided* (*trail ?T′*) = *0*›
  using *hd-T′* **by** (*cases* ‹*trail* (*f l*)›) *auto*
then have ‹*backtrack-lvl ?T′* = *0*›
  using *inv-f-l* **unfolding** *cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-M-level-inv-def*
  by *auto*
then show *?thesis*
  using *bt bt-lvl-f-l*[*of l*] ‹*l* < *j*› *confl-S-T* **by** (*auto simp*: *f-0 elim*: *rulesE*)
qed
qed

lemma *conflict-full-skip-or-resolve-backtrack-backjump-l*:
assumes
  conf: ‹*conflict S T*› **and**
  full: ‹*full skip-or-resolve T U*› **and**
  bt: ‹*backtrack U V*› **and**
  inv: ‹*cdcl*$_W$*-all-struct-inv S*›
shows ‹*cdcl*$_W$*-with-strategy.backjump-l S V*›
proof −
have *inv-U*: ‹*cdcl*$_W$*-all-struct-inv U*›
  by (*metis cdcl*$_W$*-stgy.conflict′ cdcl*$_W$*-stgy-cdcl*$_W$*-all-struct-inv*
    conf full full-def inv rtranclp-cdcl$_W$*-all-struct-inv-inv*
    rtranclp-skip-or-resolve-rtranclp-cdcl$_W$*-restart*)
then have *inv-V*: ‹*cdcl*$_W$*-all-struct-inv V*›
  by (*metis backtrack bt cdcl*$_W$*-bj-cdcl*$_W$*-stgy cdcl*$_W$*-stgy-cdcl*$_W$*-all-struct-inv*)
obtain *C* **where**
  C-S: ‹*C* ∈# *clauses S*› **and**
  S-Not-C: ‹*trail S* ⊨*as CNot C*› **and**
  tr-T-S: ‹*trail T* = *trail S*› **and**
  T: ‹*T* ∼ *update-conflicting* (*Some C*) *S*› **and**
  clss-T-S: ‹*clauses T* = *clauses S*›
  using *conf* **by** (*auto elim*: *rulesE*)
have *s-o-r*: ‹*skip-or-resolve*** *T U*›
  using *full* **unfolding** *full-def* **by** *blast*
then have
  ‹∃ *M*. *trail T* = *M* @ *trail U*› **and**
  bt-T-U: ‹*backtrack-lvl T* = *backtrack-lvl U*› **and**
  bt-lvl-T-U: ‹*backtrack-lvl T* = *backtrack-lvl U*› **and**
  clss-T-U: ‹*clauses T* = *clauses U*› **and**
  init-T-U: ‹*init-clss T* = *init-clss U*› **and**
  learned-T-U: ‹*learned-clss T* = *learned-clss U*›
  using *skip-or-resolve-state-change*[*of T U*] **by** *blast+*
then obtain *M* **where** *M*: ‹*trail T* = *M* @ *trail U*›
  by *blast*
obtain *D D′* :: *′v clause* **and** *K L* :: *′v literal* **and**
  M1 M2 :: (*′v*, *′v clause*) *ann-lit list* **and** *i* :: *nat* **where**
  confl-D: *conflicting U* = *Some* (*add-mset L D*) **and**
  decomp: (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail U*)) **and**

lev-L-U: *get-level* (*trail U*) *L* = *backtrack-lvl U* **and**
max-D-L-U: *get-level* (*trail U*) *L* = *get-maximum-level* (*trail U*) (*add-mset L D′*) **and**
i: *get-maximum-level* (*trail U*) *D′* ≡ *i* **and**
lev-K-U: *get-level* (*trail U*) *K* = *i* + *1* **and**
V: *V* ∼ *cons-trail* (*Propagated L* (*add-mset L D′*))
(*reduce-trail-to M1*
(*add-learned-cls* (*add-mset L D′*)
(*update-conflicting None U*))) **and**
U-L-D′: ‹*clauses U* ⊨*pm add-mset L D′*› **and**
D-D′: ‹*D′* ⊆# *D*›
using *bt* **by** (*auto elim*!: *rulesE*)
let *?D′* = ‹*add-mset L D′*›
obtain *M′* **where** *M′*: ‹*trail U* = *M′* @ *M2* @ *Decided K* # *M1*›
using *decomp* **by** *auto*
have ‹*clauses V* = {#*?D′*#} + *clauses U*›
using *V* **by** *auto*
moreover have ‹*trail V* = (*Propagated L ?D′*) # *trail* (*reduce-trail-to M1 U*)›
using *V T M tr-T-S*[*symmetric*] *M′ clss-T-U*[*symmetric*] **unfolding** *state-eq*$_{NOT}$*-def*
by (*auto simp del*: *state-simp dest*!: *state-simp*(*1*))
ultimately have *V′*: ‹*V* ∼$_{NOT}$
cons-trail (*Propagated L dummy-cls*) (*reduce-trail-to*$_{NOT}$ *M1* (*add-learned-cls ?D′ S*))›
using *V T M tr-T-S*[*symmetric*] *M′ clss-T-U*[*symmetric*] **unfolding** *state-eq*$_{NOT}$*-def*
by (*auto simp del*: *state-simp*
simp: *trail-reduce-trail-to*$_{NOT}$*-drop drop-map drop-tl clss-T-S*)
have ‹*no-dup* (*trail V*)›
using *inv-V V* **unfolding** *cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-M-level-inv-def* **by** *blast*
then have *undef-L*: ‹*undefined-lit M1 L*›
using *V decomp* **by** (*auto simp*: *defined-lit-map*)

have ‹*atm-of L* ∈ *atms-of-mm* (*init-clss V*)›
using *inv-V V decomp* **unfolding** *cdcl*$_W$*-all-struct-inv-def no-strange-atm-def* **by** *auto*
moreover have *init-clss-VU-S*: ‹*init-clss V* = *init-clss S*›
‹*init-clss U* = *init-clss S*›‹*learned-clss U* = *learned-clss S*›
using *T V init-T-U learned-T-U* **by** *auto*
ultimately have *atm-L*: ‹*atm-of L* ∈ *atms-of-mm* (*clauses S*)›
by (*auto simp*: *clauses-def*)

have ‹*distinct-mset ?D′*› **and** ‹¬ *tautology ?D′*›
using *inv-U confl-D decomp D-D′* **unfolding** *cdcl*$_W$*-all-struct-inv-def distinct-cdcl*$_W$*-state-def*
apply *simp-all*
using *inv-V V not-tautology-mono*[*OF D-D′*] *distinct-mset-mono*[*OF D-D′*]
unfolding *cdcl*$_W$*-all-struct-inv-def*
apply (*auto simp add*: *tautology-add-mset*)
done
have ‹*L* ∉# *D′*›
using ‹*distinct-mset ?D′*› **by** (*auto simp*: *not-in-iff*)
have *bj*: ‹*backjump-l-conds-stgy C D′ L S V*›
apply (*rule exI*[*of - T*], *rule exI*[*of - U*])
using ‹*distinct-mset ?D′*› ‹¬ *tautology ?D′*› *conf full bt confl-D*
‹*L* ∉# *D′*› *V T*
by (*auto*)

have *M1-D′*: *M1* ⊨*as CNot D′*
using *backtrack-M1-CNot-D′*[*of U D′* ‹*i*› *K M1 M2 L* ‹*add-mset L D*› *V* ‹*Propagated L* (*add-mset L*

D')›]
inv-U confl-D decomp lev-L-U max-D-L-U i lev-K-U V U-L-D′ D-D′
unfolding $cdcl_W$-all-struct-inv-def $cdcl_W$-conflicting-def $cdcl_W$-M-level-inv-def
by (auto simp: subset-mset-trans-add-mset)
show ?thesis
apply (rule $cdcl_W$-with-strategy.backjump-l.intros[of S - K
convert-trail-from-W M1 - L - C D′])
apply (simp add: tr-T-S[symmetric] M′ M; fail)
using V′ **apply** (simp; fail)
using C-S **apply** (simp; fail)
using S-Not-C **apply** (simp; fail)
using undef-L **apply** (simp; fail)
using atm-L **apply** (simp; fail)
using U-L-D′ init-clss-VU-S **apply** (simp add: clauses-def; fail)
apply (simp; fail)
using M1-D′ **apply** (simp; fail)
using bj ‹distinct-mset ?D′› ‹¬ tautology ?D′› **by** auto
qed

lemma is-decided-o-convert-ann-lit-from-W[simp]:
‹is-decided o convert-ann-lit-from-W = is-decided›
apply (rule ext)
apply (rename-tac x, case-tac x)
apply (auto simp: comp-def)
done

lemma $cdcl_W$-with-strategy-propagate$_{NOT}$-propagate-iff[iff]:
‹$cdcl_W$-with-strategy.propagate$_{NOT}$ S T ⟷ propagate S T› (**is** ?NOT ⟷ ?W)
proof (rule iffI)
assume ?NOT
then show ?W **by** auto
next
assume ?W
then obtain E L **where**
‹conflicting S = None› **and**
E: ‹E ∈# clauses S› **and**
LE: ‹L ∈# E› **and**
tr-E: ‹trail S ⊨as CNot (remove1-mset L E)› **and**
undef: ‹undefined-lit (trail S) L› **and**
T: ‹T ∼ cons-trail (Propagated L E) S›
by (auto elim!: propagateE)
show ?NOT
apply (rule $cdcl_W$-with-strategy.propagate$_{NOT}$[of L ‹remove1-mset L E›])
using LE E **apply** (simp; fail)
using tr-E **apply** (simp; fail)
using undef **apply** (simp; fail)
using ‹?W› **apply** (simp; fail)
using T **by** (simp add: state-eq$_{NOT}$-def clauses-def)
qed

interpretation $cdcl_W$-with-strategy: $cdcl_{NOT}$-merge-bj-learn **where**
trail = λS. convert-trail-from-W (trail S) **and**
clauses$_{NOT}$ = clauses **and**
prepend-trail = λL S. cons-trail (convert-ann-lit-from-NOT L) S **and**
tl-trail = λS. tl-trail S **and**

$add\text{-}cls_{NOT} = \lambda C\ S.\ add\text{-}learned\text{-}cls\ C\ S$ **and**
$remove\text{-}cls_{NOT} = \lambda C\ S.\ remove\text{-}cls\ C\ S$ **and**
decide-conds = *decide-conds* **and**
propagate-conds = *propagate-conds* **and**
forget-conds = λ- -. *False* **and**
backjump-l-cond = *backjump-l-conds-stgy* **and**
inv = inv_{NOT}*-stgy*
proof (*unfold-locales*, *goal-cases*)
case (*2 S T*)
then show *?case*
using $cdcl_W$*-with-strategy-*$cdcl_{NOT}$*-merged-bj-learn-*$cdcl_W$*-stgy*[*of S T*]
$cdcl_W$*-with-strategy-*$cdcl_{NOT}$*-merged-bj-learn-conflict*[*of S T*]
rtranclp-$cdcl_W$*-stgy-*$cdcl_W$*-all-struct-inv rtranclp-*$cdcl_W$*-stgy-no-smaller-propa*
rtranclp-$cdcl_W$*-stgy-*$cdcl_W$*-stgy-invariant rtranclp-*$cdcl_W$*-stgy-propagated-clauses-clauses*
by *blast*
next
case (*1 C′ S C F′ K F L*)
have ‹*count-decided* (*convert-trail-from-W* (*trail S*)) > *0*›
unfolding ‹*convert-trail-from-W* (*trail S*) = *F′* @ *Decided K* # *F*› **by** *simp*
then have ‹*count-decided* (*trail S*) > *0*›
by *simp*
then have ‹*backtrack-lvl S* > *0*›
using ‹inv_{NOT}*-stgy S*› **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def* **by** *auto*
have ∃ *T U V*. *conflict S T* ∧ *full skip-or-resolve T U* ∧ *backtrack U V*
apply (*rule conflicting-clause-bt-lvl-gt-0-backjump*)
using ‹inv_{NOT}*-stgy S*› **apply** (*auto*; *fail*)[]
using ‹*C* ∈# *clauses S*› **apply** (*simp*; *fail*)
using ‹*convert-trail-from-W* (*trail S*) ⊨*as CNot C*› **apply** (*simp*; *fail*)
using ‹*backtrack-lvl S* > *0*› **by** (*simp*; *fail*)
then show *?case*
using *conflict-full-skip-or-resolve-backtrack-backjump-l* ‹inv_{NOT}*-stgy S*› **by** *blast*
next
case (*3 L S*) **note** *atm* = *this*(*1*,*2*) **and** *inv* = *this*(*3*) **and** *sat* = *this*(*4*)
moreover have ‹*Ex*($cdcl_W$*-with-strategy.backjump-l S*)› **if** ‹*conflict S T*› **for** *T*
proof −
have ‹∃ *C*. *C* ∈# *clauses S* ∧ *trail S* ⊨*as CNot C*›
using *that* **by** (*auto elim*: *rulesE*)
then obtain *C* **where** ‹*C* ∈# *clauses S*› **and** ‹*trail S* ⊨*as CNot C*› **by** *blast*
have ‹*backtrack-lvl S* > *0*›
proof (*rule ccontr*)
assume ‹¬ *?thesis*›
then have ‹*backtrack-lvl S* = *0*›
by *simp*
then have ‹*count-decided* (*trail S*) = *0*›
using *inv* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def* **by** *simp*
then have ‹*get-all-ann-decomposition* (*trail S*) = [([], *trail S*)]›
by (*auto simp*: *filter-empty-conv no-decision-get-all-ann-decomposition count-decided-0-iff*)
then have ‹*set-mset* (*clauses S*) ⊨*ps unmark-l* (*trail S*)›
using *3*(*3*) **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *auto*
obtain *I* **where**
consistent: ‹*consistent-interp I*› **and**
I-S: ‹*I* ⊨*m clauses S*› **and**
tot: ‹*total-over-m I* (*set-mset* (*clauses S*))›
using *sat* **by** (*auto simp*: *satisfiable-def*)
have ‹*total-over-m I* (*set-mset* (*clauses S*)) ∧ *total-over-m I* (*unmark-l* (*trail S*))›
using *tot inv* **unfolding** $cdcl_W$*-all-struct-inv-def no-strange-atm-def*

by (*auto simp*: *clauses-def total-over-set-def total-over-m-def*)
then have ⟨*I* ⊨*s unmark-l* (*trail S*)⟩
using ⟨*set-mset* (*clauses S*) ⊨*ps unmark-l* (*trail S*)⟩ *consistent I-S*
unfolding *true-clss-clss-def clauses-def*
by *auto*

have ⟨*I* ⊨*s CNot C*⟩
by (*meson* ⟨*trail S* ⊨*as CNot C*⟩ ⟨*I* ⊨*s unmark-l* (*trail S*)⟩ *set-mp true-annots-true-cls true-cls-def true-clss-def true-clss-singleton-lit-of-implies-incl true-lit-def*)
moreover have ⟨*I* ⊨ *C*⟩
using ⟨*C* ∈# *clauses S*⟩ **and** ⟨*I* ⊨*m clauses S*⟩ **unfolding** *true-cls-mset-def* **by** *auto*
ultimately show *False*
using *consistent consistent-CNot-not* **by** *blast*
qed
then show *?thesis*
using *conflicting-clause-bt-lvl-gt-0-backjump*[*of S C*]
conflict-full-skip-or-resolve-backtrack-backjump-l[*of S*]
⟨*C* ∈# *clauses S*⟩ ⟨*trail S* ⊨*as CNot C*⟩ *inv* **by** *fast*
qed
moreover {
have *atm*: ⟨*atms-of-mm* (*clauses S*) = *atms-of-mm* (*init-clss S*)⟩
using *3*(*3*) **unfolding** $cdcl_W$*-all-struct-inv-def no-strange-atm-def*
by (*auto simp*: *clauses-def*)
have ⟨*decide S* (*cons-trail* (*Decided L*) *S*)⟩
apply (*rule decide-rule*)
using *3* **by** (*auto simp*: *atm*) **}**
moreover have ⟨*cons-trail* (*Decided L*) $S \sim_{NOT}$ *cons-trail* (*Decided L*) *S*⟩
by (*simp add*: *state-eq*$_{NOT}$*-def del*: *state-simp*)
ultimately show ∃ *T*. $cdcl_W$*-with-strategy.decide*$_{NOT}$ *S T* ∨
$cdcl_W$*-with-strategy.propagate*$_{NOT}$ *S T* ∨
$cdcl_W$*-with-strategy.backjump-l S T*
using $cdcl_W$*-with-strategy.decide*$_{NOT}$*.intros*[*of S L cons-trail* (*Decided L*) *S*]
by *auto*
qed

thm $cdcl_W$*-with-strategy.full-cdcl*$_{NOT}$*-merged-bj-learn-final-state*

end

end
theory *CDCL-W-Full*
imports *CDCL-W-Termination CDCL-WNOT*
begin

context *conflict-driven-clause-learning*$_W$
begin
lemma *rtranclp-cdcl*$_W$*-merge-stgy-distinct-mset-clauses*:
assumes
invR: $cdcl_W$*-all-struct-inv R* **and**
st: $cdcl_W$*-s*$'^{**}$ *R S* **and**
smaller: ⟨*no-smaller-propa R*⟩ **and**
dist: *distinct-mset* (*clauses R*)
shows *distinct-mset* (*clauses S*)
using *rtranclp-cdcl*$_W$*-stgy-distinct-mset-clauses*[*OF - invR dist smaller*]
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*)

end

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

Chapter 3

Extensions on Weidenbach's CDCL

We here extend our calculus.

3.1 Restarts

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

This is an unrestricted version.

inductive $cdcl_W$*-restart-stgy* **for** S T :: ‹$'st \times nat$› **where**
‹$cdcl_W$*-stgy* (*fst* S) (*fst* T) $\Longrightarrow$ *snd* S = *snd* T $\Longrightarrow$ $cdcl_W$*-restart-stgy* S T› |
‹*restart* (*fst* S) (*fst* T) $\Longrightarrow$ *snd* T = *Suc* (*snd* S) $\Longrightarrow$ $cdcl_W$*-restart-stgy* S T›

lemma $cdcl_W$*-stgy-*$cdcl_W$*-restart*: ‹$cdcl_W$*-stgy* S S' $\Longrightarrow$ $cdcl_W$*-restart* S S'›
by (*induction rule*: $cdcl_W$*-stgy.induct*) *auto*

lemma $cdcl_W$*-restart-stgy-*$cdcl_W$*-restart*:
‹$cdcl_W$*-restart-stgy* S T $\Longrightarrow$ $cdcl_W$*-restart* (*fst* S) (*fst* T)›
by (*induction rule*: $cdcl_W$*-restart-stgy.induct*)
(*auto dest*: $cdcl_W$*-stgy-*$cdcl_W$*-restart simp*: $cdcl_W$*-restart.simps* $cdcl_W$*-rf.restart*)

lemma *rtranclp-*$cdcl_W$*-restart-stgy-*$cdcl_W$*-restart*:
‹$cdcl_W$*-restart-stgy*** S T $\Longrightarrow$ $cdcl_W$*-restart*** (*fst* S) (*fst* T)›
by (*induction rule*: *rtranclp-induct*)
(*auto dest*: $cdcl_W$*-restart-stgy-*$cdcl_W$*-restart*)

lemma $cdcl_W$*-stgy-*$cdcl_W$*-restart-stgy*:
‹$cdcl_W$*-stgy* S T $\Longrightarrow$ $cdcl_W$*-restart-stgy* (S, n) (T, n)›
using $cdcl_W$*-restart-stgy.intros* [*of* ‹(S, n)› ‹(T, n)›]
by *auto*

lemma *rtranclp-*$cdcl_W$*-stgy-*$cdcl_W$*-restart-stgy*:
‹$cdcl_W$*-stgy*** S T $\Longrightarrow$ $cdcl_W$*-restart-stgy*** (S, n) (T, n)›
apply (*induction rule*: *rtranclp-induct*)
subgoal by *auto*
subgoal for T U
by (*auto dest*!: $cdcl_W$*-stgy-*$cdcl_W$*-restart-stgy*[*of* - - n])
done

lemma $cdcl_W$*-restart-dcl*$_W$*-all-struct-inv*:
‹$cdcl_W$*-restart-stgy* S T $\Longrightarrow$ $cdcl_W$*-all-struct-inv* (*fst* S) $\Longrightarrow$ $cdcl_W$*-all-struct-inv* (*fst* T)›

using *cdcl*$_W$*-all-struct-inv-inv*[*OF cdcl*$_W$*-restart-stgy-cdcl*$_W$*-restart*] **.**

lemma *rtranclp-cdcl*$_W$*-restart-dcl*$_W$*-all-struct-inv*:
‹*cdcl*$_W$*-restart-stgy*** *S T* $\Longrightarrow$ *cdcl*$_W$*-all-struct-inv* (*fst S*) $\Longrightarrow$ *cdcl*$_W$*-all-struct-inv* (*fst T*)›
by (*induction rule*: *rtranclp-induct*)
(*auto intro*: *cdcl*$_W$*-restart-dcl*$_W$*-all-struct-inv*)

lemma *restart-cdcl*$_W$*-stgy-invariant*:
‹*restart S T* $\Longrightarrow$ *cdcl*$_W$*-stgy-invariant T*›
by (*auto simp*: *restart.simps cdcl*$_W$*-stgy-invariant-def state-prop no-smaller-confl-def*)

lemma *cdcl*$_W$*-restart-dcl*$_W$*-stgy-invariant*:
‹*cdcl*$_W$*-restart-stgy S T* $\Longrightarrow$ *cdcl*$_W$*-all-struct-inv* (*fst S*) $\Longrightarrow$ *cdcl*$_W$*-stgy-invariant* (*fst S*) $\Longrightarrow$
cdcl$_W$*-stgy-invariant* (*fst T*)›
apply (*induction rule*: *cdcl*$_W$*-restart-stgy.induct*)
subgoal using *cdcl*$_W$*-stgy-cdcl*$_W$*-stgy-invariant* **.**
subgoal by (*auto dest*!: *cdcl*$_W$*-rf.intros cdcl*$_W$*-restart.intros simp*: *restart-cdcl*$_W$*-stgy-invariant*)
done

lemma *rtranclp-cdcl*$_W$*-restart-dcl*$_W$*-stgy-invariant*:
‹*cdcl*$_W$*-restart-stgy*** *S T* $\Longrightarrow$ *cdcl*$_W$*-all-struct-inv* (*fst S*) $\Longrightarrow$ *cdcl*$_W$*-stgy-invariant* (*fst S*) $\Longrightarrow$
cdcl$_W$*-stgy-invariant* (*fst T*)›
apply (*induction rule*: *rtranclp-induct*)
subgoal by *auto*
subgoal by (*auto simp*: *rtranclp-cdcl*$_W$*-restart-dcl*$_W$*-all-struct-inv cdcl*$_W$*-restart-dcl*$_W$*-stgy-invariant*)
done

end

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

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

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

init-state :: ⟨*'v clauses* ⇒ *'st*⟩ +
fixes
f :: ⟨*nat* ⇒ *nat*⟩

locale $cdcl_W$*-restart-restart* =
$cdcl_W$*-restart-restart-ops* +
assumes
f: ⟨*unbounded f*⟩

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. The same applies for the $cdcl_W$-$stgy^{+\downarrow}$ *S T*: With a $cdcl_W$-$stgy^{\downarrow}$ *S T*, this rules could be applied one after the other, doing nothing each time.

context $cdcl_W$*-restart-restart-ops*
begin
inductive $cdcl_W$*-merge-with-restart* **where**
restart-step:
⟨($cdcl_W$-$stgy^{\frown\frown}$(*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* $cdcl_W$*-stgy S T* ⟹ $cdcl_W$*-merge-with-restart* (*S*, *n*) (*T*, *Suc n*)⟩

lemma $cdcl_W$*-merge-with-restart-rtranclp-*$cdcl_W$*-restart*:
⟨$cdcl_W$*-merge-with-restart S T* ⟹ $cdcl_W$-$restart^{**}$ (*fst S*) (*fst T*)⟩
by (*induction rule*: $cdcl_W$*-merge-with-restart.induct*)
(*auto dest*!: *relpowp-imp-rtranclp rtranclp-*$cdcl_W$*-stgy-rtranclp-*$cdcl_W$*-restart* $cdcl_W$*-restart.rf*
$cdcl_W$*-rf.restart tranclp-into-rtranclp simp*: *full1-def*)

lemma $cdcl_W$*-merge-with-restart-increasing-number*:
⟨$cdcl_W$*-merge-with-restart S T* ⟹ *snd T* = *1* + *snd S*⟩
by (*induction rule*: $cdcl_W$*-merge-with-restart.induct*) *auto*

lemma ⟨*full1* $cdcl_W$*-stgy S T* ⟹ $cdcl_W$*-merge-with-restart* (*S*, *n*) (*T*, *Suc n*)⟩
using *restart-full* **by** *blast*

lemma $cdcl_W$*-all-struct-inv-learned-clss-bound*:
assumes *inv*: ⟨$cdcl_W$*-all-struct-inv S*⟩
shows ⟨*set-mset* (*learned-clss S*) ⊆ *simple-clss* (*atms-of-mm* (*init-clss S*))⟩
proof
fix *C*
assume *C*: ⟨*C* ∈ *set-mset* (*learned-clss S*)⟩
have ⟨*distinct-mset C*⟩
using *C inv* **unfolding** $cdcl_W$*-all-struct-inv-def distinct-*$cdcl_W$*-state-def distinct-mset-set-def*
by *auto*
moreover have ⟨¬*tautology C*⟩
using *C inv* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-learned-clause-alt-def* **by** *auto*
moreover
have ⟨*atms-of C* ⊆ *atms-of-mm* (*learned-clss S*)⟩
using *C* **by** *auto*
then have ⟨*atms-of C* ⊆ *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* ∈ *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-*$cdcl_W$*-restart rtranclp-*$cdcl_W$*-restart-init-clss* **by** *blast*

lemma (**in** $cdcl_W$*-restart-restart*)
‹*wf* {(*T*, *S*). $cdcl_W$*-all-struct-inv* (*fst S*) ∧ $cdcl_W$*-merge-with-restart S T*}›
proof (*rule ccontr*)
assume ‹¬ *?thesis*›
then obtain *g* **where**
g: ‹$\bigwedge$*i*. $cdcl_W$*-merge-with-restart* (*g i*) (*g* (*Suc i*))› **and**
inv: ‹$\bigwedge$*i*. $cdcl_W$*-all-struct-inv* (*fst* (*g i*))›
unfolding *wf-iff-no-infinite-down-chain* **by** *fast*
{ fix *i*
have ‹*init-clss* (*fst* (*g i*)) = *init-clss* (*fst* (*g 0*))›
apply (*induction i*)
apply *simp*
using *g inv* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** (*metis* $cdcl_W$*-merge-with-restart-init-clss*)
} note *init-g* = *this*
let *?S* = ‹*g 0*›
have ‹*finite* (*atms-of-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* (λ*i*. *f* (*snd* (*g i*)))›
using *f* **unfolding** *bounded-def* **by** (*metis add.commute f less-or-eq-imp-le snd-g*
not-bounded-nat-exists-larger not-le le-iff-add)

obtain *k* **where**
f-g-k: ‹*f* (*snd* (*g k*)) > *card* (*simple-clss* (*atms-of-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*: $cdcl_W$*-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** ‹$cdcl_W$*-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$_W$-stgy: ‹(*cdcl$_W$-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$_W$-stgy*** (*fst* (*g k*)) *T*›
using *cdcl$_W$-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$_W$-restart*
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$_W$-restart*
rtranclp-cdcl$_W$-restart-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: ‹*no-smaller-propa* (*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*› **unfolding** *clauses-def*
by (*metis distinct-mset-union fstI restartE subset-mset.le-iff-add union-assoc*)
qed

inductive *cdcl$_W$-restart-with-restart* **where**
restart-step:
‹*cdcl$_W$-stgy*** *S T* ⟹
card (*set-mset* (*learned-clss T*)) − *card* (*set-mset* (*learned-clss S*)) > *f n* ⟹
restart T U ⟹
cdcl$_W$-restart-with-restart (*S*, *n*) (*U*, *Suc n*)› |
restart-full: ‹*full1 cdcl$_W$-stgy S T* ⟹ *cdcl$_W$-restart-with-restart* (*S*, *n*) (*T*, *Suc n*)›

lemma *cdcl*$_W$*-restart-with-restart-rtranclp-cdcl*$_W$*-restart*:
‹*cdcl*$_W$*-restart-with-restart S T* ⟹ *cdcl*$_W$*-restart*** (*fst S*) (*fst T*)›
apply (*induction rule*: *cdcl*$_W$*-restart-with-restart.induct*)
by (*auto dest*!: *relpowp-imp-rtranclp tranclp-into-rtranclp cdcl*$_W$*-restart.rf*
cdcl$_W$*-rf.restart rtranclp-cdcl*$_W$*-stgy-rtranclp-cdcl*$_W$*-restart*
simp: *full1-def*)

lemma *cdcl*$_W$*-restart-with-restart-increasing-number*:
‹*cdcl*$_W$*-restart-with-restart S T* ⟹ *snd T* = *1* + *snd S*›
by (*induction rule*: *cdcl*$_W$*-restart-with-restart.induct*) *auto*

lemma ‹*full1 cdcl*$_W$*-stgy S T* ⟹ *cdcl*$_W$*-restart-with-restart* (*S*, *n*) (*T*, *Suc n*)›
using *restart-full* **by** *blast*

lemma *cdcl*$_W$*-restart-with-restart-init-clss*:
‹*cdcl*$_W$*-restart-with-restart S T* ⟹ *cdcl*$_W$*-M-level-inv* (*fst S*) ⟹
init-clss (*fst S*) = *init-clss* (*fst T*)›
using *cdcl*$_W$*-restart-with-restart-rtranclp-cdcl*$_W$*-restart rtranclp-cdcl*$_W$*-restart-init-clss* **by** *blast*

theorem (**in** *cdcl*$_W$*-restart-restart*)
‹*wf* {(*T*, *S*). *cdcl*$_W$*-all-struct-inv* (*fst S*) ∧ *cdcl*$_W$*-restart-with-restart S T*}›
proof (*rule ccontr*)
assume ‹¬ *?thesis*›
then obtain *g* **where**
g: ‹⋀*i*. *cdcl*$_W$*-restart-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*$_W$*-restart-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*: ‹⋀*i*. *snd* (*g i*) = *i* + *snd* (*g 0*)›
apply (*induct-tac i*)
apply *simp*
by (*metis Suc-eq-plus1-left add-Suc cdcl*$_W$*-restart-with-restart-increasing-number g*)
then have *snd-g-0*: ‹⋀*i*. *i* > *0* ⟹ *snd* (*g i*) = *i* + *snd* (*g 0*)›
by *blast*
have *unbounded-f-g*: ‹*unbounded* (λ*i*. *f* (*snd* (*g i*)))›
using *f* **unfolding** *bounded-def* **by** (*metis add.commute f less-or-eq-imp-le snd-g*
not-bounded-nat-exists-larger not-le le-iff-add)

obtain *k* **where**
f-g-k: ‹*f* (*snd* (*g k*)) > *card* (*simple-clss* (*atms-of-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.

have *H*: *False* **if** ‹*no-step cdcl*$_W$*-stgy* (*fst* (*g i*))› **for** *i*
using *g*[*of i*] *that*
proof (*induction rule*: *cdcl*$_W$*-restart-with-restart.induct*)

case (*restart-step S T n*) **note** $H = this(1)$ **and** $c = this(2)$ **and** $n\text{-}s = this(4)$
obtain S' **where** ‹$cdcl_W$*-stgy S S'*›
using *H c* **by** (*subst* (*asm*) *rtranclp-unfold*) (*auto dest*!: *tranclpD*)
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
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**
$cdcl_W$*-stgy*: ‹$cdcl_W$*-stgy*** (*fst* (*g k*)) *T*›
using *g*[*of k*] *H*[*of* ‹*Suc k*›] **by** (*force simp*: $cdcl_W$*-restart-with-restart.simps full1-def*)
have ‹$cdcl_W$*-all-struct-inv T*›
using *inv*[*of k*] *rtranclp-*$cdcl_W$*-all-struct-inv-inv rtranclp-*$cdcl_W$*-stgy-rtranclp-*$cdcl_W$*-restart*
$cdcl_W$*-stgy* **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-*$cdcl_W$*-restart rtranclp-*$cdcl_W$*-restart-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 $cdcl_W$*-restart-with-restart-distinct-mset-clauses*:
assumes *invR*: ‹$cdcl_W$*-all-struct-inv* (*fst R*)› **and**
st: ‹$cdcl_W$*-restart-with-restart R S*› **and**
dist: ‹*distinct-mset* (*clauses* (*fst R*))› **and**
R: ‹*no-smaller-propa* (*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 S T 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*› **unfolding** *clauses-def*
by (*metis distinct-mset-union fstI restartE subset-mset.le-iff-add union-assoc*)

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 $(2::'a)^{k-1} \le i$ and $i \le (2::'a)^k - (1::'a)$

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*. $2 \,\hat{}\, (k - 1) \leq i \wedge i < 2 \,\hat{}\, k - 1$›
  have ‹$2 \,\hat{}\, (?k - 1) \leq i \wedge i < 2 \,\hat{}\, ?k - 1$›
    by (*rule someI-ex*) (*use 2 exists-luby-decomp* **in** *blast*)
  then show *?case*

  proof −
    have ‹$\forall n\ na.\ \neg\ (1::nat) \leq n \vee 1 \leq n \,\hat{}\, na$›
      by (*meson one-le-power*)
    then have *f1*: ‹$(1::nat) \leq 2 \,\hat{}\, (?k - 1)$›
      using *one-le-numeral* **by** *blast*
    have *f2*: ‹$i - 2 \,\hat{}\, (?k - 1) + 2 \,\hat{}\, (?k - 1) = i$›
      using ‹$2 \,\hat{}\, (?k - 1) \leq i \wedge i < 2 \,\hat{}\, ?k - 1$› *le-add-diff-inverse2* **by** *blast*
    have *f3*: ‹$2 \,\hat{}\, ?k - 1 \neq Suc\ 0$›
      using *f1* ‹$2 \,\hat{}\, (?k - 1) \leq i \wedge i < 2 \,\hat{}\, ?k - 1$› **by** *linarith*
    have ‹$2 \,\hat{}\, ?k - (1::nat) \neq 0$›
      using ‹$2 \,\hat{}\, (?k - 1) \leq i \wedge i < 2 \,\hat{}\, ?k - 1$› *gr-implies-not0* **by** *blast*
    then have *f4*: ‹$2 \,\hat{}\, ?k \neq (1::nat)$›
      by *linarith*
    have *f5*: ‹$\forall n\ na.$ *if* $na = 0$ *then* $(n::nat) \,\hat{}\, na = 1$ *else* $n \,\hat{}\, na = n * n \,\hat{}\, (na - 1)$›
      by (*simp add*: *power-eq-if*)
    then have ‹$?k \neq 0$›
      using *f4* **by** *meson*
    then have ‹$2 \,\hat{}\, (?k - 1) \neq Suc\ 0$›
      using *f5 f3* **by** *presburger*
    then have ‹$Suc\ 0 < 2 \,\hat{}\, (?k - 1)$›
      using *f1* **by** *linarith*
    then show *?thesis*
      using *f2 less-than-iff* **by** *presburger*
  qed
qed

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

lemma *two-pover-n-eq-two-power-n′-eq*:
  assumes *H*: ‹$(2::nat) \,\hat{}\, (k::nat) - 1 = 2 \,\hat{}\, k' - 1$›
  shows ‹$k' = k$›
proof −
  have ‹$(2::nat) \,\hat{}\, (k::nat) = 2 \,\hat{}\, k'$›
    using *H* **by** (*metis One-nat-def Suc-pred zero-less-numeral zero-less-power*)
  then show *?thesis* **by** *simp*
qed

lemma *luby-sequence-core-two-power-minus-one*:
  ‹*luby-sequence-core* $(2\hat{}k - 1) = 2\hat{}(k-1)$› (**is** ‹$?L = ?K$›)
proof −
  have *decomp*: ‹$\exists ka.\ 2 \,\hat{}\, k - 1 = 2 \,\hat{}\, ka - 1$›
    by *auto*
  have ‹$?L = 2\hat{}((SOME\ k'.\ (2::nat)\hat{}k - 1 = 2\hat{}k' - 1) - 1)$›
    apply (*subst luby-sequence-core.simps*, *subst decomp*)
    by *simp*

moreover have ‹(*SOME* k'. (*2*::*nat*)^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* ^ (k − *1*) + *1*)›
proof −
  have *H*: ‹¬ (∃*ka*. i = *2* ^ *ka* − *1*)›
    proof (*rule ccontr*)
      assume ‹¬ *?thesis*›
      then obtain k'::*nat* **where** k': ‹i = *2* ^ k' − *1*› **by** *blast*
      have ‹(*2*::*nat*) ^ k' − *1* < *2* ^ k − *1*›
        using *i-k* **unfolding** k' **.**
      then have ‹(*2*::*nat*) ^ k' < *2* ^ k›
        by *linarith*
      then have ‹k' < k›
        by *simp*
      have ‹*2* ^ (k − *1*) ≤ *2* ^ k' − (*1*::*nat*)›
        using *k-i* **unfolding** k' **.**
      then have ‹(*2*::*nat*) ^ (k−*1*) < *2* ^ 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 ‹⋀k k'. *2* ^ (k − *Suc 0*) ≤ i ⟹ i < *2* ^ k − *Suc 0* ⟹ *2* ^ (k' − *Suc 0*) ≤ i ⟹
    i < *2* ^ k' − *Suc 0* ⟹ k = k'›
    by (*meson different-luby-decomposition-false linorder-neqE-nat*)
  then have k: ‹(*SOME* k. *2* ^ (k − *Suc 0*) ≤ i ∧ i < *2* ^ k − *Suc 0*) = k›
    using *k-i i-k* **by** *auto*
  show *?thesis*
    apply (*subst luby-sequence-core.simps*[*of i*], *subst H*)
    by (*simp add*: k)
qed

lemma *unbounded-luby-sequence-core*: ‹*unbounded luby-sequence-core*›

unfolding *bounded-def*
proof
  assume ‹∃ *b*. ∀ *n*. *luby-sequence-core n* ≤ *b*›
  then obtain *b* **where** *b*: ‹⋀*n*. *luby-sequence-core n* ≤ *b*›
    by *metis*
  have ‹*luby-sequence-core* (*2*^(*b*+*1*) − *1*) = *2*^*b*›
    using *luby-sequence-core-two-power-minus-one*[*of* ‹*b*+*1*›] **by** *simp*
  moreover have ‹(*2*::*nat*)^*b* > *b*›
    by (*induction b*) *auto*
  ultimately show *False* **using** *b*[*of* ‹*2*^(*b*+*1*) − *1*›] **by** *linarith*
qed

abbreviation *luby-sequence* :: ‹*nat* ⇒ *nat*› **where**
‹*luby-sequence n* ≡ *ur* ∗ *luby-sequence-core n*›

lemma *bounded-luby-sequence*: ‹*unbounded luby-sequence*›
  using *bounded-const-product*[*of ur*] *luby-sequence-axioms*
  luby-sequence-def unbounded-luby-sequence-core **by** *blast*

lemma *luby-sequence-core-0*: ‹*luby-sequence-core 0* = *1*›
proof −
  have *0*: ‹(*0*::*nat*) = *2*^*0*−*1*›
    by *auto*
  show *?thesis*
    by (*subst 0*, *subst luby-sequence-core-two-power-minus-one*) *simp*
qed

lemma ‹*luby-sequence-core n* ≥ *1*›
proof (*induction n rule*: *nat-less-induct-case*)
  case *0*
  then show *?case* **by** (*simp add*: *luby-sequence-core-0*)
next
  case (*Suc n*) **note** *IH* = *this*

  consider
    (*interv*) *k* **where** ‹*2* ^ (*k* − *1*) ≤ *Suc n*› **and** ‹*Suc n* < *2* ^ *k* − *1*› |
    (*pow2*) *k* **where** ‹*Suc n* = *2* ^ *k* − *Suc 0*›
    using *exists-luby-decomp*[*of* ‹*Suc n*›] **by** *auto*

  then show *?case*
     proof *cases*
       case *pow2*
       show *?thesis*
         using *luby-sequence-core-two-power-minus-one pow2* **by** *auto*
     next
       case *interv*
       have *n*: ‹*Suc n* − *2* ^ (*k* − *1*) + *1* < *Suc n*›
         by (*metis Suc-1 Suc-eq-plus1 add.commute add-diff-cancel-left′ add-less-mono1 gr0I*
           interv(*1*) *interv*(*2*) *le-add-diff-inverse2 less-Suc-eq not-le power-0 power-one-right*
           power-strict-increasing-iff)
       show *?thesis*
         apply (*subst luby-sequence-core-not-two-power-minus-one*[*OF interv*])
         using *IH n* **by** *auto*
     qed
qed
end

locale *luby-sequence-restart* =
luby-sequence ur +
conflict-driven-clause-learning$_W$
— functions for the state:
state-eq state
— access functions:
trail init-clss learned-clss conflicting
— changing state:
cons-trail tl-trail add-learned-cls remove-cls
update-conflicting

— get state:
init-state
for
ur :: *nat* **and**
state-eq :: ⟨′*st* ⇒ ′*st* ⇒ *bool*⟩ (**infix** ∼ *50*) **and**
state :: ⟨′*st* ⇒ (′*v*, ′*v clause*) *ann-lits* × ′*v clauses* × ′*v clauses* × ′*v clause option* × ′*b*⟩ **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**
conflicting :: ⟨′*st* ⇒ ′*v clause option*⟩ **and**

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

init-state :: ⟨′*v clauses* ⇒ ′*st*⟩
begin

sublocale *cdcl*$_W$*-restart-restart* **where**
f = *luby-sequence*
by *unfold-locales* (*use bounded-luby-sequence* **in** *blast*)

end

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

3.2 Incremental SAT solving

locale *state*$_W$*-adding-init-clause-no-state* =
state$_W$*-no-state*
state-eq
state
— functions about the state:
— getter:
trail init-clss learned-clss conflicting
— setter:

cons-trail tl-trail add-learned-cls remove-cls
update-conflicting

— Some specific states:
init-state
for
state-eq :: $'st \Rightarrow 'st \Rightarrow bool$ (**infix** $\sim$ *50*) **and**
state :: $'st \Rightarrow ('v, 'v\ clause)\ ann\text{-}lits \times 'v\ clauses \times 'v\ clauses \times 'v\ clause\ option \times 'b$ **and**
trail :: $'st \Rightarrow ('v, 'v\ clause)\ ann\text{-}lits$ **and**
init-clss :: $'st \Rightarrow 'v\ clauses$ **and**
learned-clss :: $'st \Rightarrow 'v\ clauses$ **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-conflicting :: $'v\ clause\ option \Rightarrow 'st \Rightarrow 'st$ **and**

init-state :: $'v\ clauses \Rightarrow 'st$ +
fixes
add-init-cls :: $'v\ clause \Rightarrow 'st \Rightarrow 'st$
assumes
add-init-cls:
$state\ st = (M, N, U, S') \Longrightarrow$
$state\ (add\text{-}init\text{-}cls\ C\ st) = (M, \{\#C\#\} + N, U, S')$

locale $state_W$*-adding-init-clause-ops* =
$state_W$*-adding-init-clause-no-state*
state-eq
state
— functions about the state:
— getter:
trail init-clss learned-clss conflicting
— setter:
cons-trail tl-trail add-learned-cls remove-cls update-conflicting

— Some specific states:
init-state
add-init-cls
for
state-eq :: $'st \Rightarrow 'st \Rightarrow bool$ (**infix** $\sim$ *50*) **and**
state :: $'st \Rightarrow ('v, 'v\ clause)\ ann\text{-}lits \times 'v\ clauses \times 'v\ clauses \times 'v\ clause\ option \times 'b$ **and**
trail :: $'st \Rightarrow ('v, 'v\ clause)\ ann\text{-}lits$ **and**
init-clss :: $'st \Rightarrow 'v\ clauses$ **and**
learned-clss :: $'st \Rightarrow 'v\ clauses$ **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-conflicting :: $'v\ clause\ option \Rightarrow 'st \Rightarrow 'st$ **and**

init-state :: *'v clauses* ⇒ *'st* **and**
add-init-cls :: *'v clause* ⇒ *'st* ⇒ *'st* +
assumes
state-prop[*simp*]:
⟨*state S* = (*trail S*, *init-clss S*, *learned-clss S*, *conflicting S*, *additional-info S*)⟩

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

— Some specific states:
init-state add-init-cls
for
state-eq :: *'st* ⇒ *'st* ⇒ *bool* (**infix** ∼ *50*) **and**
state :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* × *'v clauses* × *'v clauses* × *'v clause option* ×
'b **and**
trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **and**
init-clss :: *'st* ⇒ *'v clauses* **and**
learned-clss :: *'st* ⇒ *'v clauses* **and**
conflicting :: *'st* ⇒ *'v clause option* **and**

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

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

sublocale $state_W$
by *unfold-locales auto*

lemma
trail-add-init-cls[*simp*]:
trail (*add-init-cls C st*) = *trail st* **and**
init-clss-add-init-cls[*simp*]:
init-clss (*add-init-cls C st*) = {#*C*#} + *init-clss st*
and
learned-clss-add-init-cls[*simp*]:
learned-clss (*add-init-cls C st*) = *learned-clss st* **and**
conflicting-add-init-cls[*simp*]:
conflicting (*add-init-cls C st*) = *conflicting st*
using *add-init-cls*[*of st - - - - C*] **by** (*cases state st*; *auto*; *fail*)+

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

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

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

init-state :: *'v clauses* ⇒ *'st* **and**
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*

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 (*tl-trail* S)) |
cut-trail-wrt-clause C (*Propagated* L - # M) S =

(*if* $-L \in\#$ *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)

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 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 simp*: *no-dup-def*)
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 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* ⊨*as CNot C*
  shows
    (*trail* ((*cut-trail-wrt-clause C* (*trail T*) *T*))) ⊨*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 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* ⊨*a mma*) ⟶ *M* ⊨*as* {{#− *l*#} |*l*. *l* ∈# *C*}
        using *true-annots-def* **by** *blast*
      have *mma* ∈ {{#− *l*#} |*l*. *l* ∈# *C*} ⟶ *trail T* ⊨*a mma*
        using *CNot-def M bt* **by** (*metis* (*no-types*) *true-annots-def*)
      then have *M* ⊨*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-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* ⊨*a mma*) ⟶ *M* ⊨*as* {{#− *l*#} |*l*. *l* ∈# *C*}
        using *true-annots-def* **by** *blast*
      have *mma* ∈ {{#− *l*#} |*l*. *l* ∈# *C*} ⟶ *trail T* ⊨*a mma*
        using *CNot-def M bt* **by** (*metis* (*no-types*) *true-annots-def*)
      then have *M* ⊨*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
qed

lemma *cut-trail-wrt-clause-hd-trail-in-or-empty-trail*:
$((\forall L \in\# C.\ -L \notin \textit{lits-of-l}\ (\textit{trail}\ T)) \wedge \textit{trail}\ (\textit{cut-trail-wrt-clause}\ C\ (\textit{trail}\ T)\ T) = [])$
$\vee\ (-\textit{lit-of}\ (\textit{hd}\ (\textit{trail}\ (\textit{cut-trail-wrt-clause}\ C\ (\textit{trail}\ T)\ T))) \in\#\ C$
$\wedge\ \textit{length}\ (\textit{trail}\ (\textit{cut-trail-wrt-clause}\ C\ (\textit{trail}\ T)\ T)) \geq 1)$
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 tl-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-trail T*] **and** *M* = *this*(*2*)[*symmetric*]
then show *?case* **by** *simp force*
qed

We can fully run $cdcl_W$*-restart-s* or add a clause. Remark that we use $cdcl_W$*-restart-s* to avoid an explicit *skip*, *resolve*, and *backtrack* normalisation to get rid of the conflict C if possible.

inductive *incremental-cdcl*$_W$ :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* **where**
add-confl:
trail S ⊨*asm init-clss S* ⟹ *distinct-mset C* ⟹ *conflicting S* = *None* ⟹
trail S ⊨*as CNot C* ⟹
full cdcl$_W$*-stgy*
(*update-conflicting* (*Some C*)
(*add-init-cls C* (*cut-trail-wrt-clause C* (*trail S*) *S*))) *T* ⟹
incremental-cdcl$_W$ *S T* |
add-no-confl:
trail S ⊨*asm init-clss S* ⟹ *distinct-mset C* ⟹ *conflicting S* = *None* ⟹
¬*trail S* ⊨*as CNot C* ⟹
full cdcl$_W$*-stgy* (*add-init-cls C S*) *T* ⟹
incremental-cdcl$_W$ *S T*

lemma *cdcl*$_W$*-all-struct-inv-add-new-clause-and-update-cdcl*$_W$*-all-struct-inv*:
assumes
inv-T: *cdcl*$_W$*-all-struct-inv T* **and**
tr-T-N[*simp*]: *trail T* ⊨*asm N* **and**
tr-C[*simp*]: *trail T* ⊨*as CNot C* **and**
[*simp*]: *distinct-mset C*
shows *cdcl*$_W$*-all-struct-inv* (*add-new-clause-and-update C T*) (**is** *cdcl*$_W$*-all-struct-inv ?T'*)
proof −
let *?T* = *update-conflicting* (*Some C*)
(*add-init-cls C* (*cut-trail-wrt-clause C* (*trail T*) *T*))
obtain *M* **where**
M: *trail T* = *M* @ *trail* (*cut-trail-wrt-clause C* (*trail T*) *T*)
using *trail-cut-trail-wrt-clause*[*of T C*] **by** *blast*
have *H*[*dest*]: ⋀*x*. *x* ∈ *lits-of-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*]: ⋀*x*. *x* ∈ *set* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*)) ⟹
x ∈ *set* (*trail T*)
using *inv-T arg-cong*[*OF M*, *of set*] **by** *auto*

have *H-proped*:⋀*x*. *x* ∈ *set* (*get-all-mark-of-propagated* (*trail* (*cut-trail-wrt-clause C*

(*trail T*) *T*))) $\Longrightarrow$ *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 simp*: *no-dup-def*)

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 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* $\Longrightarrow$ ¬*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* ⊨*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* (*clauses T*) (*get-all-ann-decomposition* (*trail T*))
  using *inv-T* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *auto*
have *all-decomposition-implies-m* (*clauses ?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* (*clauses T*) ⊨*ps unmark-l* (*b′* @ *b*)
      using *decomp-T* **unfolding** *all-decomposition-implies-def* **by** *fastforce*
    then have *unmark-l a* ∪ *set-mset* (*clauses ?T*) ⊨*ps unmark-l* (*b′* @ *b*)
      by (*simp add*: *clauses-def*)
    then show *unmark-l a* ∪ *set-mset* (*clauses ?T*) ⊨*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-alt-def*
by (*auto dest*!: *H-proped simp*: *clauses-def*)
show *?thesis*
using ⟨*all-decomposition-implies-m* (*clauses ?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*) $\forall$ *L*$\in$#*C*. − *L* $\notin$ *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*))) $\in$# *C* **and**
1 $\leq$ *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 *trail T* = (*MT* @ *M′*) @ *Decided K* # *M*
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*)[*simplified*] **by** (*auto simp*: *lits-of-def no-dup-def*)
qed
qed
show *?thesis* **using** *L L′ C*
unfolding $cdcl_W$*-stgy-invariant-def* $cdcl_W$*-all-struct-inv-def*
by (*auto simp*: *add-new-clause-and-update-def get-level-def count-decided-def intro*: *rev-bexI*)
qed
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* **and**
learned-entailed: ⟨*cdcl*$_W$*-learned-clauses-entailed-by-init S*⟩
shows
cdcl$_W$*-all-struct-inv T* **and**
cdcl$_W$*-stgy-invariant T* **and**
learned-entailed: ⟨*cdcl*$_W$*-learned-clauses-entailed-by-init 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 *inv'*: *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*$_W$*-restart 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*)

case *3* **show** *?case*
using *learned-entailed rtranclp-cdcl*$_W$*-learned-clauses-entailed*[*of ?T T*] *add-confl inv'*
unfolding *cdcl*$_W$*-all-struct-inv-def full-def*
by (*auto simp*: *cdcl*$_W$*-learned-clauses-entailed-by-init-def*
dest!: *rtranclp-cdcl*$_W$*-stgy-rtranclp-cdcl*$_W$*-restart*)
next
case (*add-no-confl C T*)
have *inv'*: *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-alt-def*
by (*auto 9 1 simp*: *all-decomposition-implies-insert-single clauses-def*)

case *1*
show *?case*
using *inv' 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*) $\longrightarrow$ ¬ *M* ⊨*as CNot C*
using ⟨¬ *trail S* ⊨*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* ⟨¬ *trail S* ⊨*as CNot C*⟩ *inv* **unfolding** *cdcl*$_W$*-stgy-invariant-def*
no-smaller-confl-def eq-commute[*of - trail -*] *cdcl*$_W$*-M-level-inv-def cdcl*$_W$*-all-struct-inv-def*
by (*auto simp*: *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*)

case *3*
have ‹*cdcl*$_W$*-learned-clauses-entailed-by-init* (*add-init-cls C S*)›
using *learned-entailed* **by** (*auto simp*: *cdcl*$_W$*-learned-clauses-entailed-by-init-def*)
then show *?case*
using *add-no-confl*(*5*) *learned-entailed rtranclp-cdcl*$_W$*-learned-clauses-entailed*[*of - T*] *add-confl inv'*
unfolding *cdcl*$_W$*-all-struct-inv-def full-def*
by (*auto simp*: *cdcl*$_W$*-learned-clauses-entailed-by-init-def*
dest!: *rtranclp-cdcl*$_W$*-stgy-rtranclp-cdcl*$_W$*-restart*)
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* **and**
learned-entailed: ‹*cdcl*$_W$*-learned-clauses-entailed-by-init S*›
shows
cdcl$_W$*-all-struct-inv T* **and**
cdcl$_W$*-stgy-invariant T* **and**
‹*cdcl*$_W$*-learned-clauses-entailed-by-init T*›
using *inc* **apply** *induction*
using *inv* **apply** *simp*
using *s-inv* **apply** *simp*
using *learned-entailed* **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* **and**
learned-entailed: ‹*cdcl*$_W$*-learned-clauses-entailed-by-init 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*
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 *C conf dist full incremental-cdcl*$_W$*.add-confl incremental-cdcl*$_W$*-inv*
incremental-cdcl$_W$*-inv inv learned-entailed*
‹*full cdcl*$_W$*-stgy T T*› *full-cdcl*$_W$*-stgy-inv-normal-form*
s-inv tr **by** *blast*
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*
using ‹*full cdcl*$_W$*-stgy T T*› *full-cdcl*$_W$*-stgy-inv-normal-form C conf dist full*

incremental-cdcl$_W$.*add-no-confl incremental-cdcl*$_W$-*inv inv learned-entailed*
s-inv tr **by** *blast*
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* **and**
learned-entailed: ‹*cdcl*$_W$-*learned-clauses-entailed-by-init 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 learned-entailed)

end

end
theory *DPLL-CDCL-W-Implementation*
imports
Entailment-Definition.Partial-Annotated-Herbrand-Interpretation
CDCL-W-Level
begin

Chapter 4

List-based Implementation of DPLL and CDCL

We can now 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.

4.1 Simple List-Based Implementation of the DPLL and CDCL

The idea of the list-based implementation is to test the stack: the theories about the calculi, adapting the theorems to a simple implementation and the code exportation. The implementation are very simple ans simply iterate over-and-over on lists.

4.1.1 Common Rules

Propagation

The following theorem holds:

lemma *lits-of-l-unfold*:
  $(\forall\, c \in set\ C.\ -c \in lits\text{-}of\text{-}l\ Ms) \longleftrightarrow Ms \models\!as\ CNot\ (mset\ C)$
  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* $\Rightarrow$ $('a, 'b)$ *ann-lits* $\Rightarrow$ $'a$ *literal option*
 where
 is-unit-clause l M =
   $(case\ List.filter\ (\lambda a.\ atm\text{-}of\ a \notin atm\text{-}of\ `\ lits\text{-}of\text{-}l\ M)\ l\ of$
     $a\ \#\ [] \Rightarrow if\ M \models\!as\ CNot\ (mset\ l - \{\#a\#\})\ then\ Some\ a\ else\ None$
   $|\ \text{-} \Rightarrow None)$

definition *is-unit-clause-code* :: $'a$ *literal list* $\Rightarrow$ $('a, 'b)$ *ann-lits*
  $\Rightarrow$ $'a$ *literal option* **where**
 is-unit-clause-code l M =
   $(case\ List.filter\ (\lambda a.\ atm\text{-}of\ a \notin atm\text{-}of\ `\ lits\text{-}of\text{-}l\ M)\ l\ of$
     $a\ \#\ [] \Rightarrow if\ (\forall\, c \in set\ (remove1\ a\ l).\ -c \in lits\text{-}of\text{-}l\ M)\ then\ Some\ a\ else\ None$
   $|\ \text{-} \Rightarrow None)$

lemma *is-unit-clause-is-unit-clause-code*[*code*]:
is-unit-clause l M = *is-unit-clause-code l M*
proof −
have *1*: $\bigwedge a$. (∀ *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*←*l* . *atm-of a* ∉ *atm-of* ‘ *lits-of-l M*] *of* [] ⇒ *None*
| [*a*] ⇒ *if M* ⊨*as CNot* (*mset l* − {#*a*#}) *then Some a else None*
| *a* # *ab* # *xa* ⇒ *Map.empty xa*) = *Some a*
using *assms* **unfolding** *is-unit-clause-def* **.**
then have *a* ∈ *set* [*a*←*l* . *atm-of a* ∉ *atm-of* ‘ *lits-of-l M*]
apply (*cases* [*a*←*l* . *atm-of a* ∉ *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* ∉ *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* ⟹ *M* ⊨*as CNot* (*mset l* − {#*a*#})
unfolding *is-unit-clause-def*
proof −
assume (*case* [*a*←*l* . *atm-of a* ∉ *atm-of* ‘ *lits-of-l M*] *of* [] ⇒ *None*
| [*a*] ⇒ *if M* ⊨*as CNot* (*mset l* − {#*a*#}) *then Some a else None*
| *a* # *ab* # *xa* ⇒ *Map.empty xa*) = *Some a*
then show *?thesis*
apply (*cases* [*a*←*l* . *atm-of a* ∉ *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* ⟹ *a* ∈ *set l*
unfolding *is-unit-clause-def*
proof −
assume (*case* [*a*←*l* . *atm-of a* ∉ *atm-of* ‘ *lits-of-l M*] *of* [] ⇒ *None*
| [*a*] ⇒ *if M* ⊨*as CNot* (*mset l* − {#*a*#}) *then Some a else None*
| *a* # *ab* # *xa* ⇒ *Map.empty xa*) = *Some a*
then show *a* ∈ *set l*
by (*cases* [*a*←*l* . *atm-of a* ∉ *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* ⇒ (*'a*, *'b*) *ann-lits*
  ⇒ (*'a literal* × *'a literal list*) *option* **where**
find-first-unit-clause (*a* # *l*) *M* =
  (*case is-unit-clause a M of*
    None ⇒ *find-first-unit-clause l M*
  | *Some L* ⇒ *Some* (*L*, *a*)) |
find-first-unit-clause [] - = *None*

lemma *find-first-unit-clause-some*:
  find-first-unit-clause l M = *Some* (*a*, *c*)
  ⟹ *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
  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*:
  c ∈ *set l* ⟹ *M* ⊨*as CNot* (*mset c* − {#*a*#}) ⟹ *undefined-lit M a* ⟹ *a* ∈ *set c*
  ⟹ *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* (*λlit. 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 (*λlit. lit* ∉ *M* ∧ −*lit* ∉ *M*) *a* = *None* ⟷ *atm-of* ‘ *set a* ⊆ *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* (*λlit. lit* ∉ *M* ∧ −*lit* ∉ *M*) *a* = *Some b* ⟹ *b* ∈ *set a* ∧ *b* ∉ *M* ∧ −*b* ∉ *M*
unfolding *find-Some-iff* **by** (*metis nth-mem*)

lemma *find-first-unused-var-None*[*iff*]:
find-first-unused-var l M = *None* ⟷ (∀ *a* ∈ *set l*. *atm-of* ‘ *set a* ⊆ *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 ¬(∀ *a* ∈ *set l*. *atm-of* ‘ *set a* ⊆ *atm-of* ‘ *M*)
proof −
have *find-first-unused-var l M* ≠ *None*
using *assms* **by** (*cases find-first-unused-var l M*) *auto*
then show ¬(∀ *a* ∈ *set l*. *atm-of* ‘ *set a* ⊆ *atm-of* ‘ *M*) **by** *auto*
qed

lemma *find-first-unused-var-Some*:
find-first-unused-var l M = *Some a* ⟹ (∃ *m* ∈ *set l*. *a* ∈ *set m* ∧ *a* ∉ *M* ∧ −*a* ∉ *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* ⟹ *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*

4.1.2 CDCL specific functions

Level

fun *maximum-level-code*:: *'a literal list* ⇒ (*'a*, *'b*) *ann-lits* ⇒ *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-add-mset*)

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* $i = k \wedge j < i$ *then Some* (L, j) *else find-level-decomp* M Ls (L#D) k
)

lemma *find-level-decomp-some*:
  assumes *find-level-decomp* M Ls D k = *Some* (L, j)
  shows $L \in$ *set* Ls ∧ *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*)

  define *find* **where** *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) ⟹
     $L \in$ *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*: $L \in$ *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 *auto*
  } 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**
    $L \neq L'$ ⟶ $L \in$ *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.commute add-diff-cancel-left' add-mset-add-single mset.simps*(*2*)
        option.inject prod.inject)
    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 simp*: *no-dup-def 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*: *no-dup-def 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, Some D) =
(*case find-level-decomp M D* [] (*count-decided M*) *of*
None ⇒ (*M, N, U, Some D*)
| *Some (L, j)* ⇒
(*case bt-cut j M of*
Some (Decided - # Ls) ⇒ (*Propagated L D # Ls, N, D # U, None*)
| - ⇒ (*M, N, U, Some D*))
) |
do-backtrack-step S = S

end
theory *DPLL-W-Implementation*

imports *DPLL-CDCL-W-Implementation DPLL-W HOL−Library.Code-Target-Numeral*
begin

4.1.3 Simple Implementation of DPLL

Combining the propagate and decide: a DPLL step

definition $DPLL\text{-}step$ :: $int\ dpll_W\text{-}ann\text{-}lits \times int\ literal\ list\ list$
  $\Rightarrow int\ dpll_W\text{-}ann\text{-}lits \times int\ literal\ list\ list$ **where**
$DPLL\text{-}step = (\lambda(Ms,\ N).$
  $(case\ find\text{-}first\text{-}unit\text{-}clause\ N\ Ms\ of$
    $Some\ (L,\ \text{-}) \Rightarrow (Propagated\ L\ ()\ \#\ Ms,\ N)$
  $|\ \text{-} \Rightarrow$
    $if\ \exists\ C \in set\ N.\ (\forall\ c \in set\ C.\ -c \in lits\text{-}of\text{-}l\ Ms)$
    $then$
      $(case\ backtrack\text{-}split\ Ms\ of$
        $(\text{-},\ L\ \#\ M) \Rightarrow (Propagated\ (-\ (lit\text{-}of\ L))\ ()\ \#\ M,\ N)$
      $|\ (\text{-},\ \text{-}) \Rightarrow (Ms,\ N)$
      $)$
    $else$
    $(case\ find\text{-}first\text{-}unused\text{-}var\ N\ (lits\text{-}of\text{-}l\ Ms)\ of$
        $Some\ a \Rightarrow (Decided\ a\ \#\ Ms,\ N)$
      $|\ None \Rightarrow (Ms,\ N))))$

Example of propagation:

value $DPLL\text{-}step\ ([Decided\ (Neg\ 1)],\ [[Pos\ (1::int),\ Neg\ 2]])$

We define the conversion function between the states as defined in *Prop-DPLL* (with multisets) and here (with lists).

abbreviation $toS \equiv \lambda(Ms::(int,\ unit)\ ann\text{-}lits)$
                    $(N::\ int\ literal\ list\ list).\ (Ms,\ mset\ (map\ mset\ N))$
abbreviation $toS' \equiv \lambda(Ms::(int,\ unit)\ ann\text{-}lits,$
                    $N::\ int\ literal\ list\ list).\ (Ms,\ mset\ (map\ mset\ N))$

Proof of correctness of *DPLL-step*

lemma $DPLL\text{-}step\text{-}is\text{-}a\text{-}dpll_W\text{-}step$:
  assumes *step*: $(Ms',\ N') = DPLL\text{-}step\ (Ms,\ N)$
  and *neq*: $(Ms,\ N) \neq (Ms',\ N')$
  shows $dpll_W\ (toS\ Ms\ N)\ (toS\ Ms'\ N')$
proof −
  let $?S = (Ms,\ mset\ (map\ mset\ N))$
  { fix $L\ E$
    assume *unit*: $find\text{-}first\text{-}unit\text{-}clause\ N\ Ms = Some\ (L,\ E)$
    then have $Ms'N$: $(Ms',\ N') = (Propagated\ L\ ()\ \#\ Ms,\ N)$
      using *step* **unfolding** *DPLL-step-def* **by** *auto*
    obtain C **where**
      C: $C \in set\ N$ **and**
      Ms: $Ms \models as\ CNot\ (mset\ C - \{\#L\#\})$ **and**
      undef: $undefined\text{-}lit\ Ms\ L$ **and**
      $L \in set\ C$ **using** $find\text{-}first\text{-}unit\text{-}clause\text{-}some[OF\ unit]$ **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 $Ms\ undef\ C\ \langle L \in set\ C\rangle$ **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*: $\exists$ *C* $\in$ *set N*. *Ms* $\models$*as CNot* (*mset C*)
then obtain *C* **where** *C*: *C* $\in$ *set N* **and** *Ms*: *Ms* $\models$*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 simp*: *lits-of-l-unfold*)
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-of L*)) () # *M*, *N*)
using *step exC* **unfolding** *DPLL-step-def bt prod.case unit* **by** (*auto simp*: *lits-of-l-unfold*)
ultimately have *?thesis* **by** *auto*
}
moreover
{ **assume** *unit*: *find-first-unit-clause N Ms = None*
assume *exC*: ¬ ($\exists$ *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 simp*: *lits-of-l-unfold*)
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 simp*: *lits-of-l-unfold*)
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 lits-of-l-unfold*)

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*: ¬ (∃ *C* ∈ *set N*. *Ms* ⊨*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 lits-of-l-unfold split*: *option.splits*)
then have *a*: ∀ *a* ∈ *set N*. *atm-of* ‘ *set a* ⊆ *atm-of* ‘ (*lits-of-l Ms*) **by** *auto*
have *fst* (*toS Ms N*) ⊨*asm snd* (*toS Ms N*) **unfolding** *true-annots-def CNot-def Ball-def*
proof *clarify*
fix *x*
assume *x*: *x* ∈ *set-mset* (*clauses* (*toS Ms N*))
then have ¬*Ms* ⊨*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*) ⊨*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* ⇒ *int literal list list* ⇒ *int dpll$_W$-ann-lits* × *int literal list list* **where**
DPLL-ci Ms N =
(*if* ¬*dpll$_W$-all-inv* (*Ms*, *mset* (*map mset N*))
then (*Ms*, *N*)
else
let (*Ms'*, *N'*) = *DPLL-step* (*Ms*, *N*) *in*
if (*Ms'*, *N'*) = (*Ms*, *N*) *then* (*Ms*, *N*) *else DPLL-ci Ms' N*)
by *fast+*
termination
proof (*relation* {(*S'*, *S*). (*toS' S'*, *toS' S*) ∈ {(*S'*, *S*). *dpll$_W$-all-inv S* ∧ *dpll$_W$ S S'*}})
show *wf* {(*S'*, *S*).(*toS' S'*, *toS' S*) ∈ {(*S'*, *S*). *dpll$_W$-all-inv S* ∧ *dpll$_W$ S S'*}}
using *wf-if-measure-f*[*OF wf-dpll$_W$*, *of toS'*] **by** *auto*
next
fix *Ms* :: *int dpll$_W$-ann-lits* **and** *N x xa y*
assume¬ ¬ *dpll$_W$-all-inv* (*toS Ms N*)
and *step*: *x* = *DPLL-step* (*Ms*, *N*)
and *x*: (*xa*, *y*) = *x*
and (*xa*, *y*) ≠ (*Ms*, *N*)
then show ((*xa*, *N*), *Ms*, *N*) ∈ {(*S'*, *S*). (*toS' S'*, *toS' S*) ∈ {(*S'*, *S*). *dpll$_W$-all-inv S* ∧ *dpll$_W$ S S'*}}

using *DPLL-step-is-a-dpll*$_W$*-step dpll*$_W$*-same-clauses split-conv* **by** *fastforce*
qed

No invariant tested **function** (*domintros*) *DPLL-part*:: *int dpll*$_W$*-ann-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* $\wedge$ *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*) $\neq$ (*Ms*, *N*)
then have *DPLL-ci Ms' N* = *DPLL-part Ms' N* $\wedge$ *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* $\wedge$ *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) $\neq$ (*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) $\neq$ (*Ms*, *N*)› *prod.sel*(*2*) *snd-DPLL-step*)
ultimately have *?case* **by** (*metis* (*mono-tags*, *hide-lams*) *IH S* ‹(S_1, S_2) $\neq$ (*Ms*, *N*)›
‹*DPLL-ci Ms N* = *DPLL-ci* S_1 *N*› ‹$dpll_W$*-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** *wf-*$dpll_W$*-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*) $\neq$ (*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*) ≠ (*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 *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*)}

proof −
obtain M N **where**
S: ‹*rough-state-of* S = (M, N)›
by (*cases* ‹*rough-state-of* S›)
obtain M' N' **where**
S': ‹*DPLL-step* (*rough-state-of* S) = (M', N')›
by (*cases* ‹*DPLL-step* (*rough-state-of* S)›)
have ‹$dpll_W{}^{**}$ (*toS* M N) (*toS* M' N')›
by (*metis DPLL-step-is-a-dpll*$_W$*-step* S S' *fst-conv r-into-rtranclp rtranclp.rtrancl-refl snd-conv*)
then show *?thesis*
using *rough-state-of*[*of S*] **unfolding** S' **unfolding** S **by** (*auto intro*: *rtranclp-dpll*$_W$*-all-inv*)
qed

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-state-of* T', *rough-state-of* T)
∈ $\{(S', S)$. (*toS'* S', *toS'* S)
∈ $\{(S', S)$. *dpll*$_W$*-all-inv* S ∧ *dpll*$_W$ S $S'\}\}\}$)
show *wf* $\{(b, a)$.
(*rough-state-of* b, *rough-state-of* a)
∈ $\{(b, a)$. (*toS'* b, *toS'* a)
∈ $\{(b, a)$. *dpll*$_W$*-all-inv* a ∧ *dpll*$_W$ a $b\}\}\}$
using *wf-if-measure-f*[*OF wf-if-measure-f*[*OF wf-dpll*$_W$, *of toS'*], *of rough-state-of*] **.**
next
fix S x
assume x: x = *DPLL-step'* S
and $x \neq S$
have *dpll*$_W$*-all-inv* (*case rough-state-of S of* (*Ms*, N) ⇒ (*Ms*, *mset* (*map mset* N)))
by (*metis* (*no-types*, *lifting*) *case-prodE mem-Collect-eq old.prod.case rough-state-of*)
moreover have *dpll*$_W$ (*case rough-state-of S of* (*Ms*, N) ⇒ (*Ms*, *mset* (*map mset* N)))
(*case rough-state-of* (*DPLL-step'* S) *of* (*Ms*, N) ⇒ (*Ms*, *mset* (*map mset* N)))
proof −
obtain *Ms* N **where** *Ms*: (*Ms*, N) = *rough-state-of* S **by** (*cases rough-state-of S*) *auto*
have *dpll*$_W$*-all-inv* (*toS'* (*Ms*, N)) **using** *calculation* **unfolding** *Ms* **by** *blast*
moreover obtain *Ms'* N' **where** *Ms'*: (*Ms'*, N') = *rough-state-of* (*DPLL-step'* S)
by (*cases rough-state-of* (*DPLL-step'* S)) *auto*
ultimately have *dpll*$_W$*-all-inv* (*toS'* (*Ms'*, N')) **unfolding** *Ms'*
by (*metis* (*no-types*, *lifting*) *case-prod-unfold mem-Collect-eq rough-state-of*)

have *dpll*$_W$ (*toS Ms N*) (*toS Ms'* N')
apply (*rule DPLL-step-is-a-dpll*$_W$*-step*[*of Ms'* N' *Ms N*])
unfolding *Ms Ms'* **using** ‹$x \neq S$› *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-state-of* T', *rough-state-of* T)
∈ $\{(S', S)$. (*toS'* S', *toS'* S) ∈ $\{(S', S)$. *dpll*$_W$*-all-inv* S ∧ *dpll*$_W$ S $S'\}\}\}$
by (*auto simp add*: x)

qed

lemma [*code*]:
DPLL-tot S =
  (*let S′* = *DPLL-step′ S in*
   if S′ = *S then S else DPLL-tot S′*) **by** *auto*

lemma *DPLL-tot-DPLL-step-DPLL-tot*[*simp*]: *DPLL-tot* (*DPLL-step′ S*) = *DPLL-tot S*
  apply (*cases DPLL-step′ S* = *S*)
  apply *simp*
  unfolding *DPLL-tot.simps*[*of S*] **by** (*simp del*: *DPLL-tot.simps*)

lemma *DOPLL-step′-DPLL-tot*[*simp*]:
  DPLL-step′ (*DPLL-tot S*) = *DPLL-tot S*
  by (*rule DPLL-tot.induct*[*of* λ*S*. *DPLL-step′* (*DPLL-tot S*) = *DPLL-tot S S*])
    (*metis* (*full-types*) *DPLL-tot.simps*)

lemma *DPLL-tot-final-state*:
  assumes *DPLL-tot S* = *S*
  shows *conclusive-dpll$_W$-state* (*toS′* (*rough-state-of S*))
proof −
  have *DPLL-step′ S* = *S* **using** *assms*[*symmetric*] *DOPLL-step′-DPLL-tot* **by** *metis*
  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
theory *CDCL-W-Implementation*
imports *DPLL-CDCL-W-Implementation CDCL-W-Termination*
HOL−Library.Code-Target-Numeral
begin

4.1.4 List-based CDCL Implementation

We here have a very simple implementation of Weidenbach's CDCL, based on the same principle as the implementation of DPLL: iterating over-and-over on lists. We do not use any fancy data-structure (see the two-watched literals for a better suited data-structure).

The goal was (as for DPLL) to test the infrastructure and see if an important lemma was missing to prove the correctness and the termination of a simple implementation.

Types and Instantiation

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

type-synonym $'a$ $cdcl_W$*-restart-mark* = $'a$ *clause*

type-synonym $'v$ $cdcl_W$*-restart-ann-lit* = ($'v$, $'v$ $cdcl_W$*-restart-mark*) *ann-lit*
type-synonym $'v$ $cdcl_W$*-restart-ann-lits* = ($'v$, $'v$ $cdcl_W$*-restart-mark*) *ann-lits*
type-synonym $'v$ $cdcl_W$*-restart-state* =
  $'v$ $cdcl_W$*-restart-ann-lits* × $'v$ *clauses* × $'v$ *clauses* × $'v$ *clause option*

abbreviation *raw-trail* :: $'a \times 'b \times 'c \times 'd \Rightarrow 'a$ **where**
raw-trail ≡ (λ(M, -). M)

abbreviation *raw-cons-trail* :: $'a \Rightarrow 'a$ *list* $\times 'b \times 'c \times 'd \Rightarrow 'a$ *list* $\times 'b \times 'c \times 'd$
  where
raw-cons-trail ≡ (λL (M, S). (L#M, S))

abbreviation *raw-tl-trail* :: $'a$ *list* $\times 'b \times 'c \times 'd \Rightarrow 'a$ *list* $\times 'b \times 'c \times 'd$ **where**
raw-tl-trail ≡ (λ(M, S). (*tl* M, S))

abbreviation *raw-init-clss* :: $'a \times 'b \times 'c \times 'd \Rightarrow 'b$ **where**
raw-init-clss ≡ λ(M, N, -). N

abbreviation *raw-learned-clss* :: $'a \times 'b \times 'c \times 'd \Rightarrow 'c$ **where**
raw-learned-clss ≡ λ(M, N, U, -). U

abbreviation *raw-conflicting* :: $'a \times 'b \times 'c \times 'd \Rightarrow 'd$ **where**
raw-conflicting ≡ λ(M, N, U, D). D

abbreviation *raw-update-conflicting* :: $'d \Rightarrow 'a \times 'b \times 'c \times 'd \Rightarrow 'a \times 'b \times 'c \times 'd$
  where
raw-update-conflicting ≡ λS (M, N, U, -). (M, N, U, S)

abbreviation *S0-*$cdcl_W$*-restart* N ≡ (([], N, {#}, *None*):: $'v$ $cdcl_W$*-restart-state*)

abbreviation *raw-add-learned-clss* **where**
raw-add-learned-clss ≡ λC (M, N, U, S). (M, N, {#C#} + U, S)

abbreviation *raw-remove-cls* **where**
raw-remove-cls ≡ λC (M, N, U, S). (M, *removeAll-mset* C N, *removeAll-mset* C U, S)

lemma *raw-trail-conv*: *raw-trail* (M, N, U, D) = M **and**
  clauses-conv: *raw-init-clss* (M, N, U, D) = N **and**
  raw-learned-clss-conv: *raw-learned-clss* (M, N, U, D) = U **and**
  raw-conflicting-conv: *raw-conflicting* (M, N, U, D) = D
  by *auto*

lemma *state-conv*:
  $S = (raw\text{-}trail\ S,\ raw\text{-}init\text{-}clss\ S,\ raw\text{-}learned\text{-}clss\ S,\ raw\text{-}conflicting\ S)$
  by (*cases S*) *auto*

definition *state* **where**
‹$state\ S = (raw\text{-}trail\ S,\ raw\text{-}init\text{-}clss\ S,\ raw\text{-}learned\text{-}clss\ S,\ raw\text{-}conflicting\ S,\ ())$›

interpretation $state_W$
  (=)
  state
  raw-trail raw-init-clss raw-learned-clss raw-conflicting
  $\lambda L\ (M,\ S).\ (L\ \#\ M,\ S)$
  $\lambda(M,\ S).\ (tl\ M,\ S)$
  $\lambda C\ (M,\ N,\ U,\ S).\ (M,\ N,\ add\text{-}mset\ C\ U,\ S)$
  $\lambda C\ (M,\ N,\ U,\ S).\ (M,\ removeAll\text{-}mset\ C\ N,\ removeAll\text{-}mset\ C\ U,\ S)$
  $\lambda D\ (M,\ N,\ U,\ \text{-}).\ (M,\ N,\ U,\ D)$
  $\lambda N.\ ([],\ N,\ \{\#\},\ None)$
  by *unfold-locales* (*auto simp*: *state-def*)

declare *state-simp*[*simp del*]

interpretation $conflict\text{-}driven\text{-}clause\text{-}learning_W$
  (=) *state*
  raw-trail raw-init-clss raw-learned-clss
  raw-conflicting
  $\lambda L\ (M,\ S).\ (L\ \#\ M,\ S)$
  $\lambda(M,\ S).\ (tl\ M,\ S)$
  $\lambda C\ (M,\ N,\ U,\ S).\ (M,\ N,\ add\text{-}mset\ C\ U,\ S)$
  $\lambda C\ (M,\ N,\ U,\ S).\ (M,\ removeAll\text{-}mset\ C\ N,\ removeAll\text{-}mset\ C\ U,\ S)$
  $\lambda D\ (M,\ N,\ U,\ \text{-}).\ (M,\ N,\ U,\ D)$
  $\lambda N.\ ([],\ N,\ \{\#\},\ None)$
  by *unfold-locales auto*

declare *clauses-def*[*simp*]

lemma *reduce-trail-to-empty-trail*[*simp*]:
  $reduce\text{-}trail\text{-}to\ F\ ([],\ aa,\ ab,\ b) = ([],\ aa,\ ab,\ b)$
  using *reduce-trail-to.simps* **by** *auto*

lemma *reduce-trail-to'*:
  $reduce\text{-}trail\text{-}to\ F\ S =$
    $((if\ length\ (raw\text{-}trail\ S) \geq length\ F$
    $then\ drop\ (length\ (raw\text{-}trail\ S) - length\ F)\ (raw\text{-}trail\ S)$
    $else\ []),\ raw\text{-}init\text{-}clss\ S,\ raw\text{-}learned\text{-}clss\ S,\ raw\text{-}conflicting\ S)$
    (**is** *?S* = -)
proof (*induction F S rule*: *reduce-trail-to.induct*)
  case (*1 F S*) **note** *IH* = *this*
  show *?case*
  proof (*cases raw-trail S*)
    case *Nil*
    then show *?thesis* **using** *IH* **by** (*cases S*) *auto*
  next
    case (*Cons L M*)
    then show *?thesis*
      apply (*cases Suc* (*length M*) > *length F*)

prefer *2* **using** *IH reduce-trail-to-length-ne*[*of S F*] **apply** (*cases S*) **apply** *auto*[]
apply (*subgoal-tac Suc* (*length M*) − *length F* = *Suc* (*length M* − *length F*))
using *reduce-trail-to-length-ne*[*of S F*] *IH* **by** (*cases S*) *auto*
qed
qed

Definition of the rules

Types **lemma** *true-raw-init-clss-remdups*[*simp*]:
I ⊨*s* (*mset* ∘ *remdups*) ' *N* ⟷ *I* ⊨*s mset* ' *N*
by (*simp add*: *true-clss-def*)

lemma *true-clss-raw-remdups-mset-mset*[*simp*]:
‹*I* ⊨*s* (λ*L*. *remdups-mset* (*mset L*)) ' *N'* ⟷ *I* ⊨*s mset* ' *N'*›
by (*simp add*: *true-clss-def*)

declare *satisfiable-carac*[*iff del*]
lemma *satisfiable-mset-remdups*[*simp*]:
satisfiable ((*mset* ∘ *remdups*) ' *N*) ⟷ *satisfiable* (*mset* ' *N*)
satisfiable ((λ*L*. *remdups-mset* (*mset L*)) ' *N'*) ⟷ *satisfiable* (*mset* ' *N'*)
unfolding *satisfiable-carac*[*symmetric*] **by** *simp-all*

type-synonym *'v* $cdcl_W$*-restart-state-inv-st* = (*'v*, *'v literal list*) *ann-lit list* ×
'v literal list list × *'v literal list list* × *'v literal list option*

We need some functions to convert between our abstract state *'v* $cdcl_W$*-restart-state* and the concrete state *'v* $cdcl_W$*-restart-state-inv-st*.

fun *convert* :: (*'a*, *'c list*) *ann-lit* ⇒ (*'a*, *'c multiset*) *ann-lit* **where**
convert (*Propagated L C*) = *Propagated L* (*mset C*) |
convert (*Decided K*) = *Decided K*

abbreviation *convertC* :: *'a list option* ⇒ *'a multiset option* **where**
convertC ≡ *map-option mset*

lemma *convert-Propagated*[*elim*!]:
convert z = *Propagated L C* ⟹ (∃ *C'*. *z* = *Propagated L C'* ∧ *C* = *mset C'*)
by (*cases z*) *auto*

lemma *is-decided-convert*[*simp*]: *is-decided* (*convert x*) = *is-decided x*
by (*cases x*) *auto*

lemma *is-decided-convert-is-decided*[*simp*]: ‹(*is-decided* ∘ *convert*) = (*is-decided*)›
by *auto*

lemma *get-level-map-convert*[*simp*]:
get-level (*map convert M*) *x* = *get-level M x*
by (*induction M rule*: *ann-lit-list-induct*) (*auto simp*: *comp-def get-level-def*)

lemma *get-maximum-level-map-convert*[*simp*]:
get-maximum-level (*map convert M*) *D* = *get-maximum-level M D*
by (*induction D*) (*auto simp add*: *get-maximum-level-add-mset*)

lemma *count-decided-convert*[*simp*]:
‹*count-decided* (*map convert M*) = *count-decided M*›
by (*auto simp*: *count-decided-def*)

lemma *atm-lit-of-convert*[*simp*]:
lit-of (*convert x*) = *lit-of x*
by (*cases x*) *auto*

lemma *no-dup-convert*[*simp*]:
⟨*no-dup* (*map convert M*) = *no-dup M*⟩
by (*auto simp*: *no-dup-def image-image comp-def*)

Conversion function

fun *toS* :: $'v$ $cdcl_W$*-restart-state-inv-st* ⇒ $'v$ $cdcl_W$*-restart-state* **where**
toS (*M*, *N*, *U*, *C*) = (*map convert M*, *mset* (*map mset N*), *mset* (*map mset U*), *convertC C*)

Definition an abstract type

typedef $'v$ $cdcl_W$*-restart-state-inv* = {*S*::$'v$ $cdcl_W$*-restart-state-inv-st*. $cdcl_W$*-all-struct-inv* (*toS S*)}
morphisms *rough-state-of state-of*
proof
show ([],[], [], *None*) ∈ {*S*. $cdcl_W$*-all-struct-inv* (*toS S*)}
by (*auto simp add*: $cdcl_W$*-all-struct-inv-def*)
qed

instantiation $cdcl_W$*-restart-state-inv* :: (*type*) *equal*
begin
definition *equal*-$cdcl_W$*-restart-state-inv* :: $'v$ $cdcl_W$*-restart-state-inv* ⇒
$'v$ $cdcl_W$*-restart-state-inv* ⇒ *bool* **where**
equal-$cdcl_W$*-restart-state-inv S S'* = (*rough-state-of S* = *rough-state-of S'*)
instance
by *standard* (*simp add*: *rough-state-of-inject equal*-$cdcl_W$*-restart-state-inv-def*)
end

lemma *lits-of-map-convert*[*simp*]: *lits-of-l* (*map convert M*) = *lits-of-l M*
by (*induction M rule*: *ann-lit-list-induct*) *simp-all*

lemma *undefined-lit-map-convert*[*iff*]:
undefined-lit (*map convert M*) *L* ⟷ *undefined-lit M L*
by (*auto simp add*: *defined-lit-map image-image*)

lemma *true-annot-map-convert*[*simp*]: *map convert M* ⊨*a N* ⟷ *M* ⊨*a N*
by (*simp-all add*: *true-annot-def image-image lits-of-def*)

lemma *true-annots-map-convert*[*simp*]: *map convert M* ⊨*as N* ⟷ *M* ⊨*as N*
unfolding *true-annots-def* **by** *auto*

lemmas *propagateE*
lemma *find-first-unit-clause-some-is-propagate*:
assumes *H*: *find-first-unit-clause* (*N* @ *U*) *M* = *Some* (*L*, *C*)
shows *propagate* (*toS* (*M*, *N*, *U*, *None*)) (*toS* (*Propagated L C* # *M*, *N*, *U*, *None*))
using *assms*
by (*auto dest*!: *find-first-unit-clause-some simp add*: *propagate.simps*
intro!: *exI*[*of - mset C* − {#*L*#}])

The Transitions

Propagate **definition** *do-propagate-step* :: ⟨$'v$ $cdcl_W$*-restart-state-inv-st* ⇒ $'v$ $cdcl_W$*-restart-state-inv-st*⟩
where
do-propagate-step S =

```
  (case S of
    (M, N, U, None) ⇒
      (case find-first-unit-clause (N @ U) M of
        Some (L, C) ⇒ (Propagated L C # M, N, U, None)
      | None ⇒ (M, N, U, None))
  | S ⇒ S)

lemma do-propagate-step:
  do-propagate-step S ≠ S ⟹ propagate (toS S) (toS (do-propagate-step S))
  apply (cases S, cases raw-conflicting S)
  using find-first-unit-clause-some-is-propagate[of raw-init-clss S raw-learned-clss S raw-trail S]
  by (auto simp add: do-propagate-step-def split: option.splits)

lemma do-propagate-step-option[simp]:
  raw-conflicting S ≠ None ⟹ do-propagate-step S = S
  unfolding do-propagate-step-def by (cases S, cases raw-conflicting S) auto

lemma do-propagate-step-no-step:
  assumes prop-step: do-propagate-step S = S
  shows no-step propagate (toS S)
proof (standard, standard)
  fix T
  assume propagate (toS S) T
  then obtain M N U C L E where
    toSS: toS S = (M, N, U, None) and
    LE: L ∈# E and
    T: T = (Propagated L E # M, N, U, None) and
    MC: M ⊨as CNot C and
    undef: undefined-lit M L and
    CL: C + {#L#} ∈# N + U
    apply − by (cases toS S) (auto elim!: propagateE)
  let ?M = raw-trail S
  let ?N = raw-init-clss S
  let ?U = raw-learned-clss S
  let ?D = None
  have S: S = (?M, ?N, ?U, ?D)
    using toSS by (cases S, cases raw-conflicting S) simp-all
  have S: toS S = toS (?M, ?N, ?U, ?D)
    unfolding S[symmetric] by simp

  have
    M: M = map convert ?M and
    N: N = mset (map mset ?N) and
    U: U = mset (map mset ?U)
    using toSS[unfolded S] by auto

  obtain D where
    DCL: mset D = C + {#L#} and
    D: D ∈ set (?N @ ?U)
    using CL unfolding N U by auto
  obtain C' L' where
    setD: set D = set (L' # C') and
    C': mset C' = C and
    L: L = L'
    using DCL by (metis add-mset-add-single ex-mset list.simps(15) set-mset-add-mset-insert
      set-mset-mset)
```

have *find-first-unit-clause* (*?N* @ *?U*) *?M* $\neq$ *None*
apply (*rule find-first-unit-clause-none*[*of D ?N* @ *?U ?M L*, *OF D*])
using *MC setD DCL M MC* **unfolding** *C'*[*symmetric*] **apply** *auto*[*1*]
using *M undef* **apply** *auto*[*1*]
unfolding *setD L* **by** *auto*
then show *False* **using** *prop-step S* **unfolding** *do-propagate-step-def* **by** (*cases S*) *auto*
qed

Conflict **fun** *find-conflict* **where**
find-conflict M [] = *None* |
find-conflict M (*N* # *Ns*) = (*if* ($\forall$ *c* $\in$ *set N*. $-c$ $\in$ *lits-of-l M*) *then Some N else find-conflict M Ns*)

lemma *find-conflict-Some*:
find-conflict M Ns = *Some N* $\Longrightarrow$ *N* $\in$ *set Ns* $\wedge$ *M* $\models$*as CNot* (*mset N*)
by (*induction Ns rule*: *find-conflict.induct*)
(*auto split*: *if-split-asm simp*: *lits-of-l-unfold*)

lemma *find-conflict-None*:
find-conflict M Ns = *None* $\longleftrightarrow$ ($\forall$ *N* $\in$ *set Ns*. $\neg$*M* $\models$*as CNot* (*mset N*))
by (*induction Ns*) (*auto simp*: *lits-of-l-unfold*)

lemma *find-conflict-None-no-confl*:
find-conflict M (*N*@*U*) = *None* $\longleftrightarrow$ *no-step conflict* (*toS* (*M*, *N*, *U*, *None*))
by (*auto simp add*: *find-conflict-None conflict.simps*)

definition *do-conflict-step* :: ‹*'v* $cdcl_W$*-restart-state-inv-st* $\Rightarrow$ *'v* $cdcl_W$*-restart-state-inv-st*› **where**
do-conflict-step S =
(*case S of*
(*M*, *N*, *U*, *None*) $\Rightarrow$
(*case find-conflict M* (*N* @ *U*) *of*
Some a $\Rightarrow$ (*M*, *N*, *U*, *Some a*)
| *None* $\Rightarrow$ (*M*, *N*, *U*, *None*))
| *S* $\Rightarrow$ *S*)

lemma *do-conflict-step*:
do-conflict-step S $\neq$ *S* $\Longrightarrow$ *conflict* (*toS S*) (*toS* (*do-conflict-step S*))
apply (*cases S*, *cases raw-conflicting S*)
unfolding *conflict.simps do-conflict-step-def*
by (*auto dest*!:*find-conflict-Some split*: *option.splits*)

lemma *do-conflict-step-no-step*:
do-conflict-step S = *S* $\Longrightarrow$ *no-step conflict* (*toS S*)
apply (*cases S*, *cases raw-conflicting S*)
unfolding *do-conflict-step-def*
using *find-conflict-None-no-confl*[*of raw-trail S raw-init-clss S raw-learned-clss S*]
by (*auto split*: *option.splits elim*!: *conflictE*)

lemma *do-conflict-step-option*[*simp*]:
raw-conflicting S $\neq$ *None* $\Longrightarrow$ *do-conflict-step S* = *S*
unfolding *do-conflict-step-def* **by** (*cases S*, *cases raw-conflicting S*) *auto*

lemma *do-conflict-step-raw-conflicting*[*dest*]:
do-conflict-step S $\neq$ *S* $\Longrightarrow$ *raw-conflicting* (*do-conflict-step S*) $\neq$ *None*
unfolding *do-conflict-step-def* **by** (*cases S*, *cases raw-conflicting S*) (*auto split*: *option.splits*)

definition *do-cp-step* **where**

do-cp-step S =
  (*do-propagate-step o do-conflict-step*) *S*

lemma $cdcl_W$*-all-struct-inv-rough-state*[*simp*]: $cdcl_W$*-all-struct-inv* (*toS* (*rough-state-of S*))
  using *rough-state-of* **by** *auto*

lemma [*simp*]: $cdcl_W$*-all-struct-inv* (*toS S*) $\Longrightarrow$ *rough-state-of* (*state-of S*) = *S*
  by (*simp add*: *state-of-inverse*)

Skip **fun** *do-skip-step* :: *'v* $cdcl_W$*-restart-state-inv-st* ⇒ *'v* $cdcl_W$*-restart-state-inv-st* **where**
do-skip-step (*Propagated L C* # *Ls*, *N*, *U*, *Some D*) =
  (*if* −*L* ∉ *set D* ∧ *D* ≠ []
  then (*Ls*, *N*, *U*, *Some D*)
  else (*Propagated L C* #*Ls*, *N*, *U*, *Some D*)) |
do-skip-step S = *S*

lemma *do-skip-step*:
  do-skip-step S ≠ *S* $\Longrightarrow$ *skip* (*toS S*) (*toS* (*do-skip-step S*))
  apply (*induction S rule*: *do-skip-step.induct*)
  by (*auto simp add*: *skip.simps*)

lemma *do-skip-step-no*:
  do-skip-step S = *S* $\Longrightarrow$ *no-step skip* (*toS S*)
  by (*induction S rule*: *do-skip-step.induct*)
    (*auto simp add*: *other split*: *if-split-asm elim*: *skipE*)

lemma *do-skip-step-raw-trail-is-None*[*iff*]:
  do-skip-step S = (*a*, *b*, *c*, *None*) ⟷ *S* = (*a*, *b*, *c*, *None*)
  by (*cases S rule*: *do-skip-step.cases*) *auto*

Resolve **fun** *maximum-level-code*:: *'a literal list* ⇒ (*'a*, *'a literal list*) *ann-lit list* ⇒ *nat*
  where
maximum-level-code [] - = *0* |
maximum-level-code (*L* # *Ls*) *M* = *max* (*get-level M L*) (*maximum-level-code Ls M*)

lemma *maximum-level-code-eq-get-maximum-level*[*code*, *simp*]:
  maximum-level-code D M = *get-maximum-level M* (*mset D*)
  by (*induction D*) (*auto simp add*: *get-maximum-level-add-mset*)

fun *do-resolve-step* :: *'v* $cdcl_W$*-restart-state-inv-st* ⇒ *'v* $cdcl_W$*-restart-state-inv-st* **where**
do-resolve-step (*Propagated L C* # *Ls*, *N*, *U*, *Some D*) =
  (*if* −*L* ∈ *set D* ∧ *maximum-level-code* (*remove1* (−*L*) *D*) (*Propagated L C* # *Ls*) = *count-decided Ls*
  then (*Ls*, *N*, *U*, *Some* (*remdups* (*remove1 L C* @ *remove1* (−*L*) *D*)))
  else (*Propagated L C* # *Ls*, *N*, *U*, *Some D*)) |
do-resolve-step S = *S*

lemma *do-resolve-step*:
  $cdcl_W$*-all-struct-inv* (*toS S*) $\Longrightarrow$ *do-resolve-step S* ≠ *S*
  $\Longrightarrow$ *resolve* (*toS S*) (*toS* (*do-resolve-step S*))
proof (*induction S rule*: *do-resolve-step.induct*)
  case (*1 L C M N U D*)
  then have
    − *L* ∈ *set D* **and**
    M: *maximum-level-code* (*remove1* (−*L*) *D*) (*Propagated L C* # *M*) = *count-decided M*
    by (*cases mset D* − {#− *L*#} = {#},

auto dest!: *get-maximum-level-exists-lit-of-max-level*[*of - Propagated L C # M*]
split: *if-split-asm*)+
have *every-mark-is-a-conflict* (*toS* (*Propagated L C # M*, *N*, *U*, *Some D*))
using *1*(*1*) **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-conflicting-def* **by** *fast*
then have $L \in$ *set C* **by** *fastforce*
then obtain C' **where** *C*: *mset C* = *add-mset L* C'
by (*metis in-multiset-in-set insert-DiffM*)
obtain D' **where** *D*: *mset D* = *add-mset* ($-L$) D'
using ‹$-$ $L \in$ *set D*› **by** (*metis in-multiset-in-set insert-DiffM*)
have $D'L$: D' + {#$-$ L#} $-$ {#$-$L#} = D' **by** (*auto simp add*: *multiset-eq-iff*)

have *CL*: *mset C* $-$ {#L#} + {#L#} = *mset C* **using** ‹$L \in$ *set C*› **by** (*auto simp add*: *multiset-eq-iff*)
have *get-maximum-level* (*Propagated L* (C' + {#L#}) # *map convert M*) D' = *count-decided M*
using *M*[*simplified*] **unfolding** *maximum-level-code-eq-get-maximum-level C*[*symmetric*] *CL*
by (*metis D D'L* ‹*add-mset L C'* = *mset C*› *add-mset-add-single convert.simps*(*1*)
get-maximum-level-map-convert list.simps(*9*))
then have
resolve
(*map convert* (*Propagated L C # M*), *mset* ‘# *mset N*, *mset* ‘# *mset U*, *Some* (*mset D*))
(*map convert M*, *mset* ‘# *mset N*, *mset* ‘# *mset U*,
Some (((*mset D* $-$ {#$-$L#}) ∪# (*mset C* $-$ {#L#}))))
unfolding *resolve.simps*
by (*simp add*: *C D*)
moreover have
(*map convert* (*Propagated L C # M*), *mset* ‘# *mset N*, *mset* ‘# *mset U*, *Some* (*mset D*))
= *toS* (*Propagated L C # M*, *N*, *U*, *Some D*)
by *auto*
moreover
have *distinct-mset* (*mset C*) **and** *distinct-mset* (*mset D*)
using ‹$cdcl_W$*-all-struct-inv* (*toS* (*Propagated L C # M*, *N*, *U*, *Some D*))›
unfolding $cdcl_W$*-all-struct-inv-def distinct-*$cdcl_W$*-state-def*
by *auto*
then have (*mset C* $-$ {#L#}) ∪# (*mset D* $-$ {#$-$ L#}) =
remdups-mset (*mset C* $-$ {#L#} + (*mset D* $-$ {#$-$ L#}))
by (*auto simp*: *distinct-mset-rempdups-union-mset*)
then have (*map convert M*, *mset* ‘# *mset N*, *mset* ‘# *mset U*,
Some ((*mset D* $-$ {#$-$ L#}) ∪# (*mset C* $-$ {#L#})))
= *toS* (*do-resolve-step* (*Propagated L C # M*, *N*, *U*, *Some D*))
using ‹$-$ $L \in$ *set D*› *M* **by** (*auto simp*: *ac-simps*)
ultimately show *?case*
by *simp*
qed *auto*

lemma *do-resolve-step-no*:
do-resolve-step S = *S* $\Longrightarrow$ *no-step resolve* (*toS S*)
apply (*cases S*; *cases hd* (*raw-trail S*);*cases raw-trail S*; *cases raw-conflicting S*)
by (*auto*
elim!: *resolveE split*: *if-split-asm*
dest!: *union-single-eq-member*
simp del: *in-multiset-in-set get-maximum-level-map-convert*
simp: *get-maximum-level-map-convert*[*symmetric*] *count-decided-def*)

lemma *rough-state-of-state-of-resolve*[*simp*]:
$cdcl_W$*-all-struct-inv* (*toS S*) $\Longrightarrow$
rough-state-of (*state-of* (*do-resolve-step S*)) = *do-resolve-step S*
apply (*rule state-of-inverse*)

apply (*cases do-resolve-step S = S*)
 apply (*simp*; *fail*)
by (*metis* (*mono-tags*, *lifting*) *bj* $cdcl_W$*-all-struct-inv-inv do-resolve-step mem-Collect-eq other resolve*)

lemma *do-resolve-step-raw-trail-is-None*[*iff*]:
do-resolve-step S = (*a*, *b*, *c*, *None*) $\longleftrightarrow$ *S* = (*a*, *b*, *c*, *None*)
by (*cases S rule*: *do-resolve-step.cases*) *auto*

Backjumping **lemma** *get-all-ann-decomposition-map-convert*:
(*get-all-ann-decomposition* (*map convert M*)) =
map (λ(*a*, *b*). (*map convert a*, *map convert b*)) (*get-all-ann-decomposition M*)
apply (*induction M rule*: *ann-lit-list-induct*)
 apply *simp*
by (*rename-tac L xs*, *case-tac get-all-ann-decomposition xs*; *auto*)+

lemma *do-backtrack-step*:
assumes
db: *do-backtrack-step S* $\neq$ *S* **and**
inv: $cdcl_W$*-all-struct-inv* (*toS S*)
shows *backtrack* (*toS S*) (*toS* (*do-backtrack-step S*))
proof (*cases S*, *cases raw-conflicting S*, *goal-cases*)
case (*1 M N U E*)
then show *?case* **using** *db* **by** *auto*
next
case (*2 M N U E C*) **note** *S* = *this*(*1*) **and** *confl* = *this*(*2*)
have *E*: *E* = *Some C* **using** *S confl* **by** *auto*

obtain *L j* **where** *fd*: *find-level-decomp M C* [] (*count-decided M*) = *Some* (*L*, *j*)
using *db* **unfolding** *S E* **by** (*cases C*) (*auto split*: *if-split-asm option.splits list.splits annotated-lit.splits*)
have
$L \in$ *set C* **and**
j: *get-maximum-level M* (*mset* (*remove1 L C*)) = *j* **and**
levL: *get-level M L* = *count-decided M*
using *find-level-decomp-some*[*OF fd*] **by** *auto*
obtain *C′* **where** *C*: *mset C* = *add-mset L* (*mset C′*)
using ‹$L \in$ *set C*› **by** (*metis ex-mset in-multiset-in-set insert-DiffM*)
obtain *M2* **where** *M2*: *bt-cut j M* = *Some M2*
using *db fd* **unfolding** *S E* **by** (*auto split*: *option.splits*)
have *no-dup M*
using *inv* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def S*
by (*auto simp*: *comp-def*)
then obtain *M1 K c* **where**
M1: *M2* = *Decided K* # *M1* **and** *lev-K*: *get-level M K* = *j* + *1* **and**
c: *M* = *c* @ *M2*
using *bt-cut-some-decomp*[*OF - M2*] **by** (*cases M2*) *auto*
have *j* $\leq$ *count-decided M* **unfolding** *c j*[*symmetric*]
by (*metis* (*mono-tags*, *lifting*) *count-decided-ge-get-maximum-level*)
have *max-l-j*: *maximum-level-code C′ M* = *j*
using *db fd M2 C* **unfolding** *S E* **by** (*auto*
split: *option.splits list.splits annotated-lit.splits*
dest!: *find-level-decomp-some*)[*1*]
have *get-maximum-level M* (*mset C*) $\geq$ *count-decided M*
using ‹$L \in$ *set C*› *levL get-maximum-level-ge-get-level* **by** (*metis set-mset-mset*)
moreover have *get-maximum-level M* (*mset C*) $\leq$ *count-decided M*

using *count-decided-ge-get-maximum-level* **by** *blast*
ultimately have *max-lev-count-dec*: *get-maximum-level M* (*mset C*) = *count-decided M* **by** *auto*

have *clss-C*: ‹*clauses* (*toS S*) ⊨*pm mset C*› **and**
M-C: ‹*M* ⊨*as CNot* (*mset C*)› **and**
lev-inv: $cdcl_W$*-M-level-inv* (*toS S*)
using *inv* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-learned-clause-alt-def S E*
$cdcl_W$*-conflicting-def*
by *auto*
obtain *M2'* **where** *M2'*: (*M2*, *M2'*) ∈ *set* (*get-all-ann-decomposition M*)
using *bt-cut-in-get-all-ann-decomposition*[*OF* ‹*no-dup M*› *M2*] **by** *metis*
have *decomp*:
(*Decided K* # (*map convert M1*),
(*map convert M2'*)) ∈
set (*get-all-ann-decomposition* (*map convert M*))
using *imageI*[*of* - - λ(*a*, *b*). (*map convert a*, *map convert b*), *OF M2'*] *j*
unfolding *S E M1* **by** (*simp add*: *get-all-ann-decomposition-map-convert*)
have *decomp'*:
(*Decided K* # (*map convert M1*),
(*map convert M2'*)) ∈
set (*get-all-ann-decomposition* (*raw-trail* (*toS S*)))
using *imageI*[*of* - - λ(*a*, *b*). (*map convert a*, *map convert b*), *OF M2'*] *j*
unfolding *S E M1* **by** (*simp add*: *get-all-ann-decomposition-map-convert*)

show *?case*
apply (*rule* $backtrack_W$*-rule*[*of* ‹*toS S*› *L* ‹*remove1-mset L* (*mset C*)› *K* ‹*map convert M1*› ‹*map convert M2'*›
j])
subgoal using ‹*L* ∈ *set C*› **unfolding** *S E M1* **by** *auto*
subgoal using *M2' decomp* **unfolding** *S* **by** *auto*
subgoal using *levL* **unfolding** *S E M1* **by** *auto*
subgoal using ‹*L* ∈ *set C*› *levL* ‹*get-maximum-level M* (*mset C*) = *count-decided M*›
unfolding *S E M1* **by** *auto*
subgoal using *j* **unfolding** *S E M1* **by** *auto*
subgoal using ‹*L* ∈ *set C*› *lev-K* **unfolding** *S E M1* **by** *auto*
subgoal using *S confl fd M2 M1 decomp* ‹*L* ∈ *set C*› **by** (*auto simp*: *reduce-trail-to' M2 c*)
subgoal using *inv* **unfolding** $cdcl_W$*-all-struct-inv-def S* **by** *fast*
subgoal using *inv* **unfolding** $cdcl_W$*-all-struct-inv-def S* **by** *fast*
subgoal using *inv* **unfolding** $cdcl_W$*-all-struct-inv-def S* **by** *fast*
done
qed

lemma *map-eq-list-length*:
map f L = *L'* ⟹ *length L* = *length L'*
by *auto*

lemma *map-mmset-of-mlit-eq-cons*:
assumes *map convert M* = *a* @ *c*
obtains *a' c'* **where**
M = *a'* @ *c'* **and**
a = *map convert a'* **and**
c = *map convert c'*
using *that*[*of take* (*length a*) *M drop* (*length a*) *M*]
assms **by** (*metis append-eq-conv-conj append-take-drop-id drop-map take-map*)

lemma *Decided-convert-iff*:

Decided K = *convert za* ⟷ *za* = *Decided K*
by (*cases za*) *auto*

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

lemma *do-backtrack-step-no*:
assumes
db: *do-backtrack-step S* = *S* **and**
inv: $cdcl_W$*-all-struct-inv* (*toS S*) **and**
ns: ‹*no-step skip* (*toS S*)› ‹*no-step resolve* (*toS S*)›
shows *no-step backtrack* (*toS S*)
proof (*rule ccontr*, *cases S*, *cases raw-conflicting S*, *goal-cases*)
case *1*
then show *?case* **using** *db* **by** (*auto split*: *option.splits elim*: *backtrackE*)
next
case (*2 M N U E C*) **note** *bt* = *this*(*1*) **and** *S* = *this*(*2*) **and** *confl* = *this*(*3*)
have *E*: *E* = *Some C* **using** *S confl* **by** *auto*
obtain *T'* **where** ‹*simple-backtrack* (*toS S*) *T'*›
using *no-analyse-backtrack-Ex-simple-backtrack*[*of* ‹*toS S*›]
bt inv ns **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *meson*
then obtain *K j M1 M2 L D* **where**
CE: *map-option mset* (*raw-conflicting S*) = *Some* (*add-mset L D*) **and**
decomp: (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*raw-trail S*)) **and**
levL: *get-level* (*raw-trail S*) *L* = *count-decided* (*raw-trail* (*toS S*)) **and**
k: *get-level* (*raw-trail S*) *L* = *get-maximum-level* (*raw-trail S*) (*add-mset L D*) **and**
j: *get-maximum-level* (*raw-trail S*) *D* ≡ *j* **and**
lev-K: *get-level* (*raw-trail S*) *K* = *Suc j*
apply *clarsimp*
apply (*elim simple-backtrackE*)
apply (*cases S*)
by (*auto simp add*: *get-all-ann-decomposition-map-convert reduce-trail-to*
Decided-convert-iff)
obtain *c* **where** *c*: *raw-trail S* = *c* @ *M2* @ *Decided K* # *M1*
using *decomp* **by** *blast*
have *n-d*: *no-dup M*
using *inv S* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def*
by (*auto simp*: *comp-def*)
then have *count-decided* (*raw-trail* (*toS S*)) > *j*
using *j count-decided-ge-get-maximum-level*[*of raw-trail S D*]
count-decided-ge-get-level[*of raw-trail S K*] *decomp lev-K*
unfolding *k S* **by** (*auto simp*: *get-all-ann-decomposition-map-convert*)
have *CD*: *mset C* = *add-mset L D*
using *CE confl* **by** *auto*
then have *L-C*: ‹*L* ∈ *set C*›
using *set-mset-mset* **by** *fastforce*
have *find-level-decomp M C* [] (*count-decided* (*raw-trail* (*toS S*))) ≠ *None*
apply *rule*
apply (*drule find-level-decomp-none*[*of* - - - - *L* ‹*remove1 L C*›])
using *L-C CD* ‹*count-decided* (*raw-trail* (*toS S*)) > *j*› *mset-eq-setD S levL* **unfolding** *k*[*symmetric*]
j[*symmetric*]
by (*auto simp*: *ac-simps*)

then obtain *L' j'* **where** *fd-some*: *find-level-decomp M C* [] (*count-decided* (*raw-trail* (*toS S*))) = *Some* (*L'*, *j'*)
by (*cases find-level-decomp M C* [] (*count-decided* (*raw-trail* (*toS S*)))) *auto*
have *L'*: *L'* = *L*

proof (*rule ccontr*)
  assume ¬ *?thesis*
  then have $L' \in\# D$
    using *fd-some find-level-decomp-some set-mset-mset*
    by (*metis CD insert-iff set-mset-add-mset-insert*)
  then have *get-level M L'* ≤ *get-maximum-level M D*
    using *get-maximum-level-ge-get-level* **by** *blast*
  then show *False*
    using ‹*count-decided* (*raw-trail* (*toS S*)) > *j*› *j*
      find-level-decomp-some[*OF fd-some*] *S* **by** *auto*
qed
then have *j'*: *j'* = *j* **using** *find-level-decomp-some*[*OF fd-some*] *j S CD* **by** *auto*

obtain *c' M1'* **where** *cM*: *M* = *c'* @ *Decided K* # *M1'*
  apply (*rule map-mmset-of-mlit-eq-cons*[*of M map convert* (*c* @ *M2*)
    map convert (*Decided K* # *M1*)])
    using *c S* **apply** *simp*
  apply (*rule map-mmset-of-mlit-eq-cons*[*of - map convert* [*Decided K*] *map convert M1*])
   apply *auto*[]
  apply (*rename-tac a b' aa b*, *case-tac aa*)
   apply *auto*[]
  apply (*rename-tac a b' aa b*, *case-tac aa*)
  by *auto*
have *btc-none*: *bt-cut j M* ≠ *None*
  apply (*rule bt-cut-not-none*[*of M*])
    using *n-d cM S lev-K S* **apply** *blast+*
  using *lev-K S* **by** *auto*
show *?case* **using** *db n-d fd-some L' j' btc-none* **unfolding** *S E*
  by (*auto dest*: *bt-cut-some-decomp*)
qed

lemma *rough-state-of-state-of-backtrack*[*simp*]:
  assumes *inv*: $cdcl_W$*-all-struct-inv* (*toS S*)
  shows *rough-state-of* (*state-of* (*do-backtrack-step S*)) = *do-backtrack-step S*
proof (*rule state-of-inverse*)
  consider
    (*step*) *backtrack* (*toS S*) (*toS* (*do-backtrack-step S*)) |
     (*0*) *do-backtrack-step S* = *S*
    using *do-backtrack-step inv* **by** *blast*
  then show *do-backtrack-step S* ∈ {*S*. $cdcl_W$*-all-struct-inv* (*toS S*)}
  proof *cases*
    case *0*
    thus *?thesis* **using** *inv* **by** *simp*
  next
    case *step*
    then show *?thesis*
      using *inv*
      by (*auto dest*!: $cdcl_W$*-restart.other* $cdcl_W$*-o.bj* $cdcl_W$*-bj.backtrack* *intro*: $cdcl_W$*-all-struct-inv-inv*)
  qed
qed

Decide **fun** *do-decide-step* **where**
do-decide-step (*M*, *N*, *U*, *None*) =
  (*case find-first-unused-var N* (*lits-of-l M*) *of*
    None ⇒ (*M*, *N*, *U*, *None*)
  | *Some L* ⇒ (*Decided L* # *M*, *N*, *U*, *None*)) |

do-decide-step S = S

lemma *do-decide-step*:
do-decide-step S ≠ S ⟹ decide (toS S) (toS (do-decide-step S))
apply (*cases S*, *cases raw-conflicting S*)
defer
apply (*auto split*: *option.splits simp add*: *decide.simps*
dest: *find-first-unused-var-undefined find-first-unused-var-Some*
intro: *atms-of-atms-of-ms-mono*)[*1*]
proof −
fix *a* :: (*'a*, *'a literal list*) *ann-lit list* **and**
b :: *'a literal list list* **and** *c* :: *'a literal list list* **and**
e :: *'a literal list option*
{
fix *a* :: (*'a*, *'a literal list*) *ann-lit list* **and**
b :: *'a literal list list* **and** *c* :: *'a literal list list* **and**
x2 :: *'a literal* **and** *m* :: *'a literal list*
assume *a1*: *m ∈ set b*
assume *x2 ∈ set m*
then have *f2*: *atm-of x2 ∈ atms-of* (*mset m*)
by *simp*
have ⋀*f*. (*f m*::*'a literal multiset*) ∈ *f ‘ set b*
using *a1* **by** *blast*
then have ⋀*f*. (*atms-of* (*f m*)::*'a set*) ⊆ *atms-of-ms* (*f ‘ set b*)
using *atms-of-atms-of-ms-mono* **by** *blast*
then have ⋀*n f*. (*n*::*'a*) ∈ *atms-of-ms* (*f ‘ set b*) ∨ *n ∉ atms-of* (*f m*)
by (*meson contra-subsetD*)
then have *atm-of x2 ∈ atms-of-ms* (*mset ‘ set b*)
using *f2* **by** *blast*
} **note** *H = this*
{
fix *m* :: *'a literal list* **and** *x2*
have *m ∈ set b ⟹ x2 ∈ set m ⟹ x2 ∉ lits-of-l a ⟹ − x2 ∉ lits-of-l a ⟹*
∃ aa∈set b. ¬ atm-of ‘ set aa ⊆ atm-of ‘ lits-of-l a
by (*meson atm-of-in-atm-of-set-in-uminus contra-subsetD rev-image-eqI*)
} **note** *H' = this*

assume *do-decide-step S ≠ S* **and**
S = (*a*, *b*, *c*, *e*) **and**
raw-conflicting S = None
then show *decide* (*toS S*) (*toS* (*do-decide-step S*))
using *H H'* **by** (*auto split*: *option.splits simp*: *decide.simps defined-lit-map lits-of-def*
image-image atm-of-eq-atm-of dest!: *find-first-unused-var-Some*)
qed

lemma *do-decide-step-no*:
do-decide-step S = S ⟹ no-step decide (*toS S*)
apply (*cases S*, *cases raw-conflicting S*)
apply (*auto simp*: *atms-of-ms-mset-unfold Decided-Propagated-in-iff-in-lits-of-l lits-of-def*
dest!: *atm-of-in-atm-of-set-in-uminus*
elim!: *decideE*
split: *option.splits*)+
using *atm-of-eq-atm-of* **by** *blast*+

lemma *rough-state-of-state-of-do-decide-step*[*simp*]:

$cdcl_W$*-all-struct-inv* (*toS S*) $\Longrightarrow$ *rough-state-of* (*state-of* (*do-decide-step S*)) = *do-decide-step S*
proof (*subst state-of-inverse*, *goal-cases*)
case *1*
then show *?case*
by (*cases do-decide-step S* = *S*)
(*auto dest*: *do-decide-step decide other intro*: $cdcl_W$*-all-struct-inv-inv*)
qed *simp*

lemma *rough-state-of-state-of-do-skip-step*[*simp*]:
$cdcl_W$*-all-struct-inv* (*toS S*) $\Longrightarrow$ *rough-state-of* (*state-of* (*do-skip-step S*)) = *do-skip-step S*
apply (*subst state-of-inverse*, *cases do-skip-step S* = *S*)
apply *simp*
by (*blast dest*: *other skip bj do-skip-step* $cdcl_W$*-all-struct-inv-inv*)+

Code generation

Type definition There are two invariants: one while applying conflict and propagate and one for the other rules

declare *rough-state-of-inverse*[*simp add*]
definition *Con* **where**
Con xs = *state-of* (*if* $cdcl_W$*-all-struct-inv* (*toS* (*fst xs*, *snd xs*)) *then xs*
else ([], [], [], *None*))

lemma [*code abstype*]:
Con (*rough-state-of S*) = *S*
using *rough-state-of*[*of S*] **unfolding** *Con-def* **by** *simp*

definition *do-cp-step'* **where**
do-cp-step' S = *state-of* (*do-cp-step* (*rough-state-of S*))

typedef *'v* $cdcl_W$*-restart-state-inv-from-init-state* =
{*S*:: *'v* $cdcl_W$*-restart-state-inv-st*. $cdcl_W$*-all-struct-inv* (*toS S*)
$\wedge$ $cdcl_W$*-stgy*** (*S0-*$cdcl_W$*-restart* (*raw-init-clss* (*toS S*))) (*toS S*)}
morphisms *rough-state-from-init-state-of state-from-init-state-of*
proof
show ([],[], [], *None*) $\in$ {*S*. $cdcl_W$*-all-struct-inv* (*toS S*)
$\wedge$ $cdcl_W$*-stgy*** (*S0-*$cdcl_W$*-restart* (*raw-init-clss* (*toS S*))) (*toS S*)}
by (*auto simp add*: $cdcl_W$*-all-struct-inv-def*)
qed

instantiation $cdcl_W$*-restart-state-inv-from-init-state* :: (*type*) *equal*
begin
definition *equal-*$cdcl_W$*-restart-state-inv-from-init-state* :: *'v* $cdcl_W$*-restart-state-inv-from-init-state* $\Rightarrow$
'v $cdcl_W$*-restart-state-inv-from-init-state* $\Rightarrow$ *bool* **where**
equal-$cdcl_W$*-restart-state-inv-from-init-state S S'* $\longleftrightarrow$
(*rough-state-from-init-state-of S* = *rough-state-from-init-state-of S'*)
instance
by *standard* (*simp add*: *rough-state-from-init-state-of-inject*
equal-$cdcl_W$*-restart-state-inv-from-init-state-def*)
end

definition *ConI* **where**
ConI S = *state-from-init-state-of* (*if* $cdcl_W$*-all-struct-inv* (*toS* (*fst S*, *snd S*))
$\wedge$ $cdcl_W$*-stgy*** (*S0-*$cdcl_W$*-restart* (*raw-init-clss* (*toS S*))) (*toS S*) *then S else* ([], [], [], *None*))

lemma [*code abstype*]:
ConI (*rough-state-from-init-state-of S*) = *S*
using *rough-state-from-init-state-of*[*of S*] **unfolding** *ConI-def*
by (*simp add*: *rough-state-from-init-state-of-inverse*)

definition *id-of-I-to*:: $'v$ $cdcl_W$*-restart-state-inv-from-init-state* ⇒ $'v$ $cdcl_W$*-restart-state-inv* **where**
id-of-I-to S = *state-of* (*rough-state-from-init-state-of S*)

lemma [*code abstract*]:
rough-state-of (*id-of-I-to S*) = *rough-state-from-init-state-of S*
unfolding *id-of-I-to-def* **using** *rough-state-from-init-state-of*[*of S*] **by** *auto*

lemma *in-clauses-rough-state-of-is-distinct*:
c∈*set* (*raw-init-clss* (*rough-state-of S*) @ *raw-learned-clss* (*rough-state-of S*)) ⟹ *distinct c*
apply (*cases rough-state-of S*)
using *rough-state-of*[*of S*] **by** (*auto simp add*: *distinct-mset-set-distinct* $cdcl_W$*-all-struct-inv-def*
distinct-$cdcl_W$*-state-def*)

The other rules **fun** *do-if-not-equal* **where**
do-if-not-equal [] *S* = *S* |
do-if-not-equal (*f* # *fs*) *S* =
(*let T* = *f S in*
if T ≠ *S then T else do-if-not-equal fs S*)

fun *do-cdcl-step* **where**
do-cdcl-step S =
do-if-not-equal [*do-conflict-step*, *do-propagate-step*, *do-skip-step*, *do-resolve-step*,
do-backtrack-step, *do-decide-step*] *S*

lemma *do-cdcl-step*:
assumes *inv*: $cdcl_W$*-all-struct-inv* (*toS S*) **and**
st: *do-cdcl-step S* ≠ *S*
shows $cdcl_W$*-stgy* (*toS S*) (*toS* (*do-cdcl-step S*))
using *st* **by** (*auto simp add*: *do-skip-step do-resolve-step do-backtrack-step do-decide-step*
do-conflict-step
do-propagate-step do-conflict-step-no-step do-propagate-step-no-step
$cdcl_W$*-stgy.intros* $cdcl_W$*-bj.intros* $cdcl_W$*-o.intros inv Let-def*)

lemma *do-cdcl-step-no*:
assumes *inv*: $cdcl_W$*-all-struct-inv* (*toS S*) **and**
st: *do-cdcl-step S* = *S*
shows *no-step* $cdcl_W$ (*toS S*)
using *st inv* **by** (*auto split*: *if-split-asm elim*: $cdcl_W$*-bjE*
simp add: *Let-def* $cdcl_W$*-bj.simps* $cdcl_W$*.simps do-conflict-step*
do-propagate-step do-conflict-step-no-step do-propagate-step-no-step
elim!: $cdcl_W$*-o.cases*
dest!: *do-skip-step-no do-resolve-step-no do-backtrack-step-no do-decide-step-no*)

lemma *rough-state-of-state-of-do-cdcl-step*[*simp*]:
rough-state-of (*state-of* (*do-cdcl-step* (*rough-state-of S*))) = *do-cdcl-step* (*rough-state-of S*)
proof (*cases do-cdcl-step* (*rough-state-of S*) = *rough-state-of S*)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
have $cdcl_W$ (*toS* (*rough-state-of S*)) (*toS* (*do-cdcl-step* (*rough-state-of S*)))

using *False* $cdcl_W$*-all-struct-inv-rough-state* $cdcl_W$*-stgy-*$cdcl_W$ *do-cdcl-step* **by** *blast*
then have $cdcl_W$*-all-struct-inv* (*toS* (*do-cdcl-step* (*rough-state-of S*)))
using $cdcl_W$*-all-struct-inv-inv* $cdcl_W$*-all-struct-inv-rough-state* $cdcl_W$*-*$cdcl_W$*-restart* **by** *blast*
then show *?thesis*
by (*simp add*: *CollectI state-of-inverse*)
qed

definition *do-*$cdcl_W$*-stgy-step* :: *'v* $cdcl_W$*-restart-state-inv* ⇒ *'v* $cdcl_W$*-restart-state-inv* **where**
do-$cdcl_W$*-stgy-step S* =
state-of (*do-cdcl-step* (*rough-state-of S*))

lemma *rough-state-of-do-*$cdcl_W$*-stgy-step*[*code abstract*]:
rough-state-of (*do-*$cdcl_W$*-stgy-step S*) = *do-cdcl-step* (*rough-state-of S*)
apply (*cases do-cdcl-step* (*rough-state-of S*) = *rough-state-of S*)
unfolding *do-*$cdcl_W$*-stgy-step-def* **apply** *simp*
using *do-cdcl-step*[*of rough-state-of S*] *rough-state-of-state-of-do-cdcl-step* **by** *blast*

definition *do-*$cdcl_W$*-stgy-step'* **where**
do-$cdcl_W$*-stgy-step' S* = *state-from-init-state-of* (*rough-state-of* (*do-*$cdcl_W$*-stgy-step* (*id-of-I-to S*)))

Correction of the transformation **lemma** *do-*$cdcl_W$*-stgy-step*:
assumes *do-*$cdcl_W$*-stgy-step S* ≠ *S*
shows $cdcl_W$*-stgy* (*toS* (*rough-state-of S*)) (*toS* (*rough-state-of* (*do-*$cdcl_W$*-stgy-step S*)))
proof −
have *do-cdcl-step* (*rough-state-of S*) ≠ *rough-state-of S*
by (*metis* (*no-types*) *assms do-*$cdcl_W$*-stgy-step-def rough-state-of-inject*
rough-state-of-state-of-do-cdcl-step)
then have $cdcl_W$*-stgy* (*toS* (*rough-state-of S*)) (*toS* (*do-cdcl-step* (*rough-state-of S*)))
using $cdcl_W$*-all-struct-inv-rough-state do-cdcl-step* **by** *blast*
then show *?thesis*
by (*metis* (*no-types*) *do-*$cdcl_W$*-stgy-step-def rough-state-of-state-of-do-cdcl-step*)
qed

lemma *length-raw-trail-toS*[*simp*]:
length (*raw-trail* (*toS S*)) = *length* (*raw-trail S*)
by (*cases S*) *auto*

lemma *raw-conflicting-noTrue-iff-toS*[*simp*]:
raw-conflicting (*toS S*) ≠ *None* ⟷ *raw-conflicting S* ≠ *None*
by (*cases S*) *auto*

lemma *raw-trail-toS-neq-imp-raw-trail-neq*:
raw-trail (*toS S*) ≠ *raw-trail* (*toS S'*) ⟹ *raw-trail S* ≠ *raw-trail S'*
by (*cases S*, *cases S'*) *auto*

lemma *do-cp-step-neq-raw-trail-increase*:
∃ *c. raw-trail* (*do-cp-step S*) = *c* @ *raw-trail S* ∧ (∀ *m* ∈ *set c*. ¬ *is-decided m*)
by (*cases S*, *cases raw-conflicting S*)
(*auto simp add*: *do-cp-step-def do-conflict-step-def do-propagate-step-def split*: *option.splits*)

lemma *do-cp-step-raw-conflicting*:
raw-conflicting (*rough-state-of S*) ≠ *None* ⟹ *do-cp-step' S* = *S*
unfolding *do-cp-step'-def do-cp-step-def* **by** *simp*

lemma *do-decide-step-not-raw-conflicting-one-more-decide*:
assumes

raw-conflicting S = None **and**
do-decide-step S $\neq$ *S*
shows *Suc* (*length* (*filter is-decided* (*raw-trail S*)))
= *length* (*filter is-decided* (*raw-trail* (*do-decide-step S*)))
using *assms* **by** (*cases S*) (*auto simp*: *Let-def split*: *if-split-asm option.splits*
dest!: *find-first-unused-var-Some-not-all-incl*)

lemma *do-decide-step-not-raw-conflicting-one-more-decide-bt*:
assumes *raw-conflicting S* $\neq$ *None* **and**
do-decide-step S $\neq$ *S*
shows *length* (*filter is-decided* (*raw-trail S*)) < *length* (*filter is-decided* (*raw-trail* (*do-decide-step S*)))
using *assms* **by** (*cases S*, *cases raw-conflicting S*)
(*auto simp add*: *Let-def split*: *if-split-asm option.splits*)

lemma *count-decided-raw-trail-toS*:
count-decided (*raw-trail* (*toS S*)) = *count-decided* (*raw-trail S*)
by (*cases S*) (*auto simp*: *comp-def*)

lemma *rough-state-of-state-of-do-skip-step-rough-state-of*[*simp*]:
rough-state-of (*state-of* (*do-skip-step* (*rough-state-of S*))) = *do-skip-step* (*rough-state-of S*)
using $cdcl_W$*-all-struct-inv-rough-state rough-state-of-state-of-do-skip-step* **by** *blast*

lemma *raw-conflicting-do-resolve-step-iff*[*iff*]:
raw-conflicting (*do-resolve-step S*) = *None* $\longleftrightarrow$ *raw-conflicting S* = *None*
by (*cases S rule*: *do-resolve-step.cases*)
(*auto simp add*: *Let-def split*: *option.splits*)

lemma *raw-conflicting-do-skip-step-iff*[*iff*]:
raw-conflicting (*do-skip-step S*) = *None* $\longleftrightarrow$ *raw-conflicting S* = *None*
by (*cases S rule*: *do-skip-step.cases*)
(*auto simp add*: *Let-def split*: *option.splits*)

lemma *raw-conflicting-do-decide-step-iff*[*iff*]:
raw-conflicting (*do-decide-step S*) = *None* $\longleftrightarrow$ *raw-conflicting S* = *None*
by (*cases S rule*: *do-decide-step.cases*)
(*auto simp add*: *Let-def split*: *option.splits*)

lemma *raw-conflicting-do-backtrack-step-imp*[*simp*]:
do-backtrack-step S $\neq$ *S* $\Longrightarrow$ *raw-conflicting* (*do-backtrack-step S*) = *None*
apply (*cases S rule*: *do-backtrack-step.cases*)
apply (*auto simp add*: *Let-def split*: *option.splits list.splits*
) — TODO splitting should solve the goal
apply (*rename-tac dec tr*)
by (*case-tac dec*) *auto*

lemma *do-skip-step-eq-iff-raw-trail-eq*:
do-skip-step S = *S* $\longleftrightarrow$ *raw-trail* (*do-skip-step S*) = *raw-trail S*
by (*cases S rule*: *do-skip-step.cases*) *auto*

lemma *do-decide-step-eq-iff-raw-trail-eq*:
do-decide-step S = *S* $\longleftrightarrow$ *raw-trail* (*do-decide-step S*) = *raw-trail S*
by (*cases S rule*: *do-decide-step.cases*) (*auto split*: *option.split*)

lemma *do-backtrack-step-eq-iff-raw-trail-eq*:
assumes *no-dup* (*raw-trail S*)
shows *do-backtrack-step S* = *S* $\longleftrightarrow$ *raw-trail* (*do-backtrack-step S*) = *raw-trail S*

using *assms* **apply** (*cases S rule*: *do-backtrack-step.cases*)
apply (*auto split*: *option.split list.splits*
simp: *comp-def*
dest!: *bt-cut-in-get-all-ann-decomposition*) — TODO splitting should solve the goal
apply (*rename-tac dec tr tra*)
by (*case-tac dec*) *auto*

lemma *do-resolve-step-eq-iff-raw-trail-eq*:
do-resolve-step S = *S* ⟷ *raw-trail* (*do-resolve-step S*) = *raw-trail S*
by (*cases S rule*: *do-resolve-step.cases*) *auto*

lemma *do-cdcl*$_W$*-stgy-step-no*:
assumes *S*: *do-cdcl*$_W$*-stgy-step S* = *S*
shows *no-step cdcl*$_W$*-stgy* (*toS* (*rough-state-of S*))
proof −
have *do-cdcl-step* (*rough-state-of S*) = *rough-state-of S*
by (*metis assms rough-state-of-do-cdcl*$_W$*-stgy-step*)
then show *?thesis*
using *cdcl*$_W$*-all-struct-inv-rough-state cdcl*$_W$*-stgy-cdcl*$_W$ *do-cdcl-step-no* **by** *blast*
qed

lemma *toS-rough-state-of-state-of-rough-state-from-init-state-of*[*simp*]:
toS (*rough-state-of* (*state-of* (*rough-state-from-init-state-of S*)))
= *toS* (*rough-state-from-init-state-of S*)
using *rough-state-from-init-state-of*[*of S*] **by** (*auto simp add*: *state-of-inverse*)

lemma *cdcl*$_W$*-stgy-is-rtranclp-cdcl*$_W$*-restart*:
cdcl$_W$*-stgy S T* ⟹ *cdcl*$_W$*-restart*** *S T*
by (*simp add*: *cdcl*$_W$*-stgy-tranclp-cdcl*$_W$*-restart rtranclp-unfold*)

lemma *cdcl*$_W$*-stgy-init-raw-init-clss*:
cdcl$_W$*-stgy S T* ⟹ *cdcl*$_W$*-M-level-inv S* ⟹ *raw-init-clss S* = *raw-init-clss T*
using *cdcl*$_W$*-stgy-no-more-init-clss* **by** *blast*

lemma *clauses-toS-rough-state-of-do-cdcl*$_W$*-stgy-step*[*simp*]:
raw-init-clss (*toS* (*rough-state-of* (*do-cdcl*$_W$*-stgy-step* (*state-of* (*rough-state-from-init-state-of S*)))))
= *raw-init-clss* (*toS* (*rough-state-from-init-state-of S*)) (**is** - = *raw-init-clss* (*toS ?S*))
apply (*cases do-cdcl*$_W$*-stgy-step* (*state-of ?S*) = *state-of ?S*)
apply *simp*
by (*metis cdcl*$_W$*-stgy-no-more-init-clss do-cdcl*$_W$*-stgy-step*
toS-rough-state-of-state-of-rough-state-from-init-state-of)

lemma *rough-state-from-init-state-of-do-cdcl*$_W$*-stgy-step'*[*code abstract*]:
rough-state-from-init-state-of (*do-cdcl*$_W$*-stgy-step' S*) =
rough-state-of (*do-cdcl*$_W$*-stgy-step* (*id-of-I-to S*))
proof −
let *?S* = (*rough-state-from-init-state-of S*)
have *cdcl*$_W$*-stgy*** (*S0-cdcl*$_W$*-restart* (*raw-init-clss* (*toS* (*rough-state-from-init-state-of S*))))
(*toS* (*rough-state-from-init-state-of S*))
using *rough-state-from-init-state-of*[*of S*] **by** *auto*
moreover have *cdcl*$_W$*-stgy***
(*toS* (*rough-state-from-init-state-of S*))
(*toS* (*rough-state-of* (*do-cdcl*$_W$*-stgy-step*
(*state-of* (*rough-state-from-init-state-of S*)))))
using *do-cdcl*$_W$*-stgy-step*[*of state-of ?S*]

by (*cases* $do\text{-}cdcl_W\text{-}stgy\text{-}step$ (*state-of ?S*) = *state-of ?S*) *auto*
ultimately show *?thesis*
unfolding $do\text{-}cdcl_W\text{-}stgy\text{-}step'\text{-}def$ *id-of-I-to-def*
by (*auto intro*!: *state-from-init-state-of-inverse*)
qed

All rules together **function** $do\text{-}all\text{-}cdcl_W\text{-}stgy$ **where**
$do\text{-}all\text{-}cdcl_W\text{-}stgy$ *S* =
(*let T* = $do\text{-}cdcl_W\text{-}stgy\text{-}step'$ *S in*
if T = *S then S else* $do\text{-}all\text{-}cdcl_W\text{-}stgy$ *T*)
by *fast*+
termination
proof (*relation* {(*T*, *S*).
($cdcl_W\text{-}restart\text{-}measure$ (*toS* (*rough-state-from-init-state-of T*)),
$cdcl_W\text{-}restart\text{-}measure$ (*toS* (*rough-state-from-init-state-of S*)))
∈ *lexn less-than 3*}, *goal-cases*)
case *1*
show *?case* **by** (*rule wf-if-measure-f*) (*auto intro*!: *wf-lexn wf-less*)
next
case (*2 S T*) **note** *T* = *this*(*1*) **and** *ST* = *this*(*2*)
let *?S* = *rough-state-from-init-state-of S*
have *S*: $cdcl_W\text{-}stgy^{**}$ ($S0\text{-}cdcl_W\text{-}restart$ (*raw-init-clss* (*toS ?S*))) (*toS ?S*)
using *rough-state-from-init-state-of*[*of S*] **by** *auto*
moreover have $cdcl_W\text{-}stgy$ (*toS* (*rough-state-from-init-state-of S*))
(*toS* (*rough-state-from-init-state-of T*))
proof −
have $\bigwedge c$. *rough-state-of* (*state-of* (*rough-state-from-init-state-of c*)) =
rough-state-from-init-state-of c
using *rough-state-from-init-state-of state-of-inverse* **by** *fastforce*
then have *diff*: $do\text{-}cdcl_W\text{-}stgy\text{-}step$ (*state-of* (*rough-state-from-init-state-of S*))
≠ *state-of* (*rough-state-from-init-state-of S*)
using *ST T* **by** (*metis* (*no-types*) *id-of-I-to-def rough-state-from-init-state-of-inject*
$rough\text{-}state\text{-}from\text{-}init\text{-}state\text{-}of\text{-}do\text{-}cdcl_W\text{-}stgy\text{-}step'$)
have *rough-state-of* ($do\text{-}cdcl_W\text{-}stgy\text{-}step$ (*state-of* (*rough-state-from-init-state-of S*)))
= *rough-state-from-init-state-of* ($do\text{-}cdcl_W\text{-}stgy\text{-}step'$ *S*)
by (*simp add*: *id-of-I-to-def* $rough\text{-}state\text{-}from\text{-}init\text{-}state\text{-}of\text{-}do\text{-}cdcl_W\text{-}stgy\text{-}step'$)
then show *?thesis*
using $do\text{-}cdcl_W\text{-}stgy\text{-}step$ *T diff* **unfolding** *id-of-I-to-def* $do\text{-}cdcl_W\text{-}stgy\text{-}step$ **by** *fastforce*
qed
moreover have *invs*: $cdcl_W\text{-}all\text{-}struct\text{-}inv$ (*toS* (*rough-state-from-init-state-of S*))
using *rough-state-from-init-state-of*[*of S*] **by** *auto*
moreover {
have $cdcl_W\text{-}all\text{-}struct\text{-}inv$ ($S0\text{-}cdcl_W\text{-}restart$ (*raw-init-clss* (*toS* (*rough-state-from-init-state-of S*))))
using *invs* **by** (*cases rough-state-from-init-state-of S*)
(*auto simp add*: $cdcl_W\text{-}all\text{-}struct\text{-}inv\text{-}def$ $distinct\text{-}cdcl_W\text{-}state\text{-}def$)
then have ‹*no-smaller-propa* (*toS* (*rough-state-from-init-state-of S*))›
using $rtranclp\text{-}cdcl_W\text{-}stgy\text{-}no\text{-}smaller\text{-}propa$[*OF S*]
by (*auto simp*: *empty-trail-no-smaller-propa*) **}**
ultimately show *?case*
using $tranclp\text{-}cdcl_W\text{-}stgy\text{-}S0\text{-}decreasing$
by (*auto intro*!: $cdcl_W\text{-}stgy\text{-}step\text{-}decreasing$[*of*]
simp del: $cdcl_W\text{-}restart\text{-}measure.simps$)
qed

thm $do\text{-}all\text{-}cdcl_W\text{-}stgy.induct$
lemma $do\text{-}all\text{-}cdcl_W\text{-}stgy\text{-}induct$:

$(\bigwedge S.\ (do\text{-}cdcl_W\text{-}stgy\text{-}step'\ S \neq S \Longrightarrow P\ (do\text{-}cdcl_W\text{-}stgy\text{-}step'\ S)) \Longrightarrow P\ S) \Longrightarrow P\ a0$
using *do-all-cdcl*$_W$*-stgy.induct* **by** *metis*

lemma *no-step-cdcl*$_W$*-stgy-cdcl*$_W$*-restart-all*:
fixes *S* :: *'a cdcl*$_W$*-restart-state-inv-from-init-state*
shows *no-step cdcl*$_W$*-stgy* (*toS* (*rough-state-from-init-state-of* (*do-all-cdcl*$_W$*-stgy S*)))
apply (*induction S rule*: *do-all-cdcl*$_W$*-stgy-induct*)
apply (*rename-tac S*, *case-tac do-cdcl*$_W$*-stgy-step' S* $\neq$ *S*)
proof −
fix *Sa* :: *'a cdcl*$_W$*-restart-state-inv-from-init-state*
assume *a1*: $\neg$ *do-cdcl*$_W$*-stgy-step' Sa* $\neq$ *Sa*
{ fix *pp*
have (*if True then Sa else do-all-cdcl*$_W$*-stgy Sa*) = *do-all-cdcl*$_W$*-stgy Sa*
using *a1* **by** *auto*
then have $\neg$ *cdcl*$_W$*-stgy* (*toS* (*rough-state-from-init-state-of* (*do-all-cdcl*$_W$*-stgy Sa*))) *pp*
using *a1* **by** (*metis* (*no-types*) *do-cdcl*$_W$*-stgy-step-no id-of-I-to-def*
rough-state-from-init-state-of-do-cdcl$_W$*-stgy-step' rough-state-of-inverse*) **}**
then show *no-step cdcl*$_W$*-stgy* (*toS* (*rough-state-from-init-state-of* (*do-all-cdcl*$_W$*-stgy Sa*)))
by *fastforce*
next
fix *Sa* :: *'a cdcl*$_W$*-restart-state-inv-from-init-state*
assume *a1*: *do-cdcl*$_W$*-stgy-step' Sa* $\neq$ *Sa*
$\Longrightarrow$ *no-step cdcl*$_W$*-stgy* (*toS* (*rough-state-from-init-state-of*
(*do-all-cdcl*$_W$*-stgy* (*do-cdcl*$_W$*-stgy-step' Sa*))))
assume *a2*: *do-cdcl*$_W$*-stgy-step' Sa* $\neq$ *Sa*
have *do-all-cdcl*$_W$*-stgy Sa* = *do-all-cdcl*$_W$*-stgy* (*do-cdcl*$_W$*-stgy-step' Sa*)
by (*metis* (*full-types*) *do-all-cdcl*$_W$*-stgy.simps*)
then show *no-step cdcl*$_W$*-stgy* (*toS* (*rough-state-from-init-state-of* (*do-all-cdcl*$_W$*-stgy Sa*)))
using *a2 a1* **by** *presburger*
qed

lemma *do-all-cdcl*$_W$*-stgy-is-rtranclp-cdcl*$_W$*-stgy*:
cdcl$_W$*-stgy*** (*toS* (*rough-state-from-init-state-of S*))
(*toS* (*rough-state-from-init-state-of* (*do-all-cdcl*$_W$*-stgy S*)))
proof (*induction S rule*: *do-all-cdcl*$_W$*-stgy-induct*)
case (*1 S*) **note** *IH* = *this*(*1*)
show *?case*
proof (*cases do-cdcl*$_W$*-stgy-step' S* = *S*)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
have *f2*: *do-cdcl*$_W$*-stgy-step* (*id-of-I-to S*) = *id-of-I-to S* $\longrightarrow$
rough-state-from-init-state-of (*do-cdcl*$_W$*-stgy-step' S*)
= *rough-state-of* (*state-of* (*rough-state-from-init-state-of S*))
using *rough-state-from-init-state-of-do-cdcl*$_W$*-stgy-step'*
by (*simp add*: *id-of-I-to-def rough-state-from-init-state-of-do-cdcl*$_W$*-stgy-step'*)
have *f3*: *do-all-cdcl*$_W$*-stgy S* = *do-all-cdcl*$_W$*-stgy* (*do-cdcl*$_W$*-stgy-step' S*)
by (*metis* (*full-types*) *do-all-cdcl*$_W$*-stgy.simps*)
have *cdcl*$_W$*-stgy* (*toS* (*rough-state-from-init-state-of S*))
(*toS* (*rough-state-from-init-state-of* (*do-cdcl*$_W$*-stgy-step' S*)))
= *cdcl*$_W$*-stgy* (*toS* (*rough-state-of* (*id-of-I-to S*)))
(*toS* (*rough-state-of* (*do-cdcl*$_W$*-stgy-step* (*id-of-I-to S*))))
using *rough-state-from-init-state-of-do-cdcl*$_W$*-stgy-step'*
toS-rough-state-of-state-of-rough-state-from-init-state-of
by (*simp add*: *id-of-I-to-def rough-state-from-init-state-of-do-cdcl*$_W$*-stgy-step'*)

then show *?thesis*
using *f3 f2 IH do-cdcl$_W$-stgy-step* **by** *fastforce*
qed
qed

Final theorem:

lemma *DPLL-tot-correct*:
assumes
r: *rough-state-from-init-state-of* (*do-all-cdcl$_W$-stgy* (*state-from-init-state-of*
(([], *map remdups N*, [], *None*)))) = *S* **and**
S: (M', N', U', *E*) = *toS S*
shows (*E* ≠ *Some* {#} ∧ *satisfiable* (*set* (*map mset N*)))
∨ (*E* = *Some* {#} ∧ *unsatisfiable* (*set* (*map mset N*)))
proof −
let *?N* = *map remdups N*
have *inv*: *cdcl$_W$-all-struct-inv* (*toS* ([], *map remdups N*, [], *None*))
unfolding *cdcl$_W$-all-struct-inv-def distinct-cdcl$_W$-state-def distinct-mset-set-def* **by** *auto*
then have *S0*: *rough-state-of* (*state-of* ([], *map remdups N*, [], *None*))
= ([], *map remdups N*, [], *None*) **by** *simp*
have *1*: *full cdcl$_W$-stgy* (*toS* ([], *?N*, [], *None*)) (*toS S*)
unfolding *full-def* **apply** *rule*
using *do-all-cdcl$_W$-stgy-is-rtranclp-cdcl$_W$-stgy*[*of*
state-from-init-state-of ([], *map remdups N*, [], *None*)] *inv*
no-step-cdcl$_W$-stgy-cdcl$_W$-restart-all
apply (*auto simp del*: *do-all-cdcl$_W$-stgy.simps simp*: *state-from-init-state-of-inverse*
r[*symmetric*] *comp-def*)[]
using *do-all-cdcl$_W$-stgy-is-rtranclp-cdcl$_W$-stgy*[*of*
state-from-init-state-of ([], *map remdups N*, [], *None*)] *inv*
no-step-cdcl$_W$-stgy-cdcl$_W$-restart-all
by (*force simp*: *state-from-init-state-of-inverse r*[*symmetric*] *comp-def*)
moreover have *2*: *finite* (*set* (*map mset ?N*)) **by** *auto*
moreover have *3*: *distinct-mset-set* (*set* (*map mset ?N*))
unfolding *distinct-mset-set-def* **by** *auto*
moreover
have *cdcl$_W$-all-struct-inv* (*toS S*)
by (*metis* (*no-types*) *cdcl$_W$-all-struct-inv-rough-state r*
toS-rough-state-of-state-of-rough-state-from-init-state-of)
then have *cons*: *consistent-interp* (*lits-of-l* M')
unfolding *cdcl$_W$-all-struct-inv-def cdcl$_W$-M-level-inv-def S*[*symmetric*] **by** *auto*
moreover
have *raw-init-clss* (*toS* ([], *?N*, [], *None*)) = *raw-init-clss* (*toS S*)
apply (*rule rtranclp-cdcl$_W$-stgy-no-more-init-clss*)
using *1* **unfolding** *full-def* **by** (*auto simp add*: *rtranclp-cdcl$_W$-stgy-rtranclp-cdcl$_W$-restart*)
then have N': *mset* (*map mset ?N*) = N'
using *S*[*symmetric*] **by** *auto*
have (*E* ≠ *Some* {#} ∧ *satisfiable* (*set* (*map mset ?N*)))
∨ (*E* = *Some* {#} ∧ *unsatisfiable* (*set* (*map mset ?N*)))
using *full-cdcl$_W$-stgy-final-state-conclusive* **unfolding** N' **apply** *rule*
using *1* **apply** (*simp*; *fail*)
using *3* **apply** (*simp add*: *comp-def*; *fail*)
using *S*[*symmetric*] N' **apply** (*auto*; *fail*)[*1*]
using *S*[*symmetric*] N' *cons* **by** (*fastforce simp*: *true-annots-true-cls*)
then show *?thesis* **by** *auto*
qed

The Code The SML code is skipped in the documentation, but stays to ensure that some version of the exported code is working. The only difference between the generated code and the one used here is the export of the constructor ConI.

theory *CDCL-Abstract-Clause-Representation*
imports *Entailment-Definition.Partial-Herbrand-Interpretation*
begin

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

4.1.5 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 :: $'cls \Rightarrow 'v$ *clause*
begin
end

The two following locales are the *exact same* locale, but we need two different locales. Otherwise, instantiating *raw-clss* would lead to duplicate constants.

locale *abstract-with-index* =
  fixes
    get-lit :: $'a \Rightarrow 'it \Rightarrow 'conc$ *option* **and**
    convert-to-mset :: $'a \Rightarrow 'conc$ *multiset*
  assumes
    in-clss-mset-cls[*dest*]:
      get-lit Cs a $=$ *Some e* $\Longrightarrow e \in\#$ *convert-to-mset Cs* **and**
    in-mset-cls-exists-preimage:
      $b \in\#$ *convert-to-mset Cs* $\Longrightarrow \exists\, b'.$ *get-lit Cs* $b' =$ *Some b*

locale *abstract-with-index2* =
  fixes
    get-lit :: $'a \Rightarrow 'it \Rightarrow 'conc$ *option* **and**
    convert-to-mset :: $'a \Rightarrow 'conc$ *multiset*
  assumes
    in-clss-mset-clss[*dest*]:
      get-lit Cs a $=$ *Some e* $\Longrightarrow e \in\#$ *convert-to-mset Cs* **and**
    in-mset-clss-exists-preimage:
      $b \in\#$ *convert-to-mset Cs* $\Longrightarrow \exists\, b'.$ *get-lit Cs* $b' =$ *Some b*

locale *raw-clss* =
  abstract-with-index get-lit mset-cls +
  abstract-with-index2 get-cls mset-clss
  for
    get-lit :: $'cls \Rightarrow 'lit \Rightarrow 'v$ *literal option* **and**
    mset-cls :: $'cls \Rightarrow 'v$ *clause* **and**

get-cls :: *'clss* ⇒ *'cls-it* ⇒ *'cls option* **and**
mset-clss:: *'clss* ⇒ *'cls multiset*
begin

definition *cls-lit* :: *'cls* ⇒ *'lit* ⇒ *'v literal* (**infix** ↓ *49*) **where**
C ↓ *a* ≡ *the* (*get-lit C a*)

definition *clss-cls* :: *'clss* ⇒ *'cls-it* ⇒ *'cls* (**infix** ⇓ *49*) **where**
C ⇓ *a* ≡ *the* (*get-cls C a*)

definition *in-cls* :: *'lit* ⇒ *'cls* ⇒ *bool* (**infix** ∈↓ *49*) **where**
a ∈↓ *Cs* ≡ *get-lit Cs a* ≠ *None*

definition *in-clss* :: *'cls-it* ⇒ *'clss* ⇒ *bool* (**infix** ∈⇓ *49*) **where**
a ∈⇓ *Cs* ≡ *get-cls Cs a* ≠ *None*

definition *raw-clss* **where**
raw-clss S ≡ *image-mset mset-cls* (*mset-clss S*)

end

experiment
begin
fun *safe-nth* **where**
safe-nth (*x* # -) *0* = *Some x* |
safe-nth (- # *xs*) (*Suc n*) = *safe-nth xs n* |
safe-nth [] - = *None*

lemma *safe-nth-nth*: *n* < *length l* ⟹ *safe-nth l n* = *Some* (*nth l n*)
by (*induction l n rule*: *safe-nth.induct*) *auto*

lemma *safe-nth-None*: *n* ≥ *length l* ⟹ *safe-nth l n* = *None*
by (*induction l n rule*: *safe-nth.induct*) *auto*

lemma *safe-nth-Some-iff*: *safe-nth l n* = *Some m* ⟷ *n* < *length l* ∧ *m* = *nth l n*
apply (*rule iffI*)
defer apply (*auto simp*: *safe-nth-nth*)[]
by (*induction l n rule*: *safe-nth.induct*) *auto*

lemma *safe-nth-None-iff*: *safe-nth l n* = *None* ⟷ *n* ≥ *length l*
apply (*rule iffI*)
defer apply (*auto simp*: *safe-nth-None*)[]
by (*induction l n rule*: *safe-nth.induct*) *auto*

interpretation *abstract-with-index*
safe-nth
mset
apply *unfold-locales*
apply (*simp add*: *safe-nth-Some-iff*)
by (*metis in-set-conv-nth safe-nth-nth set-mset-mset*)

interpretation *abstract-with-index2*
safe-nth
mset
apply *unfold-locales*

apply (*simp add*: *safe-nth-Some-iff*)
by (*metis in-set-conv-nth safe-nth-nth set-mset-mset*)

interpretation *list-cls*: *raw-clss*
safe-nth mset
safe-nth mset
by *unfold-locales*
end

end
theory *CDCL-W-Abstract-State*
imports *CDCL-W-Full CDCL-W-Restart*

begin

4.2 Instantiation of Weidenbach's CDCL by Multisets

We first instantiate the locale of Weidenbach's locale. Then we refine it to a 2-WL program.

type-synonym $'v$ $cdcl_W$*-restart-mset* = ($'v$, $'v$ *clause*) *ann-lit list* ×
$'v$ *clauses* ×
$'v$ *clauses* ×
$'v$ *clause option*

We use definition, otherwise we could not use the simplification theorems we have already shown.

fun *trail* :: $'v$ $cdcl_W$*-restart-mset* ⇒ ($'v$, $'v$ *clause*) *ann-lit list* **where**
trail (*M*, -) = *M*

fun *init-clss* :: $'v$ $cdcl_W$*-restart-mset* ⇒ $'v$ *clauses* **where**
init-clss (-, *N*, -) = *N*

fun *learned-clss* :: $'v$ $cdcl_W$*-restart-mset* ⇒ $'v$ *clauses* **where**
learned-clss (-, -, *U*, -) = *U*

fun *conflicting* :: $'v$ $cdcl_W$*-restart-mset* ⇒ $'v$ *clause option* **where**
conflicting (-, -, -, *C*) = *C*

fun *cons-trail* :: ($'v$, $'v$ *clause*) *ann-lit* ⇒ $'v$ $cdcl_W$*-restart-mset* ⇒ $'v$ $cdcl_W$*-restart-mset* **where**
cons-trail L (*M*, *R*) = (*L* # *M*, *R*)

fun *tl-trail* **where**
tl-trail (*M*, *R*) = (*tl M*, *R*)

fun *add-learned-cls* **where**
add-learned-cls C (*M*, *N*, *U*, *R*) = (*M*, *N*, {#*C*#} + *U*, *R*)

fun *remove-cls* **where**
remove-cls C (*M*, *N*, *U*, *R*) = (*M*, *removeAll-mset C N*, *removeAll-mset C U*, *R*)

fun *update-conflicting* **where**
update-conflicting D (*M*, *N*, *U*, -) = (*M*, *N*, *U*, *D*)

fun *init-state* **where**
init-state N = ([], *N*, {#}, *None*)

declare *trail.simps*[*simp del*] *cons-trail.simps*[*simp del*] *tl-trail.simps*[*simp del*]
add-learned-cls.simps[*simp del*] *remove-cls.simps*[*simp del*]
update-conflicting.simps[*simp del*] *init-clss.simps*[*simp del*] *learned-clss.simps*[*simp del*]
conflicting.simps[*simp del*] *init-state.simps*[*simp del*]

lemmas $cdcl_W$*-restart-mset-state* = *trail.simps cons-trail.simps tl-trail.simps add-learned-cls.simps*
remove-cls.simps update-conflicting.simps init-clss.simps learned-clss.simps
conflicting.simps init-state.simps

definition *state* **where**
‹*state S* = (*trail S*, *init-clss S*, *learned-clss S*, *conflicting S*, ())›

interpretation $cdcl_W$*-restart-mset*: $state_W$*-ops* **where**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**

cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
.

definition *state-eq* :: *'v* $cdcl_W$*-restart-mset* ⇒ *'v* $cdcl_W$*-restart-mset* ⇒ *bool* (**infix** ∼*m 50*) **where**
‹*S* ∼*m T* ⟷ *state S* = *state T*›

interpretation $cdcl_W$*-restart-mset*: $state_W$ **where**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
state-eq = *state-eq* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
by *unfold-locales* (*auto simp*: $cdcl_W$*-restart-mset-state state-eq-def state-def*)

abbreviation *backtrack-lvl* :: *'v* $cdcl_W$*-restart-mset* ⇒ *nat* **where**
backtrack-lvl ≡ $cdcl_W$*-restart-mset.backtrack-lvl*

interpretation $cdcl_W$*-restart-mset*: *conflict-driven-clause-learning*$_W$ **where**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**

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

lemma $cdcl_W$*-restart-mset-state-eq-eq*: *state-eq* = (=)
apply (*intro ext*)
unfolding *state-eq-def*
by (*auto simp*: $cdcl_W$*-restart-mset-state state-def*)

lemma *clauses-def*: ‹$cdcl_W$*-restart-mset.clauses* (*M*, *N*, *U*, *C*) = *N* + *U*›
by (*subst* $cdcl_W$*-restart-mset.clauses-def*) (*simp add*: $cdcl_W$*-restart-mset-state*)

lemma $cdcl_W$*-restart-mset-reduce-trail-to*:
$cdcl_W$*-restart-mset.reduce-trail-to F S* =
((*if length* (*trail S*) ≥ *length F*
then drop (*length* (*trail S*) − *length F*) (*trail S*)
else []), *init-clss S*, *learned-clss S*, *conflicting S*)
(**is** *?S* = -)
proof (*induction F S rule*: $cdcl_W$*-restart-mset.reduce-trail-to.induct*)
case (*1 F S*) **note** *IH* = *this*
show *?case*
proof (*cases trail S*)
case *Nil*
then show *?thesis* **using** *IH* **by** (*cases S*) (*auto simp*: $cdcl_W$*-restart-mset-state*)
next
case (*Cons L M*)
then show *?thesis*
apply (*cases Suc* (*length M*) > *length F*)
subgoal
apply (*subgoal-tac Suc* (*length M*) − *length F* = *Suc* (*length M* − *length F*))
using $cdcl_W$*-restart-mset.reduce-trail-to-length-ne*[*of S F*] *IH* **by** *auto*
subgoal
using *IH* $cdcl_W$*-restart-mset.reduce-trail-to-length-ne*[*of S F*]
apply (*cases S*)
by (*simp add*: $cdcl_W$*-restart-mset.trail-reduce-trail-to-drop* $cdcl_W$*-restart-mset-state*)
done
qed
qed

lemma *full-*$cdcl_W$*-init-state*:
‹*full* $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy* (*init-state* {#}) *S* ⟷ *S* = *init-state* {#}›
unfolding *full-def rtranclp-unfold*
by (*subst tranclp-unfold-begin*)
(*auto simp*: $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy.simps*
$cdcl_W$*-restart-mset.conflict.simps* $cdcl_W$*-restart-mset.*$cdcl_W$*-o.simps*
$cdcl_W$*-restart-mset.propagate.simps* $cdcl_W$*-restart-mset.decide.simps*
$cdcl_W$*-restart-mset.*$cdcl_W$*-bj.simps* $cdcl_W$*-restart-mset.backtrack.simps*
$cdcl_W$*-restart-mset.skip.simps* $cdcl_W$*-restart-mset.resolve.simps*
$cdcl_W$*-restart-mset-state clauses-def*)

locale *twl-restart-ops* =
fixes
f :: ⟨*nat* ⇒ *nat*⟩
begin

interpretation *cdcl*$_W$*-restart-mset*: *cdcl*$_W$*-restart-restart-ops* **where**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**

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

end

locale *twl-restart* =
twl-restart-ops f **for** *f* :: ⟨*nat* ⇒ *nat*⟩ +
assumes
f: ⟨*unbounded f*⟩
begin

interpretation *cdcl*$_W$*-restart-mset*: *cdcl*$_W$*-restart-restart* **where**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**

state-eq = *state-eq* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state* **and**
f = *f*
by *unfold-locales* (*rule f*)

end

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

lemma *distinct-cdcl*$_W$*-state-alt-def*:
⟨*distinct-cdcl*$_W$*-state S* =
((∀ *T*. *conflicting S* = *Some T* ⟶ *distinct-mset T*) ∧

$distinct\text{-}mset\text{-}mset\ (clauses\ S) \wedge$
$(\forall L\ mark.\ Propagated\ L\ mark \in set\ (trail\ S) \longrightarrow distinct\text{-}mset\ mark))$›
unfolding *distinct-cdcl$_W$-state-def clauses-def*
by *auto*
end

lemma *cdcl$_W$-stgy-cdcl$_W$-init-state-empty-no-step*:
‹$cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}stgy\ (init\text{-}state\ \{\#\})\ S \longleftrightarrow False$›
unfolding *rtranclp-unfold*
by (*auto simp*: *cdcl$_W$-restart-mset.cdcl$_W$-stgy.simps*
cdcl$_W$-restart-mset.conflict.simps cdcl$_W$-restart-mset.cdcl$_W$-o.simps
cdcl$_W$-restart-mset.propagate.simps cdcl$_W$-restart-mset.decide.simps
cdcl$_W$-restart-mset.cdcl$_W$-bj.simps cdcl$_W$-restart-mset.backtrack.simps
cdcl$_W$-restart-mset.skip.simps cdcl$_W$-restart-mset.resolve.simps
cdcl$_W$-restart-mset-state clauses-def)

lemma *cdcl$_W$-stgy-cdcl$_W$-init-state*:
‹$cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}stgy^{**}\ (init\text{-}state\ \{\#\})\ S \longleftrightarrow S = init\text{-}state\ \{\#\}$›
unfolding *rtranclp-unfold*
by (*subst tranclp-unfold-begin*)
(*auto simp*: *cdcl$_W$-stgy-cdcl$_W$-init-state-empty-no-step simp del*: *init-state.simps*)

end