

Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

December 6, 2019

Contents

0.1	Weidenbach's DPLL	4
0.1.1	Rules	4
0.1.2	Invariants	4
0.1.3	Termination	12
0.1.4	Final States	15
1	Weidenbach's CDCL	23
1.1	Weidenbach's CDCL with Multisets	23
1.1.1	The State	23
1.1.2	CDCL Rules	33
1.1.3	Structural Invariants	41
1.1.4	CDCL Strong Completeness	69
1.1.5	Higher level strategy	70
1.1.6	Structural Invariant	98
1.1.7	Strategy-Specific Invariant	100
1.1.8	Additional Invariant: No Smaller Propagation	101
1.1.9	More Invariants: Conflict is False if no decision	106
1.1.10	Some higher level use on the invariants	111
1.1.11	Termination	116
1.2	Merging backjump rules	139
1.2.1	Inclusion of the States	139
1.2.2	More lemmas about Conflict, Propagate and Backjumping	141
1.2.3	CDCL with Merging	147
1.2.4	CDCL with Merge and Strategy	152
2	NOT's CDCL and DPLL	157
2.1	Measure	157
2.2	NOT's CDCL	161
2.2.1	Auxiliary Lemmas and Measure	161
2.2.2	Initial Definitions	162
2.2.3	DPLL with Backjumping	166
2.2.4	CDCL	184
2.2.5	CDCL with Restarts	206
2.2.6	Merging backjump and learning	214
2.2.7	Instantiations	229
2.3	Link between Weidenbach's and NOT's CDCL	244
2.3.1	Inclusion of the states	244
2.3.2	Inclusion of Weidenbach's CDCL without Strategy	246
2.3.3	Additional Lemmas between NOT and W states	247
2.3.4	Inclusion of Weidenbach's CDCL in NOT's CDCL	248

2.3.5	Inclusion of Weidenbach's CDCL with Strategy	258
3	Extensions on Weidenbach's CDCL	271
3.1	Restarts	271
3.2	Incremental SAT solving	282
4	List-based Implementation of DPLL and CDCL	295
4.1	Simple List-Based Implementation of the DPLL and CDCL	295
4.1.1	Common Rules	295
4.1.2	CDCL specific functions	298
4.1.3	Simple Implementation of DPLL	301
4.1.4	List-based CDCL Implementation	311
4.1.5	Abstract Clause Representation	332
4.2	Instantiation of Weidenbach's CDCL by Multisets	334
theory <i>DPLL-W</i>		
imports		
<i>Entailment-Definition.Partial-Herbrand-Interpretation</i>		
<i>Entailment-Definition.Partial-Annotated-Herbrand-Interpretation</i>		
<i>Weidenbach-Book-Base.Wellfounded-More</i>		
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* × *'v clauses*

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

abbreviation *clauses* :: *'v dpll_W-state* ⇒ *'v clauses* **where**
clauses ≡ *snd*

inductive *dpll_W* :: *'v dpll_W-state* ⇒ *'v dpll_W-state* ⇒ *bool* **where**
propagate: *add-mset L C ∈ # clauses S* ⇒ *trail S* ⊨_{as} *CNot C* ⇒ *undefined-lit (trail S) L*
 ⇒ *dpll_W S (Propagated L () # trail S, clauses S) |*
decided: *undefined-lit (trail S) L* ⇒ *atm-of L ∈ atms-of-mm (clauses S)*
 ⇒ *dpll_W S (Decided L # trail S, clauses S) |*
backtrack: *backtrack-split (trail S) = (M', L # M)* ⇒ *is-decided L* ⇒ *D ∈ # clauses S*
 ⇒ *trail S* ⊨_{as} *CNot D* ⇒ *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 $\bigwedge xb. xb \in \text{set } M \implies \text{atm-of (lit-of } xb) \in \text{atms-of-mm (clauses S)}$
 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)  $\subseteq$  atms-of-mm (clauses S)
shows all-decomposition-implies-m (clauses S') (get-all-ann-decomposition (trail S'))
using assms
proof (induct rule: dpllW.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))  $\cup$  set-mset (clauses S)
have ?I  $\models_p$  add-mset L C by (auto simp add: inS)
moreover have ?I  $\models_{ps}$  CNot C using true-annots-true-clss-cls cnot by fastforce
ultimately have ?I  $\models_p$  {#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)  $\neq$  []
  have 1:  $\bigwedge a b. (a, b) \in \text{set } (\text{tl } (\text{get-all-ann-decomposition } (\text{trail } S)))$ 
 $\implies (\text{unmark-l } a \cup \text{set-mset } (\text{clauses } S)) \models_{ps} \text{unmark-l } b$ 
  using IH unfolding all-decomposition-implies-def by (fastforce simp add: list.set-sel(2) n)
  moreover have 2:  $\bigwedge a c. \text{hd } (\text{get-all-ann-decomposition } (\text{trail } S)) = (a, c)$ 
 $\implies (\text{unmark-l } a \cup \text{set-mset } (\text{clauses } S)) \models_{ps} (\text{unmark-l } c)$ 
  by (metis IH all-decomposition-implies-cons-pair all-decomposition-implies-single
list.collapse n)
  moreover have 3:  $\bigwedge a c. \text{hd } (\text{get-all-ann-decomposition } (\text{trail } S)) = (a, c)$ 
 $\implies (\text{unmark-l } a \cup \text{set-mset } (\text{clauses } S)) \models_p \{ \#L\# \}$ 
  proof -
    fix a c
    assume h:  $\text{hd } (\text{get-all-ann-decomposition } (\text{trail } S)) = (a, c)$ 
    have h': trail S = c @ a using get-all-ann-decomposition-decomp h by blast
    have I: set (map unmark a)  $\cup$  set-mset (clauses S)
 $\cup$  unmark-l c  $\models_{ps}$  CNot C
    using ?I  $\models_{ps}$  CNot C unfolding h' by (simp add: Un-commute Un-left-commute)
    have
      atms-of-ms (CNot C)  $\subseteq$  atms-of-ms (set (map unmark a)  $\cup$  set-mset (clauses S))
      and
      atms-of-ms (unmark-l c)  $\subseteq$  atms-of-ms (set (map unmark a)
 $\cup$  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  $\cup$  set-mset (clauses S)  $\models_{ps}$  CNot C
      using true-clss-clss-left-right[OF - I] h 2 by auto
    then show unmark-l a  $\cup$  set-mset (clauses S)  $\models_p \{ \#L\# \}$ 
      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':  $\forall l \in \text{set } M'. \neg \text{is-decided } l$ 
  using extracted backtrack-split-fst-not-decided[of - trail S] by simp
have n: get-all-ann-decomposition (trail S)  $\neq []$  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)  $\cup$  set-mset (clauses S)  $\models_{ps} (\lambda a. \{\# \text{lit-of } a \# \})$  ' set M'
  by simp
moreover
  have unmark-l (L # M)  $\cup$  unmark-l M'  $\models_{ps}$  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)  $\cup$  set-mset (clauses S)  $\cup$  unmark-l M'
     $\models_{ps}$  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))  $\cup$  set-mset (clauses S)  $\models_{ps}$  CNot D
    using true-clss-clss-left-right by fastforce
  then have set (map unmark (L # M))  $\cup$  set-mset (clauses S)  $\models_p \{\#\}$ 
    by (metis (mono-tags, lifting) D Un-def mem-Collect-eq
      true-clss-clss-contradiction-true-clss-clss-false)
  then have IL: unmark-l M  $\cup$  set-mset (clauses S)  $\models_p \{\# - \text{lit-of } L \# \}$ 
    using true-clss-clss-false-left-right by auto
show ?case unfolding S all-decomposition-implies-def
proof
  fix x P level
  assume x:  $x \in \text{set } (\text{get-all-ann-decomposition } (\text{fst } (\text{Propagated } (- \text{lit-of } L) P \# M, \text{clauses } S)))$ 
  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  $\vee x \in \text{set } ?tl$ 
    using x
    by (cases get-all-ann-decomposition ?M')
    auto
  moreover {
    assume x':  $x \in \text{set } ?tl$ 
    have L': Decided (lit-of L) = L using decided by (cases L, auto)
    have  $x \in \text{set } (\text{get-all-ann-decomposition } (M' @ L \# M))$ 
      using x' get-all-ann-decomposition-except-last-choice-equal[of M' lit-of L P M]
      L' by (metis (no-types) M' list.set-sel(2) tl-Nil)
    then have case x of (Ls, seen)  $\Rightarrow$  unmark-l Ls  $\cup$  set-mset (clauses S)
       $\models_{ps}$  unmark-l seen
      using decided IH by (cases L) (auto simp add: S all-decomposition-implies-def)
  }
  moreover {
    assume x':  $x = ?hd$ 
    have tl: tl (get-all-ann-decomposition (M' @ L # M))  $\neq []$ 
    proof -
      have f1:  $\bigwedge ms. \text{length } (\text{get-all-ann-decomposition } (M' @ ms))$ 
        = 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)) ≠ 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*: $\{\{\#lit\text{-of } L\# \} \mid L. \text{ is-decided } L \wedge L \in \text{set } M\} = \{\}$ **using** *decided* **by** *auto*
have *atms-of-ms* $(N \cup \text{unmark-l } M) = \text{atms-of-ms } N$
using *atm-incl* **unfolding** *atms-of-ms-def lits-of-def* **by** *auto*

then have *total-over-m* *I* $(N \cup \text{unmark } (set\ M))$
using *tot* **unfolding** *total-over-m-def* **by** *auto*
then have $I \models_s \text{unmark } (set\ M)$
using *all-decomposition-implies-propagated-lits-are-implied* [*OF inv*] *cons* *I*
unfolding *true-clss-clss-def l0* **by** *auto*
then have *IM*: $I \models_s \text{unmark-l } M$ **by** *auto*
{
 fix *K*
 assume $K \in \# D$
 then have $-K \in \text{lits-of-l } M$
 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-clss-def consistent-interp-def* **by** *auto*
then show *False* **using** *I-D* **by** *blast*
qed

lemma *dpll_W-same-clauses*:

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

lemma *rtrancpl-dpll_W-inv*:

assumes *rtrancpl dpll_W* *S* *S'*
and *inv*: *all-decomposition-implies-m* (*clauses* *S*) (*get-all-ann-decomposition* (*trail* *S*))
and *atm-incl*: *atm-of* ' *lits-of-l* (*trail* *S*) \subseteq *atms-of-mm* (*clauses* *S*)
and *consistent-interp* (*lits-of-l* (*trail* *S*))
and *no-dup* (*trail* *S*)
shows *all-decomposition-implies-m* (*clauses* *S'*) (*get-all-ann-decomposition* (*trail* *S'*))
and *atm-of* ' *lits-of-l* (*trail* *S'*) \subseteq *atms-of-mm* (*clauses* *S'*)
and *clauses* *S* = *clauses* *S'*
and *consistent-interp* (*lits-of-l* (*trail* *S'*))
and *no-dup* (*trail* *S'*)
using *assms*

proof (*induct rule: rtrancpl-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* = *clauses* *S* **and**
consistent-interp (*lits-of-l* (*trail* *S*)) **and**
no-dup (*trail* *S*) **using** *assms* **by** *auto*

next

case (*step* *S'* *S''*) **note** *dpll_WStar* = *this*(1) **and** *IH* = *this*(3,4,5,6,7) **and**
dpll_W = *this*(2)

moreover
assume
inv: *all-decomposition-implies-m* (*clauses S*) (*get-all-ann-decomposition* (*trail S*)) **and**
atm-incl: *atm-of* ' *lits-of-l* (*trail S*) \subseteq *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'*) \subseteq *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** *dp_{ll}_W-same-clauses*[*OF dp_{ll}_W*] *snd* **by** *metis*

show *all-decomposition-implies-m* (*clauses S''*) (*get-all-ann-decomposition* (*trail S''*))
using *dp_{ll}_W-propagate-is-conclusion*[*OF dp_{ll}_W*] *decomp atm-incl'* **by** *auto*
show *atm-of* ' *lits-of-l* (*trail S''*) \subseteq *atms-of-mm* (*clauses S''*)
using *dp_{ll}_W-vars-in-snd-inv*[*OF dp_{ll}_W*] *atm-incl atm-incl'* **by** *auto*
show *no-dup* (*trail S''*) **using** *dp_{ll}_W-distinct-inv*[*OF dp_{ll}_W*] *no-dup'* *dp_{ll}_W* **by** *auto*
show *consistent-interp* (*lits-of-l* (*trail S''*))
using *cons'* *no-dup'* *dp_{ll}_W-consistent-interp-inv*[*OF dp_{ll}_W*] **by** *auto*
qed

definition *dp_{ll}_W-all-inv S* \equiv
(*all-decomposition-implies-m* (*clauses S*) (*get-all-ann-decomposition* (*trail S*)))
 \wedge *atm-of* ' *lits-of-l* (*trail S*) \subseteq *atms-of-mm* (*clauses S*)
 \wedge *consistent-interp* (*lits-of-l* (*trail S*))
 \wedge *no-dup* (*trail S*)

lemma *dp_{ll}_W-all-inv-dest*[*dest*]:
assumes *dp_{ll}_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*) \subseteq *atms-of-mm* (*clauses S*)
and *consistent-interp* (*lits-of-l* (*trail S*)) \wedge *no-dup* (*trail S*)
using *assms unfolding dp_{ll}_W-all-inv-def lits-of-def* **by** *auto*

lemma *rtranc_lp-dp_{ll}_W-all-inv*:
assumes *rtranc_lp dp_{ll}_W S S'*
and *dp_{ll}_W-all-inv S*
shows *dp_{ll}_W-all-inv S'*
using *assms rtranc_lp-dp_{ll}_W-inv*[*OF assms(1)*] *unfolding dp_{ll}_W-all-inv-def lits-of-def* **by** *blast*

lemma *dp_{ll}_W-all-inv*:
assumes *dp_{ll}_W S S'*
and *dp_{ll}_W-all-inv S*
shows *dp_{ll}_W-all-inv S'*
using *assms rtranc_lp-dp_{ll}_W-all-inv* **by** *blast*

lemma *rtranc_lp-dp_{ll}_W-inv-starting-from-0*:
assumes *rtranc_lp dp_{ll}_W S S'*
and *inv*: *trail S* = []
shows *dp_{ll}_W-all-inv S'*
proof –
have *dp_{ll}_W-all-inv S*
using *assms unfolding all-decomposition-implies-def dp_{ll}_W-all-inv-def* **by** *auto*
then show *?thesis* **using** *rtranc_lp-dp_{ll}_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.premis(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.premis* **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-clis* **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-clis*)
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:  $\neg ?A$ 
  have  $(\exists L. \text{undefined-lit } M \ L \wedge \text{atm-of } L \in \text{atms-of-mm } N) \vee (\exists D \in \#N. M \models_{as} \text{CNot } D)$ 
  proof -
    obtain  $D :: 'a \text{ clause}$  where  $D: D \in \# N$  and  $\neg M \models_a D$ 
    using  $n$  unfolding true-annots-def Ball-def by auto
    then have  $(\exists L. \text{undefined-lit } M \ L \wedge \text{atm-of } L \in \text{atms-of } D) \vee M \models_{as} \text{CNot } D$ 
    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  $\exists L. \text{undefined-lit } M \ L \wedge \text{atm-of } L \in \text{atms-of-mm } N$ 
  then have False using assms(2) decided by fastforce
}
moreover {
  assume  $\exists D \in \#N. M \models_{as} \text{CNot } D$ 
  then obtain  $D$  where  $DN: D \in \# N$  and  $MD: M \models_{as} \text{CNot } D$  by auto
  {
    assume  $\forall l \in \text{set } M. \neg \text{is-decided } l$ 
    moreover have  $dpll_W\text{-all-inv } ([], N)$ 
    using assms unfolding all-decomposition-implies-def  $dpll_W\text{-all-inv-def}$  by auto
    ultimately have unsatisfiable (set-mset N)
    using only-propagated-vars-unsat[of M D set-mset N] DN MD
     $\text{rtranclp-}dpll_W\text{-all-inv[OF assms(1)]}$  by force
    then have False using <?B> by blast
  }
  moreover {
    assume  $l: \exists l \in \text{set } M. \text{is-decided } l$ 
    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 $dpll_W\text{-mes } M \ n =$

$\text{map } (\lambda l. \text{if is-decided } l \text{ then } 2 \text{ else } (1::\text{nat})) (\text{rev } M) @ \text{replicate } (n - \text{length } M) \ 3$

lemma $\text{length-}dpll_W\text{-mes}:$

assumes $\text{length } M \leq n$

shows $\text{length } (dpll_W\text{-mes } M \ n) = n$

using assms **unfolding** $dpll_W\text{-mes-def}$ **by** auto

lemma $\text{distinctcard-atm-of-lit-of-eq-length}:$

assumes $\text{no-dup } S$

shows $\text{card } (\text{atm-of } \text{' lits-of-l } S) = \text{length } S$
using *assms* **by** (*induct* S) (*auto simp add: image-image lits-of-def no-dup-def*)

lemma *Cons-lexn-iff*:

shows $\langle (x \# xs, y \# ys) \in \text{lexn } R \ n \longleftrightarrow (\text{length } (x \# xs) = n \wedge \text{length } (y \# ys) = n \wedge ((x,y) \in R \vee (x = y \wedge (xs, ys) \in \text{lexn } R \ (n - 1))) \rangle$

unfolding *lexn-conv* **apply** (*rule iffI; clarify*)

subgoal for $xys \ xs \ ys \ xs' \ ys'$

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*]: $\text{length } (\text{trail } S') \leq \text{card vars}$ **and**

$\text{length } (\text{trail } S) \leq \text{card vars}$

shows

$(\text{dpll}_W\text{-mes } (\text{trail } S') (\text{card vars}), \text{dpll}_W\text{-mes } (\text{trail } S) (\text{card vars})) \in \text{lexn less-than } (\text{card vars})$

using *assms*

proof (*induct rule: dpll_W.induct*)

case (*propagate* $C \ L \ S$)

then have $m: \text{card vars} - \text{length } (\text{trail } S) = \text{Suc } (\text{card vars} - \text{Suc } (\text{length } (\text{trail } S)))$

by *fastforce*

then show $\langle (\text{dpll}_W\text{-mes } (\text{trail } (\text{Propagated } C \ () \ \# \text{trail } S, \text{clauses } S)) (\text{card vars}), \text{dpll}_W\text{-mes } (\text{trail } S) (\text{card vars})) \in \text{lexn less-than } (\text{card vars}) \rangle$

unfolding *dpll_W-mes-def* **by** *auto*

next

case (*decided* $S \ L$)

have $m: \text{card vars} - \text{length } (\text{trail } S) = \text{Suc } (\text{card vars} - \text{Suc } (\text{length } (\text{trail } S)))$

using *decided.prem[simplified]* **using** *Suc-diff-le* **by** *fastforce*

then show $\langle (\text{dpll}_W\text{-mes } (\text{trail } (\text{Decided } L \ \# \text{trail } S, \text{clauses } S)) (\text{card vars}), \text{dpll}_W\text{-mes } (\text{trail } S) (\text{card vars})) \in \text{lexn less-than } (\text{card vars}) \rangle$

unfolding *dpll_W-mes-def* **by** *auto*

next

case (*backtrack* $S \ M' \ L \ M \ D$)

moreover have $S: \text{trail } S = M' @ L \ \# \ M$

using *backtrack.hyps(1)* *backtrack-split-list-eq[of trail S]* **by** *auto*

ultimately show $\langle (\text{dpll}_W\text{-mes } (\text{trail } (\text{Propagated } (- \text{lit-of } L) \ () \ \# \ M, \text{clauses } S)) (\text{card vars}), \text{dpll}_W\text{-mes } (\text{trail } S) (\text{card vars})) \in \text{lexn less-than } (\text{card vars}) \rangle$

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*: $\text{atm-of } \text{' lits-of-l } (\text{trail } S) \subseteq \text{atms-of-mm } (\text{clauses } S)$

and *no-dup*: *no-dup* $(\text{trail } S)$

shows $(\text{dpll}_W\text{-mes } (\text{trail } S') (\text{card } (\text{atms-of-mm } (\text{clauses } S'))), \text{dpll}_W\text{-mes } (\text{trail } S) (\text{card } (\text{atms-of-mm } (\text{clauses } S)))) \in \text{lex less-than}$

proof –

have *finite* $(\text{atms-of-mm } (\text{clauses } S))$ **unfolding** *atms-of-ms-def* **by** *auto*

then have $1: \text{length } (\text{trail } S) \leq \text{card } (\text{atms-of-mm } (\text{clauses } S))$

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 dpllW-distinct-inv no-dup by blast
  have SS': clauses S' = clauses S using dpll by (auto dest!: dpllW-same-clauses)
  have atm-incl': atm-of ' lits-of-l (trail S')  $\subseteq$  atms-of-mm (clauses S')
    using atm-incl dpll dpllW-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 (dpllW-mes (trail S') (card (atms-of-mm (clauses S'))),
  dpllW-mes (trail S) (card (atms-of-mm (clauses S))))
 $\in$  lexn less-than (card (atms-of-mm (clauses S)))
using dpllW-card-decrease[OF assms(1), of atms-of-mm (clauses S)] by blast
then have (dpllW-mes (trail S') (card (atms-of-mm (clauses S'))),
  dpllW-mes (trail S) (card (atms-of-mm (clauses S))))  $\in$  lex less-than
unfolding lex-def by auto
then show (dpllW-mes (trail S') (card (atms-of-mm (clauses S'))),
  dpllW-mes (trail S) (card (atms-of-mm (clauses S))))  $\in$  lex less-than
using dpllW-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-dpllW:
  wf {(S', S). dpllW-all-inv S  $\wedge$  dpllW S S'}
apply (rule wf-wf-if-measure'[OF wf-lex-less, of -
   $\lambda S$ . dpllW-mes (trail S) (card (atms-of-mm (clauses S))))])
using dpllW-card-decrease' by fast

lemma dpllW-tranclp-star-commute:
  {(S', S). dpllW-all-inv S  $\wedge$  dpllW S S'}+ = {(S', S). dpllW-all-inv S  $\wedge$  tranclp dpllW 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 dpllW++ S' S and dpllW-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)  $\Rightarrow$  dpllW-all-inv S  $\wedge$  dpllW S S'}+ by blast
  }

```

```

moreover have  $dpll_W$ -all-inv  $S'$ 
  using  $rtrancp$ - $dpll_W$ -all-inv[ $OF$   $trancp$ -into- $rtrancp$ [ $OF$   $tranc$ -into- $tranc$ .hyps(1)]]
     $tranc$ -into- $tranc$ .prems by auto
ultimately have  $(S'', S') \in \{(pa, p). dpll_W$ -all-inv  $p \wedge dpll_W$   $p$   $pa\}^+$ 
  using  $\langle dpll_W$ -all-inv  $S' \rangle$   $tranc$ -into- $tranc$ .hyps(3) by blast
then show ?case
  using  $\langle (S', S) \in \{a. \text{case } a \text{ of } (S', S) \Rightarrow dpll_W$ -all-inv  $S \wedge dpll_W$   $S$   $S'\}^+ \rangle$  by auto
qed
}
then show ? $B \subseteq$  ? $A$  by blast
qed

```

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

lemma wf - $dpll_W$ -plus:
 $wf \{(S', ([], N)) | S'. dpll_W^{++} ([], N) S'\}$ (**is** wf ? P)
apply (rule wf -subset[OF wf - $dpll_W$ - $trancp$, 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: $\forall s \in \text{atms-of-mm}(\text{clauses } S). s \in \text{atm-of } \text{'lits-of-l'}(\text{trail } S)$

proof (rule ccontr)

assume $\neg (\forall s \in \text{atms-of-mm}(\text{clauses } S). s \in \text{atm-of } \text{'lits-of-l'}(\text{trail } S))$

then obtain L **where**

L -in-atms: $L \in \text{atms-of-mm}(\text{clauses } S)$ **and**

L -notin-trail: $L \notin \text{atm-of } \text{'lits-of-l'}(\text{trail } S)$ **by** metis

obtain L' **where** L' : $\text{atm-of } L' = L$ **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: \neg ?thesis

then have

$\neg \text{trail } S \models_{asm} \text{clauses } S$ **and**

$(\exists L \in \text{set}(\text{trail } S). \text{is-decided } L) \vee (\forall C \in \# \text{clauses } S. \neg \text{trail } S \models_{as} C \text{Not } C)$

unfolding conclusive- $dpll_W$ -state-def **by** auto

moreover {

assume $\exists L \in \text{set}(\text{trail } S). \text{is-decided } L$

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** $D \in \# \text{clauses } S$ **and** $\neg \text{trail } S \models_a D$

using $\langle \neg \text{trail } S \models_{asm} \text{clauses } S \rangle$ **unfolding** true-annots-def **by** auto

then have $\forall s \in \text{atms-of-ms} \{D\}. s \in \text{atm-of } \text{'lits-of-l'}(\text{trail } S)$

using vars **unfolding** atms-of-ms-def **by** auto

then have trail $S \models_{as} C \text{Not } D$

using all-variables-defined-not-imply-cnot[of D] $\langle \neg \text{trail } S \models_a D \rangle$ **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) dpllW.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-cl: 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-cl tr trC all-variables-defined-not-imply-cnot)
    then have False using tr C-in-cl by auto
  }
  ultimately show False by blast
qed
qed

lemma dpllW-conclusive-state-correct:
  assumes dpllW** ([], N) (M, N) and conclusive-dpllW-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-cl)
  moreover have consistent-interp ?M'
    using rtranclp-dpllW-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-dpllW-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-dpllW-all-inv[OF assms(1)]
      unfolding dpllW-all-inv-def by force
    then show False using ⟨?B⟩ by blast
  qed
qed
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 $l = \text{length } (\text{filter is-decided } l)$

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

lemma *get-level-uminus*[*simp*]: $\langle \text{get-level } M\ (-L) = \text{get-level } M\ L \rangle$
by (*auto simp: get-level-def*)

lemma *get-level-Neg-Pos*: $\langle \text{get-level } M\ (\text{Neg } L) = \text{get-level } M\ (\text{Pos } L) \rangle$
unfolding *get-level-def* **by** *auto*

lemma *count-decided-0-iff*:
 $\langle \text{count-decided } M = 0 \longleftrightarrow (\forall L \in \text{set } M. \neg \text{is-decided } L) \rangle$
by (*auto simp: count-decided-def filter-empty-conv*)

lemma
shows
count-decided-nil[*simp*]: $\langle \text{count-decided } [] = 0 \rangle$ **and**
count-decided-cons[*simp*]:
 $\langle \text{count-decided } (a \# M) = (\text{if is-decided } a \text{ then } \text{Suc } (\text{count-decided } M) \text{ else } \text{count-decided } M) \rangle$ **and**
count-decided-append[*simp*]:
 $\langle \text{count-decided } (M @ M') = \text{count-decided } M + \text{count-decided } M' \rangle$
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* $L \in \text{atm-of } \text{'lits-of-l } M$
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* $(M @ M')\ L = \text{get-level } M'\ L$
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* $(M @ M')\ L = \text{get-level } M\ L + \text{count-decided } M'$
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* $L' \neq \text{atm-of } (\text{lit-of } K)$
shows *get-level* $(K \# M)\ L' = \text{get-level } M\ L'$
using *assms* **by** (*auto simp: get-level-def*)

lemma *get-level-take-beginning*[*simp*]:

assumes $\text{atm-of } L' = \text{atm-of } (\text{lit-of } K)$
shows $\text{get-level } (K \# M) L' = \text{count-decided } (K \# M)$
using *assms* **by** (*auto simp: get-level-def count-decided-def*)

lemma *get-level-cons-if*:
 $\langle \text{get-level } (K \# M) L' =$
 $(\text{if } \text{atm-of } L' = \text{atm-of } (\text{lit-of } K) \text{ then } \text{count-decided } (K \# M) \text{ else } \text{get-level } M L') \rangle$
by *auto*

lemma *get-level-skip-beginning-not-decided*[*simp*]:
assumes *undefined-lit S L*
and $\forall s \in \text{set } S. \neg \text{is-decided } s$
shows $\text{get-level } (M @ S) L = \text{get-level } M L$
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 $\forall m \in \text{set } M. \neg \text{is-decided } m$
shows $\text{get-level } M L = 0$
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* :: $('a, 'b) \text{ ann-lits} \Rightarrow 'a \text{ clause} \Rightarrow \text{nat}$
where
 $\text{get-maximum-level } M D = \text{Max-mset } (\{\#0\# \} + \text{image-mset } (\text{get-level } M) D)$

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

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

lemma *get-maximum-level-exists-lit-of-max-level*:
 $D \neq \{\# \} \implies \exists L \in \# D. \text{get-level } M L = \text{get-maximum-level } M D$
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*]:
 $\text{get-maximum-level } [] D = 0$
unfolding *get-maximum-level-def* **by** (*simp add: image-constant-conv*)

lemma *get-maximum-level-add-mset*:
 $\text{get-maximum-level } M (\text{add-mset } L D) = \max (\text{get-level } M L) (\text{get-maximum-level } M D)$
unfolding *get-maximum-level-def* **by** *simp*

lemma *get-level-append-if*:
 $\langle \text{get-level } (M @ M') L = (\text{if } \text{defined-lit } M L \text{ then } \text{get-level } M L + \text{count-decided } M' \text{ else } \text{get-level } M L) \rangle$

$\text{else } \text{get-level } M' L \rangle$
by (*auto*)

Do not activate as [simp] rules. It breaks everything.

lemma *get-maximum-level-single*:

$\langle \text{get-maximum-level } M \{ \#x \# \} = \text{get-level } M x \rangle$
by (*auto simp: get-maximum-level-add-mset*)

lemma *get-maximum-level-plus*:

$\text{get-maximum-level } M (D + D') = \max (\text{get-maximum-level } M D) (\text{get-maximum-level } M D')$
by (*induction D (simp-all add: get-maximum-level-add-mset)*)

lemma *get-maximum-level-cong*:

assumes $\forall L \in \# D. \text{get-level } M L = \text{get-level } M' L$
shows $\langle \text{get-maximum-level } M D = \text{get-maximum-level } M' D \rangle$
using *assms* **by** (*induction D (auto simp: get-maximum-level-add-mset)*)

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

assumes $n: n > 0$
and $\text{max: get-maximum-level } M D = n$
shows $\exists L \in \# D. \text{get-level } M L = n$

proof –

have $f: \text{finite } (\text{insert } 0 ((\lambda L. \text{get-level } M L) \text{ `set-mset } D))$ **by** *auto*
then have $n \in ((\lambda L. \text{get-level } M L) \text{ `set-mset } D)$
using $n \text{ max Max-in[OF } f]$ **unfolding** *get-maximum-level-def* **by** *simp*
then show $\exists L \in \# D. \text{get-level } M L = n$ **by** *auto*

qed

lemma *get-maximum-level-skip-first[simp]*:

assumes $\text{atm-of } (\text{lit-of } K) \notin \text{atms-of } D$
shows $\text{get-maximum-level } (K \# M) D = \text{get-maximum-level } M D$
using *assms* **unfolding** *get-maximum-level-def atms-of-def*
 $\text{atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set}$
by (*smt atm-of-in-atm-of-set-in-uminus get-level-skip-beginning image-iff lit-of.simps(2)*
 $\text{multiset.map-cong0}$)

lemma *get-maximum-level-skip-beginning*:

assumes $DH: \forall x \in \# D. \text{undefined-lit } c x$
shows $\text{get-maximum-level } (c @ H) D = \text{get-maximum-level } H D$

proof –

have $(\text{get-level } (c @ H)) \text{ `set-mset } D = (\text{get-level } H) \text{ `set-mset } D$
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*:

$\text{get-maximum-level } [\text{Propagated } x21 \ x22] D = 0$
unfolding *get-maximum-level-def* **by** (*simp add: image-constant-conv*)

lemma *get-maximum-level-union-mset*:

$\text{get-maximum-level } M (A \cup \# B) = \text{get-maximum-level } M (A + B)$
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* ⊆ # *D'* ⟹ *get-maximum-level* *M* *D* ≤ *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* ≠ [] ⟹ *is-proped* (hd *M*) ⟹ *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*:
(λl. 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* ⟷ (∃ c *K* *c'*. *M* = c @ *Decided* *K* # *c'* ∧ *get-level* *M* *K* = *Suc* *i*)
(is ?*A* ⟷ ?*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-neg-Suc-count-decided[simp]:
   $\langle \text{get-level } M \ L \neq \text{Suc } (\text{count-decided } M) \rangle$ 
  using Suc-count-decided-gt-get-level[of  $M \ L$ ] by auto

lemma length-get-all-ann-decomposition:  $\langle \text{length } (\text{get-all-ann-decomposition } M) = 1 + \text{count-decided } M \rangle$ 
  by (induction M rule: ann-lit-list-induct) auto

lemma get-maximum-level-remove-non-max-lvl:
   $\langle \text{get-level } M \ a < \text{get-maximum-level } M \ D \implies$ 
   $\text{get-maximum-level } M \ (\text{remove1-mset } a \ D) = \text{get-maximum-level } M \ D \rangle$ 
  by (cases  $\langle a \in \# \ D \rangle$ )
  (auto dest!: multi-member-split simp: get-maximum-level-add-mset)

lemma exists-lit-max-level-in-negate-ann-lits:
   $\langle \text{negate-ann-lits } M \neq \{\#\} \implies \exists L \in \# \text{negate-ann-lits } M. \text{get-level } M \ L = \text{count-decided } M \rangle$ 
  by (cases  $\langle M \rangle$ ) (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: one 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 stateW-ops =
  fixes
    state :: 'st  $\Rightarrow$  ('v, 'v clause) ann-lits  $\times$  'v clauses  $\times$  'v clauses  $\times$  'v clause option  $\times$ 
      'b and
    trail :: 'st  $\Rightarrow$  ('v, 'v clause) ann-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-lit  $\Rightarrow$  'st  $\Rightarrow$  'st and
    tl-trail :: 'st  $\Rightarrow$  'st and

    add-learned-clss :: 'v clause  $\Rightarrow$  'st  $\Rightarrow$  'st and
    remove-clss :: 'v clause  $\Rightarrow$  'st  $\Rightarrow$  'st and
    update-conflicting :: 'v clause option  $\Rightarrow$  'st  $\Rightarrow$  'st and

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

  abbreviation hd-trail :: 'st  $\Rightarrow$  ('v, 'v clause) ann-lit where
    hd-trail S  $\equiv$  hd (trail S)

  definition clauses :: 'st  $\Rightarrow$  'v clauses where
    clauses S = init-clss S + learned-clss S

  abbreviation resolve-clss :: ('a literal  $\Rightarrow$  'a clause  $\Rightarrow$  'a clause  $\Rightarrow$  'a clause) where
    resolve-clss L D' E  $\equiv$  remove1-mset ( $-L$ ) D'  $\cup\#$  remove1-mset L E

  abbreviation state-butlast :: 'st  $\Rightarrow$  ('v, 'v clause) ann-lits  $\times$  'v clauses  $\times$  'v clauses
     $\times$  'v clause option where
    state-butlast S  $\equiv$  (trail S, init-clss S, learned-clss S, conflicting S)

  definition additional-info :: 'st  $\Rightarrow$  'b where
    additional-info S = ( $\lambda(-, -, -, -, D).$  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 stateW-no-state =
  stateW-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 ⇒ '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 +
assumes
  state-eq-ref[simp, intro]: ⟨S ~ S⟩ and
  state-eq-sym: ⟨S ~ T ⟷ T ~ S⟩ and
  state-eq-trans: ⟨S ~ T ⟹ T ~ U' ⟹ S ~ U'⟩ and
  state-eq-state: ⟨S ~ T ⟹ state S = state T⟩ and

  cons-trail:
    ∧S'. state st = (M, S') ⟹
      state (cons-trail L st) = (L # M, S') and

  tl-trail:
    ∧S'. state st = (M, S') ⟹ state (tl-trail st) = (tl M, S') and

  remove-cls:
    ∧S'. state st = (M, N, U, S') ⟹
      state (remove-cls C st) =
        (M, removeAll-mset C N, removeAll-mset C U, S') and

  add-learned-cls:
    ∧S'. state st = (M, N, U, S') ⟹
      state (add-learned-cls C st) = (M, N, {#C#} + U, S') and

  update-conflicting:

```

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

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

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

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

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

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

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

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

tl-trail-update-conflicting:
 $\langle \text{tl-trail } (\text{update-conflicting } D \text{ } T) \sim \text{update-conflicting } D \text{ } (\text{tl-trail } T) \rangle \text{ and}$

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

update-conflicting-itself:
 $\langle \bigwedge D \text{ } S'. \text{conflicting } S' = D \implies \text{update-conflicting } D \text{ } S' \sim S' \rangle$

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-eq :: 'st \Rightarrow 'st \Rightarrow bool (**infix** \sim 50) **and**

state :: 'st \Rightarrow ('v, 'v clause) ann-lits \times 'v clauses \times 'v clauses \times 'v clause option \times 'b **and**

trail :: 'st \Rightarrow ('v, 'v clause) ann-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-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-prop[simp]:
     $\langle \text{state } S = (\text{trail } S, \text{init-clss } S, \text{learned-clss } S, \text{conflicting } S, \text{additional-info } S) \rangle$ 
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) =  $\{\#C\# \} + \text{learned-clss st}$  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*) = {#}
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'*: $\langle S \sim S' \implies T \sim S' \implies T \sim S \rangle$

by (*meson state-eq-trans state-eq-sym*)

abbreviation *backtrack-lvl* :: '*st* \Rightarrow nat' **where**

$\langle \text{backtrack-lvl } S \equiv \text{count-decided } (\text{trail } S) \rangle$

named-theorems *state-simp* $\langle \text{contains all theorems of the form } @\{\text{term } \langle S \sim T \implies P \ S = P \ T \rangle\}.$

These theorems can cause a signifecant blow-up of the simp-space

lemma

shows

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

lemma *state-eq-conflicting-None*:

$S \sim T \implies \text{conflicting } T = \text{None} \implies \text{conflicting } S = \text{None}$

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* \Rightarrow '*st* \Rightarrow '*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 $\text{trail } S = F' @ F$
shows $\text{trail } (\text{reduce-trail-to } F S) = F$
using *assms* **by** (*induction* F' *arbitrary*: S) (*auto simp*: *reduce-trail-to-length-ne*)

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

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

lemma *init-clss-update-trail*[*simp*]:
 $\text{init-clss } (\text{reduce-trail-to } F S) = \text{init-clss } S$
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*]:
 $\text{learned-clss } (\text{reduce-trail-to } F S) = \text{learned-clss } S$
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*]:
 $\text{conflicting } (\text{reduce-trail-to } F S) = \text{None} \longleftrightarrow \text{conflicting } S = \text{None}$
apply (*induction* $F S$ *rule*: *reduce-trail-to.induct*)
by (*metis* *conflicting-update-trail*)

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

lemma *reduce-trail-to-add-learned-clss*[*simp*]:
 $\text{trail } (\text{reduce-trail-to } F (\text{add-learned-clss } C S)) = \text{trail } (\text{reduce-trail-to } F S)$
by (*rule* *trail-eq-reduce-trail-to-eq*) *auto*

lemma *reduce-trail-to-remove-learned-clss*[*simp*]:
 $\text{trail } (\text{reduce-trail-to } F (\text{remove-clss } C S)) = \text{trail } (\text{reduce-trail-to } F S)$
by (*rule* *trail-eq-reduce-trail-to-eq*) *auto*

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

lemma *reduce-trail-to-length*:
 $\text{length } M = \text{length } M' \implies \text{reduce-trail-to } M S = \text{reduce-trail-to } M' S$
apply (*induction* $M S$ *rule*: *reduce-trail-to.induct*)
by (*simp add*: *reduce-trail-to.simps*)

lemma *trail-reduce-trail-to-drop*:
trail (*reduce-trail-to* *F S*) =
 (if *length* (*trail S*) \geq *length F*
 then *drop* (*length* (*trail S*) – *length F*) (*trail S*)
 else [])
apply (*induction F S* rule: *reduce-trail-to.induct*)
apply (*rename-tac F S*, *case-tac trail S*)
apply (*auto*; *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*) \in *set* (*get-all-ann-decomposition* (*trail S*))
shows *trail* (*reduce-trail-to M1 S*) = *M1*
proof –
obtain *K* **where**
L: *L* = *Decided K*
using *H* **by** (*cases L*) (*auto dest!: in-get-all-ann-decomposition-decided-or-empty*)
obtain *c* **where**
tr-S: *trail S* = *c* @ *M2* @ *L* # *M1*
using *H* **by** *auto*
show ?thesis
by (*rule reduce-trail-to-trail-tl-trail-decomp[of - c @ M2 K]*)
(*auto simp: tr-S L*)
qed

lemma *reduce-trail-to-state-eq*:
 $\langle S \sim S' \implies \text{length } M = \text{length } M' \implies \text{reduce-trail-to } M S \sim \text{reduce-trail-to } M' S' \rangle$
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* \longleftrightarrow *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* \longleftrightarrow *conflicting S* = *None*
by *fastforce+*

lemma *reduce-trail-to-compow-tl-trail-le*:
assumes $\langle \text{length } M < \text{length } (\text{trail } M') \rangle$
shows $\langle \text{reduce-trail-to } M M' = (\text{tl-trail}^{\sim}(\text{length } (\text{trail } M') - \text{length } M)) M' \rangle$
proof –
have [*simp*]: $\langle (\forall ka. k \neq \text{Suc } ka) \longleftrightarrow k = 0 \rangle$ **for** *k*
by (*cases k*) *auto*
show ?thesis
using *assms*
apply (*induction M \equiv M S \equiv M' arbitrary: M M' rule: reduce-trail-to.induct*)
subgoal for *F S*
by (*subst reduce-trail-to.simps; cases* $\langle \text{length } F < \text{length } (\text{trail } S) - \text{Suc } 0 \rangle$)
(*auto simp: less-iff-Suc-add funpow-swap1*)
done

qed

lemma *reduce-trail-to-compow-tl-trail-eq*:

$\langle \text{length } M = \text{length } (\text{trail } M') \implies \text{reduce-trail-to } M M' = (\text{tl-trail} \sim (\text{length } (\text{trail } M') - \text{length } M)) M' \rangle$

by auto

lemma *reduce-trail-to-compow-tl-trail*:

$\langle \text{length } M \leq \text{length } (\text{trail } M') \implies \text{reduce-trail-to } M M' = (\text{tl-trail} \sim (\text{length } (\text{trail } M') - \text{length } M)) M' \rangle$

using *reduce-trail-to-compow-tl-trail-eq*[of $M M'$]

reduce-trail-to-compow-tl-trail-le[of $M M'$]

by (cases $\langle \text{length } M < \text{length } (\text{trail } M') \rangle$) auto

lemma *tl-trail-reduce-trail-to-cons*:

$\langle \text{length } (L \# M) < \text{length } (\text{trail } M') \implies \text{tl-trail } (\text{reduce-trail-to } (L \# M) M') = \text{reduce-trail-to } M M' \rangle$

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*:

$\langle (\text{tl-trail} \sim n) (\text{add-learned-cls } D S) \sim \text{add-learned-cls } D ((\text{tl-trail} \sim n) S) \rangle$

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*:

$\langle \text{length } M \leq \text{length } (\text{trail } S) \implies$

$\text{reduce-trail-to } M (\text{add-learned-cls } D S) \sim \text{add-learned-cls } D (\text{reduce-trail-to } M S) \rangle$

by (cases $\langle \text{length } M < \text{length } (\text{trail } S) \rangle$)

(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*:

$\langle (\text{tl-trail} \sim n) (\text{update-conflicting } D S) \sim \text{update-conflicting } D ((\text{tl-trail} \sim n) S) \rangle$

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*:

$\langle \text{length } M \leq \text{length } (\text{trail } S) \implies$

$\text{reduce-trail-to } M (\text{update-conflicting } D S) \sim \text{update-conflicting } D (\text{reduce-trail-to } M S) \rangle$

by (cases $\langle \text{length } M < \text{length } (\text{trail } S) \rangle$)

(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]:

$\langle \text{additional-info } (\text{cons-trail } L S) = \text{additional-info } S \rangle$ and

additional-info-tl-trail[simp]:

$\text{additional-info } (\text{tl-trail } S) = \text{additional-info } S$ and

additional-info-add-learned-cls-unfolded:

$\text{additional-info } (\text{add-learned-cls } U S) = \text{additional-info } S$ and

additional-info-update-conflicting[simp]:

$\text{additional-info } (\text{update-conflicting } D S) = \text{additional-info } S$ and

additional-info-remove-cls[simp]:

$\text{additional-info } (\text{remove-cls } C S) = \text{additional-info } S$ 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 stateW 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-learningW =
  stateW
  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** ~ 50) **and**

state :: 'st \Rightarrow ('v, 'v clause) ann-lits \times 'v clauses \times 'v clauses \times 'v clause option \times 'b **and**

trail :: 'st \Rightarrow ('v, 'v clause) ann-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-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 \Rightarrow

E $\in \#$ clauses *S* \Rightarrow

L $\in \#$ *E* \Rightarrow

trail S \models_{as} CNot (*E* - { $\#L\#$ }) \Rightarrow

undefined-lit (*trail S*) *L* \Rightarrow

T \sim *cons-trail* (*Propagated L E*) *S* \Rightarrow

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 \Rightarrow

D $\in \#$ clauses *S* \Rightarrow

trail S \models_{as} CNot *D* \Rightarrow

T \sim *update-conflicting* (*Some D*) *S* \Rightarrow

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*) \Rightarrow

(*Decided K* $\#$ *M1*, *M2*) \in *set* (*get-all-ann-decomposition* (*trail S*)) \Rightarrow

get-level (*trail S*) *L* = *backtrack-lvl S* \Rightarrow

get-level (*trail S*) *L* = *get-maximum-level* (*trail S*) (*add-mset L D'*) \Rightarrow

get-maximum-level (*trail S*) *D'* $\equiv i$ \Rightarrow

get-level (*trail S*) *K* = *i* + 1 \Rightarrow

D' $\subseteq \#$ *D* \Rightarrow

clauses S \models_{pm} *add-mset L D'* \Rightarrow

T \sim *cons-trail* (*Propagated L* (*add-mset L D'*))

(*reduce-trail-to M1*

(*add-learned-cls* (*add-mset L D'*))

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

inductive-cases backtrackE : $\text{backtrack } S \ T$

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

inductive $\text{simple-backtrack} :: 'st \Rightarrow 'st \Rightarrow \text{bool}$ **for** $S :: 'st$ **where**

$\text{simple-backtrack-rule}$:

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

inductive-cases simple-backtrackE : $\text{simple-backtrack } S \ T$

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

inductive $\text{backtrackg} :: 'st \Rightarrow 'st \Rightarrow \text{bool}$ **for** $S :: 'st$ **where**

backtrackg-rule :

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

inductive-cases backtrackgE : $\text{backtrackg } S \ T$

inductive $\text{decide} :: 'st \Rightarrow 'st \Rightarrow \text{bool}$ **for** $S :: 'st$ **where**

decide-rule :

$\text{conflicting } S = \text{None} \Rightarrow$
 $\text{undefined-lit } (\text{trail } S) \ L \Rightarrow$
 $\text{atm-of } L \in \text{atms-of-mm } (\text{init-clss } S) \Rightarrow$
 $T \sim \text{cons-trail } (\text{Decided } L) \ S \Rightarrow$
 $\text{decide } S \ T$

inductive-cases decideE : $\text{decide } S \ T$

inductive $\text{skip} :: 'st \Rightarrow 'st \Rightarrow \text{bool}$ **for** $S :: 'st$ **where**

skip-rule :

$\text{trail } S = \text{Propagated } L \ C' \ \# \ M \Rightarrow$
 $\text{conflicting } S = \text{Some } E \Rightarrow$
 $-L \notin \# \ E \Rightarrow$
 $E \neq \{\#\} \Rightarrow$

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

inductive-cases *skipE*: $\text{skip } S \ T$

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

inductive *resolve* :: $'st \Rightarrow 'st \Rightarrow \text{bool}$ **for** $S :: 'st$ **where**

resolve-rule: $\text{trail } S \neq [] \implies$

$\text{hd-trail } S = \text{Propagated } L \ E \implies$

$L \in \# \ E \implies$

$\text{conflicting } S = \text{Some } D' \implies$

$-L \in \# \ D' \implies$

$\text{get-maximum-level } (\text{trail } S) ((\text{remove1-mset } (-L) \ D')) = \text{backtrack-lvl } S \implies$

$T \sim \text{update-conflicting } (\text{Some } (\text{resolve-cls } L \ D' \ E))$

$(\text{tl-trail } S) \implies$

$\text{resolve } S \ T$

inductive-cases *resolveE*: $\text{resolve } S \ T$

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* :: $'st \Rightarrow 'st \Rightarrow \text{bool}$ **for** $S :: 'st$ **where**

restart: $\text{state } S = (M, N, U, \text{None}, S') \implies$

$U' \subseteq \# \ U \implies$

$\text{state } T = ([], N, U', \text{None}, S') \implies$

$\text{restart } S \ T$

inductive-cases *restartE*: $\text{restart } S \ T$

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

inductive *forget* :: $'st \Rightarrow 'st \Rightarrow \text{bool}$ **where**

forget-rule:

$\text{conflicting } S = \text{None} \implies$

$C \in \# \ \text{learned-clss } S \implies$

$\neg(\text{trail } S) \models_{\text{asm}} \text{clauses } S \implies$

$C \notin \text{set } (\text{get-all-mark-of-propagated } (\text{trail } S)) \implies$

$C \notin \# \ \text{init-clss } S \implies$

$\text{removeAll-mset } C \ (\text{clauses } S) \models_{\text{pm}} C \implies$

$T \sim \text{remove-cls } C \ S \implies$

$\text{forget } S \ T$

inductive-cases *forgetE*: $\text{forget } S \ T$

inductive *cdcl_W-rf* :: $'st \Rightarrow 'st \Rightarrow \text{bool}$ **for** $S :: 'st$ **where**

restart: $\text{restart } S \ T \implies \text{cdcl}_W\text{-rf } S \ T \mid$

forget: $\text{forget } S \ T \implies \text{cdcl}_W\text{-rf } S \ T$

inductive *cdcl_W-bj* :: $'st \Rightarrow 'st \Rightarrow \text{bool}$ **where**

skip: $\text{skip } S \ S' \implies \text{cdcl}_W\text{-bj } S \ S' \mid$

resolve: $\text{resolve } S \ S' \implies \text{cdcl}_W\text{-bj } S \ S' \mid$

backtrack: $\text{backtrack } S \ S' \implies \text{cdcl}_W\text{-bj } S \ S'$

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

inductive $cdcl_W-o :: 'st \Rightarrow 'st \Rightarrow bool$ **for** $S :: 'st$ **where**
decide: $decide\ S\ S' \Longrightarrow cdcl_W-o\ S\ S' \mid$
bj: $cdcl_W-bj\ S\ S' \Longrightarrow cdcl_W-o\ S\ S'$

inductive $cdcl_W-restart :: 'st \Rightarrow 'st \Rightarrow bool$ **for** $S :: 'st$ **where**
propagate: $propagate\ S\ S' \Longrightarrow cdcl_W-restart\ S\ S' \mid$
conflict: $conflict\ S\ S' \Longrightarrow cdcl_W-restart\ S\ S' \mid$
other: $cdcl_W-o\ S\ S' \Longrightarrow cdcl_W-restart\ S\ S' \mid$
rf: $cdcl_W-rf\ S\ S' \Longrightarrow cdcl_W-restart\ S\ S'$

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

inductive $cdcl_W :: 'st \Rightarrow 'st \Rightarrow bool$ **for** $S :: 'st$ **where**
W-propagate: $propagate\ S\ S' \Longrightarrow cdcl_W\ S\ S' \mid$
W-conflict: $conflict\ S\ S' \Longrightarrow cdcl_W\ S\ S' \mid$
W-other: $cdcl_W-o\ S\ S' \Longrightarrow cdcl_W\ S\ S'$

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

lemma *rtrancpl-cdcl_W-cdcl_W-restart*:
 $\langle cdcl_W^{**}\ S\ T \Longrightarrow cdcl_W-restart^{**}\ S\ T \rangle$
apply (*induction rule*: *rtrancpl-induct*)
apply (*auto*; *fail*)[]
by (*meson cdcl_W-cdcl_W-restart rtrancpl.rtrancpl-into-rtrancpl*)

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: $\bigwedge T. propagate\ S\ T \Longrightarrow P\ S\ T$ **and**
conflict: $\bigwedge T. conflict\ S\ T \Longrightarrow P\ S\ T$ **and**
forget: $\bigwedge T. forget\ S\ T \Longrightarrow P\ S\ T$ **and**
restart: $\bigwedge T. restart\ S\ T \Longrightarrow P\ S\ T$ **and**
decide: $\bigwedge T. decide\ S\ T \Longrightarrow P\ S\ T$ **and**
skip: $\bigwedge T. skip\ S\ T \Longrightarrow P\ S\ T$ **and**
resolve: $\bigwedge T. resolve\ S\ T \Longrightarrow P\ S\ T$ **and**
backtrack: $\bigwedge T. backtrack\ S\ T \Longrightarrow P\ S\ T$
shows $P\ S\ S'$
using *assms(1)*
proof (*induct S' rule*: *cdcl_W-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: cdclW-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: cdclW-bj.induct) auto
qed
next
  case (rf S')
  then show ?case
  by (induct rule: cdclW-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: $cdcl_W\text{-restart } S \ S' \text{ and}$

propagateH: $\bigwedge C \ L \ T. \text{ conflicting } S = \text{None} \implies$

$C \in \# \text{ clauses } S \implies$

$L \in \# \ C \implies$

$\text{trail } S \models_{as} C \text{Not } (\text{remove1-mset } L \ C) \implies$

$\text{undefined-lit } (\text{trail } S) \ L \implies$

$T \sim \text{cons-trail } (\text{Propagated } L \ C) \ S \implies$

$P \ S \ T \text{ and}$

conflictH: $\bigwedge D \ T. \text{ conflicting } S = \text{None} \implies$

$D \in \# \text{ clauses } S \implies$

$\text{trail } S \models_{as} C \text{Not } D \implies$

$T \sim \text{update-conflicting } (\text{Some } D) \ S \implies$

$P \ S \ T \text{ and}$

forgetH: $\bigwedge C \ T. \text{ conflicting } S = \text{None} \implies$

$C \in \# \text{ learned-clss } S \implies$

$\neg(\text{trail } S) \models_{asm} \text{clauses } S \implies$

$C \notin \text{set } (\text{get-all-mark-of-propagated } (\text{trail } S)) \implies$

$C \notin \# \text{ init-clss } S \implies$

$\text{removeAll-mset } C \ (\text{clauses } S) \models_{pm} C \implies$

$T \sim \text{remove-clss } C \ S \implies$

$P \ S \ T \text{ and}$

restartH: $\bigwedge T \ U. \text{ conflicting } S = \text{None} \implies$

$\text{state } T = ([], \text{init-clss } S, U, \text{None}, \text{additional-info } S) \implies$

$U \subseteq \# \text{ learned-clss } S \implies$

$P \ S \ T \text{ and}$

decideH: $\bigwedge L \ T. \text{ conflicting } S = \text{None} \implies$

$\text{undefined-lit } (\text{trail } S) \ L \implies$

$\text{atm-of } L \in \text{atms-of-mm } (\text{init-clss } S) \implies$

$T \sim \text{cons-trail } (\text{Decided } L) \ S \implies$

$P \ S \ T \text{ and}$

skipH: $\bigwedge L \ C' \ M \ E \ T.$

$\text{trail } S = \text{Propagated } L \ C' \ \# \ M \implies$

$\text{conflicting } S = \text{Some } E \implies$

$\neg L \notin \# \ E \implies E \neq \{\#\} \implies$

$T \sim \text{tl-trail } S \implies$

$P \ S \ T \text{ and}$

resolveH: $\bigwedge L \ E \ M \ D \ T.$

$\text{trail } S = \text{Propagated } L \ E \ \# \ M \implies$

```

L ∈ # E ⇒
hd-trail S = Propagated L E ⇒
conflicting S = Some D ⇒
-L ∈ # D ⇒
get-maximum-level (trail S) ((remove1-mset (-L) D)) = backtrack-lvl S ⇒
T ~ update-conflicting
  (Some (resolve-cls L D E)) (tl-trail S) ⇒
P S T and
backtrackH:  $\bigwedge L D K i M1 M2 T D'$ .
  conflicting S = Some (add-mset L D) ⇒
  (Decided K # M1, M2) ∈ set (get-all-ann-decomposition (trail S)) ⇒
  get-level (trail S) L = backtrack-lvl S ⇒
  get-level (trail S) L = get-maximum-level (trail S) (add-mset L D') ⇒
  get-maximum-level (trail S) D' ≡ i ⇒
  get-level (trail S) K = i+1 ⇒
  D' ⊆ # D ⇒
  clauses S ⊨pm add-mset L D' ⇒
  T ~ cons-trail (Propagated L (add-mset L D'))
    (reduce-trail-to M1
      (add-learned-cls (add-mset L D')
        (update-conflicting None S))) ⇒
  P S T
shows P S S'
using cdclW-restart
proof (induct S S' rule: cdclW-restart-all-rules-induct)
case (propagate S')
then show ?case
  by (auto elim!: propagateE intro!: propagateH)
next
case (conflict S')
then show ?case
  by (auto elim!: conflictE intro!: conflictH)
next
case (restart S')
then show ?case
  by (auto elim!: restartE intro!: restartH)
next
case (decide T)
then show ?case
  by (auto elim!: decideE intro!: decideH)
next
case (backtrack S')
then show ?case by (auto elim!: backtrackE intro!: backtrackH simp del: state-simp)
next
case (forget S')
then show ?case by (auto elim!: forgetE intro!: forgetH)
next
case (skip S')
then show ?case by (auto elim!: skipE intro!: skipH)
next
case (resolve S')
then show ?case
  by (cases trail S) (auto elim!: resolveE intro!: resolveH)
qed

```

lemma *cdcl_W-o-induct*[*consumes 1*, *case-names decide skip resolve backtrack*]:

fixes $S :: 'st$
assumes $cdcl_W\text{-restart}$: $cdcl_W\text{-o } S \text{ } T$ **and**
 $decideH$: $\bigwedge L \text{ } T. \text{ conflicting } S = \text{None} \implies \text{undefined-lit } (\text{trail } S) \text{ } L$
 $\implies \text{atm-of } L \in \text{atms-of-mm } (\text{init-clss } S)$
 $\implies T \sim \text{cons-trail } (\text{Decided } L) \text{ } S$
 $\implies P \text{ } S \text{ } T$ **and**
 $skipH$: $\bigwedge L \text{ } C' \text{ } M \text{ } E \text{ } T.$
 $\text{trail } S = \text{Propagated } L \text{ } C' \text{ } \# \text{ } M \implies$
 $\text{conflicting } S = \text{Some } E \implies$
 $-L \notin \# \text{ } E \implies E \neq \{\#\} \implies$
 $T \sim \text{tl-trail } S \implies$
 $P \text{ } S \text{ } T$ **and**
 $resolveH$: $\bigwedge L \text{ } E \text{ } M \text{ } D \text{ } T.$
 $\text{trail } S = \text{Propagated } L \text{ } E \text{ } \# \text{ } M \implies$
 $L \in \# \text{ } E \implies$
 $\text{hd-trail } S = \text{Propagated } L \text{ } E \implies$
 $\text{conflicting } S = \text{Some } D \implies$
 $-L \in \# \text{ } D \implies$
 $\text{get-maximum-level } (\text{trail } S) ((\text{remove1-mset } (-L) \text{ } D)) = \text{backtrack-lvl } S \implies$
 $T \sim \text{update-conflicting}$
 $(\text{Some } (\text{resolve-clss } L \text{ } D \text{ } E)) (\text{tl-trail } S) \implies$
 $P \text{ } S \text{ } T$ **and**
 $backtrackH$: $\bigwedge L \text{ } D \text{ } K \text{ } i \text{ } M1 \text{ } M2 \text{ } T \text{ } D'.$
 $\text{conflicting } S = \text{Some } (\text{add-mset } L \text{ } D) \implies$
 $(\text{Decided } K \text{ } \# \text{ } M1, \text{ } M2) \in \text{set } (\text{get-all-ann-decomposition } (\text{trail } S)) \implies$
 $\text{get-level } (\text{trail } S) \text{ } L = \text{backtrack-lvl } S \implies$
 $\text{get-level } (\text{trail } S) \text{ } L = \text{get-maximum-level } (\text{trail } S) (\text{add-mset } L \text{ } D') \implies$
 $\text{get-maximum-level } (\text{trail } S) \text{ } D' \equiv i \implies$
 $\text{get-level } (\text{trail } S) \text{ } K = i+1 \implies$
 $D' \subseteq \# \text{ } D \implies$
 $\text{clauses } S \models_{pm} \text{add-mset } L \text{ } D' \implies$
 $T \sim \text{cons-trail } (\text{Propagated } L (\text{add-mset } L \text{ } D'))$
 $(\text{reduce-trail-to } M1$
 $(\text{add-learned-clss } (\text{add-mset } L \text{ } D')$
 $(\text{update-conflicting } \text{None } S))) \implies$
 $P \text{ } S \text{ } T$
shows $P \text{ } S \text{ } T$
using $cdcl_W\text{-restart}$ **apply** ($\text{induct } T \text{ rule: } cdcl_W\text{-o.induct}$)
subgoal using $\text{assms}(2)$ **by** ($\text{auto elim: } decideE; \text{fail}$)
subgoal apply ($\text{elim } cdcl_W\text{-bjE } skipE \text{ resolveE } backtrackE$)
apply ($\text{frule } skipH; \text{simp}; \text{fail}$)
apply ($\text{cases trail } S; \text{auto elim!; resolveE intro!; resolveH}; \text{fail}$)
apply ($\text{frule } backtrackH; \text{simp}; \text{fail}$)
done
done

lemma $cdcl_W\text{-o-all-rules-induct}[\text{consumes } 1, \text{ case-names } decide \text{ backtrack } skip \text{ resolve}]$:

fixes $S \text{ } T :: 'st$

assumes

$cdcl_W\text{-o } S \text{ } T$ **and**

$\bigwedge T. \text{ decide } S \text{ } T \implies P \text{ } S \text{ } T$ **and**

$\bigwedge T. \text{ backtrack } S \text{ } T \implies P \text{ } S \text{ } T$ **and**

$\bigwedge T. \text{ skip } S \text{ } T \implies P \text{ } S \text{ } T$ **and**

$\bigwedge T. \text{ resolve } S \text{ } T \implies P \text{ } S \text{ } T$

shows $P \text{ } S \text{ } T$

using assms **by** ($\text{induct } T \text{ rule: } cdcl_W\text{-o.induct}$) ($\text{auto simp: } cdcl_W\text{-bj.simps}$)


```

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

```

```

lemma backtrack-backtrackg:
   $\langle \text{backtrack } S\ T \implies \text{backtrackg } S\ T \rangle$ 
  unfolding backtrack.simps backtrackg.simps
  by blast

```

```

lemma simple-backtrack-backtrackg:
   $\langle \text{simple-backtrack } S\ T \implies \text{backtrackg } S\ T \rangle$ 
  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-skipped:
  assumes
     $L$ : get-level (trail  $S$ )  $L = \text{backtrack-lvl } S$  and
     $M1$ :  $(\text{Decided } K \# M1, M2) \in \text{set } (\text{get-all-ann-decomposition } (\text{trail } S))$  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  $?M = \text{trail } S$ 
  assume  $L\text{-in-}M1$ :  $\neg ?thesis$ 
  obtain  $M2'$  where
     $Mc$ :  $\text{trail } S = M2' @ M2 @ \text{Decided } K \# M1$ 
    using  $M1$  by blast
  have  $Kc$ :  $\langle \text{undefined-lit } M2'\ K \rangle$  and  $KM2$ :  $\langle \text{undefined-lit } M2\ K \rangle \langle \text{atm-of } L \neq \text{atm-of } K \rangle$  and
     $\langle \text{undefined-lit } M2'\ L \rangle \langle \text{undefined-lit } M2\ L \rangle$ 
    using  $L\text{-in-}M1$  no-dup unfolding  $Mc$  by (auto simp: atm-of-eq-atm-of dest: defined-lit-no-dupD)
  then have  $g\text{-}M\text{-eq-}g\text{-}M1$ : get-level  $?M\ L = \text{get-level } M1\ L$ 

```

```

    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 ≤ 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 cdclW-restart-distinctinv-1:
  assumes
    cdclW-restart S S' and
    n-d: no-dup (trail S)
  shows no-dup (trail S')
  using assms(1)
proof (induct rule: cdclW-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-skipped[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 cdclW-restart-consistent-inv-2:
  assumes
    cdclW-restart S S' and
    no-dup (trail S)
  shows consistent-interp (lits-of-l (trail S'))
  using cdclW-restart-distinctinv-1[OF assms] distinct-consistent-interp by fast

```

```

definition cdclW-M-level-inv :: 'st ⇒ bool' where
cdclW-M-level-inv S ⇔
  consistent-interp (lits-of-l (trail S))
  ∧ no-dup (trail S)

```

```

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

```

```

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

```

lemma *rtrancpl-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*: *rtrancpl-induct*) (*auto intro*: *cdcl_W-restart-consistent-inv*)

lemma *trancpl-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*: *trancpl-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 $\exists K M1 M2. (Decided K \# M1, M2) \in set (get-all-ann-decomposition (trail S)) \wedge$

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** $(Decided K \# M1, M2) \in set (get-all-ann-decomposition (trail S))$

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 (\sim)

declare *state-eq-trans[trans]*

lemma *propagate-state-eq-compatible*:

assumes

propa: propagate S T **and**

SS': S \sim S' **and**

TT': T \sim T'

shows *propagate S' T'*

proof –

obtain *C L* **where**

```

  conf: conflicting  $S = \text{None}$  and
   $C$ :  $C \in \#$  clauses  $S$  and
   $L$ :  $L \in \#$   $C$  and
   $tr$ :  $\text{trail } S \models_{as} C\text{Not } (\text{remove1-mset } L \ C)$  and
  undef: undefined-lit ( $\text{trail } S$ )  $L$  and
   $T$ :  $T \sim \text{cons-trail } (\text{Propagated } L \ C) \ S$ 
using propa by (elim propagateE) auto

have  $C'$ :  $C \in \#$  clauses  $S'$ 
  using  $SS' \ C$ 
  by (auto simp: clauses-def)
have  $T'$ :  $\langle T' \sim \text{cons-trail } (\text{Propagated } L \ C) \ S' \rangle$ 
  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  $SS' \ \text{conf } C' \ L \ tr \ \text{undef } TT' \ T \ T'$  by auto
qed

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

have  $D'$ :  $D \in \#$  clauses  $S'$ 
  using  $D \ SS' \ \text{by}$  fastforce

have  $T'$ :  $\langle T' \sim \text{update-conflicting } (\text{Some } D) \ S' \rangle$ 
  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  $SS' \ \text{conf } D' \ tr \ TT' \ T \ T'$  by auto
qed

lemma backtrack-state-eq-compatible:
assumes
  bt: backtrack  $S \ T$  and
   $SS'$ :  $S \sim S'$  and
   $TT'$ :  $T \sim T'$ 
shows backtrack  $S' \ T'$ 
proof -
obtain  $D \ L \ K \ i \ M1 \ M2 \ D'$  where
  conf: conflicting  $S = \text{Some } (\text{add-mset } L \ D)$  and
  decomp:  $(\text{Decided } K \ \# \ M1, \ M2) \in \text{set } (\text{get-all-ann-decomposition } (\text{trail } S))$  and
  lev:  $\text{get-level } (\text{trail } S) \ L = \text{backtrack-lvl } S$  and
  max:  $\text{get-level } (\text{trail } S) \ L = \text{get-maximum-level } (\text{trail } S) \ (\text{add-mset } L \ D')$  and

```

```

max-D: get-maximum-level (trail S) D'  $\equiv$  i and
lev-K: get-level (trail S) K = Suc i and
D'-D:  $\langle D' \subseteq_{\#} D \rangle$  and
NU-DL:  $\langle \text{clauses } S \models_{pm} \text{add-mset } L \ D' \rangle$  and
T:  $T \sim \text{cons-trail } (\text{Propagated } L \ (\text{add-mset } L \ D'))$ 
    (reduce-trail-to M1
      (add-learned-cls (add-mset L D')
        (update-conflicting None S)))
  using bt by (elim backtrackE) metis
let ?D =  $\langle \text{add-mset } L \ D \rangle$ 
let ?D' =  $\langle \text{add-mset } L \ D' \rangle$ 
have D': conflicting S' = Some ?D
  using SS' conf by (cases conflicting S') auto

have T'-S:  $T' \sim \text{cons-trail } (\text{Propagated } L \ ?D')$ 
  (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':  $T' \sim \text{cons-trail } (\text{Propagated } L \ ?D')$ 
  (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':  $S \sim S'$  and
    TT':  $T \sim T'$ 
  shows decide S' T'
  using assms
proof -
  obtain L :: 'v literal where
    f4: undefined-lit (trail S) L
    atm-of L  $\in$  atms-of-mm (init-clss S)
     $T \sim \text{cons-trail } (\text{Decided } L) \ S$ 
  using dec decide.simps by blast
  have cons-trail (Decided L)  $S' \sim T'$ 
  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: $T \sim \text{tl-trail } S$

using *skip* **by** (*elim skipE*) *simp*

obtain *E'* **where** *E'*: *conflicting S' = Some E'*

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

have *T'*: $\langle T' \sim \text{tl-trail } S' \rangle$

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: $T \sim \text{update-conflicting (Some (resolve-cls L D E)) (tl-trail S)}$

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'*: $\langle T' \sim \text{update-conflicting (Some (remove1-mset (- L) D') \cup \# \text{remove1-mset L E})} \rangle$
 $(\text{tl-trail } S')$

proof –

have *tl-trail S* $\sim \text{tl-trail } S'$

```

    using SS' by (auto simp: tl-trail-state-eq)
  then show ?thesis
    using T T'T  $\langle D = D' \rangle$  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':  $S \sim S'$  and
    TT':  $T \sim T'$ 
  shows forget S' T'
proof -
  obtain C where
    conf: conflicting S = None and
    C:  $C \in \#$  learned-clss S and
    tr:  $\neg(\text{trail } S) \models_{\text{asm}} \text{clauses } S$  and
    C1:  $C \notin \text{set } (\text{get-all-mark-of-propagated } (\text{trail } S))$  and
    C2:  $C \notin \#$  init-clss S and
    ent:  $\langle \text{removeAll-mset } C \text{ (clauses } S) \rangle \models_{\text{pm}} C$  and
    T:  $T \sim \text{remove-cls } C S$ 
  using forget by (elim forgetE) simp
  have T':  $\langle T' \sim \text{remove-cls } C S' \rangle$ 
  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
    cdclW-restart S T and  $\neg \text{restart } S T$  and
    S  $\sim S'$ 
    T  $\sim T'$ 
  shows cdclW-restart S' T'
  using assms by (meson backtrack backtrack-state-eq-compatible bj cdclW-restart.simps
    cdclW-o-rule-cases cdclW-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
   $cdcl_W\text{-bj } S \ T$ 
   $T \sim T'$ 
shows  $cdcl_W\text{-bj } S \ T'$ 
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
   $cdcl_W\text{-bj}^{++} S \ T$ 
   $S \sim S'$  and
   $T \sim T'$ 
shows  $cdcl_W\text{-bj}^{++} S' \ T'$ 
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  $IH = \text{this}(3)[OF \ \text{this}(4)]$ 
have  $cdcl_W\text{-restart}^{++} S \ T$ 
  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  $cdcl_W\text{-bj}^{++} T \ T'$ 
  using  $\langle U \sim T' \rangle$  cdcl_W-bj-state-eq-compatible[of T U]  $\langle cdcl_W\text{-bj } T \ U \rangle$  by auto
then show ?case
  using  $IH[of \ T]$  by auto
qed

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

lemma resolve-unique:
   $resolve \ S \ T \implies resolve \ S \ T' \implies T \sim T'$ 
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
   $cdcl_W\text{-o } S \ S'$  and
  inv: cdcl_W-M-level-inv S
shows  $init\text{-clss } S = init\text{-clss } S'$ 
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
   $cdcl_W\text{-o}^{++} S \ S'$  and
  inv: cdcl_W-M-level-inv S
shows  $init\text{-clss } S = init\text{-clss } S'$ 
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 \implies 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 \implies 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 initial 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 \models_{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-clss*)

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-clss-union-mset-true-clss-clss-or-not-true-clss-clss-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-clss 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 \implies 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** $\text{confl} = \text{this}(1)$ **and** $C = \text{this}(2)$ **and** $C-L = \text{this}(3)$ **and**
 $\text{tr} = \text{this}(4)$ **and** $\text{undef} = \text{this}(5)$ **and** $T = \text{this}(6)$
have $\text{atm-CL: } \text{atms-of } C \subseteq \text{atms-of-mm } (\text{init-clss } S)$
using C **alien unfolding no-strange-atm-def**
by (*auto simp: clauses-def dest!: multi-member-split*)
show $?case$
unfolding $\text{no-strange-atm-def}$
proof (*intro conjI allI impI, goal-cases*)
case (1 C)
then show $?case$
using $\text{confl } T \text{ undef}$ **by** *auto*
next
case (2 $L' \text{ mark}'$)
then show $?case$
using $C-L \ T \ \text{alien undef atm-CL unfolding no-strange-atm-def clauses-def}$ **by** (*auto 5 5*)
next
case 3
show $?case$ **using** $T \ \text{alien undef unfolding no-strange-atm-def}$ **by** *auto*
next
case 4
show $?case$
using $T \ \text{alien undef } C-L \ \text{atm-CL unfolding no-strange-atm-def}$ **by** (*auto simp: atms-of-def*)
qed
qed

lemma $\text{atms-of-ms-learned-clss-restart-state-in-atms-of-ms-learned-clssI:}$

$\text{atms-of-mm } (\text{learned-clss } S) \subseteq \text{atms-of-mm } (\text{init-clss } S) \implies$
 $x \in \text{atms-of-mm } (\text{learned-clss } T) \implies$
 $\text{learned-clss } T \subseteq \# \text{ learned-clss } S \implies$
 $x \in \text{atms-of-mm } (\text{init-clss } S)$
by (*meson atms-of-ms-mono contra-subsetD set-mset-mono*)

lemma (**in** $-$) $\text{atms-of-subset-mset-mono: } \langle D' \subseteq \# D \implies \text{atms-of } D' \subseteq \text{atms-of } D \rangle$
by (*auto simp: atms-of-def*)

lemma $\text{cdcl}_W\text{-restart-no-strange-atm-explicit:}$

assumes

$\text{cdcl}_W\text{-restart } S \ S' \text{ and}$

$\text{lev: cdcl}_W\text{-M-level-inv } S \text{ and}$

$\text{conf: } \forall T. \text{ conflicting } S = \text{Some } T \longrightarrow \text{atms-of } T \subseteq \text{atms-of-mm } (\text{init-clss } S) \text{ and}$

$\text{decided: } \forall L \ \text{mark}. \text{ Propagated } L \ \text{mark} \in \text{set } (\text{trail } S)$

$\longrightarrow \text{atms-of mark} \subseteq \text{atms-of-mm } (\text{init-clss } S) \text{ and}$

$\text{learned: } \text{atms-of-mm } (\text{learned-clss } S) \subseteq \text{atms-of-mm } (\text{init-clss } S) \text{ and}$

$\text{trail: atm-of ' (lits-of-l (trail } S)) \subseteq \text{atms-of-mm } (\text{init-clss } S)$

shows

$(\forall T. \text{ conflicting } S' = \text{Some } T \longrightarrow \text{atms-of } T \subseteq \text{atms-of-mm } (\text{init-clss } S')) \wedge$

$(\forall L \ \text{mark}. \text{ Propagated } L \ \text{mark} \in \text{set } (\text{trail } S') \longrightarrow \text{atms-of mark} \subseteq \text{atms-of-mm } (\text{init-clss } S')) \wedge$

$\text{atms-of-mm } (\text{learned-clss } S') \subseteq \text{atms-of-mm } (\text{init-clss } S') \wedge$

$\text{atm-of ' (lits-of-l (trail } S')) \subseteq \text{atms-of-mm } (\text{init-clss } S')$

$(\text{is } ?C \ S' \wedge ?M \ S' \wedge ?U \ S' \wedge ?V \ S')$

using $\text{assms}(1)$

proof (*induct rule: cdcl_W-restart-all-induct*)

case (*propagate C L T*) **note** $\text{confl} = \text{this}(1)$ **and** $C-L = \text{this}(2)$ **and** $\text{tr} = \text{this}(3)$ **and** $\text{undef} =$
 $\text{this}(4)$
and $T = \text{this}(5)$

```

show ?case
  using propagate-rule[OF propagate.hyps(1-3) - propagate.hyps(5,6), simplified]
  propagate.hyps(4) propagate-no-strange-atm-inv[of S T]
  conf decided learned trail unfolding no-strange-atm-def by presburger
next
case (decide L)
then show ?case using learned decided conf trail unfolding clauses-def by auto
next
case (skip L C M D)
then show ?case using learned decided conf trail by auto
next
case (conflict D T) note D-S = this(2) and T = this(4)
have D: atm-of ' set-mset D  $\subseteq$   $\bigcup$  (atms-of ' (set-mset (clauses S)))
  using D-S by (auto simp add: atms-of-def atms-of-ms-def)
moreover {
  fix xa :: 'v literal
  assume a1: atm-of ' set-mset D  $\subseteq$  ( $\bigcup$  x $\in$ set-mset (init-clss S). atms-of x)
     $\cup$  ( $\bigcup$  x $\in$ set-mset (learned-clss S). atms-of x)
  assume a2:
    ( $\bigcup$  x $\in$ set-mset (learned-clss S). atms-of x)  $\subseteq$  ( $\bigcup$  x $\in$ set-mset (init-clss S). atms-of x)
  assume xa  $\in$ # D
  then have atm-of xa  $\in$  UNION (set-mset (init-clss S)) atms-of
    using a2 a1 by (metis (no-types) Un-iff atm-of-lit-in-atms-of atms-of-def subset-Un-eq)
  then have  $\exists$  m $\in$ set-mset (init-clss S). atm-of xa  $\in$  atms-of m
    by blast
} note H = this
ultimately show ?case using conflict.premis 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:  $\bigwedge$  L mark. Propagated L mark  $\in$  set (trail S)  $\implies$  atms-of mark  $\subseteq$  atms-of-mm (init-clss S)
  using decided by simp
show ?case unfolding clauses-def apply (intro conjI)
  using conf confl T trail C unfolding clauses-def apply (auto dest!: H)[]
  using T trail C C-le apply (auto dest!: H)[]
  using T learned C-le atms-of-ms-remove-subset[of set-mset (learned-clss S)] apply auto[]
  using T trail C-le apply (auto simp: clauses-def lits-of-def)[]
done
next
case (backtrack L D K i M1 M2 T 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: cdclW-M-level-inv-decomp)
moreover have set M1  $\subseteq$  set (trail S)
  using decomp by auto
then have M: ?M T
  using decided conf confl T decomp lev lev-K D-D'
  by (auto simp: image-subset-iff clauses-def cdclW-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: cdclW-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: cdclW-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)*
 $\longleftrightarrow ((\forall T. \text{conflicting } S = \text{Some } T \longrightarrow \text{distinct-mset } T)$
 $\wedge \text{distinct-mset-mset (learned-cls } S)$
 $\wedge \text{distinct-mset-mset (init-cls } S)$
 $\wedge (\forall L \text{ mark. (Propagated } L \text{ mark} \in \text{set (trail } S) \longrightarrow \text{distinct-mset mark})))$

lemma *distinct-cdcl_W-state-decomp*:
assumes *distinct-cdcl_W-state S*
shows
 $\forall T. \text{conflicting } S = \text{Some } T \longrightarrow \text{distinct-mset } T$ **and**
 $\text{distinct-mset-mset (learned-cls } S)$ **and**
 $\text{distinct-mset-mset (init-cls } S)$ **and**
 $\forall L \text{ mark. (Propagated } L \text{ mark} \in \text{set (trail } S) \longrightarrow \text{distinct-mset mark})$
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-cdclW-state-def by auto

lemma distinct-cdclW-state-S0-cdclW-restart[simp]:
  distinct-mset-mset N  $\implies$  distinct-cdclW-state (init-state N)
  unfolding distinct-cdclW-state-def by auto

lemma distinct-cdclW-state-inv:
  assumes
    cdclW-restart S S' and
    lev-inv: cdclW-M-level-inv S and
    distinct-cdclW-state S
  shows distinct-cdclW-state S'
  using assms(1,2,2,3)
proof (induct rule: cdclW-restart-all-induct)
  case (backtrack L D K i M1 M2 D')
  then show ?case
    using lev-inv unfolding distinct-cdclW-state-def
    by (auto dest: get-all-ann-decomposition-incl distinct-mset-mono simp: cdclW-M-level-inv-decomp)
next
  case restart
  then show ?case
    unfolding distinct-cdclW-state-def distinct-mset-set-def clauses-def by auto
next
  case resolve
  then show ?case
    by (auto simp add: distinct-cdclW-state-def distinct-mset-set-def clauses-def)
qed (auto simp: distinct-cdclW-state-def distinct-mset-set-def clauses-def
  dest!: in-diffD)

lemma rtanclp-distinct-cdclW-state-inv:
  assumes
    cdclW-restart** S S' and
    cdclW-M-level-inv S and
    distinct-cdclW-state S
  shows distinct-cdclW-state S'
  using assms apply (induct rule: rtanclp-induct)
  using distinct-cdclW-state-inv rtanclp-cdclW-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
 $\forall L \text{ mark } a \text{ b. } a @ \text{Propagated } L \text{ mark} \# b = (\text{trail } S)$
 $\longrightarrow (b \models_{as} CNot (\text{mark} - \{\#L\}) \wedge L \in \# \text{ mark})$

definition *cdcl_W-conflicting :: 'st \Rightarrow bool where*
cdcl_W-conflicting S \longleftrightarrow
 $(\forall T. \text{conflicting } S = \text{Some } T \longrightarrow \text{trail } S \models_{as} CNot T) \wedge \text{every-mark-is-a-conflict } S$

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*) $D' \equiv i$ **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-conf: *conflicting S = Some D* **and**
lev-K: *get-level* (*trail S*) $K = \text{Suc } i$ **and**
T: $T \sim \text{cons-trail } K''$
(reduce-trail-to M1
(add-learned-cls (add-mset L D')
(update-conflicting None S))) **and**
conf: $\forall T. \text{conflicting } S = \text{Some } T \longrightarrow \text{trail } S \models_{\text{as}} \text{CNot } T$ **and**
D-D': $\langle D' \subseteq \# D \rangle$
shows *atms-of* $D' \subseteq \text{atm-of ' lits-of-l (tl (trail T))}$
proof (*rule ccontr*)
let $?k = \text{get-maximum-level} (\text{trail } S) (\text{add-mset } L \ D')$

have $\text{trail } S \models_{\text{as}} \text{CNot } D$ **using** *conf* *S-conf* **by** *auto*
then have $\text{trail } S \models_{\text{as}} \text{CNot } D'$
using *D-D'* **by** (*auto simp: true-annots-true-cls-def-iff-negation-in-model*)
then have *vars-of-D*: *atms-of* $D' \subseteq \text{atm-of ' lits-of-l (trail S)}$ **unfolding** *atms-of-def*
by (*meson image-subsetI true-annots-CNot-all-atms-defined*)

obtain $M0$ **where** $M: \text{trail } S = M0 @ M2 @ \text{Decided } K \# M1$
using *decomp* **by** *auto*

have *max*: $?k = \text{count-decided} (M0 @ M2 @ \text{Decided } K \# M1)$
using *S-lvl* **unfolding** M **by** *simp*
assume $a: \neg ?thesis$
then obtain L' **where**
 $L': L' \in \text{atms-of } D'$ **and**
 $L'\text{-notin-}M1: L' \notin \text{atm-of ' lits-of-l } M1$
using T *decomp inv* **by** (*auto simp: cdcl_W-M-level-inv-decomp*)

obtain L'' **where**
 $L'' \in \# D'$ **and**
 $L'': L' = \text{atm-of } L''$
using $L' L'\text{-notin-}M1$ **unfolding** *atms-of-def* **by** *auto*
then have $L'\text{-in: defined-lit } (M0 @ M2 @ \text{Decided } K \# []) L''$
using *vars-of-D* $L'\text{-notin-}M1 L'$ **unfolding** M
by (*auto dest: in-atms-of-minusD*
simp: defined-lit-append defined-lit-map lits-of-def image-Un)
have $L''\text{-}M1: \langle \text{undefined-lit } M1 L'' \rangle$
using $L'\text{-notin-}M1 L''$ **by** (*auto simp: defined-lit-map lits-of-def*)
have $\langle \text{undefined-lit } (M0 @ M2) K \rangle$
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''*:
 $\text{get-level} (\text{trail } S) L'' = \text{get-level} (M0 @ M2 @ \text{Decided } K \# []) L'' + i$
using $L'\text{-notin-}M1 L''\text{-}M1 L''$ *get-level-skip-end*[*OF* $L'\text{-in}[\text{unfolded } L'']$, *of* $M1$] M **by** *auto*
moreover {
consider
 $(M0) \text{ defined-lit } M0 L'' \mid$
 $(M2) \text{ defined-lit } M2 L'' \mid$


```

(K) L' = atm-of K
using inv L'-in unfolding L''
by (auto simp: cdclW-M-level-inv-def defined-lit-append)
then have get-level (M0 @ M2 @ Decided K # []) L'' ≥ Suc 0
proof cases
case M0
then have L' ≠ 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'' cdclW-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'' cdclW-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'' ≥ i + 1
  using lev-L'' unfolding M by simp
then have get-maximum-level (trail S) D' ≥ i + 1
  using get-maximum-level-ge-get-level[OF ⟨L'' ∈# D'⟩, of trail S] by auto
then show False using i by auto
qed

```

lemma *distinct-atms-of-incl-not-in-other*:

```

assumes
  a1: no-dup (M @ M') and
  a2: atms-of D ⊆ atm-of ' lits-of-l M' and
  a3: x ∈ atms-of D
shows x ∉ 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' ≡ 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-conf: 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
  conf: ∀ T. conflicting S = Some T ⟶ trail S ⊨as CNot T and
  D-D': ⟨D' ⊆# D⟩
shows M1 ⊨as CNot D' and
  ⟨atm-of (lit-of K'') = atm-of L ⟹ no-dup (trail T)⟩

```

proof –

obtain $M0$ **where** M : $\text{trail } S = M0 @ M2 @ \text{Decided } K \# M1$
using *decomp* **by** *auto*
have $\text{vars-of-}D$: $\text{atms-of } D' \subseteq \text{atm-of } \langle \text{ lits-of-}l \text{ } M1 \rangle$
using *backtrack-atms-of-D-in-M1* [*OF* *assms*] *decomp* T **by** *auto*
have no-dup ($\text{trail } S$) **using** *inv* **by** (*auto simp: cdcl_W-M-level-inv-decomp*)
then have $\text{vars-in-}M1$: $\forall x \in \text{atms-of } D'. x \notin \text{atm-of } \langle \text{ lits-of-}l (M0 @ M2 @ \text{Decided } K \# []) \rangle$
using $\text{vars-of-}D$ *distinct-atms-of-incl-not-in-other* [*of* $M0 @ M2 @ \text{Decided } K \# [] \text{ } M1$]
unfolding M **by** *auto*
have $\text{trail } S \models_{\text{as}} \text{CNot } D$
using *S-confl confl* **unfolding** M *true-annots-true-cls-def-iff-negation-in-model*
by (*auto dest!: in-diffD*)
then have $\text{trail } S \models_{\text{as}} \text{CNot } D'$
using $D-D'$ **unfolding** *true-annots-true-cls-def-iff-negation-in-model* **by** *auto*
then show $M1-D'$: $M1 \models_{\text{as}} \text{CNot } D'$
using *true-annots-remove-if-notin-vars* [*of* $M0 @ M2 @ \text{Decided } K \# [] \text{ } M1 \text{ } \text{CNot } D'$]
vars-in-M1 S-confl confl **unfolding** M *lits-of-def* **by** *simp*
have $M1$: $\langle \text{count-decided } M1 = i \rangle$
using *lev-K inv i vars-in-M1* **unfolding** M
by *simp*
have $\text{lev-}L$: $\langle \text{get-level } (\text{trail } S) \text{ } L = \text{backtrack-lvl } S \rangle$ **and** $\langle i < \text{backtrack-lvl } S \rangle$
using *S-lvl i lev-K*
by (*auto simp: max-def get-maximum-level-add-mset*)
have $\langle \text{no-dup } M1 \rangle$
using T *decomp inv* **by** (*auto simp: M dest: no-dup-appendD*)
moreover have $\langle \text{undefined-lit } M1 \text{ } L \rangle$
using *backtrack-lit-skipped* [*of* $S \text{ } L$, *OF - decomp*]
using $\text{lev-}L$ *inv i M1* $\langle i < \text{backtrack-lvl } S \rangle$ **unfolding** M
by (*auto simp: split: if-splits*)
moreover have $\langle \text{atm-of } (\text{lit-of } K'') = \text{atm-of } L \implies$
 $\text{undefined-lit } M1 \text{ } L \longleftrightarrow \text{undefined-lit } M1 \text{ } (\text{lit-of } K'') \rangle$
by (*simp add: defined-lit-map*)
ultimately show $\langle \text{atm-of } (\text{lit-of } K'') = \text{atm-of } L \implies \text{no-dup } (\text{trail } T) \rangle$
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 $S \text{ } S'$ **and**
inv: cdcl_W-M-level-inv S **and**
decomp: all-decomposition-implies-m (*clauses* S) (*get-all-ann-decomposition* ($\text{trail } S$)) **and**
learned: cdcl_W-learned-clause S **and**
confl: $\forall T. \text{conflicting } S = \text{Some } T \longrightarrow \text{trail } S \models_{\text{as}} \text{CNot } T$ **and**
alien: no-strange-atm S
shows *all-decomposition-implies-m* (*clauses* S') (*get-all-ann-decomposition* ($\text{trail } S'$))
using *assms*(1)

proof (*induct rule: cdcl_W-restart-all-induct*)

case *restart*

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

next

case (*forget* $C \text{ } T$) **note** $C = \text{this}(2)$ **and** $\text{cls-}C = \text{this}(6)$ **and** $T = \text{this}(7)$

show $?case$

unfolding *all-decomposition-implies-def Ball-def*

proof (*intro allI, clarify*)

fix $a \text{ } b$

```

assume  $(a, b) \in \text{set } (\text{get-all-ann-decomposition } (\text{trail } T))$ 
then have  $\text{unmark-l } a \cup \text{set-mset } (\text{clauses } S) \models_{ps} \text{unmark-l } b$ 
  using decomp T by (auto simp add: all-decomposition-implies-def)
moreover {
  have  $a1:C \in \# \text{ clauses } S$ 
    using C by (auto simp: clauses-def)
  have  $\text{clauses } T = \text{clauses } (\text{remove-cls } C S)$ 
    using T by auto
  then have  $\text{clauses } T \models_{psm} \text{clauses } S$ 
    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  $\text{unmark-l } a \cup \text{set-mset } (\text{clauses } T) \models_{ps} \text{unmark-l } b$ 
    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  $tr = \text{this}(1)$  and  $T = \text{this}(7)$ 
let ?decomp = get-all-ann-decomposition M
have  $M: \text{set } ?decomp = \text{insert } (\text{hd } ?decomp) (\text{set } (\text{tl } ?decomp))$ 
  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  $tr = \text{this}(1)$  and  $T = \text{this}(5)$ 
have  $M: \text{set } (\text{get-all-ann-decomposition } M)$ 
  =  $\text{insert } (\text{hd } (\text{get-all-ann-decomposition } M)) (\text{set } (\text{tl } (\text{get-all-ann-decomposition } M)))$ 
  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  $S = \text{this}(1)$  and  $\text{undef} = \text{this}(2)$  and  $T = \text{this}(4)$ 
show ?case using decomp T undef unfolding S all-decomposition-implies-def by auto
next
case (propagate C L T) note  $\text{propa} = \text{this}(2)$  and  $L = \text{this}(3)$  and  $S\text{-CNot-}C = \text{this}(4)$  and
 $\text{undef} = \text{this}(5)$  and  $T = \text{this}(6)$ 
obtain  $a y$  where  $ay: \text{hd } (\text{get-all-ann-decomposition } (\text{trail } S)) = (a, y)$ 
  by (cases hd (get-all-ann-decomposition (trail S)))
then have  $M: \text{trail } S = y @ a$  using get-all-ann-decomposition-decomp by blast
have  $M': \text{set } (\text{get-all-ann-decomposition } (\text{trail } S))$ 
  =  $\text{insert } (a, y) (\text{set } (\text{tl } (\text{get-all-ann-decomposition } (\text{trail } S))))$ 
  using ay by (cases get-all-ann-decomposition (trail S)) auto
have  $\text{unm-ay: unmark-l } a \cup \text{set-mset } (\text{clauses } S) \models_{ps} \text{unmark-l } y$ 
  using decomp ay unfolding all-decomposition-implies-def
  by (cases get-all-ann-decomposition (trail S)) fastforce+
then have  $a\text{-Un-N-M: unmark-l } a \cup \text{set-mset } (\text{clauses } S) \models_{ps} \text{unmark-l } (\text{trail } S)$ 
  unfolding M by (auto simp add: all-in-true-clss-clss image-Un)

have  $\text{unmark-l } a \cup \text{set-mset } (\text{clauses } S) \models_p \{\#L\# \}$  (is ?I  $\models_p -$ )
proof (rule true-clss-clss-plus-CNot)
  show ?I  $\models_p \text{add-mset } L (\text{remove1-mset } L C)$ 

```

```

    apply (rule true-clss-clss-in-imp-true-clss-cl[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)  $\models_{ps}$  CNot (remove1-mset L C)
    using S-CNot-C by (blast dest: true-annots-true-clss-clss)
  then show ?I  $\models_{ps}$  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  $\bigwedge aa\ b.$ 
   $\forall (Ls, seen) \in \text{set } (\text{get-all-ann-decomposition } (y @ a)).$ 
   $\text{unmark-l } Ls \cup \text{set-mset } (\text{clauses } S) \models_{ps} \text{unmark-l } seen \implies$ 
   $(aa, b) \in \text{set } (tl (\text{get-all-ann-decomposition } (y @ a))) \implies$ 
   $\text{unmark-l } aa \cup \text{set-mset } (\text{clauses } S) \models_{ps} \text{unmark-l } b$ 
  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  $\forall l \in \text{set } M2. \neg \text{is-decided } l$ 
  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 =  $\langle \text{add-mset } L\ D \rangle$ 
let ?D' =  $\langle \text{add-mset } L\ D' \rangle$ 
show ?case unfolding all-decomposition-implies-def
proof
  fix x
  assume  $x \in \text{set } (\text{get-all-ann-decomposition } (\text{trail } T))$ 
  then have  $x: x \in \text{set } (\text{get-all-ann-decomposition } (\text{Propagated } L\ ?D' \# M1))$ 
    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)  $x \in \text{set } ?tl$ 
  using x by (cases ?m) auto
  then show case x of (Ls, seen)  $\Rightarrow \text{unmark-l } Ls \cup \text{set-mset } (\text{clauses } T) \models_{ps} \text{unmark-l } seen$ 
proof cases
  case tl
  then have  $x \in \text{set } (\text{get-all-ann-decomposition } (\text{trail } S))$ 
    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))

```



```

    if  $\langle a @ \text{Propagated } L a \text{ mark} \# b = \text{Decided } L \# \text{trail } S \rangle$  for  $a \text{ } L a \text{ mark } b$ 
    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  $\langle b \models_{as} CNot (\text{mark} - \{\#La\}) \wedge La \in \# \text{mark} \rangle$ 
    if a @ Propagated L a mark # b = Propagated L C' # M for L a mark a b
    using mark-confl that unfolding skip.hyps(1) by simp
  ultimately show  $b \models_{as} CNot (\text{mark} - \{\#L'\}) \wedge L' \in \# \text{mark}$  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  $b \models_{as} CNot (\text{mark} - \{\#L'\}) \wedge L' \in \# \text{mark}$ 
    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  $\forall l \in \text{set } M2. \neg \text{is-decided } l$ 
  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: cdclW-M-level-inv-decomp)
let ?D = add-mset L D
let ?D' = add-mset L D'
have M1-D': M1  $\models_{as} CNot D'$ 
  using backtrack-M1-CNot-D'[of S D'  $\langle i \rangle$  K M1 M2 L  $\langle \text{add-mset } L D \rangle$  T  $\langle \text{Propagated } L (\text{add-mset } L D') \rangle$ ]
  conf lev backtrack by (auto simp: subset-mset-trans-add-mset cdclW-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  $a @ \text{Propagated } L a \text{ mark } \# b = \text{trail } T$ 
then consider
  ( $hd\text{-}tr$ )  $a = []$  and
    ( $\text{Propagated } L a \text{ mark} :: ('v, 'v \text{ clause}) \text{ ann-lit} = \text{Propagated } L ?D'$  and
       $b = M1$  |
    ( $tl\text{-}tr$ )  $tl a @ \text{Propagated } L a \text{ mark } \# b = M1$ 
    using  $M T \text{ decomp lev}$  by ( $\text{cases } a$ ) ( $\text{auto simp: } cdcl_W\text{-}M\text{-level-inv-def}$ )
then show  $b \models_{as} CNot (\text{mark} - \{\#La\# \}) \wedge La \in \# \text{ mark}$ 
proof cases
  case  $hd\text{-}tr$  note  $A = \text{this}(1)$  and  $P = \text{this}(2)$  and  $b = \text{this}(3)$ 
  show  $b \models_{as} CNot (\text{mark} - \{\#La\# \}) \wedge La \in \# \text{ mark}$ 
    using  $P M1\text{-}D'$   $b$  by auto
next
  case  $tl\text{-}tr$ 
  then obtain  $c'$  where  $c' @ \text{Propagated } L a \text{ mark } \# b = \text{trail } S$ 
    unfolding  $M$  by auto
  then show  $b \models_{as} CNot (\text{mark} - \{\#La\# \}) \wedge La \in \# \text{ mark}$ 
    using  $\text{mark-confli}$  by auto
qed
qed
qed

lemma  $cdcl_W\text{-conflicting-is-false}$ :
assumes
   $cdcl_W\text{-restart } S S'$  and
   $M\text{-lev: } cdcl_W\text{-}M\text{-level-inv } S$  and
   $\text{confli-inv: } \forall T. \text{conflicting } S = \text{Some } T \longrightarrow \text{trail } S \models_{as} CNot T$  and
   $\text{decided-confli: } \forall L \text{ mark } a b. a @ \text{Propagated } L \text{ mark } \# b = (\text{trail } S)$ 
     $\longrightarrow (b \models_{as} CNot (\text{mark} - \{\#L\# \}) \wedge L \in \# \text{ mark})$  and
   $\text{dist: distinct-}cdcl_W\text{-state } S$ 
shows  $\forall T. \text{conflicting } S' = \text{Some } T \longrightarrow \text{trail } S' \models_{as} CNot T$ 
using  $\text{assms}(1,2)$ 
proof ( $\text{induct rule: } cdcl_W\text{-restart-all-induct}$ )
  case ( $\text{skip } L C' M D T$ ) note  $tr\text{-}S = \text{this}(1)$  and  $\text{confli} = \text{this}(2)$  and  $L\text{-}D = \text{this}(3)$  and  $T = \text{this}(5)$ 
  have  $D: \text{Propagated } L C' \# M \models_{as} CNot D$  using  $\text{assms skip}$  by auto
  moreover have  $L \notin \# D$ 
  proof ( $\text{rule ccontr}$ )
    assume  $\neg ?thesis$ 
    then have  $- L \in \text{lits-of-}l M$ 
      using  $\text{in-}CNot\text{-implies-uminus}(2)[\text{of } L D \text{Propagated } L C' \# M]$ 
       $\langle \text{Propagated } L C' \# M \models_{as} CNot D \rangle$  by simp
    then show  $\text{False}$ 
      using  $M\text{-lev } tr\text{-}S$  by ( $\text{fastforce dest: } cdcl_W\text{-}M\text{-level-inv-decomp}(2)$ )
       $\text{simp: Decided-Propagated-in-iff-in-lits-of-}l$ )
  qed
  ultimately show  $?case$ 
    using  $tr\text{-}S \text{ confli } L\text{-}D T$  unfolding  $cdcl_W\text{-}M\text{-level-inv-def}$ 
    by ( $\text{auto intro: true-annots-}CNot\text{-lit-of-notin-skip}$ )
next
  case ( $\text{resolve } L C M D T$ ) note  $tr = \text{this}(1)$  and  $LC = \text{this}(2)$  and  $\text{confli} = \text{this}(4)$  and  $LD = \text{this}(5)$ 
  and  $T = \text{this}(7)$ 
  let  $?C = \text{remove1-mset } L C$ 
  let  $?D = \text{remove1-mset } (-L) D$ 
  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-cdclW-state-def by auto
  then have -L  $\notin$  # ?D using  $\langle$ distinct-mset (?D + {#- L#}) $\rangle$  by auto
  have Propagated L (?C + {#L#}) # M  $\models_{as}$  CNot ?D  $\cup$  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  $\langle$ - L  $\notin$  # ?D $\rangle$  tr
    unfolding cdclW-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 cdclW-M-level-inv-decomp)

```

```

lemma cdclW-conflicting-decomp:
  assumes cdclW-conflicting S
  shows
     $\forall T. \text{conflicting } S = \text{Some } T \longrightarrow \text{trail } S \models_{as} \text{CNot } T$  and
     $\forall L \text{ mark } a \ b. a \ @ \ \text{Propagated } L \text{ mark } \# \ b = (\text{trail } S) \longrightarrow$ 
       $(b \models_{as} \text{CNot } (\text{mark} - \{\#L\}) \wedge L \in \# \text{ mark})$ 
  using assms unfolding cdclW-conflicting-def by blast+

```

```

lemma cdclW-conflicting-decomp2:
  assumes cdclW-conflicting S and conflicting S = Some T
  shows trail S  $\models_{as}$  CNot T
  using assms unfolding cdclW-conflicting-def by blast+

```

```

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

```

```

definition cdclW-learned-clauses-entailed-by-init where
   $\langle \text{cdcl}_W\text{-learned-clauses-entailed-by-init } S \longleftrightarrow \text{init-clss } S \models_{psm} \text{learned-clss } S \rangle$ 

```

```

lemma cdclW-learned-clauses-entailed-init[simp]:
   $\langle \text{cdcl}_W\text{-learned-clauses-entailed-by-init } (\text{init-state } N) \rangle$ 
  by (auto simp: cdclW-learned-clauses-entailed-by-init-def)

```

```

lemma cdclW-learned-clauses-entailed:
  assumes
    cdclW-restart: cdclW-restart S S' and
    2: cdclW-learned-clause S and
    9:  $\langle \text{cdcl}_W\text{-learned-clauses-entailed-by-init } S \rangle$ 
  shows  $\langle \text{cdcl}_W\text{-learned-clauses-entailed-by-init } S' \rangle$ 
    using cdclW-restart 9

```

```

proof (induction rule: cdclW-restart-all-induct)
  case backtrack
  then show ?case

```

```

    using assms unfolding cdclW-learned-clause-alt-def cdclW-learned-clauses-entailed-by-init-def
    by (auto dest!: get-all-ann-decomposition-exists-prepend
      simp: clauses-def cdclW-M-level-inv-decomp dest: true-clss-clss-left-right)
qed (auto simp: cdclW-learned-clauses-entailed-by-init-def elim: true-clss-clssm-subsetE)

```


lemma *rtrancp-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*: *rtrancp-induct*)

apply (*simp*; *fail*)

using *cdcl_W-learned-clauses-entailed* *rtrancp-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 *rtrancp-cdcl_W-restart-all-inv*:

assumes

cdcl_W-restart: *rtrancp 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**

```

    cdclW-M-level-inv S' and
    no-strange-atm S' and
    distinct-cdclW-state S' and
    cdclW-conflicting S'
  using assms
proof (induct rule: rtrancpl-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 cdclW-restart-all-inv[OF H(2)]
      H by presburger
    case 2 with H(3-7)[OF this(1-6)] show ?case using cdclW-restart-all-inv[OF H(2)]
      H by presburger
    case 3 with H(3-7)[OF this(1-6)] show ?case using cdclW-restart-all-inv[OF H(2)]
      H by presburger
    case 4 with H(3-7)[OF this(1-6)] show ?case using cdclW-restart-all-inv[OF H(2)]
      H by presburger
    case 5 with H(3-7)[OF this(1-6)] show ?case using cdclW-restart-all-inv[OF H(2)]
      H by presburger
    case 6 with H(3-7)[OF this(1-6)] show ?case using cdclW-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
 $\forall T. \text{conflicting (init-state N) = Some } T \longrightarrow (\text{trail (init-state N)}) \models_{\text{as}} \text{CNot } T$ **and**
no-strange-atm (init-state N) and
consistent-interp (lits-of-l (trail (init-state N))) and
 $\forall L \text{ mark } a \text{ b. } a @ \text{Propagated } L \text{ mark } \# b = \text{trail (init-state N)} \longrightarrow$
 $(b \models_{\text{as}} \text{CNot (mark - \{\#L\# \})} \wedge L \in \# \text{ mark})$ **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: $\forall x \in \text{set } M. \neg \text{is-decided } x$ and

DN: $D \in \# \text{ clauses } S$ and

D: $M \models_{\text{as}} \text{CNot } D$ 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 $\neg \text{unsatisfiable (set-mset (N + U))}$

then obtain I where
 $I: I \models_{\text{set-mset}} N \mid I \models_{\text{set-mset}} U$ **and**
 $\text{cons: consistent-interp } I$ **and**
 $\text{tot: total-over-m } I \text{ (set-mset } N)$
unfolding *satisfiable-def* **by** *auto*
have $\text{atms-of-mm } N \cup \text{atms-of-mm } U = \text{atms-of-mm } N$
using *atm-incl state* **unfolding** *total-over-m-def no-strange-atm-def*
by (*auto simp add: clauses-def*)
then have $\text{tot-N: total-over-m } I \text{ (set-mset } N)$ **using** *tot* **unfolding** *total-over-m-def* **by** *auto*
moreover have $\text{total-over-m } I \text{ (set-mset (learned-clss } S))$
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 $I \text{ DN cons state}$ **unfolding** *true-clss-clss-def true-clss-def Ball-def*
by (*auto simp add: clauses-def*)

have $l0: \{\text{unmark } L \mid L. \text{is-decided } L \wedge L \in \text{set } M\} = \{\}$ **using** *decided* **by** *auto*
have $\text{atms-of-ms (set-mset (N+U) } \cup \text{unmark-l } M) = \text{atms-of-mm } N$
using *atm-incl state* **unfolding** *no-strange-atm-def* **by** *auto*
then have $\text{total-over-m } I \text{ (set-mset (N+U) } \cup \text{unmark-l } M)$
using *tot* **unfolding** *total-over-m-def* **by** *auto*
then have $IM: I \models_{\text{set}} \text{unmark-l } M$
using *all-decomposition-implies-propagated-lits-are-implied[OF inv]* *cons I*
unfolding *true-clss-clss-def l0* **by** *auto*
have $-K \in I$ **if** $K \in \# D$ **for** K
proof –
have $-K \in \text{lits-of-l } M$
using D **that** **unfolding** *true-annots-def* **by** *force*
then show $-K \in I$ **using** IM *true-clss-singleton-lit-of-implies-incl* **by** *fastforce*
qed
then have $\neg I \models D$ **using** *cons* **unfolding** *true-clss-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 $\forall m \in \text{set } M. \neg \text{is-decided } m$
shows $\text{set-mset } N \models_{\text{ps}} \text{unmark-l } M$
proof –
have $T: \{\text{unmark } L \mid L. \text{is-decided } L \wedge L \in \text{set } M\} = \{\}$ **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
 $\text{cdcl}_W\text{-restart: cdcl}_W\text{-restart } S \text{ } S'$ **and**
 $\text{lev: cdcl}_W\text{-M-level-inv } S$ **and**
 $[\text{simp}]: \text{conflicting } S' = \text{Some } \{\#\}$ **and**
 $\text{learned: cdcl}_W\text{-learned-clause } S$ **and**
 $\text{learned-entailed: (cdcl}_W\text{-learned-clauses-entailed-by-init } S)$

```

shows unsatisfiable (set-mset (clauses S))
using assms
proof -
  have cdclW-learned-clause S' using cdclW-restart-learned-clss cdclW-restart learned lev by auto
  then have entail-false: clauses S' ⊨pm {#} using assms(3) unfolding cdclW-learned-clause-alt-def
  by auto
  moreover have entailed: ⟨cdclW-learned-clauses-entailed-by-init S'⟩
    using cdclW-learned-clauses-entailed[OF cdclW-restart learned learned-entailed] .
  ultimately have set-mset (init-clss S') ⊨ps {{#}}
    unfolding cdclW-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: cdclW-restart-init-clss[OF assms(1)] clauses-def)
  then show ?thesis unfolding satisfiable-def true-clss-clss-def by auto
qed

```

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

```

lemma conflict-with-false-implies-terminated:
  assumes cdclW-restart S S' and conflicting S = Some {#}
  shows False
  using assms by (induct rule: cdclW-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
    cdclW-restart S S' and
    lev: cdclW-M-level-inv S and
    conflicting: cdclW-conflicting S and
    no-tauto: ∀ s ∈ # learned-clss S. ¬tautology s
  shows ∀ s ∈ # learned-clss S'. ¬tautology s
  using assms
proof (induct rule: cdclW-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: cdclW-M-level-inv-decomp)
  moreover {
    have trail S ⊨as CNot ?D
      using conflicting confl unfolding cdclW-conflicting-def by auto
    then have lits-of-l (trail S) ⊨s CNot ?D
      using true-annots-true-clss 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: cdclW-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 ($\text{trail } S \models_{asm} \text{init-clss } S$
 $\vee ((\forall L \in \text{set } (\text{trail } S). \neg \text{is-decided } L) \wedge$
 $(\exists C \in \# \text{ init-clss } S. \text{trail } S \models_{as} C \text{Not } C)))$

definition *termination-cdcl_W-restart-state* ($S :: 'st$)
 \longleftrightarrow ($\text{trail } S \models_{asm} \text{init-clss } S$
 $\vee ((\forall L \in \text{atms-of-mm } (\text{init-clss } S). L \in \text{atm-of ' lits-of-l } (\text{trail } S))$
 $\wedge (\exists C \in \# \text{ init-clss } S. \text{trail } S \models_{as} C \text{Not } C)))$

1.1.4 CDCL Strong Completeness

lemma *cdcl_W-restart-can-do-step*:

assumes

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

distinct M **and**

atm-of ' (set M) \subseteq atms-of-mm N

shows $\exists S. \text{rtrancpl } \text{cdcl}_W\text{-restart } (\text{init-state } N) S$

$\wedge \text{state-butlast } S = (\text{map } (\lambda L. \text{Decided } L) M, N, \{\#\}, \text{None})$

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** $IH = \text{this}(1)$ **and** $\text{dist} = \text{this}(2)$

have *consistent-interp* ($\text{set } M$) **and** *distinct* M **and** *atm-of ' set $M \subseteq$ atms-of-mm N*

using *Cons.prem*s(1–3) **unfolding** *consistent-interp-def* **by** *auto*

then obtain S **where**

$\text{st}: \text{cdcl}_W\text{-restart}^{**} (\text{init-state } N) S$ **and**

$S: \text{state-butlast } S = (\text{map } (\lambda L. \text{Decided } L) M, N, \{\#\}, \text{None})$

using IH **by** *blast*

let $?S_0 = \text{cons-trail } (\text{Decided } L) S$

have *undef: undefined-lit* ($\text{map } (\lambda L. \text{Decided } L) M$) L

using *Cons.prem*s(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* $L \in \text{atms-of-mm } N$ **using** *Cons.prem*s(3) **by** *auto*

moreover have *undef: undefined-lit* ($\text{trail } S$) 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*^{**} ($\text{init-state } N$) $?S_0$

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

$MN: \text{set } M \models_{sm} N$ **and**

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

dist: distinct M **and**

$atm: atm\text{-}of \text{ ` } (set\ M) \subseteq atm\text{-}of\text{-}mm\ N$
obtains S **where**
 $state\text{-}butlast\ S = (map\ (\lambda L. Decided\ L)\ M, N, \{\#\}, None)$ **and**
 $rtrancpl\ cdcl_W\text{-}restart\ (init\text{-}state\ N)\ S$ **and**
 $final\text{-}cdcl_W\text{-}restart\text{-}state\ S$
proof –
obtain S **where**
 $st: rtrancpl\ cdcl_W\text{-}restart\ (init\text{-}state\ N)\ S$ **and**
 $S: state\text{-}butlast\ S = (map\ (\lambda L. Decided\ L)\ M, N, \{\#\}, None)$
using $cdcl_W\text{-}restart\text{-}can\text{-}do\text{-}step[OF\ cons\ dist\ atm]$ **by** $auto$
have $lits\text{-}of\text{-}l\ (map\ (\lambda L. Decided\ L)\ M) = set\ M$
by $(induct\ M, auto)$
then have $map\ (\lambda L. Decided\ L)\ M \models_{asm}\ N$ **using** $MN\ true\text{-}annots\text{-}true\text{-}cls$ **by** $metis$
then have $final\text{-}cdcl_W\text{-}restart\text{-}state\ S$
using S **unfolding** $final\text{-}cdcl_W\text{-}restart\text{-}state\text{-}def$ **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 $trancpl\text{-}conflict$:
 $trancpl\ conflict\ S\ S' \implies conflict\ S\ S'$
by $(induct\ rule: trancpl.induct)\ (auto\ elim!: conflictE)$

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

lemma $trancpl\text{-}conflict\text{-}iff$:
 $full1\ conflict\ S\ S' \longleftrightarrow conflict\ S\ S'$
by $(auto\ simp: full1\text{-}def\ dest: trancpl\text{-}conflict\ no\text{-}chained\text{-}conflict)$

lemma $no\text{-}conflict\text{-}after\text{-}conflict$:
 $conflict\ S\ T \implies \neg conflict\ T\ U$
by $(auto\ elim!: conflictE\ simp: conflict.simps)$

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

inductive $cdcl_W\text{-}stgy :: 'st \Rightarrow 'st \Rightarrow bool$ **for** $S :: 'st$ **where**
 $conflict': conflict\ S\ S' \implies cdcl_W\text{-}stgy\ S\ S' \mid$

propagate': $\text{propagate } S \ S' \implies \text{cdcl}_W\text{-stgy } S \ S' \mid$
other': $\text{no-step conflict } S \implies \text{no-step propagate } S \implies \text{cdcl}_W\text{-o } S \ S' \implies \text{cdcl}_W\text{-stgy } S \ S'$

lemma $\text{cdcl}_W\text{-stgy-cdcl}_W$: $\text{cdcl}_W\text{-stgy } S \ T \implies \text{cdcl}_W \ S \ T$
by (*induction rule*: $\text{cdcl}_W\text{-stgy.induct}$) (*auto intro*: $\text{cdcl}_W\text{-intros}$)

lemma $\text{cdcl}_W\text{-stgy-induct}$ [*consumes 1, case-names conflict propagate decide skip resolve backtrack*]:
assumes
 $\langle \text{cdcl}_W\text{-stgy } S \ T \rangle$ **and**
 $\langle \bigwedge T. \text{conflict } S \ T \implies P \ T \rangle$ **and**
 $\langle \bigwedge T. \text{propagate } S \ T \implies P \ T \rangle$ **and**
 $\langle \bigwedge T. \text{no-step conflict } S \implies \text{no-step propagate } S \implies \text{decide } S \ T \implies P \ T \rangle$ **and**
 $\langle \bigwedge T. \text{no-step conflict } S \implies \text{no-step propagate } S \implies \text{skip } S \ T \implies P \ T \rangle$ **and**
 $\langle \bigwedge T. \text{no-step conflict } S \implies \text{no-step propagate } S \implies \text{resolve } S \ T \implies P \ T \rangle$ **and**
 $\langle \bigwedge T. \text{no-step conflict } S \implies \text{no-step propagate } S \implies \text{backtrack } S \ T \implies P \ T \rangle$
shows
 $\langle P \ T \rangle$
using $\text{assms}(1)$ **by** (*induction rule*: $\text{cdcl}_W\text{-stgy.induct}$)
(*auto simp*: $\text{assms}(2-)$ $\text{cdcl}_W\text{-o.simps}$ $\text{cdcl}_W\text{-bj.simps}$)

lemma $\text{cdcl}_W\text{-stgy-cases}$ [*consumes 1, case-names conflict propagate decide skip resolve backtrack*]:
assumes
 $\langle \text{cdcl}_W\text{-stgy } S \ T \rangle$ **and**
 $\langle \text{conflict } S \ T \implies P \rangle$ **and**
 $\langle \text{propagate } S \ T \implies P \rangle$ **and**
 $\langle \text{no-step conflict } S \implies \text{no-step propagate } S \implies \text{decide } S \ T \implies P \rangle$ **and**
 $\langle \text{no-step conflict } S \implies \text{no-step propagate } S \implies \text{skip } S \ T \implies P \rangle$ **and**
 $\langle \text{no-step conflict } S \implies \text{no-step propagate } S \implies \text{resolve } S \ T \implies P \rangle$ **and**
 $\langle \text{no-step conflict } S \implies \text{no-step propagate } S \implies \text{backtrack } S \ T \implies P \rangle$
shows
 $\langle P \rangle$
using $\text{assms}(1)$ **by** (*cases rule*: $\text{cdcl}_W\text{-stgy.cases}$)
(*auto simp*: $\text{assms}(2-)$ $\text{cdcl}_W\text{-o.simps}$ $\text{cdcl}_W\text{-bj.simps}$)

Invariants

lemma $\text{cdcl}_W\text{-stgy-consistent-inv}$:
assumes $\text{cdcl}_W\text{-stgy } S \ S'$ **and** $\text{cdcl}_W\text{-M-level-inv } S$
shows $\text{cdcl}_W\text{-M-level-inv } S'$
using assms **by** (*induct rule*: $\text{cdcl}_W\text{-stgy.induct}$) (*blast intro*: $\text{cdcl}_W\text{-restart-consistent-inv}$ $\text{cdcl}_W\text{-restart.intros}$)**+**

lemma $\text{rtrancpl-cdcl}_W\text{-stgy-consistent-inv}$:
assumes $\text{cdcl}_W\text{-stgy}^{**} S \ S'$ **and** $\text{cdcl}_W\text{-M-level-inv } S$
shows $\text{cdcl}_W\text{-M-level-inv } S'$
using assms **by induction** (*auto dest!*: $\text{cdcl}_W\text{-stgy-consistent-inv}$)

lemma $\text{cdcl}_W\text{-stgy-no-more-init-clss}$:
assumes $\text{cdcl}_W\text{-stgy } S \ S'$
shows $\text{init-clss } S = \text{init-clss } S'$
using assms $\text{cdcl}_W\text{-cdcl}_W\text{-restart}$ $\text{cdcl}_W\text{-restart-init-clss}$ $\text{cdcl}_W\text{-stgy-cdcl}_W$ **by blast**

lemma $\text{rtrancpl-cdcl}_W\text{-stgy-no-more-init-clss}$:
assumes $\text{cdcl}_W\text{-stgy}^{**} S \ S'$
shows $\text{init-clss } S = \text{init-clss } S'$
using assms

apply (*induct rule: rtrancpl-induct, simp*)
using *cdcl_W-stgy-no-more-init-clss* **by** (*simp add: rtrancpl-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* \Rightarrow bool' **where**
conflict-is-false-with-level $S \equiv \forall D. \text{conflicting } S = \text{Some } D \longrightarrow D \neq \{\#\}$
 $\longrightarrow (\exists L \in \# D. \text{get-level } (\text{trail } S) L = \text{backtrack-lvl } S)$

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

Literal of highest level in decided literals

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

lemma *backtrack_W-rule*:

assumes

conf: $\langle \text{conflicting } S = \text{Some } (\text{add-mset } L D) \rangle$ **and**
decomp: $\langle (\text{Decided } K \# M1, M2) \in \text{set } (\text{get-all-ann-decomposition } (\text{trail } S)) \rangle$ **and**
lev-L: $\langle \text{get-level } (\text{trail } S) L = \text{backtrack-lvl } S \rangle$ **and**
max-lev: $\langle \text{get-level } (\text{trail } S) L = \text{get-maximum-level } (\text{trail } S) (\text{add-mset } L D) \rangle$ **and**
max-D: $\langle \text{get-maximum-level } (\text{trail } S) D \equiv i \rangle$ **and**
lev-K: $\langle \text{get-level } (\text{trail } S) K = i + 1 \rangle$ **and**
T: $\langle T \sim \text{cons-trail } (\text{Propagated } L (\text{add-mset } L D))$
 $(\text{reduce-trail-to } M1$
 $(\text{add-learned-cls } (\text{add-mset } L D)$
 $(\text{update-conflicting } \text{None } S))) \rangle$ **and**
lev-inv: *cdcl_W-M-level-inv* S **and**
conf: $\langle \text{cdcl}_W\text{-conflicting } S \rangle$ **and**
learned: $\langle \text{cdcl}_W\text{-learned-clause } S \rangle$

shows $\langle \text{backtrack } S T \rangle$

using *conf decomp lev-L max-lev max-D lev-K*

proof (*rule backtrack-rule*)

let $?i = \text{get-maximum-level } (\text{trail } S) D$

let $?D = \langle \text{add-mset } L D \rangle$

show $\langle D \subseteq \# D \rangle$

by *simp*

obtain $M3$ **where**

$M3: \langle \text{trail } S = M3 @ M2 @ \text{Decided } K \# M1 \rangle$

using *decomp* **by** *auto*

have *trail-S-D*: $\langle \text{trail } S \models_{\text{as}} \text{CNot } ?D \rangle$

using *conf conf unfolding cdcl_W-conflicting-def* **by** *auto*

then have *atms-E-M1*: $\langle \text{atms-of } D \subseteq \text{atm-of } \text{'lits-of-l } M1 \rangle$

using *backtrack-atms-of-D-in-M1*[*OF - - decomp, of D ?i L ?D*]

$\langle \text{cons-trail } (\text{Propagated } L ?D) (\text{reduce-trail-to } M1 (\text{add-learned-cls } ?D (\text{update-conflicting } \text{None } S))) \rangle$
 $\langle \text{Propagated } L (\text{add-mset } L D) \rangle]$

conf lev-K decomp max-lev lev-L conf T max-D lev-inv unfolding cdcl_W-M-level-inv-def

by *auto*

have *n-d*: $\langle \text{no-dup } (M3 @ M2 @ \text{Decided } K \# M1) \rangle$


```

    using lev-inv no-dup-rev[of ⟨rev M1 @ rev M2 @ rev M3⟩, unfolded rev-append]
    by (auto simp: cdclW-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 cdclW-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: cdclW-M-level-inv S and
    conf: ⟨cdclW-conflicting S⟩ and
    learned: ⟨cdclW-learned-clause S⟩
  shows ∃ S'. cdclW-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 cdclW-M-level-inv-def by auto

```

show $\langle Ex \text{ (backtrack } S) \rangle$
using *backtrack_W-rule*[*OF S K L L-D - lev-K*] *lev-inv conf learned by auto*
then show $\langle Ex \text{ (cdcl}_W\text{-o } S) \rangle$
using *bj* **by** (*auto simp: cdcl_W-bj.simps*)
qed

lemma *no-analyse-backtrack-Ex-simple-backtrack:*

assumes
bt: $\langle \text{backtrack } S \text{ } T \rangle$ **and**
lev-inv: *cdcl_W-M-level-inv S* **and**
conf: $\langle \text{cdcl}_W\text{-conflicting } S \rangle$ **and**
learned: $\langle \text{cdcl}_W\text{-learned-clause } S \rangle$ **and**
no-dup: $\langle \text{distinct-cdcl}_W\text{-state } S \rangle$ **and**
ns-s: $\langle \text{no-step skip } S \rangle$ **and**
ns-r: $\langle \text{no-step resolve } S \rangle$
shows $\langle Ex(\text{simple-backtrack } S) \rangle$
proof –
obtain *D L K i M1 M2 D'* **where**
confl: *conflicting S = Some (add-mset L D)* **and**
decomp: $(\text{Decided } K \# M1, M2) \in \text{set } (\text{get-all-ann-decomposition } (\text{trail } S))$ **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' \equiv i* **and**
lev-K: *get-level (trail S) K = Suc i* **and**
D'-D: $\langle D' \subseteq \# D \rangle$ **and**
NU-DL: $\langle \text{clauses } S \models_{pm} \text{add-mset } L \text{ } D' \rangle$ **and**
T: $T \sim \text{cons-trail } (\text{Propagated } L \text{ (add-mset } L \text{ } D'))$
(reduce-trail-to M1
(add-learned-cls (add-mset L D')
(update-conflicting None S)))
using *bt* **by** (*elim backtrackE*) *metis*
have *n-d*: $\langle \text{no-dup } (\text{trail } S) \rangle$
using *lev-inv unfolding cdcl_W-M-level-inv-def* **by** *auto*
have *trail-S-Nil*: $\langle \text{trail } S \neq [] \rangle$
using *decomp* **by** *auto*
then have *hd-in-annot*: $\langle \text{lit-of } (\text{hd-trail } S) \in \# \text{ mark-of } (\text{hd-trail } S) \rangle$ **if** $\langle \text{is-proped } (\text{hd-trail } S) \rangle$
using *conf that unfolding cdcl_W-conflicting-def*
by (*cases (trail S); cases (hd (trail S))*) *fastforce+*
have *max-D-L-hd*: $\langle \text{get-maximum-level } (\text{trail } S) \text{ } D < \text{get-level } (\text{trail } S) \text{ } L \wedge L = \neg \text{lit-of } (\text{hd-trail } S) \rangle$
proof cases
assume *is-p*: $\langle \text{is-proped } (\text{hd } (\text{trail } S)) \rangle$
then have $\langle \neg \text{lit-of } (\text{hd } (\text{trail } S)) \in \# \text{ add-mset } L \text{ } D \rangle$
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 $\langle \text{get-maximum-level } (\text{trail } S) \text{ (remove1-mset } (\neg \text{lit-of } (\text{hd-trail } S)) \text{ (add-mset } L \text{ } D)) \neq \text{backtrack-lvl } S \rangle$
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*: $\langle \text{get-maximum-level } (\text{trail } S) \text{ (remove1-mset } (\neg \text{lit-of } (\text{hd-trail } S)) \text{ (add-mset } L \text{ } D)) < \text{backtrack-lvl } S \rangle$
using *count-decided-ge-get-maximum-level*[*of (trail S) (remove1-mset (neg lit-of (hd-trail S)) (add-mset L D))*]
by *auto*

```

then have  $\langle L = -\text{lit-of } (\text{hd-trail } S) \rangle$ 
  using get-maximum-level-ge-get-level[of  $L$   $\langle \text{remove1-mset } (- \text{ lit-of } (\text{hd-trail } S)) (\text{add-mset } L \ D) \rangle$ 
     $\langle \text{trail } S \rangle$ ] lev apply –
  by (rule ccontr) auto
then show ?thesis
  using lev-L-D lev by auto
next
assume is-p:  $\langle \neg \text{ is-proped } (\text{hd } (\text{trail } S)) \rangle$ 
obtain  $L'$  where
   $L'$ :  $\langle L' \in \# \text{ add-mset } L \ D \rangle$  and
  lev-L':  $\langle \text{get-level } (\text{trail } S) \ L' = \text{backtrack-lvl } S \rangle$ 
  using lev by auto
moreover have  $\langle \neg L' \in \text{lits-of-l } (\text{trail } S) \rangle$ 
  using conf confl L' unfolding cdclW-conflicting-def true-annots-true-cls-def-iff-negation-in-model
  by auto
moreover have  $\langle L' \notin \text{lits-of-l } (\text{trail } S) \rangle$ 
  using n-d uL'-trail by (blast dest: no-dup-consistentD)
ultimately have  $L'\text{-hd}$ :  $\langle L' = -\text{lit-of } (\text{hd-trail } S) \rangle$ 
  using is-p trail-S-Nil by (cases  $\langle \text{trail } S \rangle$ ; cases  $\langle \text{hd } (\text{trail } S) \rangle$ )
  (auto simp: get-level-cons-if atm-of-eq-atm-of split: if-splits)
have  $\langle \text{distinct-mset } (\text{add-mset } L \ D) \rangle$ 
  using no-dup confl unfolding distinct-cdclW-state-def by auto
then have  $\langle L' \notin \# \text{ remove1-mset } L' (\text{add-mset } L \ D) \rangle$ 
  using  $L'$  by (meson distinct-mem-diff-mset multi-member-last)
moreover have  $\langle \neg L' \notin \# \text{ add-mset } L \ D \rangle$ 
proof (rule ccontr)
  assume  $\langle \neg ?thesis \rangle$ 
  then have  $\langle L' \in \text{lits-of-l } (\text{trail } S) \rangle$ 
  using conf confl trail-S-Nil unfolding cdclW-conflicting-def true-annots-true-cls-def-iff-negation-in-model
  by auto
  then show False
    using n-d L'-hd by (cases  $\langle \text{trail } S \rangle$ ; cases  $\langle \text{hd } (\text{trail } S) \rangle$ )
    (auto simp: Decided-Propagated-in-iff-in-lits-of-l)
qed
ultimately have  $\langle \text{atm-of } (\text{lit-of } (\text{Decided } (- L'))) \notin \text{atms-of } (\text{remove1-mset } L' (\text{add-mset } L \ D)) \rangle$ 
  using  $\langle \neg L' \notin \# \text{ add-mset } L \ D \rangle$  by (auto simp: atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set
    atms-of-def dest: in-diffD)
then have  $\langle \text{get-maximum-level } (\text{Decided } (-L')) \ \# \ \text{tl } (\text{trail } S) \rangle (\text{remove1-mset } L' (\text{add-mset } L \ D)) =$ 
   $\text{get-maximum-level } (\text{tl } (\text{trail } S)) (\text{remove1-mset } L' (\text{add-mset } L \ D)) \rangle$ 
  by (rule get-maximum-level-skip-first)
also have  $\langle \text{get-maximum-level } (\text{tl } (\text{trail } S)) (\text{remove1-mset } L' (\text{add-mset } L \ D)) < \text{backtrack-lvl } S \rangle$ 
  using count-decided-ge-get-maximum-level[of  $\langle \text{tl } (\text{trail } S) \rangle \langle \text{remove1-mset } L' (\text{add-mset } L \ D) \rangle$ ]
  trail-S-Nil is-p by (cases  $\langle \text{trail } S \rangle$ ; cases  $\langle \text{hd } (\text{trail } S) \rangle$ ) auto
finally have lev-L'-L:  $\langle \text{get-maximum-level } (\text{trail } S) (\text{remove1-mset } L' (\text{add-mset } L \ D)) < \text{backtrack-lvl}$ 
 $S \rangle$ 
  using trail-S-Nil is-p L'-hd by (cases  $\langle \text{trail } S \rangle$ ; cases  $\langle \text{hd } (\text{trail } S) \rangle$ ) auto
then have  $\langle L = L' \rangle$ 
  using get-maximum-level-ge-get-level[of  $L$   $\langle \text{remove1-mset } L' (\text{add-mset } L \ D) \rangle$ 
     $\langle \text{trail } S \rangle$ ]  $L'$  lev-L' lev by auto
then show ?thesis
  using lev-L'-L lev L'-hd by auto
qed
let  $?i = \langle \text{get-maximum-level } (\text{trail } S) \ D \rangle$ 
obtain  $K' \ M1' \ M2'$  where
  decomp':  $\langle (\text{Decided } K' \ \# \ M1', M2') \in \text{set } (\text{get-all-ann-decomposition } (\text{trail } S)) \rangle$  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 cdclW-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
  confl: ⟨cdclW-conflicting S⟩ and
  level-inv: ⟨cdclW-M-level-inv S⟩ and
  no-dup: ⟨distinct-cdclW-state S⟩ and
  learned: ⟨cdclW-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 confl unfolding cdclW-conflicting-def by auto
then have ?M ≠ [] unfolding true-annots-def Ball-def true-annot-def true-clss-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 cdclW-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 $g\text{-}k$: $\text{get-maximum-level } (\text{trail } S) \ D \leq ?k$
using $\text{count-decided-ge-get-maximum-level}[of \ ?M] \ \text{level-inv} \ \text{unfolding} \ \text{cdcl}_W\text{-}M\text{-level-inv-def}$
by auto
have $L'\text{-}L$: $L' = -L$
proof (rule ccontr)
assume $\neg \ ?thesis$
moreover {
have $-L \in \text{lits-of-l } ?M$
using $\text{confl conf} \ \text{unfolding} \ \text{cdcl}_W\text{-conflicting-def} \ \text{by} \ \text{auto}$
then have $\langle \text{atm-of } L \neq \text{atm-of } L' \rangle$
using $\text{cdcl}_W\text{-}M\text{-level-inv-decomp}(2)[OF \ \text{level-inv}] \ M \ \text{calculation } L'$
by ($\text{auto simp: atm-of-eq-atm-of all-conj-distrib uminus-lit-swap lits-of-def no-dup-def}$) }
ultimately have $\text{get-level } (\text{hd } (\text{trail } S) \ \# \ M') \ L = \text{get-level } (\text{tl } ?M) \ L$
using $\text{cdcl}_W\text{-}M\text{-level-inv-decomp}(1)[OF \ \text{level-inv}] \ M \ \text{unfolding} \ \text{consistent-interp-def}$
by ($\text{simp add: atm-of-eq-atm-of } L' \ M'[\text{symmetric}]$)
moreover {
have $\text{count-decided } (\text{trail } S) = ?k$
using $\text{level-inv} \ \text{unfolding} \ \text{cdcl}_W\text{-}M\text{-level-inv-def} \ \text{by} \ \text{auto}$
then have $\text{count: count-decided } M' = ?k - 1$
using $\text{level-inv } M \ \text{by} \ (\text{auto simp add: } L' \ M'[\text{symmetric}])$
then have $\text{get-level } (\text{tl } ?M) \ L < ?k$
using $\text{count-decided-ge-get-level}[of \ M' \ L] \ \text{unfolding} \ k'[\text{symmetric}] \ M' \ \text{by} \ \text{auto}$ }
finally show $\text{False} \ \text{using} \ \text{lev-}L \ M \ \text{unfolding} \ M' \ \text{by} \ \text{auto}$
qed
then have L : $\text{hd } ?M = \text{Decided } (-L) \ \text{using} \ L' \ \text{by} \ \text{auto}$
have H : $\text{get-maximum-level } (\text{trail } S) \ D < ?k$
proof (rule ccontr)
assume $\neg \ ?thesis$
then have $\text{get-maximum-level } (\text{trail } S) \ D = ?k \ \text{using} \ M \ g\text{-}k \ \text{unfolding} \ L \ \text{by} \ \text{auto}$
then obtain L'' **where** $L'' \in \# \ D$ **and** $L\text{-}k$: $\text{get-level } ?M \ L'' = ?k$
using $\text{get-maximum-level-exists-lit}[of \ ?k \ ?M \ D] \ \text{unfolding} \ k'[\text{symmetric}] \ \text{by} \ \text{auto}$
have $L \neq L'' \ \text{using} \ \text{no-dup } \langle L'' \in \# \ D \rangle$
unfolding $\text{distinct-cdcl}_W\text{-state-def confl}$
by ($\text{metis } E' \ \text{add-diff-cancel-right'} \ \text{distinct-mem-diff-mset union-commute union-single-eq-member}$)
have $L'' = -L$
proof (rule ccontr)
assume $\neg \ ?thesis$
then have $\text{get-level } ?M \ L'' = \text{get-level } (\text{tl } ?M) \ L''$
using $M \ \langle L \neq L'' \rangle \ \text{get-level-skip-beginning}[of \ L'' \ \text{hd } ?M \ \text{tl } ?M] \ \text{unfolding} \ L$
by ($\text{auto simp: atm-of-eq-atm-of}$)
moreover have $\text{get-level } (\text{tl } (\text{trail } S)) \ L = 0$
using $\text{level-inv } L' \ M \ \text{unfolding} \ \text{cdcl}_W\text{-}M\text{-level-inv-def}$
by ($\text{auto simp: image-iff } L' \ L'\text{-}L$)
moreover {
have $\langle \text{backtrack-lvl } S = \text{count-decided } (\text{hd } ?M \ \# \ \text{tl } ?M) \rangle$
unfolding $M[\text{symmetric}] \ M'[\text{symmetric}] \ \dots$
then have $\text{get-level } (\text{tl } (\text{trail } S)) \ L'' < \text{backtrack-lvl } S$
using $\text{count-decided-ge-get-level}[of \ \langle \text{tl } (\text{trail } S) \rangle \ L'']$
by ($\text{auto simp: image-iff } L' \ L'\text{-}L$) }
ultimately show False
using $M[\text{unfolded } L' \ M'[\text{symmetric}]] \ L\text{-}k \ \text{by} \ (\text{auto simp: } L' \ L'\text{-}L)$
qed
then have $\text{taut: tautology } (D + \{\#L\# \})$
using $\langle L'' \in \# \ D \rangle \ \text{by} \ (\text{metis } \text{add.commute mset-subset-eqD mset-subset-eq-add-left multi-member-this tautology-minus})$
moreover have $\text{consistent-interp } (\text{lits-of-l } ?M)$

```

    using level-inv unfolding cdclW-M-level-inv-def by auto
  ultimately have  $\neg ?M \models_{as} CNot ?D$ 
    by (metis  $\langle L'' = - L \rangle \langle L'' \in \# D \rangle$  add commute consistent-interp-def
        diff-union-cancelR in-CNot-implies-uminus(2) in-diffD multi-member-this)
  moreover have  $?M \models_{as} CNot ?D$ 
    using confl no-dup conf unfolding cdclW-conflicting-def by auto
  ultimately show False by blast
qed
have confl-D:  $\langle conflicting S = Some (add-mset L D) \rangle$ 
  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  $\langle Ex (backtrack S) \rangle$ 
  using backtrack-no-decomp[OF confl-D - ] level-inv alien conf learned
  by (auto simp add: lev-L max-def n-s)

show  $\langle no-step resolve S \rangle$ 
  using L by (auto elim!: resolveE)
show  $\langle no-step skip S \rangle$ 
  using L by (auto elim!: skipE)
qed

lemma conflicting-no-false-can-do-step:
  assumes
    confl:  $\langle conflicting S = Some C \rangle$  and
    nempty:  $\langle C \neq \{\#\} \rangle$  and
    confl-k:  $\langle conflict-is-false-with-level S \rangle$  and
    conf:  $\langle cdcl_W-conflicting S \rangle$  and
    level-inv:  $\langle cdcl_W-M-level-inv S \rangle$  and
    no-dup:  $\langle distinct-cdcl_W-state S \rangle$  and
    learned:  $\langle cdcl_W-learned-clause S \rangle$  and
    alien:  $\langle no-strange-atm S \rangle$  and
    termi:  $\langle no-step cdcl_W-stgy S \rangle$ 
  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  $\langle M' = tl ?M \rangle$ 
  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  $?D \neq \{\#\}$  by auto
  have  $?M \models_{as} CNot ?D$  using confl conf unfolding cdclW-conflicting-def by auto
  then have  $?M \neq []$  unfolding true-annots-def Ball-def true-annot-def true-clss-def by force
  have M':  $?M = hd ?M \# tl ?M$  using  $\langle ?M \neq [] \rangle$  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: cdclW-stgy.intros)+
  have  $\langle no-step backtrack S \rangle$ 

```

```

    using termi by (blast intro: cdclW-stgy.intros cdclW-o.intros cdclW-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  $\langle ?M \neq [] \rangle$  - nempty confl] by (cases  $\langle$ hd-trail S $\rangle$ ) (auto)
  have g-k: get-maximum-level (trail S)  $D \leq ?k$ 
    using count-decided-ge-get-maximum-level[of ?M] level-inv unfolding cdclW-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 cdclW-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  $\langle ?M \neq [] \rangle$  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 cdclW-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' cdclW-o.intros cdclW-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 cdclW-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 (cdclW-o S)
      using resolve-rule[of S L' C , OF  $\langle$ trail S  $\neq [] \rangle$  - - confl] confl
      L'C L'D D' C' by (auto dest: cdclW-o.intros cdclW-bj.intros)
    then show False
      using n-s termi by (auto dest: other' cdclW-o.intros cdclW-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 confl
      by (auto simp: D' dest: other' cdclW-o.intros cdclW-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 {#} \wedge unsatisfiable (set-mset (clauses S))*)
 \vee (*conflicting S = None \wedge trail S \models 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-clss-empty)

then show *?thesis* **using** *Some-Empty* **by** (*auto simp: clauses-def*)

next

case *None*

have *?M \models asm ?N*

proof (*rule ccontr*)

assume *MN: \neg ?thesis*

have *all-defined: atm-of ' (lits-of-l ?M) = atms-of-mm ?N (is ?A = ?B)*

proof

show *?A \subseteq ?B* **using** *alien unfolding no-strange-atm-def clauses-def* **by** *auto*

show *?B \subseteq ?A*

proof (*rule ccontr*)

assume *$\neg ?B \subseteq ?A$*

then obtain *l* **where** *l \in ?B* **and** *l \notin ?A* **by** *auto*

then have *undefined-lit ?M (Pos l)*

using *$\langle l \notin ?A \rangle$ unfolding lits-of-def* **by** (*auto simp add: defined-lit-map*)

then have *$\exists S'. \text{cdcl}_W\text{-o } S S'$*

using *cdcl_W-o.decide[of S] decide-rule[of S (Pos l) $\langle \text{cons-trail (Decided (Pos l)) S} \rangle$*

$\langle l \in ?B \rangle$ 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** *$\neg ?M \models_a D$* **and** *$D \in \# ?N$*

using *MN unfolding lits-of-def true-annots-def Ball-def* **by** *auto*

have *atms-of D \subseteq atm-of ' (lits-of-l ?M)*

using *$\langle D \in \# ?N \rangle$ 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  $\neg$  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  $\exists S'. \text{conflict } S S'$ 
  using  $\langle \text{trail } S \models_{as} \text{CNot } D \rangle \langle D \in \# \text{ clauses } S \rangle$ 
  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-final-state-conclusive:
assumes
  termi: no-step cdclW-stgy S and
  decomp: all-decomposition-implies-m (clauses S) (get-all-ann-decomposition (trail S)) and
  learned: cdclW-learned-clause S and
  level-inv: cdclW-M-level-inv S and
  alien: no-strange-atm S and
  no-dup: distinct-cdclW-state S and
  confl: cdclW-conflicting S and
  confl-k: conflict-is-false-with-level S and
  learned-entailed:  $\langle \text{cdcl}_W\text{-learned-clauses-entailed-by-init } S \rangle$ 
shows (conflicting S = Some {#}  $\wedge$  unsatisfiable (set-mset (init-clss S)))
   $\vee$  (conflicting S = None  $\wedge$  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 cdclW-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)  $\subseteq$  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)  $\cup$  set-mset (learned-clss S)) =
    atms-of-mm (init-clss S)
    by auto

```

```

have init-clss  $S \models_{psm} \text{learned-clss } S$ 
  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  $\models_{asm} ?N$ 
proof (rule ccontr)
  assume MN:  $\neg ?thesis$ 
  have all-defined: atm-of ' (lits-of-l ?M) = atms-of-mm ?N (is ?A = ?B)
  proof
    show ?A  $\subseteq$  ?B using alien unfolding no-strange-atm-def by auto
    show ?B  $\subseteq$  ?A
    proof (rule ccontr)
      assume  $\neg ?B \subseteq ?A$ 
      then obtain l where  $l \in ?B$  and  $l \notin ?A$  by auto
      then have undefined-lit ?M (Pos l)
        using  $\langle l \notin ?A \rangle$  unfolding lits-of-def by (auto simp add: defined-lit-map)
      then have  $\exists S'. \text{cdcl}_W\text{-o } S S'$ 
        using cdclW-o.decide decide-rule  $\langle l \in ?B \rangle$  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  $\neg ?M \models_a D$  and  $D \in \# ?N$ 
    using MN unfolding lits-of-def true-annots-def Ball-def by auto
  have atms-of D  $\subseteq$  atm-of ' (lits-of-l ?M)
    using  $\langle D \in \# ?N \rangle$  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  $\models_{as} \text{CNot } D$ 
    using  $\langle \neg \text{trail } S \models_a D \rangle$  unfolding true-annot-def true-annots-true-cls
    by (fastforce simp: total-not-true-cls-true-clss-CNot)
  then have  $\exists S'. \text{conflict } S S'$ 
    using M-CNot-D  $\langle D \in \# \text{init-clss } S \rangle$ 
    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  $S S' \implies \text{cdcl}_W\text{-restart}^{++} S S'$ 
  by (simp add: cdclW-cdclW-restart cdclW-stgy-cdclW tranclp.r-into-trancl)

```

lemma *trancpl-cdcl_W-stgy-trancpl-cdcl_W-restart*:
 $cdcl_W\text{-stgy}^{++} S S' \implies cdcl_W\text{-restart}^{++} S S'$
apply (*induct rule: trancpl.induct*)
using *cdcl_W-stgy-trancpl-cdcl_W-restart* **apply** *blast*
by (*meson cdcl_W-stgy-trancpl-cdcl_W-restart trancpl-trans*)

lemma *rtrancpl-cdcl_W-stgy-rtrancpl-cdcl_W-restart*:
 $cdcl_W\text{-stgy}^{**} S S' \implies cdcl_W\text{-restart}^{**} S S'$
using *rtrancpl-unfold*[*of cdcl_W-stgy S S'*] *trancpl-cdcl_W-stgy-trancpl-cdcl_W-restart*[*of S S'*] **by** *auto*

lemma *cdcl_W-o-conflict-is-false-with-level-inv*:
assumes
 $cdcl_W\text{-o } S S'$ **and**
 $lev: cdcl_W\text{-M-level-inv } S$ **and**
 $confl\text{-inv: conflict-is-false-with-level } S$ **and**
 $n\text{-d: distinct-cdcl}_W\text{-state } S$ **and**
 $conflicting: cdcl_W\text{-conflicting } S$
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** $tr\text{-}S = this(1)$ **and** $confl = this(4)$ **and** $LD = this(5)$ **and** $T = this(7)$
have $uL\text{-not-}D: -L \notin \# \text{ remove1-mset } (-L) D$
using $n\text{-d}$ *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 $L\text{-not-}D: L \notin \# \text{ remove1-mset } (-L) D$
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 $-L \in \text{lits-of-l } (Propagated L C \# M)$
using *in-CNot-implies-uminus(2)* **by** *blast*
moreover have $no\text{-dup } (Propagated L C \# M)$
using $lev\ tr\text{-}S$ **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 $g\text{-}D: \text{get-maximum-level } (Propagated L C \# M) (\text{remove1-mset } (-L) D)$
 $= \text{get-maximum-level } M (\text{remove1-mset } (-L) D)$
by (*simp add: atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set atms-of-def*)
have $lev\text{-}L[simp]: \text{get-level } M L = 0$
using lev **unfolding** *cdcl_W-M-level-inv-def tr-S* **by** (*auto simp: lits-of-def*)

have $D: \text{get-maximum-level } M (\text{remove1-mset } (-L) D) = \text{backtrack-lvl } S$
using *resolve.hyps(6) LD* **unfolding** *tr-S* **by** (*auto simp: get-maximum-level-plus max-def g-D*)
have $\text{get-maximum-level } M (\text{remove1-mset } L C) \leq \text{backtrack-lvl } S$
using *count-decided-ge-get-maximum-level*[*of M*] lev **unfolding** *tr-S cdcl_W-M-level-inv-def* **by** *auto*
then have
 $\text{get-maximum-level } M (\text{remove1-mset } (-L) D \cup \# \text{ remove1-mset } L C) = \text{backtrack-lvl } S$
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 ∪# 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 ∈# D and
  get-level (Propagated L C' # M) La = backtrack-lvl S
  using skip confl-inv by auto
moreover {
  have atm-of La ≠ atm-of L
  proof (rule ccontr)
    assume ¬ ?thesis
    then have La: La = L using ⟨La ∈# D⟩ ⟨← L ∉# D⟩
      by (auto simp add: atm-of-eq-atm-of)
    have Propagated L C' # M ⊨as CNot D
      using conflicting tr-S D unfolding cdclW-conflicting-def by auto
    then have ←L ∈ lits-of-l M
      using ⟨La ∈# D⟩ in-CNot-implies-uminus(2)[of L D Propagated L C' # M] unfolding La
      by auto
    then show False using lev tr-S unfolding cdclW-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: cdclW-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#} ∈# local.clauses S and
    M' ⊨as CNot C and
    undefined-lit (trail S) L
proof -
  obtain E L where
    conf: conflicting S = None and
    E: E ∈# clauses S and
    LE: L ∈# E and
    tr: trail S ⊨as CNot (E - {#L#}) and
    undef: undefined-lit (trail S) L and
    T: T ∼ cons-trail (Propagated L E) S
    using assms by (elim propagateE) simp
  obtain M N U 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-clss-def*
by *force*

moreover

have *set M* \models *C* + {#*L*#}

using *MN C learned Y NS* \langle *init-clss S* = *init-clss Y* \rangle \langle *learned-clss Y* = {#} \rangle

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)  $\subseteq$  atms-of-mm N and
distM: distinct M and
length:  $n \leq \text{length } M$ 
shows
 $\exists M' S. \text{length } M' \geq n \wedge$ 
  lits-of-l  $M' \subseteq \text{set } M \wedge$ 
  no-dup  $M' \wedge$ 
  state-butlast  $S = (M', N, \{\#\}, \text{None}) \wedge$ 
  cdclW-stgy** (init-state N) S
using length
proof (induction n)
case 0
have state-butlast (init-state N) = ( $\square$ , N,  $\{\#\}$ , None)
  by auto
moreover have
   $0 \leq \text{length } \square$  and
  lits-of-l  $\square \subseteq \text{set } M$  and
  cdclW-stgy** (init-state N) (init-state N)
  and no-dup  $\square$ 
  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':  $\text{length } M' \geq n$  and
  M': lits-of-l  $M' \subseteq \text{set } M$  and
  n-d[simp]: no-dup  $M' \wedge$ 
  S: state-butlast  $S = (M', N, \{\#\}, \text{None})$  and
  st: cdclW-stgy** (init-state N) S
  by auto
have
  M: cdclW-M-level-inv S and
  alien: no-strange-atm S
  using cdclW-M-level-inv-S0-cdclW-restart rtrancpl-cdclW-stgy-consistent-inv st apply blast
using cdclW-M-level-inv-S0-cdclW-restart no-strange-atm-S0 rtrancpl-cdclW-restart-no-strange-atm-inv
  rtrancpl-cdclW-stgy-rtrancpl-cdclW-restart st by blast

{ assume no-step:  $\neg \text{no-step propagate } S$ 
  then obtain S' where S': propagate S S'
    by auto
  have lev: cdclW-M-level-inv S'
    using M S' rtrancpl-cdclW-restart-consistent-inv rtrancpl-propagate-is-rtrancpl-cdclW-restart by
blast
  then have n-d'[simp]: no-dup (trail S')
    unfolding cdclW-M-level-inv-def by auto
  have length (trail S)  $\leq \text{length } (\text{trail } S') \wedge \text{lits-of-l } (\text{trail } S') \subseteq \text{set } M$ 
    using S' cdclW-propagate-conflict-completeness[OF assms(1-3), of S] M' S
    by (auto simp: comp-def)
  moreover have cdclW-stgy S S' using S' by (simp add: cdclW-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  $stS'$ :  $cdcl_W\text{-stgy}^{**} (init\text{-state } N) S'$ 
    using  $st\ cdcl_W\text{-stgy.propagate}[OF\ S']$  by ( $auto\ simp: r\text{-into-rtranclp}$ )
  then have  $init\text{-clss } S' = N$ 
    using  $rtranclp\text{-}cdcl_W\text{-stgy-no-more-init-clss}$  by  $fastforce$ }
moreover {
  have
     $[simp]: learned\text{-clss } S' = \{\#\}$  and
     $[simp]: init\text{-clss } S' = init\text{-clss } S$  and
     $[simp]: conflicting\ S' = None$ 
    using  $S\ S'$  by ( $auto\ elim: propagateE$ )
  have  $state\text{-butlast } S' = (trail\ S', N, \{\#\}, None)$ 
    using  $S$  by  $auto$  }
moreover
have  $cdcl_W\text{-stgy}^{**} (init\text{-state } N) S'$ 
  apply ( $rule\ rtranclp.rtrancl\text{-into-rtrancl}$ )
  using  $st$  apply  $simp$ 
  using  $\langle cdcl_W\text{-stgy } S\ S' \rangle$  by  $simp$ 
ultimately have  $?case$ 
  apply –
  apply ( $rule\ exI[of\ \text{trail } S'], rule\ exI[of\ S']$ )
  by  $auto$ 
}
moreover {
  assume  $no\text{-step: no-step propagate } S$ 
  have  $?case$ 
  proof ( $cases\ length\ M' \geq Suc\ n$ )
    case  $True$ 
    then show  $?thesis$  using  $l\text{-}M'\ M'\ st\ M\ alien\ S\ n\text{-}d$  by  $blast$ 
  next
  case  $False$ 
  then have  $n'$ :  $length\ M' = n$  using  $l\text{-}M'$  by  $auto$ 
  have  $no\text{-confl: no-step conflict } S$ 
  proof –
  { fix  $D$ 
    assume  $D \in \# N$  and  $M' \models_{as} CNot\ D$ 
    then have  $set\ M \models D$  using  $MN$  unfolding  $true\text{-clss-def}$  by  $auto$ 
    moreover have  $set\ M \models_s CNot\ D$ 
      using  $\langle M' \models_{as} CNot\ D \rangle\ M'$ 
      by ( $metis\ le\text{-iff-sup}\ true\text{-annots-true-clss}\ true\text{-clss-union-increase}$ )
    ultimately have  $False$  using  $cons\ consistent\text{-}CNot\text{-not}$  by  $blast$ 
  }
  then show  $?thesis$ 
    using  $S$  by ( $auto\ simp: true\text{-clss-def comp-def rev-map}$ 
       $clauses\text{-def elim!:: conflictE}$ )
  }
qed
have  $lenM$ :  $length\ M = card\ (set\ M)$  using  $distM$  by ( $induction\ M$ )  $auto$ 
have  $no\text{-dup } M'$  using  $S\ M$  unfolding  $cdcl_W\text{-}M\text{-level-inv-def}$  by  $auto$ 
then have  $card\ (lits\text{-of-l } M') = length\ M'$ 
  by ( $induction\ M'$ ) ( $auto\ simp\ add: lits\text{-of-def card-insert-if defined-lit-map}$ )
then have  $lits\text{-of-l } M' \subset set\ M$ 
  using  $n\ M'\ n'\ lenM$  by  $auto$ 
then obtain  $L$  where  $L: L \in set\ M$  and  $undef\text{-}m: L \notin lits\text{-of-l } M'$  by  $auto$ 
moreover have  $undef$ :  $undefined\text{-lit } M'\ L$ 
  using  $M'\ Decided\text{-Propagated-in-iff-in-lits-of-l calculation}(1,2)\ cons$ 
   $consistent\text{-interp-def}$  by ( $metis\ (no\text{-types, lifting})\ subset\text{-eq}$ )

```

```

moreover have atm-of  $L \in \text{atms-of-mm } (\text{init-clss } S)$ 
  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  $?S' = \text{cons-trail } (\text{Decided } L) S$ 
have lits-of-l (trail  $?S'$ )  $\subseteq \text{set } M$  using  $L M' S$  undef by auto
moreover have no-strange-atm  $?S'$ 
  using alien dec  $M$  by (meson cdclW-restart-no-strange-atm-inv decide other)
have cdclW-M-level-inv  $?S'$ 
  using  $M$  dec rtrancpl-mono[of decide cdclW-restart] by (meson cdclW-restart-consistent-inv
    decide other)
then have lev'': cdclW-M-level-inv  $?S'$ 
  using  $S$  rtrancpl-cdclW-restart-consistent-inv rtrancpl-propagate-is-rtrancpl-cdclW-restart
  by blast
then have n-d'': no-dup (trail  $?S'$ )
  unfolding cdclW-M-level-inv-def by auto
have length (trail  $S$ )  $\leq \text{length } (\text{trail } ?S') \wedge \text{lits-of-l } (\text{trail } ?S') \subseteq \text{set } M$ 
  using  $S L M' S$  undef by simp
then have Suc  $n \leq \text{length } (\text{trail } ?S') \wedge \text{lits-of-l } (\text{trail } ?S') \subseteq \text{set } M$ 
  using l-M'  $S$  undef by auto
moreover have S'': state-butlast  $?S' = (\text{trail } ?S', N, \{\#\}, \text{None})$ 
  using  $S$  undef n-d'' lev'' by auto
moreover have cdclW-stgy** (init-state  $N$ )  $?S'$ 
  using  $S''$  no-step no-confl st dec by (auto dest: decide cdclW-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 : $\text{set } M \models_s \text{set-mset } N$ **and**

cons: consistent-interp (set M) **and**

tot: total-over-m (set M) (set-mset N) **and**

atm-incl: atm-of ' (set M) $\subseteq \text{atms-of-mm } N$ **and**

distM: distinct M

shows

$\exists M' S. \text{lits-of-l } M' = \text{set } M \wedge$

state-butlast $S = (M', N, \{\#\}, \text{None}) \wedge$

cdcl_W-stgy** (init-state N) S

proof –

have $\langle \exists M' S. \text{lits-of-l } M' \subseteq \text{set } M \wedge$

no-dup $M' \wedge \text{length } M' = n \wedge$

state-butlast $S = (M', N, \{\#\}, \text{None}) \wedge$

cdcl_W-stgy** (init-state N) $S \rangle$

if $\langle n \leq \text{length } M \rangle$ **for** $n :: \text{nat}$

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 $M' \subseteq \text{set } M$ **and**

n-d[simp]: no-dup M' **and**


```

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

{ assume no-step:  $\neg$ no-step propagate S
  then obtain S' where S': propagate S S'
    by auto
  have lev: cdclW-M-level-inv S'
    using M S' rtranclp-cdclW-restart-consistent-inv rtranclp-propagate-is-rtranclp-cdclW-restart by
blast
  then have n-d'[simp]: no-dup (trail S')
    unfolding cdclW-M-level-inv-def by auto
  have length (trail S) ≤ length (trail S') ∧ lits-of-l (trail S') ⊆ set M
    using S' cdclW-propagate-conflict-completeness[OF assms(1-3), of S] M' S
    by (auto simp: comp-def)
  moreover have cdclW-stgy S S' using S' by (simp add: cdclW-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': cdclW-stgy** (init-state N) S'
      using st cdclW-stgy.propagate'[OF S'] by (auto simp: r-into-rtranclp)
    then have init-clss S' = N
      using rtranclp-cdclW-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 cdclW-stgy** (init-state N) S'
    apply (rule rtranclp.rtrancl-into-rtrancl)
    using st apply simp
    using cdclW-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-conflict: 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  $\langle M' \models_{as} CNot D \rangle M'$ 
  by (metis le-iff-sup true-annots-true-clss 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 cdclW-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'  $\subseteq$  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 cdclW-restart-no-strange-atm-inv decide other)
have cdclW-M-level-inv ?S'
  using M dec rtranclp-mono[of decide cdclW-restart] by (meson cdclW-restart-consistent-inv
    decide other)
then have lev'': cdclW-M-level-inv ?S'
  using S rtranclp-cdclW-restart-consistent-inv rtranclp-propagate-is-rtranclp-cdclW-restart
    by blast
then have n-d'': no-dup (trail ?S')
  unfolding cdclW-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 cdclW-stgy** (init-state N) ?S'
  using S'' no-step no-confl st dec by (auto dest: decide cdclW-stgy.intros)
ultimately have ?case using n-d'' L M' by (auto intro!: exI[of -  $\langle Decided L \# trail S \rangle$ ] exI[of -
 $\langle ?S' \rangle$ ])
}
ultimately show ?case by blast
qed
from this[of  $\langle length M \rangle$ ] obtain M' S where
  M'-M:  $\langle lits-of-l M' \subseteq set M \rangle$  and
  n-d:  $\langle no-dup M' \rangle$  and
   $\langle length M' = length M \rangle$  and
   $\langle state-butlast S = (M', N, \{ \# \}, None) \wedge cdcl_W-stgy^{**} (init-state N) S \rangle$ 
  by auto
moreover have  $\langle lits-of-l M' = set M \rangle$ 

```

```

apply (rule card-subset-eq)
subgoal by auto
subgoal using  $M'-M$  .
  subgoal using  $M'-M$   $n-d$   $no-dup-length-eq-card-atm-of-lits-of-l$   $[OF\ n-d]$   $M'-M$   $\langle finite\ (set\ M) \rangle$ 
   $distinct-card[OF\ distM]$   $calculation(3)$ 
     $card-image-le[of\ \langle lits-of-l\ M' \rangle\ atm-of]$   $card-seteq[OF\ \langle finite\ (set\ M) \rangle, of\ \langle lits-of-l\ M' \rangle]$ 
  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 : $set\ M \models_s set-mset\ N$ **and**
 $cons$: $consistent-interp\ (set\ M)$ **and**
 tot : $total-over-m\ (set\ M)\ (set-mset\ N)$ **and**
 $atm-incl$: $atm-of\ \cdot\ (set\ M) \subseteq atms-of-mm\ N$ **and**
 $distM$: $distinct\ M$

shows

$\exists M' k S.$
 $lits-of-l\ M' = set\ M \wedge$
 $state-butlast\ S = (M', N, \{\#\}, None) \wedge$
 $cdcl_W-stgy^{**}\ (init-state\ N)\ S \wedge$
 $final-cdcl_W-restart-state\ S$

proof –

from $cdcl_W-stgy-strong-completeness-n[OF\ assms, of\ length\ M]$

obtain $M' T$ **where**

l : $length\ M \leq length\ M'$ **and**
 $M'-M$: $lits-of-l\ M' \subseteq set\ M$ **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 $card\ (set\ ((map\ (\lambda l. atm-of\ (lit-of\ l))\ M')) = length\ M'$
using $distinct-card\ no-dup$ **by** ($fastforce\ simp$: $lits-of-def\ image-image\ no-dup-def$) }

moreover have $card\ (lits-of-l\ M') = card\ (set\ ((map\ (\lambda l. atm-of\ (lit-of\ l))\ M'))$

using $no-dup$ **by** ($induction\ M'$) ($auto\ simp\ add$: $defined-lit-map\ card-insert-if\ lits-of-def$)

ultimately have $card\ (set\ M) \leq card\ (lits-of-l\ M')$ **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 $M' \models_{asm}\ N$
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-conf* ($S :: 'st$) \equiv
 $(\forall M K M' D. \text{trail } S = M' @ \text{Decided } K \# M \longrightarrow D \in \# \text{ clauses } S \longrightarrow \neg M \models_{as} CNot D)$

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

lemma *cdcl_W-o-no-smaller-conf-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-conf* S
shows *no-smaller-conf* S'
using *assms*(1,2) **unfolding** *no-smaller-conf-def*
proof (*induct rule*: *cdcl_W-o-induct*)
case (*decide* $L T$) **note** *conf* = *this*(1) **and** *undef* = *this*(2) **and** $T = \text{this}(4)$
have [simp]: *clauses* $T = \text{clauses } S$
 using $T \text{ undef}$ **by** *auto*
show ?*case*
proof (*intro allI impI*)
fix $M'' K M' Da$
assume *trail* $T = M'' @ \text{Decided } K \# M'$ **and** $Da \in \# \text{ local.clauses } T$
then have *trail* $S = \text{tl } M'' @ \text{Decided } K \# M'$
 $\vee (M'' = [] \wedge \text{Decided } K \# M' = \text{Decided } L \# \text{trail } S)$
 using $T \text{ undef}$ **by** (*cases* M'') *auto*
moreover {
 assume *trail* $S = \text{tl } M'' @ \text{Decided } K \# M'$
 then have $\neg M' \models_{as} CNot Da$
 using $D T \text{ undef conf}$ *smaller* **unfolding** *no-smaller-conf-def* *smaller* **by** *fastforce*
 }
moreover {
 assume $\text{Decided } K \# M' = \text{Decided } L \# \text{trail } S$
 then have $\neg M' \models_{as} CNot Da$ **using** *smaller* $D \text{ conf}$ $T \text{ n-s}$ **by** (*auto simp: conflict.simps*)
 }
ultimately show $\neg M' \models_{as} CNot Da$ **by** *fast*
qed
next
case *resolve*
then show ?*case* **using** *smaller max-lev* **unfolding** *no-smaller-conf-def* **by** *auto*
next
case *skip*
then show ?*case* **using** *smaller max-lev* **unfolding** *no-smaller-conf-def* **by** *auto*
next
case (*backtrack* $L D K i M1 M2 T D'$) **note** *conf* = *this*(1) **and** *decomp* = *this*(2) **and**
 $T = \text{this}(9)$
obtain c **where** $M: \text{trail } S = c @ M2 @ \text{Decided } K \# M1$
 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: cdclW-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 cdclW-M-level-inv ?S'
        using cdclW-restart-consistent-inv[OF - lev] other[OF bj]
        by (auto intro: cdclW-bj.intros)
      then have no-dup (Propagated L D # M1)
        using decomp lev unfolding cdclW-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 cdclW-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  $M'$ :  $\text{trail } S' = M'' @ \text{Decided } K \# M'$ 
and  $D \in \# \text{ clauses } S'$ 
obtain  $M N U C L$  where
   $S$ :  $\text{state-butlast } S = (M, N, U, \text{None})$  and
   $S'$ :  $\text{state-butlast } S' = (\text{Propagated } L (C + \{\#L\}) \# M, N, U, \text{None})$  and
   $C + \{\#L\} \in \# \text{ clauses } S$  and
   $M \models_{as} C \text{Not } C$  and
   $\text{undefined-lit } M L$ 
  using  $\text{propagate}$  by ( $\text{auto elim: propagate-high-levelE}$ )
have  $\text{tl } M'' @ \text{Decided } K \# M' = \text{trail } S$  using  $M' S S'$ 
  by ( $\text{metis Pair-inject list.inject list.sel(3) annotated-lit.distinct(1) self-append-conv2}$ 
     $\text{tl-append2}$ )
then have  $\neg M' \models_{as} C \text{Not } D$ 
  using  $\langle D \in \# \text{ clauses } S' \rangle$   $n\text{-l } S S'$   $\text{clauses-def}$  unfolding  $\text{no-smaller-conflict-def}$  by  $\text{auto}$ 
then show  $\neg M' \models_{as} C \text{Not } D$  by  $\text{auto}$ 
qed

```

```

lemma  $\text{cdcl}_W\text{-stgy-no-smaller-conflict}$ :
  assumes  $\text{cdcl}_W\text{-stgy } S S'$ 
  and  $n\text{-l: no-smaller-conflict } S$ 
  and  $\text{conflict-is-false-with-level } S$ 
  and  $\text{cdcl}_W\text{-M-level-inv } S$ 
  shows  $\text{no-smaller-conflict } S'$ 
  using  $\text{assms}$ 
proof ( $\text{induct rule: cdcl}_W\text{-stgy.induct}$ )
  case ( $\text{conflict}' S'$ )
    then show  $?case$  using  $\text{conflict-no-smaller-conflict-inv[of } S S']$  by  $\text{blast}$ 
  next
    case ( $\text{propagate}' S'$ )
      then show  $?case$  using  $\text{propagate-no-smaller-conflict-inv[of } S S']$  by  $\text{blast}$ 
  next
    case ( $\text{other}' S'$ )
      then show  $?case$ 
        using  $\text{cdcl}_W\text{-o-no-smaller-conflict-inv[of } S]$  by  $\text{auto}$ 
qed

```

```

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

```

assume
empty: $D \neq \{\#\}$ **and**
lev: $\forall L \in \#D. \text{get-level}(\text{trail } T) L \neq \text{backtrack-lvl } T$
moreover {
 have $\text{get-level}(\text{trail } T) L \leq \text{backtrack-lvl } T$ **if** $L \in \#D$ **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 $\text{get-level}(\text{trail } T) L < \text{backtrack-lvl } T$ **if** $L \in \#D$ **for** L
 using *lev that by fastforce* } **note** $\text{lev}' = \text{this}$
ultimately have $\text{count-decided}(\text{trail } T) > 0$
 using *M-lev-T unfolding cdcl_W-M-level-inv-def by (cases D) fastforce+*
then have $\exists x \in \text{set}(\text{trail } T). \text{is-decided } x$
 unfolding *no-dup-def count-decided-def* **by** *cases auto*
have $\exists M2 L M1. \text{trail } T = M2 @ \text{Decided } L \# M1 \wedge (\forall m \in \text{set } M2. \neg \text{is-decided } m)$
 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**
 $\text{tr-}T$: $\text{trail } T = M2 @ \text{Decided } L \# M1$ **and** $\text{nm}: \forall m \in \text{set } M2. \neg \text{is-decided } m$
 by *blast*
moreover {
 have $\text{get-level}(\text{trail } T) L_a = \text{backtrack-lvl } T$ **if** $L_a \in \text{lits-of-l } M2$ **for** L_a
 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 $\text{tr-}T: M2 @ \text{Decided } L \# M1 = (M2 @ [\text{Decided } L]) @ M1$
 by *auto*
 have $\text{get-level}(\text{trail } T) L = \text{backtrack-lvl } T$
 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 $\text{trail } S = \text{trail } T$
 using T **by** *auto*
ultimately have $M1 \models_{as} \text{CNot } D$
 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-conflict-def)*
qed
qed

lemma *cdcl_W-stgy-ex-lit-of-max-level:*

assumes
 cdcl_W-stgy $S S'$ **and**
 n-l: no-smaller-conflict 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-conflict* S'

```

    using conflict'.hyps conflict-no-smaller-conflict-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 cdclW-o-conflict-is-false-with-level-inv[OF o] other'.prems by blast
qed

lemma rtranclp-cdclW-stgy-no-smaller-conflict-inv:
  assumes
    cdclW-stgy** S S' and
    n-l: no-smaller-conflict S and
    cls-false: conflict-is-false-with-level S and
    lev: cdclW-M-level-inv S and
    dist: distinct-cdclW-state S and
    conflicting: cdclW-conflicting S and
    decomp: all-decomposition-implies-m (clauses S) (get-all-ann-decomposition (trail S)) and
    learned: cdclW-learned-clause S and
    alien: no-strange-atm S
  shows no-smaller-conflict 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-conflict S' and conflict-is-false-with-level S'
  using IH by blast+
moreover have cdclW-M-level-inv S'
  using st lev rtranclp-cdclW-stgy-rtranclp-cdclW-restart
  by (blast intro: rtranclp-cdclW-restart-consistent-inv)+
moreover have distinct-cdclW-state S'
  using rtranclp-distinct-cdclW-state-inv[of S S'] lev rtranclp-cdclW-stgy-rtranclp-cdclW-restart[OF st]
  dist by auto
moreover have cdclW-conflicting S'
  using rtranclp-cdclW-restart-all-inv(6)[of S S'] st alien conflicting decomp dist learned lev
  rtranclp-cdclW-stgy-rtranclp-cdclW-restart by blast
ultimately show ?case
  using cdclW-stgy-no-smaller-conflict[OF cdcl] cdclW-stgy-ex-lit-of-max-level[OF cdcl] cdcl
  by (auto simp del: simp add: cdclW-stgy.simps elim!: propagateE)
qed

```

Final States are Conclusive

theorem 2.9.9 page 97 of Weidenbach's book

```

lemma full-cdclW-stgy-final-state-conclusive:
  fixes S' :: 'st
  assumes full: full cdclW-stgy (init-state N) S'
  and no-d: distinct-mset-mset N
  shows (conflicting S' = Some {#} ∧ unsatisfiable (set-mset (init-cls S')))

```


$\vee (\text{conflicting } S' = \text{None} \wedge \text{trail } S' \models_{\text{asm}} \text{init-clss } S')$
proof –
 let $?S = \text{init-state } N$
have
 termi: $\forall S''. \neg \text{cdcl}_W\text{-stgy } S' S''$ **and**
 step: $\text{cdcl}_W\text{-stgy}^{**} ?S S'$ **using** *full unfolding full-def* **by** *auto*
have
 learned: $\text{cdcl}_W\text{-learned-clause } S'$ **and**
 level-inv: $\text{cdcl}_W\text{-M-level-inv } S'$ **and**
 alien: $\text{no-strange-atm } S'$ **and**
 no-dup: $\text{distinct-cdcl}_W\text{-state } S'$ **and**
 confl: $\text{cdcl}_W\text{-conflicting } S'$ **and**
 decomp: $\text{all-decomposition-implies-m (clauses } S') (\text{get-all-ann-decomposition (trail } S'))$
 using *no-d* $\text{trancpl-cdcl}_W\text{-stgy-trancpl-cdcl}_W\text{-restart[of } ?S S']$ *step*
 *rtrancpl-cdcl}_W\text{-restart-all-inv}(1-6)[\text{of } ?S S']
 unfolding *rtrancpl-unfold* **by** *auto*
have *confl-k*: $\text{conflict-is-false-with-level } S'$
 using *rtrancpl-cdcl}_W\text{-stgy-no-smaller-confl-inv[OF step]* *no-d* **by** *auto*
have *learned-entailed*: $\langle \text{cdcl}_W\text{-learned-clauses-entailed-by-init } S' \rangle$
 using *rtrancpl-cdcl}_W\text{-learned-clauses-entailed[of } \langle ?S \rangle \langle S' \rangle] *step*
 by (*simp add: rtrancpl-cdcl}_W\text{-stgy-rtrancpl-cdcl}_W\text{-restart}*)

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

lemma *cdcl}_W\text{-o-fst-empty-conflicting-false*:
assumes
 $\text{cdcl}_W\text{-o } S S'$ **and**
 $\text{trail } S = []$ **and**
 $\text{conflicting } S \neq \text{None}$
shows *False*
using *assms* **by** (*induct rule: cdcl}_W\text{-o-induct}*) *auto*

lemma *cdcl}_W\text{-stgy-fst-empty-conflicting-false*:
assumes
 $\text{cdcl}_W\text{-stgy } S S'$ **and**
 $\text{trail } S = []$ **and**
 $\text{conflicting } S \neq \text{None}$
shows *False*
using *assms* **apply** (*induct rule: cdcl}_W\text{-stgy.induct}*)
 apply (*auto elim: conflictE; fail*)[]
 apply (*auto elim: propagateE; fail*)[]
using *cdcl}_W\text{-o-fst-empty-conflicting-false* **by** *blast*

lemma *cdcl}_W\text{-o-conflicting-is-false*:
 $\text{cdcl}_W\text{-o } S S' \implies \text{conflicting } S = \text{Some } \{\#\} \implies \text{False}$
by (*induction rule: cdcl}_W\text{-o-induct}*) *auto*

lemma *cdcl}_W\text{-stgy-conflicting-is-false*:
 $\text{cdcl}_W\text{-stgy } S S' \implies \text{conflicting } S = \text{Some } \{\#\} \implies \text{False}$
apply (*induction rule: cdcl}_W\text{-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' \implies conflicting S = Some {#} \implies S' = S*
apply (*induction rule*: *rtranclp-induct*)
apply *simp*
using *cdcl_W-stgy-conflicting-is-false* **by** *blast*

definition *conflict-or-propagate* :: '*st* \Rightarrow '*st* \Rightarrow *bool* **where**
conflict-or-propagate S T \longleftrightarrow conflict S T \vee propagate S T

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

lemma *conflict-or-propagate-intros*:
conflict S T \implies conflict-or-propagate S T
propagate S T \implies 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 {#} \wedge unsatisfiable (set-mset N)*)
 \vee (*conflicting S' = None \wedge trail S' \models_{asm} N \wedge satisfiable (set-mset N)*)

proof –

have *N*: *init-clss S' = N*
using *full* **unfolding** *full-def* **by** (*auto dest*: *rtranclp-cdcl_W-stgy-no-more-init-clss*)

consider

(*confl*) *conflicting S' = Some {#} and unsatisfiable (set-mset (init-clss S'))*
| (*sat*) *conflicting S' = None and trail S' \models_{asm} init-clss S'*
using *full-cdcl_W-stgy-final-state-conclusive*[*OF assms*] **by** *auto*

then show *?thesis*

proof *cases*

case *confl*

then show *?thesis* **by** (*auto simp*: *N*)

next

case *sat*

have *cdcl_W-M-level-inv (init-state N)* **by** *auto*

then have *cdcl_W-M-level-inv S'*

using *full rtranclp-cdcl_W-stgy-consistent-inv* **unfolding** *full-def* **by** *blast*

then have *consistent-interp (lits-of-l (trail S'))*

unfolding *cdcl_W-M-level-inv-def* **by** *blast*

moreover have *lits-of-l (trail S') \models_s set-mset (init-clss S')*

using *sat(2)* **by** (*auto simp add*: *true-annots-def true-annot-def true-clss-def*)

ultimately have *satisfiable (set-mset (init-clss S'))* **by** *simp*

then show *?thesis* **using** *sat* **unfolding** *N* **by** *blast*

qed

qed

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**

$cdcl_W\text{-all-struct-inv } S \longleftrightarrow$
 $no\text{-strange-atm } S \wedge$
 $cdcl_W\text{-M-level-inv } S \wedge$
 $(\forall s \in \# \text{ learned-clss } S. \neg \text{tautology } s) \wedge$
 $distinct\text{-}cdcl_W\text{-state } S \wedge$
 $cdcl_W\text{-conflicting } S \wedge$
 $all\text{-decomposition-implies-m } (clauses\ S) (get\text{-all-ann-decomposition } (trail\ S)) \wedge$
 $cdcl_W\text{-learned-clause } S$

lemma $cdcl_W\text{-all-struct-inv-inv}$:

assumes $cdcl_W\text{-restart } S\ S'$ **and** $cdcl_W\text{-all-struct-inv } S$

shows $cdcl_W\text{-all-struct-inv } S'$

unfolding $cdcl_W\text{-all-struct-inv-def}$

proof (intro $HOL.conjI$)

show $no\text{-strange-atm } S'$

using $cdcl_W\text{-restart-all-inv}[OF\ assms(1)]\ assms(2)$ **unfolding** $cdcl_W\text{-all-struct-inv-def}$ **by** *auto*

show $cdcl_W\text{-M-level-inv } S'$

using $cdcl_W\text{-restart-all-inv}[OF\ assms(1)]\ assms(2)$ **unfolding** $cdcl_W\text{-all-struct-inv-def}$ **by** *fast*

show $distinct\text{-}cdcl_W\text{-state } S'$

using $cdcl_W\text{-restart-all-inv}[OF\ assms(1)]\ assms(2)$ **unfolding** $cdcl_W\text{-all-struct-inv-def}$ **by** *fast*

show $cdcl_W\text{-conflicting } S'$

using $cdcl_W\text{-restart-all-inv}[OF\ assms(1)]\ assms(2)$ **unfolding** $cdcl_W\text{-all-struct-inv-def}$ **by** *fast*

show $all\text{-decomposition-implies-m } (clauses\ S') (get\text{-all-ann-decomposition } (trail\ S'))$

using $cdcl_W\text{-restart-all-inv}[OF\ assms(1)]\ assms(2)$ **unfolding** $cdcl_W\text{-all-struct-inv-def}$ **by** *fast*

show $cdcl_W\text{-learned-clause } S'$

using $cdcl_W\text{-restart-all-inv}[OF\ assms(1)]\ assms(2)$ **unfolding** $cdcl_W\text{-all-struct-inv-def}$ **by** *fast*

show $\forall s \in \# \text{ learned-clss } S'. \neg \text{tautology } s$

using $assms(1)[THEN\ learned-clss-are-not-tautologies]\ assms(2)$

unfolding $cdcl_W\text{-all-struct-inv-def}$ **by** *fast*

qed

lemma $rtrancp\text{-}cdcl_W\text{-all-struct-inv-inv}$:

assumes $cdcl_W\text{-restart}^{**} S\ S'$ **and** $cdcl_W\text{-all-struct-inv } S$

shows $cdcl_W\text{-all-struct-inv } S'$

using $assms$ **by** *induction* (auto intro: $cdcl_W\text{-all-struct-inv-inv}$)

lemma $cdcl_W\text{-stgy-}cdcl_W\text{-all-struct-inv}$:

$cdcl_W\text{-stgy } S\ T \implies cdcl_W\text{-all-struct-inv } S \implies cdcl_W\text{-all-struct-inv } T$

by (*meson* $cdcl_W\text{-stgy-trancp-}cdcl_W\text{-restart } rtrancp\text{-}cdcl_W\text{-all-struct-inv-inv } rtrancp\text{-unfold}$)

lemma $rtrancp\text{-}cdcl_W\text{-stgy-}cdcl_W\text{-all-struct-inv}$:

$cdcl_W\text{-stgy}^{**} S\ T \implies cdcl_W\text{-all-struct-inv } S \implies cdcl_W\text{-all-struct-inv } T$

by (*induction rule*: $rtrancp\text{-induct}$) (auto intro: $cdcl_W\text{-stgy-}cdcl_W\text{-all-struct-inv}$)

lemma $beginning\text{-not-decided-invert}$:

assumes $A: M @ A = M' @ Decided\ K \# H$ **and**

$nm: \forall m \in \text{set } M. \neg \text{is-decided } m$

shows $\exists M. A = M @ Decided\ K \# H$

proof –

have $A = \text{drop } (\text{length } M) (M' @ Decided\ K \# H)$

using $arg\text{-cong}[OF\ A, \text{of } \text{drop } (\text{length } M)]$ **by** *auto*

moreover have $\text{drop } (\text{length } M) (M' @ Decided\ K \# H) = \text{drop } (\text{length } M) M' @ Decided\ K \# H$

using nm **by** (*metis* (*no-types*, *lifting*) $A\ \text{drop-Cons}'\ \text{drop-append annotated-lit.disc}(1)\ \text{not-gr0}$

$\text{nth-append nth-append-length nth-mem zero-less-diff}$)

finally show $?thesis$ **by** *fast*

qed

1.1.7 Strategy-Specific Invariant

definition *cdcl_W-stgy-invariant* where

cdcl_W-stgy-invariant $S \longleftrightarrow$
 $\text{conflict-is-false-with-level } S$
 $\wedge \text{no-smaller-confl } S$

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 *rtrancp-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* *rtrancp-cdcl_W-all-struct-inv-inv*

rtrancp-cdcl_W-stgy-rtrancp-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: $\langle \text{cdcl}_W\text{-learned-clauses-entailed-by-init } S \rangle$

shows *conflicting* $T = \text{Some } \{\#\} \wedge \text{unsatisfiable } (\text{set-mset } (\text{init-clss } S))$

$\vee \text{conflicting } T = \text{None} \wedge \text{trail } T \models_{\text{asm}} \text{init-clss } S \wedge \text{satisfiable } (\text{set-mset } (\text{init-clss } S))$

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* *rtrancp-cdcl_W-stgy-rtrancp-cdcl_W-restart* *full full-def inv*

rtrancp-cdcl_W-all-struct-inv-inv)

by (*metis* *full full-def inv inv-s* *rtrancp-cdcl_W-stgy-cdcl_W-stgy-invariant*)

moreover have $\langle \text{cdcl}_W\text{-learned-clauses-entailed-by-init } T \rangle$

using *inv* *learned-entailed* **unfolding** *cdcl_W-all-struct-inv-def*

using *rtrancp-cdcl_W-learned-clauses-entailed* *rtrancp-cdcl_W-stgy-rtrancp-cdcl_W-restart*[*OF st*]

by *blast*

ultimately have *conflicting* $T = \text{Some } \{\#\} \wedge \text{unsatisfiable } (\text{set-mset } (\text{init-clss } T))$

$\vee \text{conflicting } T = \text{None} \wedge \text{trail } T \models_{\text{asm}} \text{init-clss } T$

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 $\langle \text{cdcl}_W\text{-all-struct-inv } T \rangle$ **unfolding** *cdcl_W-all-struct-inv-def* *cdcl_W-M-level-inv-def*
by *auto*
moreover have *init-clss* $S = \text{init-clss } T$
using *inv* **unfolding** *cdcl_W-all-struct-inv-def*
by (*metis* *rtrancp-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 {#} \wedge unsatisfiable (set-mset (clauses T))*
 \vee *conflicting T = None \wedge 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* *rtrancp-cdcl_W-stgy-rtrancp-cdcl_W-restart* *full* *full-def* *inv*
rtrancp-cdcl_W-all-struct-inv-inv)
by (*metis* *full* *full-def* *inv* *inv-s* *rtrancp-cdcl_W-stgy-cdcl_W-stgy-invariant*)
ultimately have *conflicting T = Some {#} \wedge unsatisfiable (set-mset (clauses T))*
 \vee *conflicting T = None \wedge trail T \models_{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 $\langle \text{cdcl}_W\text{-all-struct-inv } T \rangle$ **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* :: $\langle 'st \Rightarrow bool \rangle$ **where**

no-smaller-propa ($S :: 'st$) \equiv

$(\forall M K M' D L. \text{trail } S = M' @ \text{Decided } K \# M \longrightarrow D + \{\#L\} \in \# \text{ clauses } S \longrightarrow \text{undefined-lit } M$
 $L \longrightarrow \neg M \models_{as} CNot D)$

lemma *propagated-cons-eq-append-decide-cons*:

Propagated L E # Ms = M' @ Decided K # M \longleftrightarrow

M' $\neq []$ \wedge Ms = tl M' @ Decided K # M \wedge 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*:

$\langle C \in \text{set } (\text{get-all-mark-of-propagated } M) \longleftrightarrow (\exists L. \text{Propagated } L \ C \in \text{set } M) \rangle$

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  $- L \notin \text{ lits-of-l } M$ 
  proof (rule ccontr)
    define MM where  $\langle MM = \text{tl } M' \rangle$ 
    assume  $\langle \neg ?thesis \rangle$ 
    then have  $\langle - L \notin \text{ lits-of-l } (M' @ [\text{Decided } K']) \rangle$ 
      using n-d-T unfolding tr by (auto simp: lits-of-def no-dup-def)
    have  $\langle \text{undefined-lit } (M' @ [\text{Decided } K']) L \rangle$ 
      apply (rule no-dup-uminus-append-in-atm-notin)
      using n-d-T  $\langle \neg - L \notin \text{ lits-of-l } M \rangle$  unfolding tr by auto
    moreover have  $M' = \text{Propagated } L ?D' \# MM$ 
      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  $L\text{-max} \in \# D' \vee L \in \# D'$ 
    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-cls-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 cdclW-stgy-no-smaller-propa:
  assumes
    cdcl:  $\langle \text{cdcl}_W\text{-stgy } S T \rangle$  and
    smaller-propa:  $\langle \text{no-smaller-propa } S \rangle$  and
    inv:  $\langle \text{cdcl}_W\text{-all-struct-inv } S \rangle$ 
  shows  $\langle \text{no-smaller-propa } T \rangle$ 
  using cdcl
proof (cases rule: cdclW-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: cdclW-all-struct-inv T
  using cdcl cdclW-stgy-cdclW-all-struct-inv inv by blast
have ⟨trail S ⊨as CNot (the (conflicting S))⟩ and ⟨no-dup (trail S)⟩
  using inv o unfolding cdclW-all-struct-inv-def
  by (auto simp: cdclW-M-level-inv-def cdclW-conflicting-def
      elim: rulesE)
then show ?thesis
  using backtrack-no-smaller-propa[of S T] n-s o smaller-propa
  by auto
qed

lemma rtrancpl-cdclW-stgy-no-smaller-propa:
  assumes
    cdcl: ⟨cdclW-stgy** S T⟩ and
    smaller-propa: ⟨no-smaller-propa S⟩ and
    inv: ⟨cdclW-all-struct-inv S⟩
  shows ⟨no-smaller-propa T⟩
  using cdcl apply (induction rule: rtrancpl-induct)
  subgoal using smaller-propa by simp
  subgoal using inv by (auto intro: rtrancpl-cdclW-stgy-cdclW-all-struct-inv
      cdclW-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: ⟨cdclW-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 cdclW-all-struct-inv-def cdclW-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 cdclW-all-struct-inv-def cdclW-conflicting-def

```

```

  M M' K by blast
then have C:  $\langle C = \{\#\} + \{\#K\# \} \rangle$ 
  using size-C K by (cases C) auto
have  $\langle \text{undefined-lit } M1 K \rangle$ 
  using nd unfolding M' K decomp' by simp
moreover have  $\langle \{\#\} + \{\#K\# \} \in \# \text{ clauses } S \rangle$ 
  using struct unfolding cdclW-all-struct-inv-def cdclW-learned-clause-alt-def M M' K C
  reasons-in-clauses-def
  by auto
moreover have  $\langle M1 \models_{as} CNot \{\#\} \rangle$ 
  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* :: $\langle 'st \Rightarrow bool \rangle$ **where**
 $\langle \text{conflict-non-zero-unless-level-0 } S \longleftrightarrow$
 $(\text{conflicting } S = \text{Some } \{\#\} \longrightarrow \text{count-decided } (\text{trail } S) = 0) \rangle$

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

lemma *cdcl_W-restart-no-false-clause*:
assumes
 $\langle \text{cdcl}_W\text{-restart } S T \rangle$
 $\langle \text{no-false-clause } S \rangle$
shows $\langle \text{no-false-clause } T \rangle$
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
 $\langle \text{cdcl}_W\text{-restart } S T \rangle$
 $\langle \text{no-false-clause } S \rangle$ **and**
 $\langle \text{conflict-non-zero-unless-level-0 } S \rangle$
shows $\langle \text{conflict-non-zero-unless-level-0 } T \rangle$
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 *rtrancp-cdcl_W-restart-no-false-clause*:
assumes
 $\langle \text{cdcl}_W\text{-restart}^{**} S T \rangle$
 $\langle \text{no-false-clause } S \rangle$
shows $\langle \text{no-false-clause } T \rangle$
using *assms* **by** (*induction rule: rtrancp-induct*) (*auto intro: cdcl_W-restart-no-false-clause*)

lemma *rtrancp-cdcl_W-restart-conflict-non-zero-unless-level-0*:
assumes
 $\langle \text{cdcl}_W\text{-restart}^{**} S T \rangle$

$\langle \text{no-false-clause } S \rangle$ and
 $\langle \text{conflict-non-zero-unless-level-0 } S \rangle$
shows $\langle \text{conflict-non-zero-unless-level-0 } T \rangle$
using *assms* **by** (*induction rule*: *rtrancpl-induct*)
(auto intro: rtrancpl-cdcl_W-restart-no-false-clause cdcl_W-restart-conflict-non-zero-unless-level-0)

definition *propagated-clauses-clauses* :: '*st* \Rightarrow bool' **where**
 $\langle \text{propagated-clauses-clauses } S \equiv \forall L K. \text{Propagated } L K \in \text{set } (\text{trail } S) \longrightarrow K \in \# \text{ clauses } S \rangle$

lemma *propagate-single-literal-clause-get-level-is-0*:
assumes
smaller: $\langle \text{no-smaller-propa } S \rangle$ and
propa-tr: $\langle \text{Propagated } L \{ \#L\# \} \in \text{set } (\text{trail } S) \rangle$ and
n-d: $\langle \text{no-dup } (\text{trail } S) \rangle$ and
propa: $\langle \text{propagated-clauses-clauses } S \rangle$
shows $\langle \text{get-level } (\text{trail } S) L = 0 \rangle$

proof (*rule ccontr*)
assume *H*: $\langle \neg ?thesis \rangle$
then obtain *M M' K* **where**
tr: $\langle \text{trail } S = M' @ \text{Decided } K \# M \rangle$ and
nm: $\langle \forall m \in \text{set } M. \neg \text{is-decided } m \rangle$
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*: $\langle \neg L \notin \text{lits-of-l } (\text{trail } S) \rangle$
using *n-d propa-tr unfolding lits-of-def* **by** (*fastforce simp: no-dup-cannot-not-lit-and-uminus*)
then have [*iff*]: $\langle \text{defined-lit } M' L \longleftrightarrow L \in \text{lits-of-l } M' \rangle$
by (*auto simp add: tr Decided-Propagated-in-iff-in-lits-of-l*)
have $\langle \text{get-level } M L = 0 \rangle$ **for** *L*
using *nm* **by** *auto*
have [*simp*]: $\langle L \neq -K \rangle$
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 $\langle L \in \text{lits-of-l } (M' @ [\text{Decided } K]) \rangle$
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 $\langle \text{get-level } M L = 0 \rangle$ **by** (*auto simp: atm-of-eq-atm-of uminus-lit-swap lits-of-def*)
then have $\langle \text{undefined-lit } M L \rangle$
using *n-d unfolding tr* **by** (*auto simp: defined-lit-map lits-of-def image-Un no-dup-def*)
moreover have $\{ \# \} + \{ \#L\# \} \in \# \text{ clauses } S$
using *propa propa-tr unfolding propagated-clauses-clauses-def* **by** *auto*
moreover have $M \models_{\text{as}} \text{CNot } \{ \# \}$
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*]: $\langle D \equiv \text{the } (\text{conflicting } S) \rangle$
defines [*simp*]: $\langle M \equiv \text{trail } S \rangle$
defines $\langle D' \equiv \text{filter-mset } (\lambda L. \text{get-level } M L > 0) D \rangle$

```

assumes
  ns-s: ⟨no-step skip S⟩ and
  ns-r: ⟨no-step resolve S⟩ and
  inv-s: cdclW-stgy-invariant S and
  inv: cdclW-all-struct-inv S and
  conf: ⟨conflicting S ≠ None⟩ ⟨conflicting S ≠ Some {#}⟩ and
  M-empty: ⟨M ≈ []⟩
shows
  clauses S ⊨pm D' and
  ⟨← lit-of (hd M) ∈# D'⟩
proof −
  define D0 where D0: ⟨D0 = filter-mset (λL. get-level M L = 0) D⟩
  have D-D0-D': ⟨D = D0 + D'⟩
    using multiset-partition[of D (λL. get-level M L = 0)]
    unfolding D0 D'-def by auto
  have
    confl: ⟨cdclW-conflicting S⟩ and
    decomp: ⟨all-decomposition-implies-m (clauses S) (get-all-ann-decomposition (trail S))⟩ and
    learned: ⟨cdclW-learned-clause S⟩ and
    M-lev: ⟨cdclW-M-level-inv S⟩ and
    alien: ⟨no-strange-atm S⟩
    using inv unfolding cdclW-all-struct-inv-def by fast+
  have cls-D: ⟨clauses S ⊨pm D⟩
    using learned conf unfolding cdclW-learned-clause-alt-def by auto
  have M-CNot-D: ⟨trail S ⊨as CNot D⟩ and m-conf: ⟨every-mark-is-a-conflict S⟩
    using conf confl unfolding cdclW-conflicting-def by auto
  have n-d: ⟨no-dup M⟩
    using M-lev unfolding cdclW-M-level-inv-def by auto
  have uhd-D: ⟨← lit-of (hd M) ∈# D⟩
    using ns-s ns-r conf M-empty inv-s M-CNot-D n-d
    unfolding cdclW-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 grOI le-0-eq)
    then show False
      using ns-s ns-r conf M-empty m-conf 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: cdclW-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-clb-def true-clb-mset-def true-lit-def uminus-Pos)

have

H1: (total-over-m I (set-mset (clauses S) \cup {D0 + D'}) = total-over-m I (set-mset (clauses S)))

and

H2: (total-over-m I (set-mset (clauses S) \cup {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 \models_{pm} D')*

using *clss-D clss-D0 I-D0 unfolding D-D0-D' true-clss-clb-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 *(\neg lit-of (hd M) \in # 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 \in lits-of-l (trail S)) and

lev: (get-level (trail S) L = 0)

shows

(clauses S \models_{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#} \in 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-clb)*

next

case *False*

then obtain *K M1 M2 M3 where*

decomp': ((Decided K # M1, M2) \in 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 ⟨ $([], M1) \in \text{set } (\text{get-all-ann-decomposition } (\text{trail } S))$ ⟩
  using hd-get-all-ann-decomposition-skip-some[of Nil M1 M1 ⟨- @ -⟩] decomp'
  by auto
then have ⟨set-mset (clauses S)  $\models_{ps}$  unmark-l M1⟩
  using decomp
  unfolding all-decomposition-implies-def by auto
moreover {
  have ⟨ $L \in \text{lits-of-l } M1$ ⟩
    using n-d lev M3 in-trail
    by (cases ⟨undefined-lit ( $M2' @ \text{Decided } K \# []$ ) L⟩) (auto dest: in-lits-of-l-defined-litD)
  then have ⟨ $\{\#L\# \} \in \text{unmark-l } M1$ ⟩
    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-clss)
qed
qed

```

1.1.10 Some higher level use on the invariants

In later refinement we mostly use 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-conf S ⟹
cdclW-all-struct-inv S ⟹
conflict-is-false-with-level T⟩
by (rule conflict-conflict-is-false-with-level) (auto simp: cdclW-all-struct-inv-def)

```

lemma *cdcl_W-stgy-ex-lit-of-max-level-all-inv:*

```

assumes
  cdclW-stgy S S' and
  n-l: no-smaller-conf S and
  conflict-is-false-with-level S and
  cdclW-all-struct-inv S
shows conflict-is-false-with-level S'
by (rule cdclW-stgy-ex-lit-of-max-level) (use assms in ⟨auto simp: cdclW-all-struct-inv-def⟩)

```

lemma *cdcl_W-o-conflict-is-false-with-level-inv-all-inv:*

```

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

```

lemma *no-step-cdcl_W-total:*

```

assumes

```

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

lemma $\text{cdcl}_W\text{-Ex-cdcl}_W\text{-stgy}$:
assumes
 $\langle \text{cdcl}_W \ S \ T \rangle$
shows $\langle \text{Ex}(\text{cdcl}_W\text{-stgy } S) \rangle$
using *assms* **by** (meson *assms* $\text{cdcl}_W.\text{simps}$ $\text{cdcl}_W\text{-stgy.simps}$)

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


```

    then have  $\langle \neg \text{lit-of } L \notin \text{lits-of-l } M \rangle$ 
      unfolding  $M$  by (auto simp: defined-lit-map lits-of-def uminus-lit-swap)
    }
  ultimately have  $L\text{-}D$ :  $\langle \text{lit-of } L \notin \# D \rangle$ 
    using  $M\text{-}D$  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  $L' = \text{this}(1)$ 
    moreover have  $\langle \text{atm-of } L' = \text{atm-of } L\text{-}k \rangle$ 
      using  $\text{lev-}L\text{-}k$  count-decided-ge-get-level[of  $M$   $L\text{-}k$ ] unfolding  $M$  dec  $L'$ 
      by (auto simp: get-level-cons-if split: if-splits)
    then have  $\langle L' = -L\text{-}k \rangle$ 
      using  $L\text{-}k$   $L\text{-}D$   $L'$  by (auto simp: atm-of-eq-atm-of)
    then show ?thesis using  $L\text{-}k$  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
     $nss$ :  $\langle \text{no-step skip } S \rangle$  and
     $nsr$ :  $\langle \text{no-step resolve } S \rangle$  and
     $invs$ :  $\langle \text{cdcl}_W\text{-all-struct-inv } S \rangle$  and
     $stgy$ :  $\langle \text{cdcl}_W\text{-stgy-invariant } S \rangle$  and
     $confl$ :  $\langle \text{conflicting } S \neq \text{None} \rangle$  and
     $confl'$ :  $\langle \text{conflicting } S \neq \text{Some } \{\# \} \rangle$ 
  shows no-skip-no-resolve-single-highest-level:
     $\langle \text{the } (\text{conflicting } S) =$ 
       $\text{add-mset } (\neg(\text{lit-of } (\text{hd } (\text{trail } S)))) \{\#L \in \# \text{the } (\text{conflicting } S)\}.$ 
       $\text{get-level } (\text{trail } S) L < \text{local.backtrack-lvl } S\# \rangle$  (is ?A) and
     $\text{no-skip-no-resolve-level-lvl-nonzero}$ :
     $\langle 0 < \text{backtrack-lvl } S \rangle$  (is ?B) and
     $\text{no-skip-no-resolve-level-get-maximum-lvl-le}$ :
     $\langle \text{get-maximum-level } (\text{trail } S) (\text{remove1-mset } (\neg(\text{lit-of } (\text{hd } (\text{trail } S)))) (\text{the } (\text{conflicting } S)))$ 
       $< \text{backtrack-lvl } S \rangle$  (is ?C)
  proof -
    define  $K$  where  $\langle K \equiv \text{lit-of } (\text{hd } (\text{trail } S)) \rangle$ 
    have  $K$ :  $\langle \neg K \in \# \text{the } (\text{conflicting } S) \rangle$ 
      using no-step-skip-hd-in-conflicting[OF  $stgy$   $invs$   $nss$   $confl$   $confl'$ ]
      unfolding  $K\text{-def}$  .
    have
       $\langle \text{no-strange-atm } S \rangle$  and
       $\text{lev}$ :  $\langle \text{cdcl}_W\text{-}M\text{-level-inv } S \rangle$  and
       $\langle \forall s \in \# \text{learned-clss } S. \neg \text{tautology } s \rangle$  and
       $\text{dist}$ :  $\langle \text{distinct-cdcl}_W\text{-state } S \rangle$  and
       $\text{conf}$ :  $\langle \text{cdcl}_W\text{-conflicting } S \rangle$  and
       $\langle \text{all-decomposition-implies-}m (\text{local.clauses } S)$ 
         $(\text{get-all-ann-decomposition } (\text{trail } S)) \rangle$  and
       $\text{learned}$ :  $\langle \text{cdcl}_W\text{-learned-clause } S \rangle$ 
    using  $invs$  unfolding  $\text{cdcl}_W\text{-all-struct-inv-def}$ 
    by auto

```

```

obtain  $D$  where
   $D[simp]: \langle \text{conflicting } S = \text{Some } (\text{add-mset } (-K) D) \rangle$ 
  using  $\text{confl } K$  by ( $\text{auto dest: multi-member-split}$ )

have  $\text{dist}: \langle \text{distinct-mset } (\text{the } (\text{conflicting } S)) \rangle$ 
  using  $\text{dist confl unfolding distinct-cdcl}_W\text{-state-def}$  by  $\text{auto}$ 
then have  $[iff]: \langle L \notin \# \text{ remove1-mset } L (\text{the } (\text{conflicting } S)) \rangle$  for  $L$ 
  by ( $\text{meson distinct-mem-diff-mset union-single-eq-member}$ )
from this[ $\text{of } K$ ] have  $[simp]: \langle -K \notin \# D \rangle$  using  $\text{dist}$  by  $\text{auto}$ 

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

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

```

```

case proped: True
obtain C M where
  [simp]:  $\langle \text{trail } S = \text{Propagated } K \ C \ \# \ M \rangle$ 
  using tr proped K-def
  by (cases  $\langle \text{trail } S \rangle$ ; cases  $\langle \text{hd } (\text{trail } S) \rangle$ )
  (auto simp: K-def)
have  $\langle a \ @ \ \text{Propagated } L \ \text{mark} \ \# \ b = \text{Propagated } K \ C \ \# \ M \longrightarrow$ 
   $b \models_{\text{as}} C \text{Not } (\text{remove1-mset } L \ \text{mark}) \wedge L \in \# \ \text{mark} \rangle$  for L mark a b
  using conf unfolding cdclW-conflicting-def
  by fastforce
from this[of  $\langle [] \rangle$ ] have [simp]:  $\langle K \in \# \ C \rangle \langle M \models_{\text{as}} C \text{Not } (\text{remove1-mset } K \ C) \rangle$ 
  by auto
have [simp]:  $\langle \text{get-maximum-level } (\text{Propagated } K \ C \ \# \ M) \ D = \text{get-maximum-level } M \ D \rangle$ 
  by (rule get-maximum-level-skip-first)
  (auto simp: atms-of-def atm-of-eq-atm-of uminus-lit-swap[symmetric])

have  $\langle \text{get-maximum-level } M \ D < \text{count-decided } M \rangle$ 
  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  $\langle \text{get-maximum-level } (\text{tl } (\text{trail } S)) \ D = \text{get-maximum-level } (\text{trail } S) \ D \rangle$ 
  apply (rule get-maximum-level-cong)
  using K-def  $\langle - \ K \notin \# \ D \rangle \langle K \notin \# \ D \rangle$ 
  apply (cases  $\langle \text{trail } S \rangle$ )
  by (auto simp: get-level-cons-if atm-of-eq-atm-of)
moreover have  $\langle \text{get-maximum-level } (\text{tl } (\text{trail } S)) \ D < \text{count-decided } (\text{trail } S) \rangle$ 
  using tr confl K proped count-decided-ge-get-maximum-level[of  $\langle \text{tl } (\text{trail } S) \rangle \ D$ ]
  by (cases  $\langle \text{trail } S \rangle$ ; cases  $\langle \text{hd } (\text{trail } S) \rangle$ )
  (auto simp: resolve.simps get-level-cons-if atm-of-eq-atm-of)
ultimately show ?thesis
  by (simp add: K-def)
qed

have H:
   $\langle \text{get-level } (\text{trail } S) \ L < \text{local.backtrack-lvl } S \rangle$ 
  if  $\langle L \in \# \ \text{remove1-mset } (-K) \ (\text{the } (\text{conflicting } S)) \rangle$ 
  for L
proof (cases  $\langle \text{is-proped } (\text{hd } (\text{trail } S)) \rangle$ )
case proped: True
obtain C M where
  [simp]:  $\langle \text{trail } S = \text{Propagated } K \ C \ \# \ M \rangle$ 
  using tr proped K-def
  by (cases  $\langle \text{trail } S \rangle$ ; cases  $\langle \text{hd } (\text{trail } S) \rangle$ )
  (auto simp: K-def)
have  $\langle a \ @ \ \text{Propagated } L \ \text{mark} \ \# \ b = \text{Propagated } K \ C \ \# \ M \longrightarrow$ 
   $b \models_{\text{as}} C \text{Not } (\text{remove1-mset } L \ \text{mark}) \wedge L \in \# \ \text{mark} \rangle$  for L mark a b
  using conf unfolding cdclW-conflicting-def
  by fastforce
from this[of  $\langle [] \rangle$ ] have [simp]:  $\langle K \in \# \ C \rangle \langle M \models_{\text{as}} C \text{Not } (\text{remove1-mset } K \ C) \rangle$ 
  by auto
have [simp]:  $\langle \text{get-maximum-level } (\text{Propagated } K \ C \ \# \ M) \ D = \text{get-maximum-level } M \ D \rangle$ 
  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-learningw
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: $\langle \text{backtrack } S \ T \rangle$ **and**
inv: $\langle \text{cdcl}_W\text{-all-struct-inv } S \rangle$ **and**
smaller: $\langle \text{no-smaller-propa } S \rangle$

shows

$\langle \text{mark-of } (\text{hd-trail } T) \notin \# \text{ clauses } S \rangle$

proof (rule *ccontr*)

assume *n-dist*: $\langle \neg \text{ ?thesis} \rangle$

obtain *K L* :: '*v* literal **and**

M1 M2 :: ('*v*, '*v* clause) ann-lit list **and** *i* :: nat **and** *D D'* **where**

confl-S: $\langle \text{conflicting } S = \text{Some } (\text{add-mset } L \ D) \rangle$ **and**

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

lev-L: $\langle \text{get-level } (\text{trail } S) \ L = \text{backtrack-lvl } S \rangle$ **and**

max-D-L: $\langle \text{get-level } (\text{trail } S) \ L = \text{get-maximum-level } (\text{trail } S) \ (\text{add-mset } L \ D') \rangle$ **and**

i: $\langle \text{get-maximum-level } (\text{trail } S) \ D' \equiv i \rangle$ **and**

lev-K: $\langle \text{get-level } (\text{trail } S) \ K = i + 1 \rangle$ **and**

T: $\langle T \sim \text{cons-trail } (\text{Propagated } L \ (\text{add-mset } L \ D')) \rangle$

$\langle \text{reduce-trail-to } M1$

$\langle \text{add-learned-cls } (\text{add-mset } L \ D')$

$\langle \text{update-conflicting } \text{None } S \rangle \rangle$ **and**

D-D': $\langle D' \subseteq \# \ D \rangle$ **and**

$\langle \text{clauses } S \models_{pm} \text{add-mset } L \ D' \rangle$

using *cdcl* **by** (auto elim!: *rulesE*)

obtain *M2'* **where** *M2'*: $\langle \text{trail } S = (M2' \ @ \ M2) \ @ \ \text{Decided } K \ \# \ M1 \rangle$

using *decomp* **by** auto

have *inv-T*: $\langle \text{cdcl}_W\text{-all-struct-inv } T \rangle$

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'*: $\langle M1 \models_{as} \text{CNot } D' \rangle$

using *backtrack-M1-CNot-D'[of S D' (i) K M1 M2 L (add-mset L D) T*

$\langle \text{Propagated } L \ (\text{add-mset } L \ D') \rangle]$ *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 $\langle \text{undefined-lit } M1 \ L \rangle$

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 $\langle D' + \{\#L\} \in \# \text{ clauses } S \rangle$

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: $\langle \text{backtrack } S \ T \rangle$ **and**

inv: $\langle \text{cdcl}_W\text{-all-struct-inv } S \rangle$ **and**

smaller: $\langle \text{no-smaller-propa } S \rangle$ **and**

smaller-conf: $\langle \text{no-smaller-conf } S \rangle$ **and**

E-subset: $\langle E \subset \# \ \text{mark-of } (\text{hd-trail } T) \rangle$

shows $\langle E \notin \# \text{ clauses } S \rangle$

proof (rule *ccontr*)

```

assume  $n\text{-dist}$ :  $\langle \neg \text{?thesis} \rangle$ 
obtain  $K\ L$  :: ' $v$  literal and
   $M1\ M2$  :: (' $v$ , ' $v$  clause) ann-lit list and  $i$  :: nat and  $D\ D'$  where
     $\text{confl-}S$ : conflicting  $S = \text{Some}(\text{add-mset } L\ D)$  and
     $\text{decomp}$ :  $(\text{Decided } K \# M1, M2) \in \text{set}(\text{get-all-ann-decomposition } (\text{trail } S))$  and
     $\text{lev-}L$ : get-level  $(\text{trail } S)\ L = \text{backtrack-lvl } S$  and
     $\text{max-}D\text{-}L$ : get-level  $(\text{trail } S)\ L = \text{get-maximum-level } (\text{trail } S)\ (\text{add-mset } L\ D')$  and
     $i$ : get-maximum-level  $(\text{trail } S)\ D' \equiv i$  and
     $\text{lev-}K$ : get-level  $(\text{trail } S)\ K = i + 1$  and
     $T$ :  $T \sim \text{cons-trail } (\text{Propagated } L\ (\text{add-mset } L\ D'))$ 
       $(\text{reduce-trail-to } M1$ 
         $(\text{add-learned-cls } (\text{add-mset } L\ D')$ 
           $(\text{update-conflicting } \text{None } S)))$  and
     $D\text{-}D'$ :  $\langle D' \subseteq \# D \rangle$  and
     $\langle \text{clauses } S \models_{pm} \text{add-mset } L\ D' \rangle$ 
using  $\text{cdcl}$  by  $(\text{auto elim!}; \text{rulesE})$ 

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

have  $M1\text{-}D'$ :  $\langle M1 \models_{as} CNot\ D' \rangle$ 
using  $\text{backtrack-}M1\text{-}CNot\text{-}D'$   $[of\ S\ D'\ \langle i \rangle\ K\ M1\ M2\ L\ \langle \text{add-mset } L\ D \rangle\ T$ 
   $(\text{Propagated } L\ (\text{add-mset } L\ D'))]$   $\text{inv confl-}S\ \text{decomp } i\ T\ D\text{-}D'\ \text{lev-}K\ \text{lev-}L\ \text{max-}D\text{-}L$ 
unfolding  $\text{cdcl}_W\text{-all-struct-inv-def cdcl}_W\text{-conflicting-def cdcl}_W\text{-M-level-inv-def}$ 
by  $(\text{auto simp: subset-mset-trans-add-mset})$ 
have  $\text{undef-}L$ :  $\langle \text{undefined-lit } M1\ L \rangle$ 
using  $\text{inv-}T\ T\ \text{decomp}$  unfolding  $\text{cdcl}_W\text{-all-struct-inv-def cdcl}_W\text{-M-level-inv-def}$ 
by  $(\text{auto simp: defined-lit-map})$ 

show False
proof  $(\text{cases } \langle L \in \# E \rangle)$ 
case True
then obtain  $E'$  where
   $E$ :  $\langle E = \text{add-mset } L\ E' \rangle$ 
by  $(\text{auto dest: multi-member-split})$ 
then have  $\langle \text{distinct-mset } E' \rangle$  and  $\langle L \notin \# E' \rangle$  and  $E'\text{-}E'$ :  $\langle E' \subseteq \# D' \rangle$ 
using  $\text{dist-}E\ T\ E\text{-subset}$  by auto
then have  $M1\text{-}E'$ :  $\langle M1 \models_{as} CNot\ E' \rangle$ 
using  $M1\text{-}D'\ T$  unfolding  $\text{true-annots-true-cls-def-iff-negation-in-model}$ 
by  $(\text{auto dest: multi-member-split}[of\ -\ E]\ \text{mset-subset-eq-insertD})$ 
have  $\text{propa}$ :  $\langle \bigwedge M' K M L D. \text{trail } S = M' @ \text{Decided } K \# M \implies$ 
   $D + \{\#L\# \} \in \# \text{clauses } S \implies \text{undefined-lit } M\ L \implies \neg M \models_{as} CNot\ D \rangle$ 
using  $\text{smaller}$  unfolding  $\text{no-smaller-propa-def}$  by blast
show False
using  $M1\text{-}E'\ \text{propa}[of\ \langle M2' @ M2 \rangle\ K\ M1\ E',\ OF\ M2' - \text{undef-}L]$   $n\text{-dist}$  unfolding  $E$ 
by auto
next
case False

```

```

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

```

lemma $cdcl_W\text{-stgy}\text{-no}\text{-relearned}\text{-highest}\text{-subres}\text{-clause}$:

```

assumes
   $cdcl$ :  $\langle backtrack\ S\ T \rangle$  and
   $inv$ :  $\langle cdcl_W\text{-all}\text{-struct}\text{-inv } S \rangle$  and
   $smaller$ :  $\langle no\text{-}smaller\text{-propa } S \rangle$  and
   $smaller\text{-}confl$ :  $\langle no\text{-}smaller\text{-confl } S \rangle$  and
   $E\text{-subset}$ :  $\langle mark\text{-of } (hd\text{-trail } T) = add\text{-mset } (lit\text{-of } (hd\text{-trail } T))\ E \rangle$ 
shows  $\langle add\text{-mset } (-\ lit\text{-of } (hd\text{-trail } T))\ E \notin \# clauses\ S \rangle$ 
proof (rule  $ccontr$ )
  assume  $n\text{-dist}$ :  $\langle \neg ?thesis \rangle$ 
  obtain  $K\ L :: 'v\ literal$  and
     $M1\ M2 :: ('v, 'v\ clause)\ ann\text{-lit } list$  and  $i :: nat$  and  $D\ D'$  where
     $confl\text{-}S$ :  $conflicting\ S = Some\ (add\text{-mset } L\ D)$  and
     $decomp$ :  $(Decided\ K \# M1, M2) \in set\ (get\text{-all}\text{-ann}\text{-decomposition } (trail\ S))$  and
     $lev\text{-}L$ :  $get\text{-level } (trail\ S)\ L = backtrack\text{-lvl } S$  and
     $max\text{-}D\text{-}L$ :  $get\text{-level } (trail\ S)\ L = get\text{-maximum}\text{-level } (trail\ S)\ (add\text{-mset } L\ D')$  and
     $i$ :  $get\text{-maximum}\text{-level } (trail\ S)\ D' \equiv i$  and
     $lev\text{-}K$ :  $get\text{-level } (trail\ S)\ K = i + 1$  and
     $T$ :  $T \sim cons\text{-trail } (Propagated\ L\ (add\text{-mset } L\ D'))$ 
    (reduce-trail-to  $M1$ 
      (add-learned-cls (add-mset  $L\ D'$ )
        (update-conflicting None  $S$ ))) and
     $D\text{-}D'$ :  $\langle D' \subseteq \# D \rangle$  and
     $\langle clauses\ S \models_{pm} add\text{-mset } L\ D' \rangle$ 
  using  $cdcl$  by (auto elim!: rulesE)

```

```

obtain  $M2'$  where  $M2'$ :  $\langle trail\ S = (M2' @ M2) @ Decided\ K \# M1 \rangle$ 
  using  $decomp$  by auto
have  $inv\text{-}T$ :  $\langle cdcl_W\text{-all}\text{-struct}\text{-inv } T \rangle$ 
  using  $cdcl$   $cdcl_W\text{-stgy}\text{-}cdcl_W\text{-all}\text{-struct}\text{-inv } inv$   $W\text{-other } backtrack\ bj$ 
   $cdcl_W\text{-all}\text{-struct}\text{-inv}\text{-inv } cdcl_W\text{-}cdcl_W\text{-restart}$  by blast
have  $\langle distinct\text{-mset } (add\text{-mset } L\ D') \rangle$ 
  using  $inv\text{-}T\ T$  unfolding  $cdcl_W\text{-all}\text{-struct}\text{-inv}\text{-def}$   $distinct\text{-}cdcl_W\text{-state}\text{-def}$ 
  by auto

```

```

have  $M1\text{-}D'$ :  $\langle M1 \models_{as} CNot\ D' \rangle$ 
  using  $backtrack\text{-}M1\text{-}CNot\text{-}D'$ [of  $S\ D'$   $\langle i \rangle K M1 M2 L \langle add\text{-mset } L\ D \rangle T$ 
     $\langle Propagated\ L\ (add\text{-mset } L\ D') \rangle]$   $inv\ confl\text{-}S\ decomp\ i\ T\ D\text{-}D'\ lev\text{-}K\ lev\text{-}L\ max\text{-}D\text{-}L$ 
  unfolding  $cdcl_W\text{-all}\text{-struct}\text{-inv}\text{-def}$   $cdcl_W\text{-conflicting}\text{-def}$   $cdcl_W\text{-}M\text{-level}\text{-inv}\text{-def}$ 
  by (auto simp: subset-mset-trans-add-mset)
have  $undef\text{-}L$ :  $\langle undefined\text{-lit } M1\ L \rangle$ 
  using  $inv\text{-}T\ T\ decomp$  unfolding  $cdcl_W\text{-all}\text{-struct}\text{-inv}\text{-def}$   $cdcl_W\text{-}M\text{-level}\text{-inv}\text{-def}$ 

```

```

  by (auto simp: defined-lit-map)
then have undef-uL: ⟨undefined-lit M1 (−L)⟩
  by auto
have propa: ⟨ $\bigwedge M' K M L D. \text{trail } S = M' @ \text{Decided } K \# M \implies$ 
   $D + \{\#L\} \in \# \text{ clauses } S \implies \text{undefined-lit } M L \implies \neg M \models_{as} CNot D$ ⟩
  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: ⟨ $\bigwedge M' K M L D. \text{trail } S = M' @ \text{Decided } K \# M \implies$ 
   $D + \{\#L\} \in \# \text{ clauses } S \implies \text{undefined-lit } M L \implies \neg M \models_{as} CNot D$ ⟩
  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: ⟨cdclW-stgy S T⟩ and
  inv: cdclW-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: ⟨ $\neg \text{distinct-mset (clauses T)}$ ⟩
  then have ⟨backtrack S T⟩
    using cdcl dist by (auto simp: cdclW-stgy.simps cdclW-o.simps cdclW-bj.simps
      elim: propagateE conflictE decideE skipE resolveE)
  then show False
    using n-dist cdclW-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: ⟨cdclW-stgy S T⟩ and
  inv: cdclW-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: ⟨ $\neg ?thesis$ ⟩
  then have ⟨backtrack S T⟩
    using cdcl dist by (auto simp: cdclW-stgy.simps cdclW-o.simps cdclW-bj.simps
      elim: propagateE conflictE decideE skipE resolveE)
  then show False
    using n-dist cdclW-stgy-no-relearned-clause[of S T] dist
    by (auto simp: inv smaller clauses-def elim!: rulesE)
qed

```

lemma *rtrancp-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 ∉# (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 ∈# 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 $n\text{-dist}$: $\langle \neg \text{?thesis} \rangle$
then have $\langle \text{backtrack } S \ T \rangle$
 using cdcl dist **by** $(\text{auto simp: cdcl}_W\text{-stgy.simps cdcl}_W\text{-o.simps cdcl}_W\text{-bj.simps}$
 $\text{elim: propagateE conflictE decideE skipE resolveE})$
then show False
 using $n\text{-dist cdcl}_W\text{-stgy-no-relearned-clause}$ $[of \ S \ T]$
 by $(\text{auto simp: inv smaller clauses-def elim!: rulesE}$
 $\text{dest!: in-diffD})$

qed

lemma $\text{rtranclp-cdcl}_W\text{-stgy-distinct-mset-clauses-new-abs}$:

assumes
 st : $\text{cdcl}_W\text{-stgy}^{**} \ R \ S$ **and**
 $invR$: $\text{cdcl}_W\text{-all-struct-inv } R$ **and**
 $no\text{-smaller}$: $\langle \text{no-smaller-propa } R \rangle$ **and**
 $\langle \text{distinct-mset (learned-clss } R - A) \rangle$
shows $\text{distinct-mset (learned-clss } S - A)$
using assms **by** $(\text{induction rule: rtranclp-induct})$
 $(\text{auto simp: cdcl}_W\text{-stgy-distinct-mset rtranclp-cdcl}_W\text{-stgy-no-smaller-propa}$
 $\text{rtranclp-cdcl}_W\text{-stgy-cdcl}_W\text{-all-struct-inv}$
 $\text{cdcl}_W\text{-stgy-learned-distinct-mset-new})$

lemma $\text{rtranclp-cdcl}_W\text{-stgy-distinct-mset-clauses-new}$:

assumes
 st : $\text{cdcl}_W\text{-stgy}^{**} \ R \ S$ **and**
 $invR$: $\text{cdcl}_W\text{-all-struct-inv } R$ **and**
 $no\text{-smaller}$: $\langle \text{no-smaller-propa } R \rangle$
shows $\text{distinct-mset (learned-clss } S - \text{learned-clss } R)$
using assms **by** $(\text{rule rtranclp-cdcl}_W\text{-stgy-distinct-mset-clauses-new-abs}) \text{ auto}$

Decrease of a Measure

fun $\text{cdcl}_W\text{-restart-measure}$ **where**

$\text{cdcl}_W\text{-restart-measure } S =$
 $[(\mathcal{I}::\text{nat}) \wedge (\text{card (atms-of-mm (init-clss } S)) - \text{card (set-mset (learned-clss } S))},$
 $\text{if conflicting } S = \text{None then } 1 \text{ else } 0,$
 $\text{if conflicting } S = \text{None then } \text{card (atms-of-mm (init-clss } S)) - \text{length (trail } S)$
 $\text{else length (trail } S)$
 $]$

lemma $\text{length-model-le-vars}$:

assumes
 $no\text{-strange-atm } S$ **and**
 $no\text{-d}$: $\text{no-dup (trail } S)$ **and**
 $\text{finite (atms-of-mm (init-clss } S))}$
shows $\text{length (trail } S) \leq \text{card (atms-of-mm (init-clss } S))$

proof –

obtain $M \ N \ U \ k \ D$ **where** S : $\text{state } S = (M, N, U, k, D)$ **by** $(\text{cases state } S, \text{auto})$
have $\text{finite (atm-of ' lits-of-l (trail } S))}$
 using $\text{assms}(1, \mathcal{I})$ **unfolding** S **by** $(\text{auto simp add: finite-subset})$
have $\text{length (trail } S) = \text{card (atm-of ' lits-of-l (trail } S))}$
 using $\text{no-dup-length-eq-card-atm-of-lits-of-l no-d}$ **by** blast
then show ?thesis **using** $\text{assms}(1)$ **unfolding** $\text{no-strange-atm-def}$
by $(\text{auto simp add: assms}(\mathcal{I}) \text{ card-mono})$

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
 $\forall s \in \# \text{learned-clss } S. \neg \text{tautology } s$
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))*
 $\leq \text{card (simple-clss (atms-of-mm (learned-clss S)))}$
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:
 $\neg(\text{learned-clss } S \subseteq \# \text{learned-clss } S' \wedge [] = \text{trail } S' \wedge \text{conflicting } S' = \text{None})$
and
no-forget: learned-clss S ⊆ # learned-clss S' and
no-relearn: $\bigwedge S'. \text{backtrack } S S' \implies \text{mark-of (hd-trail } S') \notin \# \text{learned-clss } S$
and
alien: no-strange-atm S and
M-level: cdcl_W-M-level-inv S and
no-taut: $\forall s \in \# \text{learned-clss } S. \neg \text{tautology } s$ 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) ∈ le_{rn} 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)*
 $\subseteq \text{atms-of-mm (init-clss S)}$
using *undef unfolding no-strange-atm-def* **by** *auto*

```

then have card (atm-of ' lits-of-l (Propagated L C # trail S))
  ≤ card (atms-of-mm (init-clss S))
  by (meson atms-of-ms-finite card-mono finite-set-mset)
then have length (Propagated L C # trail S) ≤ card (atms-of-mm ?N)
  using no-dup-length-eq-card-atm-of-lits-of-l no-dup' by fastforce
then have H: card (atms-of-mm (init-clss S)) - length (trail S)
  = Suc (card (atms-of-mm (init-clss S)) - Suc (length (trail S)))
  by simp
show ?case using conf T undef by (auto simp: H lexn3-conv)
next
case (decide L) note conf = this(1) and undef = this(2) and T = this(4)
moreover {
  have dec: decide S (cons-trail (Decided L) S)
    using decide-rule decide.hyps by force
  then have cdclW-restart S (cons-trail (Decided L) S)
    using cdclW-restart.simps cdclW-o.intros by blast } note cdclW-restart = this
moreover {
  have lev: cdclW-M-level-inv (cons-trail (Decided L) S)
    using cdclW-restart M-level cdclW-restart-consistent-inv[OF cdclW-restart] by auto
  then have no-dup: no-dup (Decided L # trail S)
    using undef unfolding cdclW-M-level-inv-def by auto
  have no-strange-atm (cons-trail (Decided L) S)
    using M-level alien calculation(4) cdclW-restart-no-strange-atm-inv by blast
  then have length (Decided L # (trail S))
    ≤ card (atms-of-mm (init-clss S))
    using no-dup undef
    length-model-le-vars[of cons-trail (Decided L) 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-cdclW-state T
  using bt M-level distinct-cdclW-state-inv no-dup other cdclW-o.intros cdclW-bj.intros by blast
moreover have ∀ s ∈ #learned-clss T. ¬ tautology s
  using learned-clss-are-not-tautologies[OF cdclW-restart.other[OF cdclW-o.bj[OF
    cdclW-bj.backtrack[OF bt]]]] M-level no-taut confl by auto
ultimately have card (set-mset (learned-clss T)) ≤ 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: cdclW-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: cdclW-M-level-inv-decomp le3-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: ⟨cdclW-stgy S T⟩ and
  struct-inv: ⟨cdclW-all-struct-inv S⟩ and
  smaller: ⟨no-smaller-propa S⟩
shows (cdclW-restart-measure T, cdclW-restart-measure S) ∈ le3 less-than 3
proof (rule cdclW-restart-measure-decreasing)
show ⟨cdclW-restart S T⟩
  using cdcl cdclW-cdclW-restart cdclW-stgy-cdclW by blast
show ⟨¬ (learned-clss S ⊆# learned-clss T ∧ [] = trail T ∧ conflicting T = None)⟩
  using cdcl by (cases rule: cdclW-stgy-cases) (auto elim!: rulesE)
show ⟨learned-clss S ⊆# learned-clss T⟩
  using cdcl by (cases rule: cdclW-stgy-cases) (auto elim!: rulesE)
show ⟨mark-of (hd-trail S') ∉# learned-clss S⟩ if ⟨backtrack S S'⟩ for S'
  using cdclW-stgy-no-relearned-clause[of S S'] cdclW-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 ⟨cdclW-M-level-inv S⟩ and ⟨distinct-cdclW-state S⟩ and
  ⟨cdclW-conflicting S⟩ and ⟨∀ s ∈ #learned-clss S. ¬ tautology s⟩
  using struct-inv unfolding cdclW-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 *trancpl-cdcl_W-stgy-decreasing*:
fixes $R\ S\ T :: 'st$
assumes $cdcl_W\text{-stgy}^{++}\ R\ S$ **and**
 $tr: trail\ R = []$ **and**
 $cdcl_W\text{-all-struct-inv}\ R$
shows $(cdcl_W\text{-restart-measure}\ S, cdcl_W\text{-restart-measure}\ R) \in lexn\ less-than\ 3$
using *assms*
apply *induction*
using *empty-trail-no-smaller-propa cdcl_W-stgy-no-relearned-clause cdcl_W-stgy-step-decreasing*
apply *blast*
using *trancpl-into-rtrancpl[of cdcl_W-stgy R] lexn-transI[OF trans-less-than, of 3]*
 $rtrancpl\text{-}cdcl_W\text{-stgy-no-smaller-propa}$ **unfolding** *trans-def*
by (*meson cdcl_W-stgy-step-decreasing empty-trail-no-smaller-propa*
 $rtrancpl\text{-}cdcl_W\text{-stgy-cdcl}_W\text{-all-struct-inv}$)

lemma *trancpl-cdcl_W-stgy-S0-decreasing*:
fixes $R\ S\ T :: 'st$
assumes
 $pl: cdcl_W\text{-stgy}^{++}\ (init\text{-state}\ N)\ S$ **and**
 $no\text{-dup}: distinct\text{-mset-mset}\ N$
shows $(cdcl_W\text{-restart-measure}\ S, cdcl_W\text{-restart-measure}\ (init\text{-state}\ N)) \in lexn\ less-than\ 3$
proof –
have $cdcl_W\text{-all-struct-inv}\ (init\text{-state}\ N)$
using *no-dup unfolding cdcl_W-all-struct-inv-def* **by** *auto*
then show *?thesis* **using** *pl trancpl-cdcl_W-stgy-decreasing init-state-trail* **by** *blast*
qed

lemma *wf-trancpl-cdcl_W-stgy*:
 $wf\ \{(S :: 'st, init\text{-state}\ N) \mid S\ N.\ distinct\text{-mset-mset}\ N \wedge cdcl_W\text{-stgy}^{++}\ (init\text{-state}\ N)\ S\}$
apply (*rule wf-wf-if-measure'-notation2[of lexn less-than 3 - - cdcl_W-restart-measure]*)
apply (*simp add: wf wf-lexn*)
using *trancpl-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
 $dec: \langle decide\ S\ T \rangle$ **and**
 $inv: \langle cdcl_W\text{-all-struct-inv}\ T \rangle$ **and**
 $smaller: \langle no\text{-smaller-propa}\ T \rangle$ **and**
 $smaller\text{-confl}: \langle no\text{-smaller-confl}\ T \rangle$
shows $\langle \neg conflict\ T\ U \rangle$
proof (*rule ccontr*)
assume $\langle \neg ?thesis \rangle$
then obtain D **where**
 $D: \langle D \in \# clauses\ T \rangle$ **and**
 $confl: \langle trail\ T \models_{as}\ CNot\ D \rangle$
by (*auto simp: conflict.simps*)
obtain L **where**
 $\langle conflicting\ S = None \rangle$ **and**
 $undef: \langle undefined\text{-lit}\ (trail\ S)\ L \rangle$ **and**
 $\langle atm\text{-of}\ L \in atms\text{-of-mm}\ (init\text{-class}\ S) \rangle$ **and**

```

  T: ⟨T ~ cons-trail (Decided L) S⟩
  using dec by (auto simp: decide.simps)
have dist: ⟨distinct-mset D⟩
  using inv D unfolding cdclW-all-struct-inv-def distinct-cdclW-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:

$\langle \text{Suc } a \leq n \implies 2 * 2^{\wedge}(n - \text{Suc } a) = (2::\text{nat})^{\wedge}(n - a) \rangle$
 $\langle \text{Suc } a \leq n \implies 2^{\wedge}(n - \text{Suc } a) * 2 = (2::\text{nat})^{\wedge}(n - a) \rangle$
 $\langle \text{Suc } a \leq n \implies 2^{\wedge}(n - \text{Suc } a) * b * 2 = (2::\text{nat})^{\wedge}(n - a) * b \rangle$
 $\langle \text{Suc } a \leq n \implies 2^{\wedge}(n - \text{Suc } a) * (b * 2) = (2::\text{nat})^{\wedge}(n - a) * b \rangle$
 $\langle \text{Suc } a \leq n \implies 2^{\wedge}(n - \text{Suc } a) * (2 * b) = (2::\text{nat})^{\wedge}(n - a) * b \rangle$
 $\langle \text{Suc } a \leq n \implies 2 * b * 2^{\wedge}(n - \text{Suc } a) = (2::\text{nat})^{\wedge}(n - a) * b \rangle$
 $\langle \text{Suc } a \leq n \implies 2 * (b * 2^{\wedge}(n - \text{Suc } a)) = (2::\text{nat})^{\wedge}(n - a) * b \rangle$
apply (simp-all add: Suc-diff-Suc semiring-normalization-rules(27))
using Suc-diff-le **by** fastforce+

lemma decide-propa-weight:

$\langle \text{decide } S \ T \implies n \geq \text{length } (\text{trail } T) \implies \text{propa-weight } n \ (\text{trail } S) \leq \text{propa-weight } n \ (\text{trail } T) \rangle$
by (auto elim!: decideE simp: comp-list-weight-propa-trail-def
 algebra-simps pow2-times-n)

lemma propagate-propa-weight:

$\langle \text{propagate } S \ T \implies n \geq \text{length } (\text{trail } T) \implies \text{propa-weight } n \ (\text{trail } S) < \text{propa-weight } n \ (\text{trail } T) \rangle$
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 $S \rightarrow \dots \rightarrow T$. The bookkeeping part:

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

Then the simple part:

1. each backtrack increases *propa-weight*
2. but *propa-weight* is bounded by $(2::'a)^{\text{card } (\text{atms-of-mm } (\text{init-clss } S))}$ 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 $\llbracket \text{decide } ?S \ ?T; \text{cdcl}_W\text{-all-struct-inv } ?T; \text{no-smaller-propa } ?T; \text{no-smaller-confl } ?T \rrbracket \implies \neg \text{conflict } ?T \ ?U$ 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:  $\langle \text{cdcl}_W^{**} S T \rangle$  and
    conf:  $\langle \text{conflicting } S = \text{None} \rangle$  and
    inv:  $\langle \text{cdcl}_W\text{-all-struct-inv } S \rangle$ 
  shows  $\langle \text{size } (\text{learned-clss } T) \leq \text{size } (\text{learned-clss } S) + 2 \wedge (\text{card } (\text{atms-of-mm } (\text{init-clss } S))) \rangle$ 
  (is  $\langle - \leq - + ?b \rangle$ )
proof (rule ccontr)
  assume ge:  $\langle \neg ?thesis \rangle$ 
  let ?m =  $\langle \text{card } (\text{atms-of-mm } (\text{init-clss } S)) \rangle$ 
  obtain n :: nat where
    n:  $\langle (\text{cdcl}_W \sim n) S T \rangle$ 
    using cdcl unfolding rtrancp-power by fast
  then obtain f ::  $\langle \text{nat} \Rightarrow 'st \rangle$  where
    f:  $\langle \bigwedge i. i < n \implies \text{cdcl}_W (f i) (f (\text{Suc } i)) \rangle$  and
    [simp]:  $\langle f 0 = S \rangle$  and
    [simp]:  $\langle f n = T \rangle$ 
    using power-ex-decomp[OF n]
    by auto

  have cdcl-st-k:  $\langle \text{cdcl}_W^{**} S (f k) \rangle$  if  $\langle k \leq n \rangle$  for k
    using that
    apply (induction k)
    subgoal by auto
    subgoal for k using f[of k] by (auto)
    done
  let ?g =  $\langle \lambda i. \text{size } (\text{learned-clss } (f i)) \rangle$ 
  have  $\langle ?g 0 = \text{size } (\text{learned-clss } S) \rangle$ 
    by auto
  have g-n:  $\langle ?g n > ?g 0 + 2 \wedge (\text{card } (\text{atms-of-mm } (\text{init-clss } S))) \rangle$ 
    using ge by auto
  have g:  $\langle ?g (\text{Suc } i) = ?g i \vee (?g (\text{Suc } i) = \text{Suc } (?g i) \wedge \text{backtrack } (f i) (f (\text{Suc } i))) \rangle$  if  $\langle i < n \rangle$ 
    for i
    using f[OF that]
    by (cases rule: cdclW.cases)
    (auto elim: propagateE conflictE decideE backtrackE skipE resolveE
      simp: cdclW-o.simps cdclW-bj.simps)
  have g-le:  $\langle ?g i \leq i + ?g 0 \rangle$  if  $\langle i \leq n \rangle$  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:  $\langle n > ?b \rangle$ 
    using g-n ge by auto
  then have n0:  $\langle n > 0 \rangle$ 
    using not-add-less1 by fastforce
  define nth-bj where
     $\langle \text{nth-bj} = \text{rec-nat } 0 (\lambda j. (\text{LEAST } i. i > j \wedge i < n \wedge \text{backtrack } (f i) (f (\text{Suc } i)))) \rangle$ 
  have [simp]:  $\langle \text{nth-bj } 0 = 0 \rangle$ 
    by (auto simp: nth-bj-def)
  have nth-bj-Suc:  $\langle \text{nth-bj } (\text{Suc } i) = (\text{LEAST } x. \text{nth-bj } i < x \wedge x < n \wedge \text{backtrack } (f x) (f (\text{Suc } x))) \rangle$ 
    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 ⟨λx. 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: cdclW-o.simps cdclW-bj.simps cdclW.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: cdclW-o.simps cdclW-bj.simps cdclW.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
  ge-i: ⟨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:  $\langle n \rangle x \implies x \rangle nth\text{-bj } i \implies ?g (Suc\ x) = ?g (Suc\ (nth\text{-bj } i))$  for x
  apply (induction x)
subgoal by auto
subgoal for x
  by (cases  $\langle nth\text{-bj } i < x \rangle$ ) (auto dest: eq1)
done
  from this[of  $\langle n-1 \rangle$ ] have g-n-nth-bj:  $\langle ?g\ n = ?g\ (Suc\ (nth\text{-bj } i)) \rangle$ 
  using n-ge-m i-le-m le-n
by (cases  $\langle nth\text{-bj } i < n - Suc\ 0 \rangle$ )
  (auto dest: eq2)
  then have  $\langle size\ (learned\text{-clss}\ (f\ (Suc\ (nth\text{-bj } i)))) < size\ (learned\text{-clss}\ T) \rangle$ 
  using g-n i-le-m n-ge-m g-le[of  $\langle Suc\ (nth\text{-bj } i) \rangle$ ] le-n ge
   $\langle ?g\ (nth\text{-bj } i) = size\ (learned\text{-clss}\ S) + (i - 1) \rangle$ 
using Suc.IH by auto
  then show False
  using g-n i-le-m n-ge-m g-le[of  $\langle Suc\ (nth\text{-bj } i) \rangle$ ] g-n-nth-bj by auto
qed

from LeastI-ex[OF ex-larger]
have bt:  $\langle backtrack\ (f\ (nth\text{-bj}\ (Suc\ i)))\ (f\ (Suc\ (nth\text{-bj}\ (Suc\ i)))) \rangle$  and
  le:  $\langle nth\text{-bj}\ (Suc\ i) < n \rangle$  and
  nth-mono:  $\langle nth\text{-bj } i < nth\text{-bj}\ (Suc\ i) \rangle$ 
  unfolding nth-bj-Suc[symmetric]
  by auto

have g-nth-Suc-g-Suc-nth:  $\langle ?g\ (nth\text{-bj}\ (Suc\ i)) = ?g\ (Suc\ (nth\text{-bj } i)) \rangle$ 
  using g-nth-bj-eq2[of  $\langle nth\text{-bj}\ (Suc\ i) - 1 \rangle i$ ] le nth-mono
  apply auto
  by (metis Suc-pred grOI less-Suc0 less-Suc-eq less-imp-diff-less)
have H1:  $\langle size\ (learned\text{-clss}\ (f\ (Suc\ (nth\text{-bj}\ (Suc\ i)))) \rangle =$ 
   $1 + size\ (learned\text{-clss}\ (f\ (nth\text{-bj}\ (Suc\ i)))) \rangle$  if  $\langle i = 0 \rangle$ 
  using bt unfolding that
  by (auto simp: that elim: backtrackE)
have ?case if  $\langle i > 0 \rangle$ 
  using IH that nth-mono le bt gei
  by (auto elim: backtrackE simp: g-nth-Suc-g-Suc-nth)
moreover have ?case if  $\langle i = 0 \rangle$ 
  using le bt gei nth-mono IH g-nth-bj-eq2[of  $\langle nth\text{-bj}\ (Suc\ i) - 1 \rangle i$ ]
  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
   $\langle (?g\ (nth\text{-bj } i) = size\ (learned\text{-clss}\ S) + (i - 1)) \rangle$  and
  nth-bj-le:  $\langle nth\text{-bj } i < n \rangle$  and
  nth-bj-ge:  $\langle nth\text{-bj } i \geq i \rangle$  and
  bt-nth-bj:  $\langle i > 0 \implies backtrack\ (f\ (nth\text{-bj } i))\ (f\ (Suc\ (nth\text{-bj } i))) \rangle$  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: cdclW-o.simps cdclW-bj.simps cdclW.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: cdclW-o.simps cdclW-bj.simps cdclW.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-conf:  $\langle \neg \text{conflict } (f\ k) (f\ (\text{Suc } k)) \rangle$ 
  if  $\langle k > \text{nth-bj } a \rangle$  and
     $\langle k < \text{nth-conf } a \rangle$  and
    a-n:  $\langle a \leq ?b \rangle \langle a > 0 \rangle$ 
  for k a
  using not-less-Least[of k  $\langle \lambda i. i > \text{nth-bj } a \wedge i < \text{nth-bj } (\text{Suc } a) \wedge \text{conflict } (f\ i) (f\ (\text{Suc } i)) \rangle$ ] that
    nth-conf-le-nth-bj-Suc[of a]
  unfolding nth-conf-def[symmetric]
  by auto
have conflicting-after-nth-conf:  $\langle \text{conflicting } (f\ (\text{Suc } (\text{nth-conf } a) + k)) \neq \text{None} \rangle$ 
  if a-n:  $\langle a \leq ?b \rangle \langle a > 0 \rangle$  and
    k:  $\langle \text{Suc } (\text{nth-conf } a) + k < \text{nth-bj } (\text{Suc } a) \rangle$ 
  for a k
  using k
  apply (induction k)
  subgoal using nth-conf-conflicting[OF a-n] by simp
  subgoal for k
    using conflicting-still-conflicting[of  $\langle \text{Suc } (\text{nth-conf } a + k) \rangle$  a] a-n
      nth-bj-le[of a] nth-bj-le-nth-conf[of a]
    apply (cases  $\langle \text{Suc } (\text{nth-conf } a + k) < n \rangle$ )
    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-conf:  $\langle \text{conflicting } (f\ (\text{Suc } (\text{nth-bj } a) + k)) = \text{None} \rangle$ 
  if a-n:  $\langle a \leq ?b \rangle \langle a > 0 \rangle$  and
    k:  $\langle \text{Suc } (\text{nth-bj } a) + k < \text{nth-conf } a \rangle$ 
  for a k
  using k
  apply (induction k)
  subgoal using confl-None[of a] a-n by simp
  subgoal for k
    using f[of  $\langle \text{Suc } (\text{nth-bj } a) + k \rangle$ ] no-conflict-before-nth-conf[of a  $\langle \text{Suc } (\text{nth-bj } a) + k \rangle$ ] a-n
      nth-conf-le-nth-bj-Suc[of a] nth-bj-le[of  $\langle \text{Suc } a \rangle$ ]
    apply (cases  $\langle \text{Suc } (\text{nth-bj } a + k) < n \rangle$ )
    apply (auto elim!: propagateE conflictE decideE backtrackE skipE resolveE
      simp: cdclW-o.simps cdclW-bj.simps cdclW.simps)[]
    by linarith
  done
have
  ex-trail-decomp:  $\langle \exists M. \text{trail } (f\ (\text{Suc } (\text{nth-conf } a))) = M @ \text{trail } (f\ (\text{Suc } (\text{nth-conf } a + k))) \rangle$ 
  if a-n:  $\langle a \leq ?b \rangle \langle a > 0 \rangle$  and
    k:  $\langle \text{Suc } (\text{nth-conf } a) + k \leq \text{nth-bj } (\text{Suc } a) \rangle$ 
  for a k
  using k
  proof (induction k)
  case 0
  then show  $\langle ?case \rangle$  by auto
  next
  case (Suc k)
  moreover have  $\langle \text{nth-conf } a + k < n \rangle$ 
  proof -
  have nth-bj (Suc a) < n
    by (rule nth-bj-le) (use a-n(1) in simp)
  then show ?thesis
    using Suc.premis by linarith

```

```

qed
moreover have ⟨ $\exists Ma. M @ \text{trail } (f \text{ (Suc (nth-conf } a + k))) =$   

   $Ma @ \text{tl } (\text{trail } (f \text{ (Suc (nth-conf } a + k))))$ ⟩ for M
  by (cases ⟨ $\text{trail } (f \text{ (Suc (nth-conf } a + k)))$ ⟩) auto
ultimately show ?case
  using f[of ⟨ $\text{Suc (nth-conf } a + k)$ ⟩] conflicting-after-nth-conf[of a ⟨k⟩, OF a-n] Suc
    between-nth-bj-not-bt[of ⟨ $\text{Suc (nth-conf } a + k)$ ⟩ ⟨a⟩]
nth-bj-le-nth-conf[of a, OF a-n]
  apply (cases ⟨ $\text{Suc (nth-conf } a + k) < n$ ⟩)
  subgoal
    by (auto elim!: propagateE conflictE decideE skipE resolveE
      simp: cdclW-o.simps cdclW-bj.simps cdclW.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-conf:
  ⟨ $\text{propa-weight } n \text{ (trail } (f \text{ (Suc (nth-bj (Suc a)))))} > \text{propa-weight } n \text{ (trail } (f \text{ (nth-conf } a)))$ ⟩
  if a-n: ⟨ $a \leq ?b$ ⟩ ⟨ $a > 0$ ⟩ and
    n: ⟨ $n \geq \text{length } (\text{trail } (f \text{ (nth-conf } a)))$ ⟩
  for a n
proof -
  have pw0: ⟨ $\text{propa-weight } n \text{ (trail } (f \text{ (Suc (nth-conf } a))) =$   

     $\text{propa-weight } n \text{ (trail } (f \text{ (nth-conf } a)))$ ⟩ and
    [simp]: ⟨ $\text{trail } (f \text{ (Suc (nth-conf } a))) = \text{trail } (f \text{ (nth-conf } a))$ ⟩
  using nth-conf[OF a-n] by (auto elim!: conflictE)
  have H: ⟨ $\text{nth-bj (Suc } a) = \text{Suc (nth-conf } a) \vee \text{nth-bj (Suc } a) \geq \text{Suc (Suc (nth-conf } a))$ ⟩
  using nth-bj-le-nth-conf[of a, OF a-n]
  using a-n(1) nth-conf-le-nth-bj-Suc that(2) by force

  from ex-trail-decomp[of a ⟨ $\text{nth-bj (Suc } a) - (1 + \text{nth-conf } a)$ ⟩, OF a-n]
  obtain M where
    M: ⟨ $\text{trail } (f \text{ (Suc (nth-conf } a))) = M @ \text{trail } (f \text{ (nth-bj (Suc } a)))$ ⟩
  apply -
  apply (rule disjE[OF H])
  subgoal
    by auto
  subgoal
    using nth-bj-le-nth-conf[of a, OF a-n] nth-bj-ge[of ⟨ $\text{Suc } a$ ⟩] a-n
  by (auto simp add: numeral-2-eq-2)
  done
  obtain K M1 M2 D D' L where
    decomp: ⟨ $(\text{Decided } K \# M1, M2)$   

     $\in \text{set } (\text{get-all-ann-decomposition } (\text{trail } (f \text{ (nth-bj (Suc } a)))))$ ⟩ and
    ⟨ $\text{get-maximum-level } (\text{trail } (f \text{ (nth-bj (Suc } a)))) (\text{add-mset } L \text{ } D') =$   

     $\text{backtrack-lvl } (f \text{ (nth-bj (Suc } a)))$ ⟩ and
    ⟨ $\text{get-level } (\text{trail } (f \text{ (nth-bj (Suc } a)))) L = \text{backtrack-lvl } (f \text{ (nth-bj (Suc } a)))$ ⟩ and
    ⟨ $\text{get-level } (\text{trail } (f \text{ (nth-bj (Suc } a)))) K =$   

     $\text{Suc } (\text{get-maximum-level } (\text{trail } (f \text{ (nth-bj (Suc } a)))) D'$ ⟩ and
    ⟨ $D' \subseteq \# D$ ⟩ and
    ⟨ $\text{clauses } (f \text{ (nth-bj (Suc } a))) \models_{\text{pm}} \text{add-mset } L \text{ } D'$ ⟩ and
    st-Suc: ⟨ $f \text{ (Suc (nth-bj (Suc } a))) \sim$   

    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 ≤ n⟩
  for k
proof -
  have ⟨cdclW-all-struct-inv (f k)⟩
    using rtrancp-cdclW-cdclW-restart[OF cdcl-st-k[OF that]] inv
    rtrancp-cdclW-all-struct-inv-inv by blast
  then have ⟨cdclW-M-level-inv (f k)⟩ and ⟨no-strange-atm (f k)⟩
    unfolding cdclW-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 cdclW-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 rtrancp-cdclW-restart-init-clss[OF rtrancp-cdclW-cdclW-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: cdclW-o.simps cdclW-bj.simps cdclW.simps
        dest!: propagate-propa-weight[of - - ?m]
        decide-propa-weight[of - - ?m])
qed
have WTF3:  $\langle (Suc\ (nth\text{-}bj\ a + (nth\text{-}confl\ a - Suc\ (nth\text{-}bj\ a)))) = nth\text{-}confl\ a \rangle$ 
  using a-n(1) nth-bj-le-nth-confl that(2) by fastforce
have  $\langle propa\text{-}weight\ ?m\ (trail\ (f\ (Suc\ (nth\text{-}bj\ a) + k))) \rangle$ 
   $\geq propa\text{-}weight\ ?m\ (trail\ (f\ (Suc\ (nth\text{-}bj\ a)))) \rangle$ 
  if  $\langle k \leq nth\text{-}confl\ a - Suc\ (nth\text{-}bj\ a) \rangle$ 
  for k
  using that
  apply (induction k)
  subgoal by auto
  subgoal for k using ppa[of k]
    apply (cases  $\langle k < nth\text{-}confl\ a - Suc\ (nth\text{-}bj\ a) \rangle$ )
subgoal by auto
subgoal by linarith
  done
  done
  from this[of  $\langle nth\text{-}confl\ a - (Suc\ (nth\text{-}bj\ a)) \rangle$ ]
  show ?thesis
    by (auto simp: WTF3)
qed
have propa-weight-decreasing-confl:
   $\langle propa\text{-}weight\ ?m\ (trail\ (f\ (Suc\ (nth\text{-}bj\ a)))) \rangle$ 
   $< propa\text{-}weight\ ?m\ (trail\ (f\ (Suc\ (nth\text{-}bj\ (Suc\ a)))) \rangle$ 
  if a-n:  $\langle a \leq ?b \rangle \langle a > 0 \rangle$ 
  for a
proof -
  have WTF:  $\langle b < c \implies a \leq b \implies a < c \rangle$  for a b c :: nat by linarith
  have  $\langle nth\text{-}confl\ a < n \rangle$ 
    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  $\langle nth\text{-}confl\ a \rangle$ ]  $\langle nth\text{-}confl\ a < n \rangle$  by auto
  apply (rule propa-weight-decreasing-propa[OF a-n])
  done
qed
have weight1:  $\langle propa\text{-}weight\ ?m\ (trail\ (f\ (Suc\ (nth\text{-}bj\ 1)))) \rangle \geq 1 \rangle$ 
  using bt-nth-bj[of 1]
  by (auto simp: elim!: backtrackE intro!: trans-le-add1)
have  $\langle propa\text{-}weight\ ?m\ (trail\ (f\ (Suc\ (nth\text{-}bj\ (Suc\ a)))) \rangle \geq$ 
   $propa\text{-}weight\ ?m\ (trail\ (f\ (Suc\ (nth\text{-}bj\ 1)))) + a \rangle$ 
  if a-n:  $\langle a \leq ?b \rangle$ 
  for a :: nat
  using that
  apply (induction a)
  subgoal by auto
  subgoal for a
    using propa-weight-decreasing-confl[of  $\langle Suc\ a \rangle$ ]
    by auto
  done

```

```

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

```

Application of the previous theorem to an initial state:

corollary *cdcl-pow2-n-learned-clauses2*:

```

assumes
  cdcl:  $\langle \text{cdcl}_W^{**} (\text{init-state } N) \text{ } T \rangle$  and
  inv:  $\langle \text{cdcl}_W\text{-all-struct-inv } (\text{init-state } N) \rangle$ 
shows  $\langle \text{size (learned-cls } T) \leq 2 \wedge (\text{card (atms-of-mm } N) \rangle$ 
using assms cdcl-pow2-n-learned-clauses[of  $\langle \text{init-state } N \rangle \text{ } T$ ]
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 cdclW-restart.intros[intro] cdclW-bj.intros[intro] cdclW-o.intros[intro]
state-prop [simp del]

```

lemma *backtrack-no-cdcl_W-bj*:

```

assumes cdcl: cdclW-bj T U
shows  $\neg$ backtrack V T
using cdcl
apply (induction rule: cdclW-bj.induct)
  apply (elim skipE, force elim!: backtrackE simp: cdclW-M-level-inv-def)
  apply (elim resolveE, force elim!: backtrackE simp: cdclW-M-level-inv-def)
apply standard
apply (elim backtrackE)
apply (force simp add: cdclW-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 rtrancpl-cdclW-bj-skip-or-resolve-backtrack:
assumes cdclW-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) cdclW-bj++ S U
    using st unfolding rtrancpl-unfold by blast
  then show ?case
proof cases
  case SUp
  have  $\bigwedge$  T. skip-or-resolve** S T  $\Longrightarrow$  cdclW-restart** S T
    using mono-rtrancpl[of skip-or-resolve cdclW-restart]
    by (blast intro: skip-or-resolve.cases)
  then have skip-or-resolve** S U
    using bj IH backtrack-no-cdclW-bj by meson
  then show ?thesis
    using bj by (auto simp: cdclW-bj.simps dest!: skip-or-resolve.intros)
next
  case SU
  then show ?thesis
    using bj by (auto simp: cdclW-bj.simps dest!: skip-or-resolve.intros)
qed
qed

```

```

lemma rtrancpl-skip-or-resolve-rtrancpl-cdclW-restart:
skip-or-resolve** S T  $\Longrightarrow$  cdclW-restart** S T
by (induction rule: rtrancpl-induct)
  (auto dest!: cdclW-bj.intros cdclW-restart.intros cdclW-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  $x\ y \implies \text{length } (\text{trail } y) < \text{length } (\text{trail } x)$  for  $x\ y$ 
    by (auto simp: skip-or-resolve.simps elim!: skipE resolveE)
  then show ?thesis
    using wfP-if-measure[of  $\lambda\cdot. \text{True skip-or-resolve } \lambda S. \text{length } (\text{trail } S)$ ]
    by meson
qed

```

```

definition invNOT :: 'st  $\Rightarrow$  bool' where
invNOT  $\equiv \lambda S. \text{no-dup } (\text{trail } S)$ 

```

```

declare invNOT-def[simp]
end

```

```

context conflict-driven-clause-learningW
begin

```

1.2.2 More lemmas about Conflict, Propagate and Backjumping

Termination

```

lemma cdclW-bj-measure:
  assumes cdclW-bj  $S\ T$ 
  shows  $\text{length } (\text{trail } S) + (\text{if conflicting } S = \text{None then } 0 \text{ else } 1)$ 
     $> \text{length } (\text{trail } T) + (\text{if conflicting } T = \text{None then } 0 \text{ else } 1)$ 
  using assms by (induction rule: cdclW-bj.induct) (force elim!: backtrackE skipE resolveE)+

```

```

lemma wf-cdclW-bj:
  wf  $\{(b,a). \text{cdcl}_W\text{-bj } a\ b\}$ 
  apply (rule wfP-if-measure[of  $\lambda\cdot. \text{True}$ 
     $-\lambda T. \text{length } (\text{trail } T) + (\text{if conflicting } T = \text{None then } 0 \text{ else } 1), \text{simplified}$ ])
  using cdclW-bj-measure by simp

```

```

lemma cdclW-bj-exists-normal-form:
  shows  $\exists T. \text{full cdcl}_W\text{-bj } S\ T$ 
  using wf-exists-normal-form-full[OF wf-cdclW-bj] .

```

```

lemma rtrancpl-skip-state-decomp:
  assumes skip**  $S\ T$ 
  shows
     $\exists M. \text{trail } S = M @ \text{trail } T \wedge (\forall m \in \text{set } M. \neg \text{is-decided } m)$ 
     $\text{init-clss } S = \text{init-clss } T$ 
     $\text{learned-clss } S = \text{learned-clss } T$ 
     $\text{backtrack-lvl } S = \text{backtrack-lvl } T$ 
     $\text{conflicting } S = \text{conflicting } T$ 
  using assms by (induction rule: rtrancpl-induct) (auto elim!: skipE)

```

Analysing is confluent

```

lemma backtrack-reduce-trail-to-state-eq:
  assumes
     $V\ T: \langle V \sim \text{tl-trail } T \rangle$  and
    decomp:  $\langle (\text{Decided } K \# M1, M2) \in \text{set } (\text{get-all-ann-decomposition } (\text{trail } V)) \rangle$ 
  shows  $\langle \text{reduce-trail-to } M1\ (\text{add-learned-clss } E\ (\text{update-conflicting } \text{None } V))$ 
     $\sim \text{reduce-trail-to } M1\ (\text{add-learned-clss } E\ (\text{update-conflicting } \text{None } T)) \rangle$ 
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

```

```

    cdclW-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 cdclW-all-struct-inv V
    using rtrancp-mono[of skip cdclW-restart] assms(3) rtrancp-cdclW-all-struct-inv-inv mono-rtrancp
    by (auto dest!: bj other cdclW-bj.skip)
  then have cdclW-M-level-inv V
    unfolding cdclW-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 cdclW-restart** S T
    using bj cdclW-bj.skip mono-rtrancp[of skip cdclW-restart S T] other st by meson
  then have inv': cdclW-all-struct-inv T
    using rtrancp-cdclW-all-struct-inv-inv inv by blast
  have M-lev: cdclW-M-level-inv T using inv' unfolding cdclW-all-struct-inv-def by auto
  then have n-d': no-dup ?M
    using tr-M unfolding cdclW-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 cdclW-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 cdclW-conflicting-def cdclW-all-struct-inv-def tr-M by auto
  then have L' ∉ # D
    using L-L' L'-M unfolding true-annots-true-cls true-cls-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 cdclW-all-struct-inv-def cdclW-M-level-inv-def by auto
  then have [simp]: init-cls S = init-cls V and [simp]: learned-cls S = learned-cls V
    using rtrancp-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 cdclW-all-struct-inv-def cdclW-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 *rtrancpl-skip-backtrack-backtrack*

lemma *rtrancpl-skip-backtrack-backtrack-end*:

```

assumes
  skip: skip** S T and
  bt: backtrack S W and
  inv: cdclW-all-struct-inv S
shows backtrack T W
using assms
proof –
  have M-lev: cdclW-M-level-inv S
    using bt inv unfolding cdclW-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: cdclW-M-level-inv-decomp)
have cdclW-all-struct-inv T
  using mono-rtrancpl[of skip cdclW-restart] by (smt bj cdclW-bj.skip inv local.skip other
    rtrancpl-cdclW-all-struct-inv-inv)
then have [simp]: no-dup (trail T)
  unfolding cdclW-all-struct-inv-def cdclW-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 rtrancpl-skip-state-decomp(1)[OF skip] S by auto
have T: state-butlast T = (MT, 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 MT rtrancpl-skip-state-decomp[of S T] skip S by (auto simp: clauses-def)

```

have cdcl_W-all-struct-inv T

```

apply (rule rtrancp-cdclW-all-struct-inv-inv[OF - inv])
using bj cdclW-bj.skip local.skip other rtrancp-mono[of skip cdclW-restart] by blast
then have  $M_T \models_{as} CNot \ ?D$ 
unfolding cdclW-all-struct-inv-def cdclW-conflicting-def using  $T$  by auto
then have  $\forall L' \in \# ?D. \text{defined-lit } M_T \ L'$ 
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 cdclW-all-struct-inv-def cdclW-M-level-inv-def by auto
ultimately have undef-D:  $\forall L' \in \# ?D. \text{undefined-lit } MS \ L'$ 
unfolding  $M$  by (auto dest: defined-lit-no-dupD)
then have  $H: \bigwedge L'. L' \in \# D \implies \text{get-level } (trail \ S) \ L' = \text{get-level } M_T \ L'$ 
 $\text{get-level } (trail \ S) \ L = \text{get-level } M_T \ L$ 
unfolding  $M$  by (auto simp: lits-of-def)
have [simp]:  $\text{get-maximum-level } (trail \ S) \ D = \text{get-maximum-level } M_T \ D$ 
using  $\langle M_T \models_{as} CNot \ (\text{add-mset } L \ D) \rangle M \text{ nm undef-D}$  by (auto simp: get-maximum-level-skip-beginning)

have lev-l':  $\text{get-level } M_T \ L = \text{backtrack-lvl } S$ 
using lev-l nm by (auto simp: H)
have [simp]:  $\text{trail } (\text{reduce-trail-to } M1 \ T) = M1$ 
by (metis (no-types)  $M \ M_T$  append-assoc get-all-ann-decomposition-exists-prepend[OF decomp] nm
 $\text{reduce-trail-to-trail-tl-trail-decomp}$  beginning-not-decided-invert)
obtain  $c$  where  $c: \langle M_T = c \ @ \text{Decided } K \ \# \ M1 \rangle$ 
using nm decomp by (auto dest!: get-all-ann-decomposition-exists-prepend
 $\text{simp: } M_T[\text{symmetric}] \ M \text{ append-assoc}[\text{symmetric}]$ 
 $\text{simp del: append-assoc}$ 
 $\text{dest!}: \text{beginning-not-decided-invert}$ )
obtain  $c''$  where
 $c'': \langle (\text{Decided } K \ \# \ M1, c'') \in \text{set } (\text{get-all-ann-decomposition } (c \ @ \text{Decided } K \ \# \ M1)) \rangle$ 
using Decided-cons-in-get-all-ann-decomposition-append-Decided-cons[of  $K \ M1$ ] by blast
have  $W: W \sim \text{cons-trail } (\text{Propagated } L \ (\text{add-mset } L \ D')) \ (\text{reduce-trail-to } M1$ 
 $(\text{add-learned-cls } (\text{add-mset } L \ D') \ (\text{update-conflicting } None \ T)))$ 
apply (rule state-eq-trans[OF  $W$ ])
apply (rule cons-trail-state-eq)
apply (rule rtrancp-skip-backtrack-reduce-trail-to-state-eq[of - -  $K \ M1$ ])
using skip apply (simp; fail)
using  $c''$  by (auto simp:  $M_T[\text{symmetric}] \ M \ c$ )
have max-trail-S-MT-L-D':  $\langle \text{get-maximum-level } (trail \ S) \ ?D' = \text{get-maximum-level } M_T \ ?D' \rangle$ 
by (rule get-maximum-level-cong) (use  $H \ D-D'$  in auto)
then have lev-l-D':  $\text{get-level } M_T \ L = \text{get-maximum-level } M_T \ ?D'$ 
using lev-l-D  $H$  by auto
have  $i': i = \text{get-maximum-level } M_T \ D'$ 
unfolding  $i[\text{symmetric}]$ 
by (rule get-maximum-level-cong) (use  $H \ D-D'$  in auto)
have  $\text{Decided } K \ \# \ M1 \in \text{set } (\text{map fst } (\text{get-all-ann-decomposition } (trail \ S)))$ 
using Set.imageI[OF decomp, of fst] by auto
then have  $\text{Decided } K \ \# \ M1 \in \text{set } (\text{map fst } (\text{get-all-ann-decomposition } M_T))$ 
using fst-get-all-ann-decomposition-prepend-not-decided[OF nm] unfolding  $M$  by auto
then obtain  $M2'$  where  $\text{decomp}': (\text{Decided } K \ \# \ M1, M2') \in \text{set } (\text{get-all-ann-decomposition } M_T)$ 
by auto
moreover {
have undefined-lit  $MS \ K$ 
using  $\langle \text{no-dup } (trail \ S) \rangle \text{ decomp}'$  unfolding  $M \ M_T$ 
by (auto simp: lits-of-def defined-lit-map no-dup-def)
then have  $\text{get-level } (trail \ T) \ K = \text{get-level } (trail \ S) \ K$ 
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 rtrancpl-skip-or-resolve-rtrancpl-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 rtrancpl.simps skip-or-resolve.simps)+
qed

```

1.2.3 CDCL with Merging

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

lemma $rtrancpl\text{-}cdcl_W\text{-bj-rtrancpl-cdcl_W-restart}$:
 $cdcl_W\text{-bj}^{**}\ S\ T \Longrightarrow cdcl_W\text{-restart}^{**}\ S\ T$

```

using mono-rtrancp[of cdclW-bj cdclW-restart] by blast

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

lemma cdclW-merge-restart-conflicting-true-or-no-step:
  assumes cdclW-merge-restart S T
  shows conflicting T = None ∨ no-step cdclW-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 cdclW-restart U V and conflicting U = Some D
    then have False
      using n-s unfolding full-def
      by (induction rule: cdclW-restart-all-rules-induct)
        (auto dest!: cdclW-bj.intros elim: decideE propagateE conflictE forgetE restartE)
  }
  then show ?case by (cases conflicting U) fastforce+
qed (auto simp add: cdclW-rf.simps elim: propagateE decideE restartE forgetE)

inductive cdclW-merge :: 'st ⇒ 'st ⇒ bool where
  fw-propagate: propagate S S' ⇒ cdclW-merge S S' |
  fw-conflict: conflict S T ⇒ full cdclW-bj T U ⇒ cdclW-merge S U |
  fw-decide: decide S S' ⇒ cdclW-merge S S' |
  fw-forget: forget S S' ⇒ cdclW-merge S S'

lemma cdclW-merge-cdclW-merge-restart:
  cdclW-merge S T ⇒ cdclW-merge-restart S T
  by (meson cdclW-merge.cases cdclW-merge-restart.simps forget)

lemma rtrancp-cdclW-merge-trancp-cdclW-merge-restart:
  cdclW-merge** S T ⇒ cdclW-merge-restart** S T
  using rtrancp-mono[of cdclW-merge cdclW-merge-restart] cdclW-merge-cdclW-merge-restart by blast

lemma cdclW-merge-rtrancp-cdclW-restart:
  cdclW-merge S T ⇒ cdclW-restart** S T
  using cdclW-merge-cdclW-merge-restart cdclW-merge-restart-cdclW-restart by blast

lemma rtrancp-cdclW-merge-rtrancp-cdclW-restart:
  cdclW-merge** S T ⇒ cdclW-restart** S T
  using rtrancp-mono[of cdclW-merge cdclW-restart**] cdclW-merge-rtrancp-cdclW-restart by auto

lemma cdclW-all-struct-inv-trancp-cdclW-merge-trancp-cdclW-merge-cdclW-all-struct-inv:
  assumes
    inv: cdclW-all-struct-inv b

```

```

    cdclW-merge++ b a
  shows (λS T. cdclW-all-struct-inv S ∧ cdclW-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 cdclW-all-struct-inv c
    using tranclp-into-rtranclp[OF st] cdclW-merge-rtranclp-cdclW-restart assms(1)
    rtranclp-cdclW-all-struct-inv-inv rtranclp-mono[of cdclW-merge cdclW-restart**] by fastforce
  then have (λS T. cdclW-all-struct-inv S ∧ cdclW-merge S T)++ c d
    using fw by auto
  then show ?case using IH by auto
qed

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

lemma rtrancl-cdclW-conflicting-true-cdclW-merge-restart:
  assumes cdclW-restart** S V and inv: cdclW-M-level-inv S and conflicting S = None
  shows (cdclW-merge-restart** S V ∧ conflicting V = None)
    ∨ (∃ T U. cdclW-merge-restart** S T ∧ conflicting V ≠ None ∧ conflict T U ∧ cdclW-bj** U V)
  using assms
proof induction
  case base
  then show ?case by simp
next
  case (step U V) note st = this(1) and cdclW-restart = this(2) and IH = this(3)[OF this(4-)] and
    confl[simp] = this(5) and inv = this(4)
  from cdclW-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 cdclW-merge-restart.fw-r-propagate[of U V] by auto
next
  case conflict
  moreover have conflicting U = None and conflicting V ≠ 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 cdclW-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: cdclW-bj.simps)

```

```

then show ?thesis
proof cases
case s-or-r
have f1: cdclW-bj++ U V
  by (simp add: local.bj tranclp.r-into-trancl)
obtain T T' :: 'st where
  f2: cdclW-merge-restart** S U
    ∨ cdclW-merge-restart** S T ∧ conflicting U ≠ None
    ∧ conflict T T' ∧ cdclW-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
  cdclW-merge-restart** S T and
  conflicting U ≠ None and
  conflict T T' and
  cdclW-bj** T' U
  using IH confl by meson
have invU: cdclW-M-level-inv U
  using inv rtranclp-cdclW-restart-consistent-inv step.hyps(1) by blast
then have conflicting V = None
  using ⟨backtrack U V⟩ inv by (auto elim: backtrackE simp: cdclW-M-level-inv-decomp)
have full cdclW-bj T' V
  apply (rule rtranclp-fullI[of cdclW-bj T' U V])
  using ⟨cdclW-bj** T' U⟩ apply fast
  using ⟨backtrack U V⟩ backtrack-is-full1-cdclW-bj invU unfolding full1-def full-def
  by blast
then show ?thesis
  using cdclW-merge-restart.fw-r-conflict[of T T' V] ⟨conflict T T'⟩
    ⟨cdclW-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: cdclW-rf.simps elim: restartE forgetE)
ultimately show ?thesis using IH cdclW-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 \implies 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 cdclW-restart S S' using cdclW-restart.conflict by auto
  then have cdclW-M-level-inv S'
    using assms(2) cdclW-restart-consistent-inv by blast
  then obtain S'' where full cdclW-bj S' S''
    using cdclW-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 cdclW-restart.simps cdclW-merge-restart.simps cdclW-o.simps cdclW-bj.simps
  by (auto elim: skipE resolveE backtrackE conflictE decideE restartE)
qed

```

```

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

```

```

lemma rtranclp-cdclW-merge-restart-no-step-cdclW-bj:
  assumes
    cdclW-merge-restart** S T and
    conflicting S = None
  shows no-step cdclW-bj T
  using assms unfolding rtranclp-unfold
  apply (elim disjE)
  apply (force simp: cdclW-bj.simps cdclW-rf.simps elim!: rulesE)
  by (auto simp: tranclp-unfold-end simp: cdclW-merge-restart-no-step-cdclW-bj)

```

If $\text{conflicting } S \neq \text{None}$, 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-cdclW-restart-iff-full-cdclW-merge:
  assumes conf: conflicting S = None and lev: cdclW-M-level-inv S
  shows full cdclW-restart S V  $\longleftrightarrow$  full cdclW-merge-restart S V

```

```

proof
  assume full: full cdclW-merge-restart S V
  then have st: cdclW-restart** S V
    using rtranclp-mono[of cdclW-merge-restart cdclW-restart**] cdclW-merge-restart-cdclW-restart
    unfolding full-def by auto

  have n-s: no-step cdclW-merge-restart V
    using full unfolding full-def by auto
  have n-s-bj: no-step cdclW-bj V
    using rtranclp-cdclW-merge-restart-no-step-cdclW-bj confl full unfolding full-def by auto
  have  $\bigwedge S'. \text{conflict } V S' \implies \text{cdcl}_W\text{-M-level-inv } S'$ 
    using cdclW-restart.conflict cdclW-restart-consistent-inv lev rtranclp-cdclW-restart-consistent-inv st
  by blast
  then have  $\bigwedge S'. \text{conflict } V S' \implies \text{False}$ 

```

```

    using n-s n-s-bj cdclW-bj-exists-normal-form cdclW-merge-restart.simps by meson
  then have n-s-cdclW-restart: no-step cdclW-restart V
    using n-s n-s-bj by (auto simp: cdclW-restart.simps cdclW-o.simps cdclW-merge-restart.simps)
  then show full cdclW-restart S V using st unfolding full-def by auto
next
assume full: full cdclW-restart S V
have no-step cdclW-merge-restart V
  using full no-step-cdclW-restart-no-step-cdclW-merge-restart unfolding full-def by blast
moreover {
  consider
    (fw) cdclW-merge-restart** S V and conflicting V = None |
    (bj) T U where
      cdclW-merge-restart** S T and
      conflicting V ≠ None and
      conflict T U and
      cdclW-bj** U V
    using full rtrancl-cdclW-conflicting-true-cdclW-merge-restart confl lev unfolding full-def
    by meson
  then have cdclW-merge-restart** S V
  proof cases
    case fw
    then show ?thesis by fast
  next
    case (bj T U)
    have no-step cdclW-bj V
      using full unfolding full-def by (meson cdclW-o.bj other)
    then have full cdclW-bj U V
      using ⟨ cdclW-bj** U V ⟩ unfolding full-def by auto
    then have cdclW-merge-restart T V
      using ⟨ conflict T U ⟩ cdclW-merge-restart.fw-r-conflict by blast
    then show ?thesis using ⟨ cdclW-merge-restart** S T ⟩ by auto
  qed }
ultimately show full cdclW-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: cdclW-bj.simps elim!: rulesE)
consider
  (U) U = S'
  | (U') U' where cdclW-bj U U' and cdclW-bj** U' S'
  using st by (metis converse-rtrancplE)
then show ?case
proof cases
  case U
  then show ?thesis
    using n-s cdclW-o.bj local.bj other' by (meson r-into-rtrancpl)
next
  case U' note U' = this(1)
  have no-step conflict U no-step propagate U
    using U' by (fastforce simp: cdclW-bj.simps elim!: rulesE)+
  then have cdclW-stgy T U
    using n-s cdclW-stgy.simps local.bj cdclW-o.bj by meson
  then show ?thesis using IH by auto
qed
qed

lemma cdclW-s'-is-rtrancpl-cdclW-stgy:
  cdclW-s' S T  $\implies$  cdclW-stgy** S T
  by (induction rule: cdclW-s'.induct)
  (auto simp: full1-def
    dest: trancpl-into-rtrancpl rtrancpl-cdclW-bj-full1-cdclp-cdclW-stgy cdclW-stgy.intros)

lemma cdclW-stgy-cdclW-s'-no-step:
  assumes cdclW-stgy S U and cdclW-all-struct-inv S and no-step cdclW-bj U
  shows cdclW-s' S U
  using assms apply (cases rule: cdclW-stgy.cases)
  using bj'[of S U] by (auto intro: cdclW-s'.intros simp: cdclW-o.simps full1-def)

lemma rtrancpl-cdclW-stgy-connected-to-rtrancpl-cdclW-s':
  assumes cdclW-stgy** S U and inv: cdclW-M-level-inv S
  shows cdclW-s'** S U  $\vee$  ( $\exists T$ . cdclW-s'** S T  $\wedge$  cdclW-bj++ T U  $\wedge$  conflicting U  $\neq$  None)
  using assms(1)
proof induction
  case base
  then show ?case by simp
next
  case (step T V) note st = this(1) and o = this(2) and IH = this(3)
  from o show ?case
  proof cases
    case conflict'
    then have cdclW-s'l** S T
      using IH by (auto elim: conflictE)
    moreover have f2: cdclW-s'l** T V
      using cdclW-s'.conflict' conflict' by blast
    ultimately show ?thesis by auto
  next
    case propagate'
    then have cdclW-s'l** 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  $n-s = this(1,2)$  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 rtrancp.rtrancp-into-rtrancp)
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) rtrancp-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  $S-S' = this(1)$  and  $bj-T = this(2)$ 
moreover {
  have  $cdcl_W-M-level-inv T$ 
    using inv local.step(1) rtrancp-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  $S-S'$  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 confI-V step.IH by auto
  qed
qed
qed

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

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

lemma tranclp-cdclW-s'-tranclp-cdclW-restart:
  cdclW-s'++ S S'  $\implies$  cdclW-restart++ S S'
  apply (induct rule: tranclp.induct)
  using cdclW-s'-tranclp-cdclW-restart apply blast
  by (meson cdclW-s'-tranclp-cdclW-restart tranclp-trans)

lemma rtranclp-cdclW-s'-rtranclp-cdclW-restart:
  cdclW-s'** S S'  $\implies$  cdclW-restart** S S'
  using rtranclp-unfold[of cdclW-s' S S'] tranclp-cdclW-s'-tranclp-cdclW-restart[of S S'] by auto

lemma full-cdclW-stgy-iff-full-cdclW-s':
  assumes inv: cdclW-all-struct-inv S
  shows full cdclW-stgy S T  $\longleftrightarrow$  full cdclW-s' S T (is ?S  $\longleftrightarrow$  ?S')
proof
  assume ?S'
  then have cdclW-restart** S T

```

```

    using rtrancpl-cdclW-s'-rtrancpl-cdclW-restart[of S T] unfolding full-def by blast
  then have inv': cdclW-all-struct-inv T
    using rtrancpl-cdclW-all-struct-inv-inv inv by blast
  have cdclW-stgy** S T
    using ⟨?S'⟩ unfolding full-def
      using cdclW-s'-is-rtrancpl-cdclW-stgy rtrancpl-mono[of cdclW-s' cdclW-stgy**] by auto
  then show ?S
    using ⟨?S'⟩ inv' n-step-cdclW-stgy-iff-no-step-cdclW-restart-cl-cdclW-o unfolding full-def
    by blast
next
assume ?S
then have inv-T: cdclW-all-struct-inv T
  by (metis assms full-def rtrancpl-cdclW-all-struct-inv-inv
      rtrancpl-cdclW-stgy-rtrancpl-cdclW-restart)
consider
  (s') cdclW-s'** S T |
  (st) S' where cdclW-s'** S S' and cdclW-bj++ S' T and conflicting T ≠ None
  using rtrancpl-cdclW-stgy-connected-to-rtrancpl-cdclW-s'[of S T] inv ⟨?S⟩
  unfolding full-def cdclW-all-struct-inv-def
  by blast
then show ?S'
proof cases
  case s'
  then show ?thesis
    using ⟨full cdclW-stgy S T⟩ unfolding full-def
    by (metis inv-T n-step-cdclW-stgy-iff-no-step-cdclW-restart-cl-cdclW-o)
next
case (st S') note st = this(1) and bj = this(2) and confl = this(3)
  have no-step cdclW-bj T
    using ⟨?S⟩ cdclW-stgy.conflict' cdclW-stgy.intros(2) other' unfolding full-def by blast
  then have full1 cdclW-bj S' T
    using bj unfolding full1-def by blast
  then have cdclW-s' S' T
    using cdclW-s'.bj'[of S' T] by blast
  then have cdclW-s'** S T
    using st(1) by auto
  moreover have no-step cdclW-s' T
    using inv-T ⟨full cdclW-stgy S T⟩ n-step-cdclW-stgy-iff-no-step-cdclW-restart-cl-cdclW-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 $\mu_C :: nat \Rightarrow nat \Rightarrow nat\ list \Rightarrow nat$ **where**
 $\mu_C\ s\ b\ M \equiv (\sum i=0..<length\ M. M!i * b^\wedge (s + i - length\ M))$

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

lemma $\mu_C\ single[simp]$:
 $\mu_C\ s\ b\ [L] = L * b^\wedge (s - Suc\ 0)$
unfolding $\mu_C\text{-def}$ **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*:

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

lemma μ_C -*cons*:

$\mu_C \ s \ b \ (L \# \ M) = L * b \wedge (s - 1 - \text{length } M) + \mu_C \ s \ b \ M$

proof –

have $\mu_C \ s \ b \ (L \# \ M) = (\sum i=0..< \text{length } (L \# \ M). (L \# \ M)!i * b \wedge (s + i - \text{length } (L \# \ M)))$

unfolding μ_C -*def* **by** *blast*

also have $\dots = (\sum i=0..< 1. (L \# \ M)!i * b \wedge (s + i - \text{length } (L \# \ M)))$
 $+ (\sum i=1..< \text{length } (L \# \ M). (L \# \ M)!i * b \wedge (s + i - \text{length } (L \# \ M)))$

by (*rule sum.atLeastLessThan-concat[symmetric]*) *simp-all*

finally have $\mu_C \ s \ b \ (L \# \ M) = L * b \wedge (s - 1 - \text{length } M)$
 $+ (\sum i=1..< \text{length } (L \# \ M). (L \# \ M)!i * b \wedge (s + i - \text{length } (L \# \ M)))$

by *auto*

moreover {

have $(\sum i=1..< \text{length } (L \# \ M). (L \# \ M)!i * b \wedge (s + i - \text{length } (L \# \ M))) =$
 $(\sum i=0..< \text{length } M. (L \# \ M)!(\text{Suc } i) * b \wedge (s + (\text{Suc } i) - \text{length } (L \# \ M)))$

unfolding *length-Cons set-sum-atLeastLessThan-Suc* **by** *blast*

also have $\dots = (\sum i=0..< \text{length } M. M!i * b \wedge (s + i - \text{length } M))$

by *auto*

finally have $(\sum i=1..< \text{length } (L \# \ M). (L \# \ M)!i * b \wedge (s + i - \text{length } (L \# \ M))) = \mu_C \ s \ b \ M$

unfolding μ_C -*def* .

}

ultimately show *?thesis* **by** *presburger*

qed

lemma μ_C -*append*:

assumes $s \geq \text{length } (M @ M')$

shows $\mu_C \ s \ b \ (M @ M') = \mu_C \ (s - \text{length } M') \ b \ M + \mu_C \ s \ b \ M'$

proof –

have $\mu_C \ s \ b \ (M @ M') = (\sum i=0..< \text{length } (M @ M'). (M @ M')!i * b \wedge (s + i - \text{length } (M @ M')))$

unfolding μ_C -*def* **by** *blast*

moreover then have $\dots = (\sum i=0..< \text{length } M. (M @ M')!i * b \wedge (s + i - \text{length } (M @ M')))$
 $+ (\sum i=\text{length } M..< \text{length } (M @ M'). (M @ M')!i * b \wedge (s + i - \text{length } (M @ M')))$

by (*auto intro!: sum.atLeastLessThan-concat[symmetric]*)

moreover

have $\forall i \in \{0..< \text{length } M\}. (M @ M')!i * b \wedge (s + i - \text{length } (M @ M')) = M!i * b \wedge (s - \text{length } M' + i - \text{length } M)$

using $\langle s \geq \text{length } (M @ M') \rangle$ **by** (*auto simp add: nth-append ac-simps*)

then have $\mu_C \ (s - \text{length } M') \ b \ M = (\sum i=0..< \text{length } M. (M @ M')!i * b \wedge (s + i - \text{length } (M @ M')))$
 $(M @ M'))$

unfolding μ_C -*def* **by** *auto*

ultimately have $\mu_C \ s \ b \ (M @ M') = \mu_C \ (s - \text{length } M') \ b \ M$

$+ (\sum i=\text{length } M..< \text{length } (M @ M'). (M @ M')!i * b \wedge (s + i - \text{length } (M @ M')))$

by *auto*

moreover {

have $(\sum i=\text{length } M..< \text{length } (M @ M'). (M @ M')!i * b \wedge (s + i - \text{length } (M @ M')) =$
 $(\sum i=0..< \text{length } M'. M!i * b \wedge (s + i - \text{length } M'))$

unfolding *length-append set-sum-atLeastLessThan-add* **by** *auto*

then have $(\sum i=\text{length } M..< \text{length } (M @ M'). (M @ M')!i * b \wedge (s + i - \text{length } (M @ M')) = \mu_C \ s \ b \ M'$

unfolding μ_C -*def* .

}

ultimately show *?thesis* **by** *presburger*

qed

lemma μ_C -cons-non-empty-inf:
assumes M -ge-1: $\forall i \in \text{set } M. i \geq 1$ **and** $M: M \neq []$
shows $\mu_C s b M \geq b^\wedge (s - \text{length } M)$
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*: $0 \leq k \implies (k - 1) * (\sum_{i=0..<n. k^\wedge i} = k^\wedge n - (1::nat)$
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: $\forall i < \text{length } M. M!i < b$ **and**
 $s \geq \text{length } M$
shows $\mu_C s b M < b^\wedge s$

proof –

consider ($b1$) $b = 1 \mid (b) b > 1$ **using** $\langle b > 0 \rangle$ **by** (*cases* b) *auto*
then show *?thesis*

proof *cases*

case $b1$

then have $\forall i < \text{length } M. M!i = 0$ **using** M -le **by** *auto*
then have $\mu_C s b M = 0$ **unfolding** μ_C -def **by** *auto*
then show *?thesis* **using** $\langle b > 0 \rangle$ **by** *auto*

next

case b

have $\forall i \in \{0..<\text{length } M\}. M!i * b^\wedge (s + i - \text{length } M) \leq (b-1) * b^\wedge (s + i - \text{length } M)$
using M -le $\langle b > 1 \rangle$ **by** *auto*

then have $\mu_C s b M \leq (\sum_{i=0..<\text{length } M. (b-1) * b^\wedge (s + i - \text{length } M)})$
using $\langle M \neq [] \rangle \langle b > 0 \rangle$ **unfolding** μ_C -def **by** (*auto intro: sum-mono*)

also

have $\forall i \in \{0..<\text{length } M\}. (b-1) * b^\wedge (s + i - \text{length } M) = (b-1) * b^\wedge i * b^\wedge (s - \text{length } M)$
by (*metis* *Nat.add-diff-assoc2* *add.commute* *assms(4)* *mult.assoc* *power-add*)

then have $(\sum_{i=0..<\text{length } M. (b-1) * b^\wedge (s + i - \text{length } M))$
 $= (\sum_{i=0..<\text{length } M. (b-1) * b^\wedge i * b^\wedge (s - \text{length } M))$
by (*auto simp add: ac-simps*)

also have $\dots = (\sum_{i=0..<\text{length } M. b^\wedge i) * b^\wedge (s - \text{length } M) * (b-1)$
by (*simp add: sum-distrib-right sum-distrib-left ac-simps*)

finally have $\mu_C s b M \leq (\sum_{i=0..<\text{length } M. b^\wedge i) * (b-1) * b^\wedge (s - \text{length } M)$
by (*simp add: ac-simps*)

also

have $(\sum_{i=0..<\text{length } M. b^\wedge i) * (b-1) = b^\wedge (\text{length } M) - 1$
using *sum-of-powers*[*of* b $\text{length } M$] $\langle b > 1 \rangle$
by (*auto simp add: ac-simps*)

finally have $\mu_C s b M \leq (b^\wedge (\text{length } M) - 1) * b^\wedge (s - \text{length } M)$
by *auto*

also have $\dots < b^\wedge (\text{length } M) * b^\wedge (s - \text{length } M)$

```

    using <b>1> by auto
  also have ... = b ^ s
    by (metis assms(4) le-add-diff-inverse power-add)
  finally show ?thesis unfolding  $\mu_C$ -def by (auto simp add: ac-simps)
qed
qed

```

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

```

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

```

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

```

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

```

```

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

```


proof –

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

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

lemma *finite- νNOT* [*simp*]:
finite ($\nu NOT \ s \ base$)

proof –

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

lemma *acyclic- νNOT* : *acyclic* ($\nu NOT \ s \ base$)
apply (*rule acyclic-subset*[*of lenlex less-than $\nu NOT \ s \ base$*])
apply (*rule wf-acyclic*)
by (*auto simp: νNOT -def*)

lemma *wf- νNOT* : *wf* ($\nu NOT \ s \ base$)
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*:
 $\langle \text{atms-of } \{\# - \text{lit-of } x. x \in \# A \# \} = \text{atm-of } ' (\text{lit-of } ' (\text{set-mset } A)) \rangle$
unfolding *atms-of-def* **by** (*auto simp add: Fun.image-comp*)

lemma *atms-of-ms-single-image-atm-of-lit-of*:
 $\langle \text{atms-of-ms } (\text{unmark-s } A) = \text{atm-of } ' (\text{lit-of } ' A) \rangle$
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
 $\text{trail} :: \langle 'st \Rightarrow ('v, \text{unit}) \text{ ann-lits} \rangle$ **and**
 $\text{clauses}_{NOT} :: \langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**
 $\text{prepend-trail} :: \langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
 $\text{tl-trail} :: \langle 'st \Rightarrow 'st \rangle$ **and**
 $\text{add-cl}_{NOT} :: \langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
 $\text{remove-cl}_{NOT} :: \langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$
begin
abbreviation $\text{state}_{NOT} :: \langle 'st \Rightarrow ('v, \text{unit}) \text{ ann-lit list} \times 'v \text{ clauses} \rangle$ **where**
 $\langle \text{state}_{NOT} S \equiv (\text{trail } S, \text{clauses}_{NOT} S) \rangle$
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
 $\text{trail } \text{clauses}_{NOT} \text{ prepend-trail } \text{tl-trail } \text{add-cl}_{NOT} \text{ remove-cl}_{NOT}$ — related to the state
for
 $\text{trail} :: \langle 'st \Rightarrow ('v, \text{unit}) \text{ ann-lits} \rangle$ **and**
 $\text{clauses}_{NOT} :: \langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**
 $\text{prepend-trail} :: \langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
 $\text{tl-trail} :: \langle 'st \Rightarrow 'st \rangle$ **and**
 $\text{add-cl}_{NOT} :: \langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
 $\text{remove-cl}_{NOT} :: \langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ +
assumes
 $\text{prepend-trail}_{NOT}$:
 $\langle \text{state}_{NOT} (\text{prepend-trail } L \text{ st}) = (L \# \text{trail st}, \text{clauses}_{NOT} \text{ st}) \rangle$ **and**
 tl-trail_{NOT} :
 $\langle \text{state}_{NOT} (\text{tl-trail } st) = (\text{tl } (\text{trail st}), \text{clauses}_{NOT} \text{ st}) \rangle$ **and**
 add-cl_{NOT} :
 $\langle \text{state}_{NOT} (\text{add-cl}_{NOT} C \text{ st}) = (\text{trail st}, \text{add-mset } C (\text{clauses}_{NOT} \text{ st})) \rangle$ **and**
 remove-cl_{NOT} :
 $\langle \text{state}_{NOT} (\text{remove-cl}_{NOT} C \text{ st}) = (\text{trail st}, \text{removeAll-mset } C (\text{clauses}_{NOT} \text{ st})) \rangle$
begin
lemma
 $\text{trail-prepend-trail[simp]}$:
 $\langle \text{trail } (\text{prepend-trail } L \text{ st}) = L \# \text{trail st} \rangle$
and
 $\text{trail-tl-trail}_{NOT}[\text{simp}]$: $\langle \text{trail } (\text{tl-trail } st) = \text{tl } (\text{trail st}) \rangle$ **and**
 $\text{trail-add-cl}_{NOT}[\text{simp}]$: $\langle \text{trail } (\text{add-cl}_{NOT} C \text{ st}) = \text{trail st} \rangle$ **and**
 $\text{trail-remove-cl}_{NOT}[\text{simp}]$: $\langle \text{trail } (\text{remove-cl}_{NOT} C \text{ st}) = \text{trail st} \rangle$ **and**

 $\text{clauses-prepend-trail[simp]}$:
 $\langle \text{clauses}_{NOT} (\text{prepend-trail } L \text{ st}) = \text{clauses}_{NOT} \text{ st} \rangle$

and
clauses-tl-trail[simp]: $\langle \text{clauses}_{NOT} (\text{tl-trail } st) = \text{clauses}_{NOT} st \rangle$ **and**
*clauses-add-cls*_{NOT}[simp]:
 $\langle \text{clauses}_{NOT} (\text{add-cls}_{NOT} C st) = \text{add-mset } C (\text{clauses}_{NOT} st) \rangle$ **and**
*clauses-remove-cls*_{NOT}[simp]:
 $\langle \text{clauses}_{NOT} (\text{remove-cls}_{NOT} C st) = \text{removeAll-mset } C (\text{clauses}_{NOT} st) \rangle$
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 $\langle \text{state}_{NOT} st \rangle$; auto)+

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

function *reduce-trail-to*_{NOT} :: $\langle 'a \text{ list} \Rightarrow 'st \Rightarrow 'st \rangle$ **where**
 $\langle \text{reduce-trail-to}_{NOT} F S =$
 $(\text{if } \text{length } (\text{trail } S) = \text{length } F \vee \text{trail } S = [] \text{ then } S \text{ else } \text{reduce-trail-to}_{NOT} F (\text{tl-trail } S)) \rangle$
by fast+
termination by (relation $\langle \text{measure } (\lambda(-, S). \text{length } (\text{trail } S)) \rangle$) 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]: $\langle \text{trail } S = [] \implies \text{reduce-trail-to}_{NOT} F S = S \rangle$ **and**
*reduce-trail-to*_{NOT}-eq-length[simp]: $\langle \text{length } (\text{trail } S) = \text{length } F \implies \text{reduce-trail-to}_{NOT} F S = S \rangle$
by (auto simp: *reduce-trail-to*_{NOT}.simps)

lemma *reduce-trail-to*_{NOT}-length-ne[simp]:

$\langle \text{length } (\text{trail } S) \neq \text{length } F \implies \text{trail } S \neq [] \implies$
 $\text{reduce-trail-to}_{NOT} F S = \text{reduce-trail-to}_{NOT} F (\text{tl-trail } S) \rangle$
by (auto simp: *reduce-trail-to*_{NOT}.simps)

lemma *trail-reduce-trail-to*_{NOT}-length-le:

assumes $\langle \text{length } F > \text{length } (\text{trail } S) \rangle$
shows $\langle \text{trail } (\text{reduce-trail-to}_{NOT} F S) = [] \rangle$
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]:

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

lemma *clauses-reduce-trail-to*_{NOT}-Nil:

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

lemma *trail-reduce-trail-to*_{NOT}-drop:

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

```

apply (subgoal-tac  $\langle \text{Suc } (\text{length list}) - \text{length } F = \text{Suc } (\text{length list} - \text{length } F) \rangle$ )
apply simp
apply simp
done

```

```

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

```

```

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

```

```

lemma trail-eq-reduce-trail-toNOT-eq:
   $\langle \text{trail } S = \text{trail } T \implies \text{trail } (\text{reduce-trail-to}_{\text{NOT}} F S) = \text{trail } (\text{reduce-trail-to}_{\text{NOT}} F T) \rangle$ 
  apply (induction F S arbitrary: T rule: reduce-trail-toNOT.induct)
  by (metis trail-tl-trailNOT reduce-trail-toNOT-eq-length reduce-trail-toNOT-length-ne
    reduce-trail-toNOT-Nil)

```

```

lemma trail-reduce-trail-toNOT-add-clNOT[simp]:
   $\langle \text{no-dup } (\text{trail } S) \implies$ 
     $\text{trail } (\text{reduce-trail-to}_{\text{NOT}} F (\text{add-cl}_{\text{NOT}} C S)) = \text{trail } (\text{reduce-trail-to}_{\text{NOT}} F S) \rangle$ 
  by (rule trail-eq-reduce-trail-toNOT-eq) simp

```

```

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

```

```

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

```

```

abbreviation trail-weight where
   $\langle \text{trail-weight } S \equiv \text{map } ((\lambda l. 1 + \text{length } l) \circ \text{snd}) (\text{get-all-ann-decomposition } (\text{trail } S)) \rangle$ 

```

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-eqNOT ::  $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$  (infix  $\sim$  50) where
   $\langle S \sim T \longleftrightarrow \text{trail } S = \text{trail } T \wedge \text{clauses}_{\text{NOT}} S = \text{clauses}_{\text{NOT}} T \rangle$ 

```

```

lemma state-eqNOT-ref[intro, simp]:
   $\langle S \sim S \rangle$ 
  unfolding state-eqNOT-def by auto

```

```

lemma state-eqNOT-sym:
   $\langle S \sim T \longleftrightarrow T \sim S \rangle$ 
  unfolding state-eqNOT-def by auto

```

```

lemma state-eqNOT-trans:

```

$\langle S \sim T \implies T \sim U \implies S \sim U \rangle$
unfolding *state-eq_{NOT}-def* **by** *auto*

lemma

shows

state-eq_{NOT}-trail: $\langle S \sim T \implies \text{trail } S = \text{trail } T \rangle$ **and**

state-eq_{NOT}-clauses: $\langle S \sim T \implies \text{clauses}_{\text{NOT}} S = \text{clauses}_{\text{NOT}} T \rangle$

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*: $\langle S \sim T \rangle$

shows $\langle \text{reduce-trail-to}_{\text{NOT}} F S \sim \text{reduce-trail-to}_{\text{NOT}} F T \rangle$

proof –

have $\langle \text{clauses}_{\text{NOT}} (\text{reduce-trail-to}_{\text{NOT}} F S) = \text{clauses}_{\text{NOT}} (\text{reduce-trail-to}_{\text{NOT}} F T) \rangle$

using *ST* **by** *auto*

moreover have $\langle \text{trail } (\text{reduce-trail-to}_{\text{NOT}} F S) = \text{trail } (\text{reduce-trail-to}_{\text{NOT}} F T) \rangle$

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 *dp_{ll}-state*.

Definition of the Transitions

Each possible is in its own locale.

locale *propagate-ops* =

dp_{ll}-state *trail* *clauses_{NOT}* *prepend-trail* *tl-trail* *add-cl_s_{NOT}* *remove-cl_s_{NOT}*

for

trail :: $\langle 'st \Rightarrow ('v, \text{unit}) \text{ ann-lits} \rangle$ **and**

clauses_{NOT} :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**

prepend-trail :: $\langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

tl-trail :: $\langle 'st \Rightarrow 'st \rangle$ **and**

add-cl_s_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

remove-cl_s_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ +

fixes

propagate-conds :: $\langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$

begin

inductive *propagate_{NOT}* :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **where**

propagate_{NOT}[intro]: $\langle \text{add-mset } L \ C \in \# \text{ clauses}_{\text{NOT}} S \implies \text{trail } S \models_{\text{as}} C \text{Not } C$

$\implies \text{undefined-lit } (\text{trail } S) \ L$

$\implies \text{propagate-conds } (\text{Propagated } L \ ()) \ S \ T$

$\implies T \sim \text{prepend-trail } (\text{Propagated } L \ ()) \ S$

$\implies \text{propagate}_{\text{NOT}} S \ T \rangle$

inductive-cases *propagate_{NOT}E[elim]*: $\langle \text{propagate}_{\text{NOT}} S \ T \rangle$

end

locale *decide-ops* =

dp_{ll}-state *trail* *clauses_{NOT}* *prepend-trail* *tl-trail* *add-cl_s_{NOT}* *remove-cl_s_{NOT}*

for

trail :: $\langle 'st \Rightarrow ('v, \text{unit}) \text{ ann-lits} \rangle$ **and**

clauses_{NOT} :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**

prepend-trail :: $\langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

```

    tl-trail :: ⟨'st ⇒ 'st⟩ and
    add-clNOT :: ⟨'v clause ⇒ 'st ⇒ 'st⟩ and
    remove-clNOT :: ⟨'v clause ⇒ 'st ⇒ 'st⟩ +
fixes
    decide-conds :: ⟨'st ⇒ 'st ⇒ bool⟩
begin
inductive decideNOT :: ⟨'st ⇒ 'st ⇒ bool⟩ where
decideNOT[intro]:
    ⟨undefined-lit (trail S) L ⇒
    atm-of L ∈ atms-of-mm (clausesNOT S) ⇒
    T ~ prepend-trail (Decided L) S ⇒
    decide-conds S T ⇒
    decideNOT S T⟩

inductive-cases decideNOTE[elim]: ⟨decideNOT S S'⟩
end

locale backjumping-ops =
    dpll-state trail clausesNOT prepend-trail tl-trail add-clNOT remove-clNOT
for
    trail :: ⟨'st ⇒ ('v, unit) ann-lits⟩ and
    clausesNOT :: ⟨'st ⇒ 'v clauses⟩ and
    prepend-trail :: ⟨('v, unit) ann-lit ⇒ 'st ⇒ 'st⟩ and
    tl-trail :: ⟨'st ⇒ 'st⟩ and
    add-clNOT :: ⟨'v clause ⇒ 'st ⇒ 'st⟩ and
    remove-clNOT :: ⟨'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-toNOT F S)
    ⇒ C ∈ # clausesNOT S
    ⇒ trail S ⊨as CNot C
    ⇒ undefined-lit F L
    ⇒ atm-of L ∈ atms-of-mm (clausesNOT S) ∪ atm-of ' (lits-of-l (trail S))
    ⇒ clausesNOT 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 $\text{atm-of } L \in \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \cup \text{atm-of ' } (\text{lits-of-l } (\text{trail } S))$ is not implied by the condition $\text{clauses}_{\text{NOT}} S \models_{\text{pm}} \text{add-mset } L C'$ (no negation).

end

2.2.3 DPLL with Backjumping

```

locale dpll-with-backjumping-ops =
    propagate-ops trail clausesNOT prepend-trail tl-trail add-clNOT remove-clNOT propagate-conds +
    decide-ops trail clausesNOT prepend-trail tl-trail add-clNOT remove-clNOT decide-conds +
    backjumping-ops trail clausesNOT prepend-trail tl-trail add-clNOT remove-clNOT backjump-conds
for
    trail :: ⟨'st ⇒ ('v, unit) ann-lits⟩ and
    clausesNOT :: ⟨'st ⇒ 'v clauses⟩ and

```

```

prepend-trail :: ⟨('v, unit) ann-lit ⇒ 'st ⇒ 'st⟩ and
tl-trail :: ⟨'st ⇒ 'st⟩ and
add-clsNOT :: ⟨'v clause ⇒ 'st ⇒ 'st⟩ and
remove-clsNOT :: ⟨'v clause ⇒ 'st ⇒ 'st⟩ and
inv :: ⟨'st ⇒ bool⟩ and
decide-cons :: ⟨'st ⇒ 'st ⇒ bool⟩ and
backjump-cons :: ⟨'v clause ⇒ 'v clause ⇒ 'v literal ⇒ 'st ⇒ 'st ⇒ bool⟩ and
propagate-cons :: ⟨('v, unit) ann-lit ⇒ 'st ⇒ 'st ⇒ bool⟩ +
assumes
  bj-can-jump:
    ⟨∧ S C F' K F L.
      inv S ⇒
      trail S = F' @ Decided K # F ⇒
      C ∈ # clausesNOT S ⇒
      trail S ⊨as CNot C ⇒
      undefined-lit F L ⇒
      atm-of L ∈ atms-of-mm (clausesNOT S) ∪ atm-of ' (lits-of-l (F' @ Decided K # F)) ⇒
      clausesNOT S ⊨pm add-mset L C' ⇒
      F ⊨as CNot C' ⇒
      ¬no-step backjump S⟩ and
  can-propagate-or-decide-or-backjump:
    ⟨atm-of L ∈ atms-of-mm (clausesNOT S) ⇒
    undefined-lit (trail S) L ⇒
    satisfiable (set-mset (clausesNOT S)) ⇒
    inv S ⇒
    no-dup (trail S) ⇒
    ∃ T. decideNOT S T ∨ propagateNOT S T ∨ backjump S T⟩
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 $atm\text{-}of\ L \in atm\text{-}of\ ' \text{ lits-of-l } (F' @ Decided\ K \# F)$ is important, otherwise you are not sure that you can backtrack.

Definition

We define `dpll` with backjumping:

inductive `dpll-bj` :: ⟨'st ⇒ 'st ⇒ bool⟩ **for** `S` :: 'st **where**

`bj-decideNOT`: ⟨`decideNOT S S' ⇒ dpll-bj S S'`⟩ |

`bj-propagateNOT`: ⟨`propagateNOT S S' ⇒ dpll-bj S S'`⟩ |

`bj-backjump`: ⟨`backjump S S' ⇒ dpll-bj S S'`⟩

lemmas `dpll-bj-induct` = `dpll-bj.induct[split-format(complete)]`

thm `dpll-bj-induct[OF dpll-with-backjumping-ops-axioms]`

lemma `dpll-bj-all-induct[consumes 2, case-names decideNOT propagateNOT backjump]`:

fixes `S T` :: ⟨'st⟩

assumes

⟨`dpll-bj S T`⟩ **and**

⟨`inv S`⟩

⟨ $\bigwedge L\ T. \text{undefined-lit } (trail\ S)\ L \implies atm\text{-}of\ L \in atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \implies T \sim \text{prepend-trail } (Decided\ L)\ S \implies P\ S\ T$ ⟩ **and**

⟨ $\bigwedge C\ L\ T. \text{add-mset } L\ C \in \# \text{ clauses}_{NOT}\ S \implies trail\ S \models_{as} CNot\ C \implies \text{undefined-lit } (trail\ S)\ L \implies T \sim \text{prepend-trail } (Propagated\ L\ ())\ S$ ⟩

$\Rightarrow P \ S \ T$ and
 $\langle \bigwedge C \ F' \ K \ F \ L \ C' \ T. \ C \in \# \text{ clauses}_{NOT} \ S \Rightarrow F' @ \text{Decided } K \ \# \ F \models_{as} CNot \ C \rangle$
 $\Rightarrow \text{trail } S = F' @ \text{Decided } K \ \# \ F$
 $\Rightarrow \text{undefined-lit } F \ L$
 $\Rightarrow \text{atm-of } L \in \text{atms-of-mm } (\text{clauses}_{NOT} \ S) \cup \text{atm-of } ' \ (\text{lits-of-l } (F' @ \text{Decided } K \ \# \ F))$
 $\Rightarrow \text{clauses}_{NOT} \ S \models_{pm} \text{add-mset } L \ C'$
 $\Rightarrow F \models_{as} CNot \ C'$
 $\Rightarrow T \sim \text{prepend-trail } (\text{Propagated } L \ ()) \ (\text{reduce-trail-to}_{NOT} \ F \ S)$
 $\Rightarrow P \ S \ T$
shows $\langle P \ S \ T \rangle$
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* *backjump* E) **using** *assms*(5) $\langle \text{inv } S \rangle$ **by** *simp*

Basic properties

First, some better suited induction principle lemma *dpll-bj-clauses*:

assumes $\langle \text{dpll-bj } S \ T \rangle$ **and** $\langle \text{inv } S \rangle$
shows $\langle \text{clauses}_{NOT} \ S = \text{clauses}_{NOT} \ T \rangle$
using *assms* **by** (*induction rule*: *dpll-bj-all-induct*) *auto*

No duplicates in the trail lemma *dpll-bj-no-dup*:

assumes $\langle \text{dpll-bj } S \ T \rangle$ **and** $\langle \text{inv } S \rangle$
and $\langle \text{no-dup } (\text{trail } S) \rangle$
shows $\langle \text{no-dup } (\text{trail } T) \rangle$
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 $\langle \text{dpll-bj } S \ T \rangle$ **and** $\langle \text{inv } S \rangle$
shows $\langle I \models_{sm} \text{clauses}_{NOT} \ S \longleftrightarrow I \models_{sm} \text{clauses}_{NOT} \ T \rangle$
using *assms* **by** (*induction rule*: *dpll-bj-all-induct*) *auto*

Clauses lemma *dpll-bj-atms-of-ms-clauses-inv*:

assumes
 $\langle \text{dpll-bj } S \ T \rangle$ **and**
 $\langle \text{inv } S \rangle$
shows $\langle \text{atms-of-mm } (\text{clauses}_{NOT} \ S) = \text{atms-of-mm } (\text{clauses}_{NOT} \ T) \rangle$
using *assms* **by** (*induction rule*: *dpll-bj-all-induct*) *auto*

lemma *dpll-bj-atms-in-trail*:

assumes
 $\langle \text{dpll-bj } S \ T \rangle$ **and**
 $\langle \text{inv } S \rangle$ **and**
 $\langle \text{atm-of } ' \ (\text{lits-of-l } (\text{trail } S)) \subseteq \text{atms-of-mm } (\text{clauses}_{NOT} \ S) \rangle$
shows $\langle \text{atm-of } ' \ (\text{lits-of-l } (\text{trail } T)) \subseteq \text{atms-of-mm } (\text{clauses}_{NOT} \ S) \rangle$
using *assms* **by** (*induction rule*: *dpll-bj-all-induct*)
(auto simp: in-plus-implys-atm-of-on-atms-of-ms reduce-trail-to_{NOT}-skip-beginning)

lemma *dpll-bj-atms-in-trail-in-set*:

assumes $\langle \text{dpll-bj } S \ T \rangle$ **and**
 $\langle \text{inv } S \rangle$ **and**
 $\langle \text{atms-of-mm } (\text{clauses}_{NOT} \ S) \subseteq A \rangle$ **and**

$\langle \text{atm-of } \langle \text{ lits-of-l } (\text{trail } S) \rangle \subseteq A \rangle$
shows $\langle \text{atm-of } \langle \text{ lits-of-l } (\text{trail } T) \rangle \subseteq A \rangle$
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
 $\langle \text{dpll-bj } S \ T \rangle$ **and**
 $\text{inv: } \langle \text{inv } S \rangle$ **and**
 $\text{decomp: } \langle \text{all-decomposition-implies-m } (\text{clauses}_{\text{NOT}} S) (\text{get-all-ann-decomposition } (\text{trail } S)) \rangle$
shows $\langle \text{all-decomposition-implies-m } (\text{clauses}_{\text{NOT}} T) (\text{get-all-ann-decomposition } (\text{trail } T)) \rangle$
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 *y₀: 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] y₀ 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 $\langle \text{unmark-l } a \cup \text{set-mset } ?N \models_p \{ \#L \# \} \rangle$ (**is** $\langle ?I \models_p \cdot \rangle$)
proof (*rule true-clss-clss-plus-CNot*)
show $\langle ?I \models_p \text{add-mset } L \ C \rangle$
using *propa propagate_{NOT}.prems* **by** (*auto dest!: true-clss-clss-in-imp-true-clss-clss*)
next
have $\langle \text{unmark-l } ?M' \models_{ps} \text{CNot } C \rangle$
using $\langle \text{trail } S \models_{as} \text{CNot } C \rangle$ *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 $\langle \text{unmark-l } a \cup \text{set-mset } (\text{clauses}_{\text{NOT}} S) \models_{ps} \text{unmark-l } a \cup \text{unmark-l } (\text{tl } y) \rangle$
using *a-Un-N-M true-clss-clss-def* **by** *blast*
then show $\langle \text{unmark-l } a \cup \text{set-mset } (\text{clauses}_{\text{NOT}} S) \models_{ps} \text{CNot } C \rangle$
using *a1* **by** (*meson true-clss-clss-left-right true-clss-clss-union-and true-clss-clss-union-l-r*)
qed
ultimately have $\langle \text{unmark-l } a \cup \text{set-mset } ?N \models_{ps} \text{unmark-l } ?M' \rangle$
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 (clausesNOT 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 (clausesNOT 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 (clausesNOT 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 (clausesNOT S) ⊢p add-mset L D⟩
  by (simp add: N-C)
have ⟨unmark-l a ∪ set-mset (clausesNOT S) ⊢p {#L#}⟩
  using a-N-D-L a-N-CNot-D by (blast intro: true-clss-clss-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 $\langle \text{get-all-ann-decomposition } F' \rangle$) *auto*
then show $?case$
using IH by (*simp add: F'*)
qed

lemma *length-get-all-ann-decomposition-length*:
 $\langle \text{length } (\text{get-all-ann-decomposition } M) \leq 1 + \text{length } M \rangle$
by (*induction M rule: ann-lit-list-induct*) *auto*

lemma *length-in-get-all-ann-decomposition-bounded*:
assumes $i: i \in \text{set } (\text{trail-weight } S)$
shows $\langle i \leq \text{Suc } (\text{length } (\text{trail } S)) \rangle$
proof –
obtain $a\ b$ **where**
 $\langle (a, b) \in \text{set } (\text{get-all-ann-decomposition } (\text{trail } S)) \rangle$ **and**
 $ib: \langle i = \text{Suc } (\text{length } b) \rangle$
using i **by** *auto*
then obtain c **where** $\langle \text{trail } S = c @ b @ a \rangle$
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:

- $1 + \text{card } (\text{atms-of-ms } A)$: $\text{card } (\text{atms-of-ms } A)$ 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.
- $2 + \text{card } (\text{atms-of-ms } A)$: $\text{card } (\text{atms-of-ms } A)$ 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* :: $\langle 'b \text{ clause set} \Rightarrow 'a \text{ list} \Rightarrow \text{nat} \rangle$ **where**
 $\langle \text{unassigned-lit } N\ M \equiv \text{card } (\text{atms-of-ms } N) - \text{length } M \rangle$

lemma *dp11-bj-trail-mes-increasing-prop*:
fixes $M :: \langle ('v, \text{unit}) \text{ ann-lits} \rangle$ **and** $N :: \langle 'v \text{ clauses} \rangle$
assumes
 $\langle \text{dp11-bj } S\ T \rangle$ **and**
 $\langle \text{inv } S \rangle$ **and**
 $NA: \langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $MA: \langle \text{atm-of } ' \text{ lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $n-d: \langle \text{no-dup } (\text{trail } S) \rangle$ **and**
 $\text{finite}: \langle \text{finite } A \rangle$
shows $\langle \mu_C (1 + \text{card } (\text{atms-of-ms } A)) (2 + \text{card } (\text{atms-of-ms } A)) (\text{trail-weight } T) \rangle$
 $\quad > \mu_C (1 + \text{card } (\text{atms-of-ms } A)) (2 + \text{card } (\text{atms-of-ms } A)) (\text{trail-weight } S) \rangle$
using *assms(1,2)*
proof (*induction rule: dp11-bj-all-induct*)
case (*propagate*_{NOT} $C\ L\ T$) **note** $CLN = \text{this}(1)$ **and** $MC = \text{this}(2)$ **and** $\text{undef-L} = \text{this}(3)$ **and** $T = \text{this}(4)$
have $\text{incl}: \langle \text{atm-of } ' \text{ lits-of-l } (\text{Propagated } L\ ()) \# \text{trail } S \rangle \subseteq \text{atms-of-ms } A$
using *propagate*_{NOT} *dp11-bj-atms-in-trail-in-set* *bj-propagate*_{NOT} $NA\ MA\ CLN$
by (*auto simp: in-plus-implies-atm-of-on-atms-of-ms*)

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

then have $\langle \text{length } (\text{Propagated } L \ () \# \text{ trail } S) \leq \text{card } (\text{atms-of-ms } A) \rangle$
using *incl finite unfolding no-dup-length-eq-card-atm-of-lits-of-l*[*OF no-dup*]
by $(\text{simp add: card-mono})$
then have *latm*: $\langle \text{unassigned-lit } A \ b = \text{Suc } (\text{unassigned-lit } A \ (\text{Propagated } L \ () \# b)) \rangle$
using *b-le-M* **by** *auto*
then show ?*case* **using** *T undef-L* **by** $(\text{auto simp: latm } M \ \mu_C\text{-cons})$
next
case $(\text{decide}_{\text{NOT}} L)$ **note** *undef-L* = *this*(1) **and** *MC* = *this*(2) **and** *T* = *this*(3)
have *incl*: $\langle \text{atm-of ' lits-of-l } (\text{Decided } L \# (\text{trail } S)) \subseteq \text{atms-of-ms } A \rangle$
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*: $\langle \text{no-dup } (\text{Decided } L \# (\text{trail } S)) \rangle$
using *defined-lit-map n-d undef-L* **by** *auto*
obtain *a b l* **where** *M*: $\langle \text{get-all-ann-decomposition } (\text{trail } S) = (a, b) \# l \rangle$
by $(\text{cases } \langle \text{get-all-ann-decomposition } (\text{trail } S) \rangle)$ *auto*

then have $\langle \text{length } (\text{Decided } L \# (\text{trail } S)) \leq \text{card } (\text{atms-of-ms } A) \rangle$
using *incl finite unfolding no-dup-length-eq-card-atm-of-lits-of-l*[*OF no-dup*]
by $(\text{simp add: card-mono})$
show ?*case* **using** *T undef-L* **by** $(\text{simp add: } \mu_C\text{-cons})$
next
case $(\text{backjump } C \ F' \ K \ F \ L \ C' \ T)$ **note** *undef-L* = *this*(4) **and** *MC* = *this*(1) **and** *tr-S* = *this*(3)
and
L = *this*(5) **and** *T* = *this*(8)
have *incl*: $\langle \text{atm-of ' lits-of-l } (\text{Propagated } L \ () \# F) \subseteq \text{atms-of-ms } A \rangle$
using *dpll-bj-atms-in-trail-in-set NA MA L* **by** (auto simp: tr-S)

have *no-dup*: $\langle \text{no-dup } (\text{Propagated } L \ () \# F) \rangle$
using *defined-lit-map n-d undef-L tr-S* **by** $(\text{auto dest: no-dup-appendD})$
obtain *a b l* **where** *M*: $\langle \text{get-all-ann-decomposition } (\text{trail } S) = (a, b) \# l \rangle$
by $(\text{cases } \langle \text{get-all-ann-decomposition } (\text{trail } S) \rangle)$ *auto*
have *b-le-M*: $\langle \text{length } b \leq \text{length } (\text{trail } S) \rangle$
using *get-all-ann-decomposition-decomp*[of $\langle \text{trail } S \rangle$] **by** $(\text{simp add: } M)$
have *fin-atms-A*: $\langle \text{finite } (\text{atms-of-ms } A) \rangle$ **using** *finite* **by** *simp*

then have *F-le-A*: $\langle \text{length } (\text{Propagated } L \ () \# F) \leq \text{card } (\text{atms-of-ms } A) \rangle$
using *incl finite unfolding no-dup-length-eq-card-atm-of-lits-of-l*[*OF no-dup*]
by $(\text{simp add: card-mono})$
have *tr-S-le-A*: $\langle \text{length } (\text{trail } S) \leq \text{card } (\text{atms-of-ms } A) \rangle$
using *n-d MA* **by** $(\text{metis fin-atms-A card-mono no-dup-length-eq-card-atm-of-lits-of-l})$
obtain *a b l* **where** *F*: $\langle \text{get-all-ann-decomposition } F = (a, b) \# l \rangle$
by $(\text{cases } \langle \text{get-all-ann-decomposition } F \rangle)$ *auto*
then have $\langle F = b \ @ \ a \rangle$
using *get-all-ann-decomposition-decomp*[of $\langle \text{Propagated } L \ () \# F \rangle$ *a*
 $\langle \text{Propagated } L \ () \# b \rangle$] **by** *simp*
then have *latm*: $\langle \text{unassigned-lit } A \ b = \text{Suc } (\text{unassigned-lit } A \ (\text{Propagated } L \ () \# b)) \rangle$

using $F\text{-le-}A$ by *simp*
 obtain *rem* where
 $\text{rem} : \langle \text{map } (\lambda a. \text{Suc } (\text{length } (\text{snd } a))) (\text{rev } (\text{get-all-ann-decomposition } (F' @ \text{Decided } K \# F))) \rangle$
 $= \text{map } (\lambda a. \text{Suc } (\text{length } (\text{snd } a))) (\text{rev } (\text{get-all-ann-decomposition } F)) @ \text{rem}$
 using *take-length-get-all-ann-decomposition-decided-sandwich*[of $F \langle \lambda a. \text{Suc } (\text{length } a) \rangle F' K$]
 unfolding *o-def* by (*metis append-take-drop-id*)
 then have *rem*: $\langle \text{map } (\lambda a. \text{Suc } (\text{length } (\text{snd } a))) (\text{get-all-ann-decomposition } (F' @ \text{Decided } K \# F)) \rangle$
 $= \text{rev rem } @ \text{map } (\lambda a. \text{Suc } (\text{length } (\text{snd } a))) ((\text{get-all-ann-decomposition } F))$
 by (*simp add: rev-map[symmetric] rev-swap*)
 have $\langle \text{length } (\text{rev rem } @ \text{map } (\lambda a. \text{Suc } (\text{length } (\text{snd } a))) (\text{get-all-ann-decomposition } F)) \rangle$
 $\leq \text{Suc } (\text{card } (\text{atms-of-ms } A))$
 using *arg-cong[OF rem, of length]* *tr-S-le-A*
 $\text{length-get-all-ann-decomposition-length}$ [of $\langle F' @ \text{Decided } K \# F \rangle$] *tr-S* by *auto*
 moreover {
 { **fix** $i :: \text{nat}$ **and** $xs :: \langle 'a \text{ list} \rangle$
 have $\langle i < \text{length } xs \implies \text{length } xs - \text{Suc } i < \text{length } xs \rangle$
 by *auto*
 then have $H : \langle i < \text{length } xs \implies \text{rev } xs ! i \in \text{set } xs \rangle$
 using *rev-nth*[of i xs] **unfolding** *in-set-conv-nth* by (*force simp add: in-set-conv-nth*)
 } **note** $H = \text{this}$
 have $\langle \forall i < \text{length } \text{rev rem}. \text{rev rem } ! i < \text{card } (\text{atms-of-ms } A) + 2 \rangle$
 using *tr-S-le-A* $\text{length-in-get-all-ann-decomposition-bounded}$ [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 $\mu_C\text{-bounded}$ [of $\langle \text{rev rem} \rangle \langle \text{card } (\text{atms-of-ms } A) + 2 \rangle \langle \text{unassigned-lit } A \ b \rangle$] $T \text{ undef-}L$
 by (*simp add: rem μ_C -append μ_C -cons $F \text{ tr-}S$*)
 qed

lemma *dp11-bj-trail-mes-decreasing-prop*:

assumes *dp11*: $\langle \text{dp11-bj } S \ T \rangle$ **and** *inv*: $\langle \text{inv } S \rangle$ **and**
 $N\text{-}A : \langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $M\text{-}A : \langle \text{atm-of } \langle \text{lits-of-l } (\text{trail } S) \rangle \subseteq \text{atms-of-ms } A \rangle$ **and**
 $nd : \langle \text{no-dup } (\text{trail } S) \rangle$ **and**
 $\text{fin-}A : \langle \text{finite } A \rangle$

shows $\langle (2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A))$
 $- \mu_C (1 + \text{card } (\text{atms-of-ms } A)) (2 + \text{card } (\text{atms-of-ms } A)) (\text{trail-weight } T)$
 $< (2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A))$
 $- \mu_C (1 + \text{card } (\text{atms-of-ms } A)) (2 + \text{card } (\text{atms-of-ms } A)) (\text{trail-weight } S) \rangle$

proof –

let $?b = \langle 2 + \text{card } (\text{atms-of-ms } A) \rangle$
 let $?s = \langle 1 + \text{card } (\text{atms-of-ms } A) \rangle$
 let $? \mu = \langle \mu_C ?s ?b \rangle$
 have $M'\text{-}A : \langle \text{atm-of } \langle \text{lits-of-l } (\text{trail } T) \rangle \subseteq \text{atms-of-ms } A \rangle$
 by (*meson* $M\text{-}A \ N\text{-}A \ \text{dp11} \ \text{dp11-bj-atms-in-trail-in-set inv}$)
 have $nd' : \langle \text{no-dup } (\text{trail } T) \rangle$
 using $\langle \text{dp11-bj } S \ T \rangle \ \text{dp11-bj-no-dup } nd \ \text{inv}$ by *blast*
 { **fix** $i :: \text{nat}$ **and** $xs :: \langle 'a \text{ list} \rangle$
 have $\langle i < \text{length } xs \implies \text{length } xs - \text{Suc } i < \text{length } xs \rangle$
 by *auto*
 then have $H : \langle i < \text{length } xs \implies xs ! i \in \text{set } xs \rangle$
 using *rev-nth*[of i xs] **unfolding** *in-set-conv-nth* by (*force simp add: in-set-conv-nth*)
 } **note** $H = \text{this}$

have $l\text{-}M\text{-}A : \langle \text{length } (\text{trail } S) \leq \text{card } (\text{atms-of-ms } A) \rangle$
 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 ⟨μC ?s ?b (trail-weight S) < μC ?s ?b (trail-weight T)⟩ by simp
moreover from μC-bounded[OF bounded-M l-trail-weight-M]
  have ⟨μC ?s ?b (trail-weight T) ≤ ?b ^ ?s⟩ by auto
ultimately show ?thesis by linarith
qed

```

lemma wf-dpll-bj:

```

assumes fin: ⟨finite A⟩
shows ⟨wf {(T, S). dpll-bj S T
  ∧ atms-of-mm (clausesNOT 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. μC (1+card (atms-of-ms A)) (2+card (atms-of-ms A)) (trail-weight S)⟩])
fix a b :: ⟨'st⟩
let ?b = ⟨2+card (atms-of-ms A)⟩
let ?s = ⟨1+card (atms-of-ms A)⟩
let ?μ = ⟨μC ?s ?b⟩
assume ab: ⟨(b, a) ∈ ?A⟩

have fin-A: ⟨finite (atms-of-ms A)⟩
  using fin by auto
have
  dpll-bj: ⟨dpll-bj a b⟩ and
  N-A: ⟨atms-of-mm (clausesNOT a) ⊆ atms-of-ms A⟩ and
  M-A: ⟨atm-of ' lits-of-l (trail a) ⊆ atms-of-ms A⟩ and
  nd: ⟨no-dup (trail a)⟩ and
  inv: ⟨inv a⟩
  using ab by auto

have M'-A: ⟨atm-of ' lits-of-l (trail b) ⊆ atms-of-ms A⟩
  by (meson M-A N-A ⟨dpll-bj a b⟩ dpll-bj-atms-in-trail-in-set inv)
have nd': ⟨no-dup (trail b)⟩
  using ⟨dpll-bj a b⟩ dpll-bj-no-dup nd inv by blast
{ fix i :: nat and xs :: ⟨'a list⟩
  have ⟨i < length xs ⟹ length xs - Suc i < length xs⟩
    by auto
  then have H: ⟨i < length xs ⟹ xs ! i ∈ set xs⟩
    using rev-nth[of i xs] unfolding in-set-conv-nth by (force simp add: in-set-conv-nth)
} note H = this

have l-M-A: ⟨length (trail a) ≤ card (atms-of-ms A)⟩
  by (simp add: fin-A M-A card-mono no-dup-length-eq-card-atm-of-lits-of-l nd)
have l-M'-A: ⟨length (trail b) ≤ card (atms-of-ms A)⟩
  by (simp add: fin-A M'-A card-mono no-dup-length-eq-card-atm-of-lits-of-l nd')
have l-trail-weight-M: ⟨length (trail-weight b) ≤ 1+card (atms-of-ms A)⟩

```

using $l\text{-}M'\text{-}A$ $\text{length-get-all-ann-decomposition-length}$ [*of* $\langle \text{trail } b \rangle$] **by** *auto*
have $\langle \mu_C \text{ ?s ?b } (\text{trail-weight } a) < \mu_C \text{ ?s ?b } (\text{trail-weight } b) \rangle$ **by** *simp*
using $\text{length-in-get-all-ann-decomposition-bounded}$ [*of* $- b$] $l\text{-}M'\text{-}A$
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 $dpll\text{-}bj\text{-}trail\text{-}mes\text{-}increasing\text{-}prop$ [*OF* $dpll\text{-}bj\text{-}inv\ N\text{-}A\ M\text{-}A\ nd\ fin$]
have $\langle \mu_C \text{ ?s ?b } (\text{trail-weight } a) < \mu_C \text{ ?s ?b } (\text{trail-weight } b) \rangle$ **by** *simp*
moreover from $\mu_C\text{-bounded}$ [*OF* $\text{bounded-}M\ l\text{-}trail\text{-}weight\text{-}M$]
have $\langle \mu_C \text{ ?s ?b } (\text{trail-weight } b) \leq ?b \wedge ?s \rangle$ **by** *auto*
ultimately show $\langle ?b \wedge ?s \leq ?b \wedge ?s \wedge$
 $\mu_C \text{ ?s ?b } (\text{trail-weight } b) \leq ?b \wedge ?s \wedge$
 $\mu_C \text{ ?s ?b } (\text{trail-weight } a) < \mu_C \text{ ?s ?b } (\text{trail-weight } b) \rangle$
by *blast*
qed

Alternative termination proof abbreviation $DPLL\text{-}mes_W$ **where**

$\langle DPLL\text{-}mes_W\ A\ M \equiv$
 $\text{map } (\lambda L. \text{ if is-decided } L \text{ then } 2::\text{nat else } 1) (\text{rev } M) \text{ @ replicate } (\text{card } A - \text{length } M) \ 3 \rangle$

lemma $\text{distinctcard-atm-of-lit-of-eq-length}$:

assumes *no-dup S*
shows $\text{card } (\text{atm-of } \langle \text{lits-of-l } S \rangle = \text{length } S$
using *assms* **by** (*induct S*) (*auto simp add: image-image lits-of-def no-dup-def*)

lemma $dpll\text{-}bj\text{-}trail\text{-}mes\text{-}decreasing\text{-}less\text{-}than$:

assumes $dpll$: $\langle dpll\text{-}bj\ S\ T \rangle$ **and** inv : $\langle inv\ S \rangle$ **and**
 $N\text{-}A$: $\langle \text{atms-of-mm } (\text{clauses}_{NOT}\ S) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $M\text{-}A$: $\langle \text{atm-of } \langle \text{lits-of-l } (\text{trail } S) \rangle \subseteq \text{atms-of-ms } A \rangle$ **and**
 nd : $\langle \text{no-dup } (\text{trail } S) \rangle$ **and**
 $fin\text{-}A$: $\langle \text{finite } A \rangle$
shows $\langle (DPLL\text{-}mes_W (\text{atms-of-ms } A) (\text{trail } T), DPLL\text{-}mes_W (\text{atms-of-ms } A) (\text{trail } S)) \in$
 $\text{lern less-than } (\text{card } ((\text{atms-of-ms } A))) \rangle$
using *assms*(1,2)

proof (*induction rule: dpll-bj-all-induct*)

case ($\text{decide}_{NOT}\ L\ T$)

define n **where**

$\langle n = \text{card } (\text{atms-of-ms } A) - \text{card } (\text{atm-of } \langle \text{lits-of-l } (\text{trail } S) \rangle) \rangle$

have [*simp*]: $\langle \text{length } (\text{trail } S) = \text{card } (\text{atm-of } \langle \text{lits-of-l } (\text{trail } S) \rangle) \rangle$
using nd **by** (*auto simp: no-dup-def lits-of-def image-image dest: distinct-card*)

have $\langle \text{atm-of } L \notin \text{atm-of } \langle \text{lits-of-l } (\text{trail } S) \rangle$

by (*metis* $\text{decide}_{NOT}.\text{hyps}(1)$ *defined-lit-map imageE in-lits-of-l-defined-litD*)

have $\langle \text{card } (\text{atms-of-ms } A) > \text{card } (\text{atm-of } \langle \text{lits-of-l } (\text{trail } S) \rangle) \rangle$

by (*metis* $N\text{-}A\ \langle \text{atm-of } L \notin \text{atm-of } \langle \text{lits-of-l } (\text{trail } S) \rangle \text{ atms-of-ms-finite card-seteq decide}_{NOT}.\text{hyps}(2)$
 $M\text{-}A\ fin\text{-}A\ \text{not-le subsetCE}$)

then have

$n\text{-}0$: $\langle n > 0 \rangle$ **and**

$n\text{-}Suc$: $\langle \text{card } (\text{atms-of-ms } A) - \text{Suc } (\text{card } (\text{atm-of } \langle \text{lits-of-l } (\text{trail } S) \rangle)) = n - 1 \rangle$

unfolding $n\text{-def}$ **by** *auto*

show $?case$

using $fin\text{-}A\ \text{decide}_{NOT}\ n\text{-}0$ **unfolding** $\text{state-eq}_{NOT}\text{-trail}$ [*OF* $\text{decide}_{NOT}(3)$]

by (*cases n*) (*auto simp: prepend-same-lern n-def[symmetric] n-Suc lern-Suc*
 $\text{simp del: state-simp}_{NOT}\ \text{lern.simps}$)

next
case (*propagate*_{NOT} *C L T*) **note** *C = this(1)* **and** *undef = this(3)* **and** *T = this(3)*
then have $\langle \text{card } (\text{atms-of-ms } A) > \text{length } (\text{trail } S) \rangle$
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* $\langle \text{card } (\text{atms-of-ms } A) - \text{length } (\text{trail } S) \rangle$)
 (*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 $\langle \text{trail } (\text{reduce-trail-to}_{\text{NOT}} F S) = F \rangle$
 by (*simp add: tr-S*)
have $\langle \text{no-dup } F \rangle$
 using *nd tr-S by (auto dest: no-dup-appendD)*
then have *card-A-F: $\langle \text{card } (\text{atms-of-ms } A) > \text{length } F \rangle$*
 using *distinctcard-atm-of-lit-of-eq-length[of $\langle \text{trail } S \rangle$ card-mono[OF - M-A] fin-A nd tr-S*
 by auto
have $\langle \text{no-dup } (F' @ F) \rangle$
 using *nd tr-S by (auto dest: no-dup-appendD)*
then have $\langle \text{no-dup } F' \rangle$
 apply (*subst (asm) no-dup-rev[symmetric]*)
 using *nd tr-S by (auto dest: no-dup-appendD)*
then have *card-A-F': $\langle \text{card } (\text{atms-of-ms } A) > \text{length } F' + \text{length } F \rangle$*
 using *distinctcard-atm-of-lit-of-eq-length[of $\langle \text{trail } S \rangle$ 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* $\langle \text{card } (\text{atms-of-ms } A) - \text{length } F \rangle$)
 (*auto simp: tr-S prepend-same-lexn lexn-Suc simp del: state-simp_{NOT} lexn.simps*)
qed
lemma
assumes *fin[simp]: $\langle \text{finite } A \rangle$*
shows $\langle \text{wf } \{(T, S). \text{dpll-bj } S T$
 $\wedge \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \subseteq \text{atms-of-ms } A \wedge \text{atm-of } \text{'lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A$
 $\wedge \text{no-dup } (\text{trail } S) \wedge \text{inv } S\} \rangle$
 (*is* $\langle \text{wf } ?A \rangle$)
unfolding *conj-commute[of $\langle \text{dpll-bj } - \rightarrow \rangle$*
apply (*rule wf-wf-if-measure'[of - - - $\langle \lambda S. \text{DPLL-mes}_W ((\text{atms-of-ms } A)) (\text{trail } S) \rangle$]*)
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 that *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 $A :: \langle 'v\ clause\ set \rangle$ **and** $S\ T :: \langle 'st \rangle$

assumes

$\langle atms-of-mm\ (clauses_{NOT}\ S) \subseteq atms-of-ms\ A \rangle$ **and**

$\langle atm-of\ ' \ lits-of-l\ (trail\ S) \subseteq atms-of-ms\ A \rangle$ **and**

$\langle no-dup\ (trail\ S) \rangle$ **and**

$\langle finite\ A \rangle$ **and**

$inv: \langle inv\ S \rangle$ **and**

$n-d: \langle no-dup\ (trail\ S) \rangle$ **and**

$n-s: \langle no-step\ dpll-bj\ S \rangle$ **and**

$decomp: \langle all-decomposition-implies-m\ (clauses_{NOT}\ S)\ (get-all-ann-decomposition\ (trail\ S)) \rangle$

shows $\langle unsatisfiable\ (set-mset\ (clauses_{NOT}\ S)) \rangle$

$\vee (trail\ S \models_{asm}\ clauses_{NOT}\ S \wedge satisfiable\ (set-mset\ (clauses_{NOT}\ S)))$

proof –

let $?N = \langle set-mset\ (clauses_{NOT}\ S) \rangle$

let $?M = \langle trail\ S \rangle$

consider

$(sat)\ \langle satisfiable\ ?N \rangle$ **and** $\langle ?M \models_{as}\ ?N \rangle$

| $(sat')\ \langle satisfiable\ ?N \rangle$ **and** $\langle \neg ?M \models_{as}\ ?N \rangle$

| $(unsat)\ \langle unsatisfiable\ ?N \rangle$

by *auto*

then show *?thesis*

proof *cases*

case *sat'* **note** $sat = this(1)$ **and** $M = this(2)$

obtain C **where** $\langle C \in ?N \rangle$ **and** $\langle \neg ?M \models_a\ C \rangle$ **using** M **unfolding** *true-annots-def* **by** *auto*

obtain $I :: \langle 'v\ literal\ set \rangle$ **where**

$\langle I \models_s\ ?N \rangle$ **and**

$cons: \langle consistent-interp\ I \rangle$ **and**

$tot: \langle total-over-m\ I\ ?N \rangle$ **and**

$atm-I-N: \langle atm-of\ 'I \subseteq atms-of-ms\ ?N \rangle$

using *sat* **unfolding** *satisfiable-def-min* **by** *auto*

let $?I = \langle I \cup \{P \mid P \in lits-of-l\ ?M \wedge atm-of\ P \notin atm-of\ 'I\} \rangle$

let $?O = \langle \{unmark\ L \mid L.\ is-decided\ L \wedge L \in set\ ?M \wedge atm-of\ (lit-of\ L) \notin atms-of-ms\ ?N\} \rangle$

have $cons-I': \langle consistent-interp\ ?I \rangle$

using *cons* **using** $\langle no-dup\ ?M \rangle$ **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 (clausesNOT 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 ∅[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 ∅ 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-clss-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: $\langle ?M = F' @ Decided K \# F \rangle$ **and**
nm: $\langle \forall f \in set F'. \neg is-decided f \rangle$
by (*metis (full-types) is-decided-ex-Decided old.unit.exhaust*)
let $?K = \langle Decided K :: ('v, unit) ann-lit \rangle$
have $\langle ?K \in set ?M \rangle$
unfolding *M-K by auto*
let $?C = \langle image-mset lit-of \{ \#L \in \#mset ?M. is-decided L \wedge L \neq ?K \# \} :: 'v clause \rangle$
let $?C' = \langle set-mset (image-mset (\lambda L :: 'v literal. \{ \#L \# \})) (?C + unmark ?K) \rangle$
have $\langle ?N \cup \{ unmark L \mid L. is-decided L \wedge L \in set ?M \} \models ps unmark-l ?M \rangle$
using *all-decomposition-implies-propagated-lits-are-implied[OF decomp]* .
moreover have $C': \langle ?C' = \{ unmark L \mid L. is-decided L \wedge L \in set ?M \} \rangle$
unfolding *M-K by standard force+*
ultimately have $N-C-M: \langle ?N \cup ?C' \models ps unmark-l ?M \rangle$
by auto
have $N-M-False: \langle ?N \cup (\lambda L. unmark L) ' (set ?M) \models ps \{ \{ \# \} \} \rangle$
unfolding *true-clss-clss-def true-annot-def Ball-def true-annot-def*
proof (*intro allI impI*)
fix $LL :: 'v literal set$
assume
tot: $\langle total-over-m LL (set-mset (clauses_{NOT} S) \cup unmark-l (trail S) \cup \{ \{ \# \} \}) \rangle$ **and**
cons: $\langle consistent-interp LL \rangle$ **and**
 $LL: \langle LL \models s set-mset (clauses_{NOT} S) \cup unmark-l (trail S) \rangle$
have $\langle total-over-m LL (CNot C) \rangle$
by (*metis $C \in \# clauses_{NOT} S$ insert-absorb tot total-over-m-CNot-toal-over-m total-over-m-insert total-over-m-union*)
then have $total-over-m LL (unmark-l (trail S) \cup CNot C)$
using *tot by force*
then show $LL \models s \{ \{ \# \} \}$
using *tot cons LL*
by (*metis (no-types) $C \in \# clauses_{NOT} S$ $\langle trail S \models as CNot C \rangle$ consistent-CNot-not true-annot-def true-clss-clss true-clss-clss-def true-clss-def true-clss-union*)
qed
have $\langle undefined-lit F K \rangle$ **using** $\langle no-dup ?M \rangle$ **unfolding** *M-K by (auto simp: defined-lit-map)*
moreover {
have $\langle ?N \cup ?C' \models ps \{ \{ \# \} \} \rangle$
proof –
have $A: \langle ?N \cup ?C' \cup unmark-l ?M = ?N \cup unmark-l ?M \rangle$
unfolding *M-K by auto*
show *?thesis*
using *true-clss-clss-left-right[OF N-C-M, of $\langle \{ \{ \# \} \} \rangle$ N-M-False unfolding A by auto*
qed
have $\langle ?N \models p image-mset uminus ?C + \{ \# - K \# \} \rangle$
unfolding *true-clss-clss-def true-clss-clss-def total-over-m-def*
proof (*intro allI impI*)
fix I
assume
tot: $\langle total-over-set I (atms-of-ms (?N \cup \{ image-mset uminus ?C + \{ \# - K \# \} \})) \rangle$ **and**
cons: $\langle consistent-interp I \rangle$ **and**
 $\langle I \models s ?N \rangle$
have $\langle (K \in I \wedge -K \notin I) \vee (-K \in I \wedge K \notin I) \rangle$
using *cons tot unfolding consistent-interp-def by (cases K) auto*
have $\langle \{ a \in set (trail S). is-decided a \wedge a \neq Decided K \} = set (trail S) \cap \{ L. is-decided L \wedge L \neq Decided K \} \rangle$
by auto
then have *tot': $\langle total-over-set I (atm-of ' lit-of ' (set ?M \cap \{ L. is-decided L \wedge L \neq Decided K \} \})) \rangle$*

```

    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-cl-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 clausesNOT prepend-trail tl-trail add-clNOT remove-clNOT inv
  decide-conds backjump-conds propagate-conds
for
  trail :: ⟨'st ⇒ ('v, unit) ann-lits⟩ and
  clausesNOT :: ⟨'st ⇒ 'v clauses⟩ and
  prepend-trail :: ⟨('v, unit) ann-lit ⇒ 'st ⇒ 'st⟩ and
  tl-trail :: ⟨'st ⇒ 'st⟩ and
  add-clNOT :: ⟨'v clause ⇒ 'st ⇒ 'st⟩ and
  remove-clNOT :: ⟨'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 rtrancpl-dpll-bj-inv:

```

assumes $\langle \text{dpll-bj}^{**} S T \rangle$ **and** $\langle \text{inv } S \rangle$
shows $\langle \text{inv } T \rangle$
using *assms* **by** (*induction rule: rtrancpl-induct*)
(auto simp add: dpll-bj-no-dup intro: dpll-bj-inv)

lemma *rtrancpl-dpll-bj-no-dup*:
assumes $\langle \text{dpll-bj}^{**} S T \rangle$ **and** $\langle \text{inv } S \rangle$
and $\langle \text{no-dup (trail } S) \rangle$
shows $\langle \text{no-dup (trail } T) \rangle$
using *assms* **by** (*induction rule: rtrancpl-induct*)
(auto simp add: dpll-bj-no-dup dest: rtrancpl-dpll-bj-inv dpll-bj-inv)

lemma *rtrancpl-dpll-bj-atms-of-ms-clauses-inv*:
assumes
 $\langle \text{dpll-bj}^{**} S T \rangle$ **and** $\langle \text{inv } S \rangle$
shows $\langle \text{atms-of-mm (clauses}_{\text{NOT}} S) = \text{atms-of-mm (clauses}_{\text{NOT}} T) \rangle$
using *assms* **by** (*induction rule: rtrancpl-induct*)
(auto dest: rtrancpl-dpll-bj-inv dpll-bj-atms-of-ms-clauses-inv)

lemma *rtrancpl-dpll-bj-atms-in-trail*:
assumes
 $\langle \text{dpll-bj}^{**} S T \rangle$ **and**
 $\langle \text{inv } S \rangle$ **and**
 $\langle \text{atm-of ' (lits-of-l (trail } S)) \subseteq \text{atms-of-mm (clauses}_{\text{NOT}} S) \rangle$
shows $\langle \text{atm-of ' (lits-of-l (trail } T)) \subseteq \text{atms-of-mm (clauses}_{\text{NOT}} T) \rangle$
using *assms* **apply** (*induction rule: rtrancpl-induct*)
using *dpll-bj-atms-in-trail dpll-bj-atms-of-ms-clauses-inv rtrancpl-dpll-bj-inv* **by** *auto*

lemma *rtrancpl-dpll-bj-sat-iff*:
assumes $\langle \text{dpll-bj}^{**} S T \rangle$ **and** $\langle \text{inv } S \rangle$
shows $\langle I \models_{\text{sm}} \text{clauses}_{\text{NOT}} S \longleftrightarrow I \models_{\text{sm}} \text{clauses}_{\text{NOT}} T \rangle$
using *assms* **by** (*induction rule: rtrancpl-induct*)
(auto dest!: dpll-bj-sat-iff simp: rtrancpl-dpll-bj-inv)

lemma *rtrancpl-dpll-bj-atms-in-trail-in-set*:
assumes
 $\langle \text{dpll-bj}^{**} S T \rangle$ **and**
 $\langle \text{inv } S \rangle$
 $\langle \text{atms-of-mm (clauses}_{\text{NOT}} S) \subseteq A \rangle$ **and**
 $\langle \text{atm-of ' (lits-of-l (trail } S)) \subseteq A \rangle$
shows $\langle \text{atm-of ' (lits-of-l (trail } T)) \subseteq A \rangle$
using *assms* **by** (*induction rule: rtrancpl-induct*)
(auto dest: rtrancpl-dpll-bj-inv
simp: dpll-bj-atms-in-trail-in-set rtrancpl-dpll-bj-atms-of-ms-clauses-inv rtrancpl-dpll-bj-inv)

lemma *rtrancpl-dpll-bj-all-decomposition-implies-inv*:
assumes
 $\langle \text{dpll-bj}^{**} S T \rangle$ **and**
 $\langle \text{inv } S \rangle$
 $\langle \text{all-decomposition-implies-m (clauses}_{\text{NOT}} S) (\text{get-all-ann-decomposition (trail } S)) \rangle$
shows $\langle \text{all-decomposition-implies-m (clauses}_{\text{NOT}} T) (\text{get-all-ann-decomposition (trail } T)) \rangle$
using *assms* **by** (*induction rule: rtrancpl-induct*)
(auto intro: dpll-bj-all-decomposition-implies-inv simp: rtrancpl-dpll-bj-inv)

lemma *rtrancpl-dpll-bj-inv-incl-dpll-bj-inv-trancpl*:
 $\langle \{(T, S). \text{dpll-bj}^{++} S T$

$\wedge \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A \wedge \text{atm-of } ' \text{ lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A$
 $\wedge \text{no-dup } (\text{trail } S) \wedge \text{inv } S \}$
 $\subseteq \{(T, S). \text{dpll-bj } S T \wedge \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A$
 $\wedge \text{atm-of } ' \text{ lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A \wedge \text{no-dup } (\text{trail } S) \wedge \text{inv } S\}^+$
(is $\langle ?A \subseteq ?B^+ \rangle$ **)**
proof *standard*
fix x
assume $x-A: \langle x \in ?A \rangle$
obtain $S T::\langle 'st \rangle$ **where**
 $x[\text{simp}]: \langle x = (T, S) \rangle$ **by** $(\text{cases } x) \text{ auto}$
have
 $\langle \text{dpll-bj}^{++} S T \rangle$ **and**
 $\langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $\langle \text{atm-of } ' \text{ lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $\langle \text{no-dup } (\text{trail } S) \rangle$ **and**
 $\langle \text{inv } S \rangle$
using $x-A$ **by** *auto*
then show $\langle x \in ?B^+ \rangle$ **unfolding** x
proof (*induction rule: tranclp-induct*)
case *base*
then show $?case$ **by** *auto*
next
case $(\text{step } T U)$ **note** $\text{step} = \text{this}(1)$ **and** $ST = \text{this}(2)$ **and** $IH = \text{this}(3)[\text{OF } \text{this}(4-7)]$
and $N-A = \text{this}(4)$ **and** $M-A = \text{this}(5)$ **and** $nd = \text{this}(6)$ **and** $inv = \text{this}(7)$

have $[\text{simp}]: \langle \text{atms-of-mm } (\text{clauses}_{NOT} S) = \text{atms-of-mm } (\text{clauses}_{NOT} T) \rangle$
using $\text{step } r\text{tranclp-dpll-bj-atms-of-ms-clauses-inv } \text{tranclp-into-rtranclp } inv$ **by** *fastforce*
have $\langle \text{no-dup } (\text{trail } T) \rangle$
using $\text{local.step } nd \text{ rtranclp-dpll-bj-no-dup } \text{tranclp-into-rtranclp } inv$ **by** *fastforce*
moreover have $\langle \text{atm-of } ' (\text{lits-of-l } (\text{trail } T)) \subseteq \text{atms-of-ms } A \rangle$
by $(\text{metis } inv \text{ M-A } N-A \text{ local.step } r\text{tranclp-dpll-bj-atms-in-trail-in-set}$
 $\text{tranclp-into-rtranclp})$
moreover have $\langle \text{inv } T \rangle$
using $inv \text{ local.step } r\text{tranclp-dpll-bj-inv } \text{tranclp-into-rtranclp}$ **by** *fastforce*
ultimately have $\langle (U, T) \in ?B \rangle$ **using** $ST \text{ N-A } M-A \text{ inv}$ **by** *auto*
then show $?case$ **using** IH **by** (*rule trancl-into-trancl2*)
qed
qed

lemma *wf-tranclp-dpll-bj*:
assumes $\text{fin}: \langle \text{finite } A \rangle$
shows $\langle \text{wf } \{(T, S). \text{dpll-bj}^{++} S T$
 $\wedge \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A \wedge \text{atm-of } ' \text{ lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A$
 $\wedge \text{no-dup } (\text{trail } S) \wedge \text{inv } S\} \rangle$
using $\text{wf-trancl}[\text{OF } \text{wf-dpll-bj}[\text{OF } \text{fin}]] \text{ rtranclp-dpll-bj-inv-incl-dpll-bj-inv-trancl}$
by (*rule wf-subset*)

lemma *dpll-bj-sat-ext-iff*:
 $\langle \text{dpll-bj } S T \implies \text{inv } S \implies I \models_{\text{sextm } \text{clauses}_{NOT} S} \longleftrightarrow I \models_{\text{sextm } \text{clauses}_{NOT} T} \rangle$
by (*simp add: dpll-bj-clauses*)

lemma *rtranclp-dpll-bj-sat-ext-iff*:
 $\langle \text{dpll-bj}^{**} S T \implies \text{inv } S \implies I \models_{\text{sextm } \text{clauses}_{NOT} S} \longleftrightarrow I \models_{\text{sextm } \text{clauses}_{NOT} T} \rangle$
by (*induction rule: rtranclp-induct*) (*simp-all add: rtranclp-dpll-bj-inv dpll-bj-sat-ext-iff*)

theorem *full-dpll-backjump-final-state*:

fixes $A :: \langle 'v \text{ clause set} \rangle$ **and** $S \ T :: \langle 'st \rangle$
assumes
 $\text{full}: \langle \text{full dpll-bj } S \ T \rangle$ **and**
 $\text{atms-S}: \langle \text{atms-of-mm } (\text{clauses}_{NOT} \ S) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $\text{atms-trail}: \langle \text{atm-of } ' \text{ lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $\text{n-d}: \langle \text{no-dup } (\text{trail } S) \rangle$ **and**
 $\langle \text{finite } A \rangle$ **and**
 $\text{inv}: \langle \text{inv } S \rangle$ **and**
 $\text{decomp}: \langle \text{all-decomposition-implies-m } (\text{clauses}_{NOT} \ S) \ (\text{get-all-ann-decomposition } (\text{trail } S)) \rangle$
shows $\langle \text{unsatisfiable } (\text{set-mset } (\text{clauses}_{NOT} \ S)) \rangle$
 $\vee (\text{trail } T \models_{asm} \text{clauses}_{NOT} \ S \wedge \text{satisfiable } (\text{set-mset } (\text{clauses}_{NOT} \ S)))$
proof –
have $\text{st}: \langle \text{dpll-bj}^{**} \ S \ T \rangle$ **and** $\langle \text{no-step dpll-bj } T \rangle$
using $\text{full unfolding full-def by fast+}$
moreover have $\langle \text{atms-of-mm } (\text{clauses}_{NOT} \ T) \subseteq \text{atms-of-ms } A \rangle$
using $\text{atms-S inv rtranclp-dpll-bj-atms-of-ms-clauses-inv st by blast}$
moreover have $\langle \text{atm-of } ' \text{ lits-of-l } (\text{trail } T) \subseteq \text{atms-of-ms } A \rangle$
using $\text{atms-S atms-trail inv rtranclp-dpll-bj-atms-in-trail-in-set st by auto}$
moreover have $\langle \text{no-dup } (\text{trail } T) \rangle$
using $\text{n-d inv rtranclp-dpll-bj-no-dup st by blast}$
moreover have $\text{inv}: \langle \text{inv } T \rangle$
using $\text{inv rtranclp-dpll-bj-inv st by blast}$
moreover
have $\text{decomp}: \langle \text{all-decomposition-implies-m } (\text{clauses}_{NOT} \ T) \ (\text{get-all-ann-decomposition } (\text{trail } T)) \rangle$
using $\langle \text{inv } S \rangle \text{ decomp rtranclp-dpll-bj-all-decomposition-implies-inv st by blast}$
ultimately have $\langle \text{unsatisfiable } (\text{set-mset } (\text{clauses}_{NOT} \ T)) \rangle$
 $\vee (\text{trail } T \models_{asm} \text{clauses}_{NOT} \ T \wedge \text{satisfiable } (\text{set-mset } (\text{clauses}_{NOT} \ T)))$
using $\langle \text{finite } A \rangle \text{ dpll-backjump-final-state by force}$
then show $?thesis$
by $(\text{meson } \langle \text{inv } S \rangle \text{ rtranclp-dpll-bj-sat-iff satisfiable-carac st true-annots-true-cls})$
qed

corollary $\text{full-dpll-backjump-final-state-from-init-state}$:

fixes $A :: \langle 'v \text{ clause set} \rangle$ **and** $S \ T :: \langle 'st \rangle$
assumes
 $\text{full}: \langle \text{full dpll-bj } S \ T \rangle$ **and**
 $\langle \text{trail } S = [] \rangle$ **and**
 $\langle \text{clauses}_{NOT} \ S = N \rangle$ **and**
 $\langle \text{inv } S \rangle$
shows $\langle \text{unsatisfiable } (\text{set-mset } N) \vee (\text{trail } T \models_{asm} N \wedge \text{satisfiable } (\text{set-mset } N)) \rangle$
using $\text{assms full-dpll-backjump-final-state[of } S \ T \ \langle \text{set-mset } N \rangle]$ **by auto**

lemma $\text{tranclp-dpll-bj-trail-mes-decreasing-prop}$:

assumes $\text{dpll}: \langle \text{dpll-bj}^{++} \ S \ T \rangle$ **and** $\text{inv}: \langle \text{inv } S \rangle$ **and**
 $N\text{-A}: \langle \text{atms-of-mm } (\text{clauses}_{NOT} \ S) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $M\text{-A}: \langle \text{atm-of } ' \text{ lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $\text{n-d}: \langle \text{no-dup } (\text{trail } S) \rangle$ **and**
 $\text{fin-A}: \langle \text{finite } A \rangle$
shows $\langle (2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A))$
 $\quad - \mu_C (1 + \text{card } (\text{atms-of-ms } A)) (2 + \text{card } (\text{atms-of-ms } A)) (\text{trail-weight } T)$
 $\quad < (2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A))$
 $\quad - \mu_C (1 + \text{card } (\text{atms-of-ms } A)) (2 + \text{card } (\text{atms-of-ms } A)) (\text{trail-weight } S) \rangle$
using dpll
proof induction
case base
then show $?case$

```

    using  $N\text{-}A$   $M\text{-}A$   $n\text{-}d$   $dpll\text{-}bj\text{-}trail\text{-}mes\text{-}decreasing\text{-}prop$   $fin\text{-}A$   $inv$  by blast
next
case (step  $T$   $U$ ) note  $st = this(1)$  and  $dpll = this(2)$  and  $IH = this(3)$ 
have  $\langle atms\text{-}of\text{-}mm (clauses_{NOT} S) = atms\text{-}of\text{-}mm (clauses_{NOT} T) \rangle$ 
  using  $rtrancp\text{-}dpll\text{-}bj\text{-}atms\text{-}of\text{-}ms\text{-}clauses\text{-}inv$  by (metis  $dpll\text{-}bj\text{-}clauses$   $dpll\text{-}bj\text{-}inv$   $inv$   $st$ 
     $trancpD$ )
then have  $N\text{-}A'$ :  $\langle atms\text{-}of\text{-}mm (clauses_{NOT} T) \subseteq atms\text{-}of\text{-}ms A \rangle$ 
  using  $N\text{-}A$  by auto
moreover have  $M\text{-}A'$ :  $\langle atm\text{-}of ' lits\text{-}of\text{-}l (trail T) \subseteq atms\text{-}of\text{-}ms A \rangle$ 
  by (meson  $M\text{-}A$   $N\text{-}A$   $inv$   $rtrancp\text{-}dpll\text{-}bj\text{-}atms\text{-}in\text{-}trail\text{-}in\text{-}set$   $st$   $dpll$ 
     $trancp.r\text{-}into\text{-}trancp$   $trancp\text{-}into\text{-}rtrancp$   $trancp\text{-}trans$ )
moreover have  $nd$ :  $\langle no\text{-}dup (trail T) \rangle$ 
  by (metis  $inv$   $n\text{-}d$   $rtrancp\text{-}dpll\text{-}bj\text{-}no\text{-}dup$   $st$   $trancp\text{-}into\text{-}rtrancp$ )
moreover have  $inv$   $T$ 
  by (meson  $dpll$   $dpll\text{-}bj\text{-}inv$   $inv$   $rtrancp\text{-}dpll\text{-}bj\text{-}inv$   $st$   $trancp\text{-}into\text{-}rtrancp$ )
ultimately show ?case
  using  $IH$   $dpll\text{-}bj\text{-}trail\text{-}mes\text{-}decreasing\text{-}prop[of T U A]$   $dpll$   $fin\text{-}A$  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 then 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-cl_{NOT} remove-cl_{NOT}
for
  trail ::  $\langle 'st \Rightarrow ('v, unit) ann\text{-}lits \rangle$  and
  clauses_{NOT} ::  $\langle 'st \Rightarrow 'v clauses \rangle$  and
  prepend-trail ::  $\langle ('v, unit) ann\text{-}lit \Rightarrow 'st \Rightarrow 'st \rangle$  and
  tl-trail ::  $\langle 'st \Rightarrow 'st \rangle$  and
  add-cl_{NOT} ::  $\langle 'v clause \Rightarrow 'st \Rightarrow 'st \rangle$  and
  remove-cl_{NOT} ::  $\langle 'v clause \Rightarrow 'st \Rightarrow 'st \rangle +$ 
fixes
  learn-conds ::  $\langle 'v clause \Rightarrow 'st \Rightarrow bool \rangle$ 
begin

inductive learn ::  $\langle 'st \Rightarrow 'st \Rightarrow bool \rangle$  where
  learn_{NOT}-rule:  $\langle clauses_{NOT} S \models_{pm} C \Rightarrow$ 
     $atms\text{-}of C \subseteq atms\text{-}of\text{-}mm (clauses_{NOT} S) \cup atm\text{-}of ' (lits\text{-}of\text{-}l (trail S)) \Rightarrow$ 
    learn-conds  $C S \Rightarrow$ 
     $T \sim add\text{-}cl_{NOT} C S \Rightarrow$ 
    learn  $S T \rangle$ 
inductive-cases learn_{NOT}E:  $\langle learn S T \rangle$ 

```

lemma learn- μ_C -stable:

```

  assumes  $\langle learn S T \rangle$  and  $\langle no\text{-}dup (trail S) \rangle$ 
  shows  $\langle \mu_C A B (trail\text{-}weight S) = \mu_C A B (trail\text{-}weight T) \rangle$ 
  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-cls*_{NOT} *remove-cls*_{NOT}
for
trail :: $\langle 'st \Rightarrow ('v, \text{unit}) \text{ ann-lits} \rangle$ **and**
*clauses*_{NOT} :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**
prepend-trail :: $\langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
tl-trail :: $\langle 'st \Rightarrow 'st \rangle$ **and**
*add-cls*_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
*remove-cls*_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle +$
fixes
forget-conds :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow \text{bool} \rangle$
begin

inductive *forget*_{NOT} :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **where**
*forget*_{NOT}:
 $\langle \text{removeAll-mset } C(\text{clauses}_{\text{NOT}} S) \models_{\text{pm}} C \Rightarrow$
 $\text{forget-conds } C S \Rightarrow$
 $C \in \# \text{ clauses}_{\text{NOT}} S \Rightarrow$
 $T \sim \text{remove-cls}_{\text{NOT}} C S \Rightarrow$

*forget*_{NOT} *S T*
inductive-cases *forget*_{NOT}*E*: $\langle \text{forget}_{\text{NOT}} S T \rangle$

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

locale *learn-and-forget*_{NOT} =
learn-ops *trail* *clauses*_{NOT} *prepend-trail* *tl-trail* *add-cls*_{NOT} *remove-cls*_{NOT} *learn-conds* +
forget-ops *trail* *clauses*_{NOT} *prepend-trail* *tl-trail* *add-cls*_{NOT} *remove-cls*_{NOT} *forget-conds*
for
trail :: $\langle 'st \Rightarrow ('v, \text{unit}) \text{ ann-lits} \rangle$ **and**
*clauses*_{NOT} :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**
prepend-trail :: $\langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
tl-trail :: $\langle 'st \Rightarrow 'st \rangle$ **and**
*add-cls*_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
*remove-cls*_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
learn-conds *forget-conds* :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow \text{bool} \rangle$
begin
inductive *learn-and-forget*_{NOT} :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$
where
lf-learn: $\langle \text{learn } S T \Rightarrow \text{learn-and-forget}_{\text{NOT}} S T \rangle$ |
lf-forget: $\langle \text{forget}_{\text{NOT}} S T \Rightarrow \text{learn-and-forget}_{\text{NOT}} S T \rangle$
end

Definition of CDCL

locale *conflict-driven-clause-learning-ops* =
dpll-with-backjumping-ops *trail* *clauses*_{NOT} *prepend-trail* *tl-trail* *add-cls*_{NOT} *remove-cls*_{NOT}
inv *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 :: $\langle 'st \Rightarrow ('v, unit) \text{ ann-lits} \rangle$ **and**
clauses_{NOT} :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**
prepend-trail :: $\langle ('v, unit) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
tl-trail :: $\langle 'st \Rightarrow 'st \rangle$ **and**
add-cls_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
remove-cls_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
inv :: $\langle 'st \Rightarrow bool \rangle$ **and**
decide-conds :: $\langle 'st \Rightarrow 'st \Rightarrow bool \rangle$ **and**
backjump-conds :: $\langle 'v \text{ clause} \Rightarrow 'v \text{ clause} \Rightarrow 'v \text{ literal} \Rightarrow 'st \Rightarrow 'st \Rightarrow bool \rangle$ **and**
propagate-conds :: $\langle ('v, unit) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \Rightarrow bool \rangle$ **and**
learn-conds forget-conds :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow bool \rangle$

begin

inductive *cdcl_{NOT}* :: $\langle 'st \Rightarrow 'st \Rightarrow bool \rangle$ **for** *S* :: $'st$ **where**

c-dpll-bj: $\langle dpll\text{-bj } S \ S' \Rightarrow cdcl_{NOT} \ S \ S' \rangle$ |

c-learn: $\langle learn \ S \ S' \Rightarrow cdcl_{NOT} \ S \ S' \rangle$ |

c-forget_{NOT}: $\langle forget_{NOT} \ S \ S' \Rightarrow cdcl_{NOT} \ S \ S' \rangle$

lemma *cdcl_{NOT}-all-induct*[*consumes 1, case-names dpll-bj learn forget_{NOT}*]:

fixes *S T* :: $\langle 'st \rangle$

assumes $\langle cdcl_{NOT} \ S \ T \rangle$ **and**

dpll: $\langle \bigwedge T. dpll\text{-bj } S \ T \Rightarrow P \ S \ T \rangle$ **and**

learning:

$\langle \bigwedge C \ T. clauses_{NOT} \ S \models_{pm} C \Rightarrow$

$atms\text{-of } C \subseteq atms\text{-of-mm } (clauses_{NOT} \ S) \cup atm\text{-of } ' (lits\text{-of-l } (trail \ S)) \Rightarrow$

$T \sim add\text{-cls}_{NOT} \ C \ S \Rightarrow$

$P \ S \ T \rangle$ **and**

forgetting: $\langle \bigwedge C \ T. removeAll\text{-mset } C \ (clauses_{NOT} \ S) \models_{pm} C \Rightarrow$

$C \in \# \ clauses_{NOT} \ S \Rightarrow$

$T \sim remove\text{-cls}_{NOT} \ C \ S \Rightarrow$

$P \ S \ T \rangle$

shows $\langle P \ S \ T \rangle$

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

$\langle cdcl_{NOT} \ S \ T \rangle$ **and**

$\langle inv \ S \rangle$ **and**

$\langle no\text{-dup } (trail \ S) \rangle$

shows $\langle no\text{-dup } (trail \ T) \rangle$

using *assms* **by** (*induction rule*: *cdcl_{NOT}-all-induct*) (*auto intro*: *dpll-bj-no-dup*)

Consistency of the trail lemma *cdcl_{NOT}-consistent*:

assumes

$\langle cdcl_{NOT} \ S \ T \rangle$ **and**

$\langle inv \ S \rangle$ **and**

$\langle no\text{-dup } (trail \ S) \rangle$

shows $\langle consistent\text{-interp } (lits\text{-of-l } (trail \ T)) \rangle$

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 $\langle \text{cdcl}_{NOT} S T \rangle$ **and** $\langle \text{inv } S \rangle$
shows $\langle \text{atms-of-mm } (\text{clauses}_{NOT} T) \subseteq \text{atms-of-mm } (\text{clauses}_{NOT} S) \cup \text{atm-of } ' (\text{lits-of-l } (\text{trail } S)) \rangle$
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 $\langle \text{cdcl}_{NOT} S T \rangle$ **and** $\langle \text{inv } S \rangle$
and $\langle \text{atm-of } ' (\text{lits-of-l } (\text{trail } S)) \subseteq \text{atms-of-mm } (\text{clauses}_{NOT} S) \rangle$
shows $\langle \text{atm-of } ' (\text{lits-of-l } (\text{trail } T)) \subseteq \text{atms-of-mm } (\text{clauses}_{NOT} S) \rangle$
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
 $\langle \text{cdcl}_{NOT} S T \rangle$ **and** $\langle \text{inv } S \rangle$ **and**
 $\langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq A \rangle$ **and**
 $\langle \text{atm-of } ' (\text{lits-of-l } (\text{trail } S)) \subseteq A \rangle$
shows $\langle \text{atm-of } ' (\text{lits-of-l } (\text{trail } T)) \subseteq A \rangle$
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 $\langle \text{cdcl}_{NOT} S T \rangle$ **and** $\langle \text{inv } S \rangle$ **and**
 $\langle \text{all-decomposition-implies-m } (\text{clauses}_{NOT} S) (\text{get-all-ann-decomposition } (\text{trail } S)) \rangle$
shows
 $\langle \text{all-decomposition-implies-m } (\text{clauses}_{NOT} T) (\text{get-all-ann-decomposition } (\text{trail } T)) \rangle$
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 $\langle (a, b) \in \text{set } (\text{get-all-ann-decomposition } (\text{trail } T)) \rangle$

then have $\langle \text{unmark-l } a \cup \text{set-mset } (\text{clauses}_{NOT} S) \models_{ps} \text{unmark-l } b \rangle$

using *decomp T* **by** (*auto simp add: all-decomposition-implies-def*)

moreover

have $a1: \langle C \in \text{set-mset } (\text{clauses}_{NOT} S) \rangle$

using *C* **by** *blast*

have $\langle \text{clauses}_{NOT} T = \text{clauses}_{NOT} (\text{remove-cls}_{NOT} C S) \rangle$

using *T state-eq_{NOT}-clauses* **by** *blast*

then have $\langle \text{set-mset } (\text{clauses}_{NOT} T) \models_{ps} \text{set-mset } (\text{clauses}_{NOT} S) \rangle$

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  $\langle \text{unmark-}l \ a \cup \text{set-mset} \ ( \text{clauses}_{NOT} \ T) \rangle$ 
   $\models_{ps} \text{unmark-}l \ b$ 
  using true-clss-clss-generalise-true-clss-clss by blast
qed
qed

Extension of models lemma cdclNOT-bj-sat-ext-iff:
  assumes  $\langle \text{cdcl}_{NOT} \ S \ T \rangle$  and  $\langle \text{inv} \ S \rangle$ 
  shows  $\langle I \models_{sextm} \text{clauses}_{NOT} \ S \longleftrightarrow I \models_{sextm} \text{clauses}_{NOT} \ T \rangle$ 
  using assms
proof (induction rule:cdclNOT-all-induct)
  case dpll-bj
  then show ?case by (simp add: dpll-bj-clauses)
next
case (learn C T) note  $T = \text{this}(3)$ 
{ fix J
  assume
     $\langle I \models_{sextm} \text{clauses}_{NOT} \ S \rangle$  and
     $\langle I \subseteq J \rangle$  and
    tot:  $\langle \text{total-over-m } J \ (\text{set-mset} \ (\text{add-mset } C \ (\text{clauses}_{NOT} \ S))) \rangle$  and
    cons:  $\langle \text{consistent-interp } J \rangle$ 
  then have  $\langle J \models_{sm} \text{clauses}_{NOT} \ S \rangle$  unfolding true-clss-ext-def by auto

  moreover
    with  $\langle \text{clauses}_{NOT} \ S \models_{pm} C \rangle$  have  $\langle J \models C \rangle$ 
    using tot cons unfolding true-clss-clss-def by auto
    ultimately have  $\langle J \models_{sm} \{\#C\} + \text{clauses}_{NOT} \ S \rangle$  by auto
  }
  then have H:  $\langle I \models_{sextm} (\text{clauses}_{NOT} \ S) \implies I \models_{sext} \text{insert } C \ (\text{set-mset} \ (\text{clauses}_{NOT} \ S)) \rangle$ 
    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 (forgetNOT C T) note cls-C = this(1) and  $T = \text{this}(3)$ 
{ fix J
  assume
     $\langle I \models_{sext} \text{set-mset} \ (\text{clauses}_{NOT} \ S) - \{C\} \rangle$  and
     $\langle I \subseteq J \rangle$  and
    tot:  $\langle \text{total-over-m } J \ (\text{set-mset} \ (\text{clauses}_{NOT} \ S)) \rangle$  and
    cons:  $\langle \text{consistent-interp } J \rangle$ 
  then have  $\langle J \models_s \text{set-mset} \ (\text{clauses}_{NOT} \ S) - \{C\} \rangle$ 
    unfolding true-clss-ext-def by (meson Diff-subset total-over-m-subset)

  moreover
    with cls-C have  $\langle J \models C \rangle$ 
    using tot cons unfolding true-clss-clss-def
    by (metis Un-commute forgetNOT.hyps(2) insert-Diff insert-is-Un order-refl
      set-mset-minus-replicate-mset(1))
    ultimately have  $\langle J \models_{sm} (\text{clauses}_{NOT} \ S) \rangle$  by (metis insert-Diff-single true-clss-insert)
  }
  then have H:  $\langle I \models_{sext} \text{set-mset} \ (\text{clauses}_{NOT} \ S) - \{C\} \implies I \models_{sextm} (\text{clauses}_{NOT} \ S) \rangle$ 
    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 $cdcl_{NOT}\text{-inv}$: $\langle \bigwedge S\ T. cdcl_{NOT}\ S\ T \implies inv\ S \implies inv\ T \rangle$
begin
sublocale *dpll-with-backjumping*
apply *unfold-locales*
using $cdcl_{NOT}.simps\ cdcl_{NOT}\text{-inv}$ **by** auto

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

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

lemma *rtranclp-cdcl_{NOT}-trail-clauses-bound*:
assumes
 $cdcl$: $\langle cdcl_{NOT}^{**}\ S\ T \rangle$ **and**
 inv : $\langle inv\ S \rangle$ **and**
 $atms\text{-clauses}\text{-}S$: $\langle atms\text{-of}\text{-}mm\ (clauses_{NOT}\ S) \subseteq A \rangle$ **and**
 $atms\text{-trail}\text{-}S$: $\langle atm\text{-of}\ ('(lits\text{-of}\text{-}l\ (trail\ S)) \subseteq A) \rangle$
shows $\langle atm\text{-of}\ ('(lits\text{-of}\text{-}l\ (trail\ T)) \subseteq A \wedge atms\text{-of}\text{-}mm\ (clauses_{NOT}\ T) \subseteq A \rangle$
using $cdcl$
proof (induction rule: *rtranclp-induct*)
case *base*
then show ?case **using** $atms\text{-clauses}\text{-}S\ atms\text{-trail}\text{-}S$ **by** *simp*
next
case (*step* $T\ U$) **note** $st = this(1)$ **and** $cdcl_{NOT} = this(2)$ **and** $IH = this(3)$
have $\langle inv\ T \rangle$ **using** $inv\ st\ rtranclp\text{-}cdcl_{NOT}\text{-inv}$ **by** *blast*
have $\langle atms\text{-of}\text{-}mm\ (clauses_{NOT}\ U) \subseteq A \rangle$
using $cdcl_{NOT}\text{-}atms\text{-of}\text{-}ms\text{-clauses}\text{-decreasing}[OF\ cdcl_{NOT}]\ IH\ \langle inv\ T \rangle$ **by** *fast*
moreover
have $\langle atm\text{-of}\ ('(lits\text{-of}\text{-}l\ (trail\ U)) \subseteq A) \rangle$
using $cdcl_{NOT}\text{-}atms\text{-in}\text{-trail}\text{-in}\text{-set}[OF\ cdcl_{NOT},\ of\ A]$
by (*meson* $atms\text{-trail}\text{-}S\ atms\text{-clauses}\text{-}S\ IH\ \langle inv\ T \rangle\ cdcl_{NOT}$)
ultimately show ?case **by** *fast*
qed

lemma *rtranclp-cdcl_{NOT}-all-decomposition-implies*:
assumes $\langle cdcl_{NOT}^{**}\ S\ T \rangle$ **and** $\langle inv\ S \rangle$ **and** $\langle no\text{-dup}\ (trail\ S) \rangle$ **and**
 $\langle all\text{-decomposition}\text{-implies}\text{-}m\ (clauses_{NOT}\ S)\ (get\text{-all}\text{-ann}\text{-decomposition}\ (trail\ S)) \rangle$
shows
 $\langle all\text{-decomposition}\text{-implies}\text{-}m\ (clauses_{NOT}\ T)\ (get\text{-all}\text{-ann}\text{-decomposition}\ (trail\ T)) \rangle$
using *assms* **by** (induction)
(auto intro: $rtranclp\text{-}cdcl_{NOT}\text{-inv}\ cdcl_{NOT}\text{-all}\text{-decomposition}\text{-implies}\ rtranclp\text{-}cdcl_{NOT}\text{-no}\text{-dup}$)

lemma *rtrancpl-cdcl_{NOT}-bj-sat-ext-iff*:
assumes $\langle \text{cdcl}_{NOT}^{**} S T \rangle$ **and** $\langle \text{inv } S \rangle$
shows $\langle I \models_{\text{sextm}} \text{clauses}_{NOT} S \longleftrightarrow I \models_{\text{sextm}} \text{clauses}_{NOT} T \rangle$
using *assms apply* (*induction rule: rtrancpl-induct*)
using *cdcl_{NOT}-bj-sat-ext-iff* **by** (*auto intro: rtrancpl-cdcl_{NOT}-inv rtrancpl-cdcl_{NOT}-no-dup*)

definition *cdcl_{NOT}-NOT-all-inv* **where**
 $\langle \text{cdcl}_{NOT}\text{-NOT-all-inv } A S \longleftrightarrow (\text{finite } A \wedge \text{inv } S \wedge \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A$
 $\wedge \text{atm-of ' lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A \wedge \text{no-dup } (\text{trail } S)) \rangle$

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

abbreviation *learn-or-forget* **where**
 $\langle \text{learn-or-forget } S T \equiv \text{learn } S T \vee \text{forget}_{NOT} S T \rangle$

lemma *rtrancpl-learn-or-forget-cdcl_{NOT}*:
 $\langle \text{learn-or-forget}^{**} S T \implies \text{cdcl}_{NOT}^{**} S T \rangle$
using *rtrancpl-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: $\langle \text{learn-or-forget}^{**} S T \rangle$ **and**
dpll: $\langle \text{dpll-bj } T U \rangle$ **and**
inv: $\langle \text{cdcl}_{NOT}\text{-NOT-all-inv } A S \rangle$
shows $\langle (2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A))$
 $- \mu_C (1 + \text{card } (\text{atms-of-ms } A)) (2 + \text{card } (\text{atms-of-ms } A)) (\text{trail-weight } U)$
 $< (2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A))$
 $- \mu_C (1 + \text{card } (\text{atms-of-ms } A)) (2 + \text{card } (\text{atms-of-ms } A)) (\text{trail-weight } S) \rangle$
(is $\langle ?\mu U < ?\mu S \rangle$)

proof –
have $\langle ?\mu S = ?\mu T \rangle$
using *l-f*
proof (*induction*)
case *base*
then show *?case* **by** *simp*
next
case (*step* *T U*)
moreover then have $\langle \text{no-dup } (\text{trail } T) \rangle$
using *rtrancpl-cdcl_{NOT}-no-dup[of S T]* *cdcl_{NOT}-NOT-all-inv-def inv*
rtrancpl-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 $\langle \text{cdcl}_{NOT}\text{-NOT-all-inv } A T \rangle$
using *rtrancpl-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
  ⟨ $\bigwedge i. \text{cdcl}_{NOT} (f\ i) (f (Suc\ i))$ ⟩ and
   $\text{inv}: \langle \text{cdcl}_{NOT-} \text{NOT-all-inv } A\ (f\ 0) \rangle$ 
shows  $\langle \exists j. \forall i \geq j. \text{learn-or-forget } (f\ i) (f (Suc\ i)) \rangle$ 
using assms
proof (induction  $\langle (2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A))$ 
   $- \mu_C (1 + \text{card } (\text{atms-of-ms } A)) (2 + \text{card } (\text{atms-of-ms } A)) (\text{trail-weight } (f\ 0)) \rangle$ 
  arbitrary: f
  rule: nat-less-induct-case)
case (Suc n) note  $IH = \text{this}(1)$  and  $\mu = \text{this}(2)$  and  $\text{cdcl}_{NOT} = \text{this}(3)$  and  $\text{inv} = \text{this}(4)$ 
consider
  (dpll-end)  $\langle \exists j. \forall i \geq j. \text{learn-or-forget } (f\ i) (f (Suc\ i)) \rangle$ 
  | (dpll-more)  $\langle \neg (\exists j. \forall i \geq j. \text{learn-or-forget } (f\ i) (f (Suc\ i))) \rangle$ 
  by blast
then show ?case
proof cases
  case dpll-end
  then show ?thesis by auto
next
  case dpll-more
  then have  $j: \langle \exists i. \neg \text{learn } (f\ i) (f (Suc\ i)) \wedge \neg \text{forget}_{NOT} (f\ i) (f (Suc\ i)) \rangle$ 
  by blast
  obtain i where
    i-learn-forget:  $\langle \neg \text{learn } (f\ i) (f (Suc\ i)) \wedge \neg \text{forget}_{NOT} (f\ i) (f (Suc\ i)) \rangle$  and
     $\langle \forall k < i. \text{learn-or-forget } (f\ k) (f (Suc\ k)) \rangle$ 
  proof  $-$ 
    obtain  $i_0$  where  $\langle \neg \text{learn } (f\ i_0) (f (Suc\ i_0)) \wedge \neg \text{forget}_{NOT} (f\ i_0) (f (Suc\ i_0)) \rangle$ 
    using j by auto
    then have  $\langle \{i. i \leq i_0 \wedge \neg \text{learn } (f\ i) (f (Suc\ i)) \wedge \neg \text{forget}_{NOT} (f\ i) (f (Suc\ i)) \} \neq \{\} \rangle$ 
    by auto
    let  $?I = \langle \{i. i \leq i_0 \wedge \neg \text{learn } (f\ i) (f (Suc\ i)) \wedge \neg \text{forget}_{NOT} (f\ i) (f (Suc\ i)) \} \rangle$ 
    let  $?i = \text{Min } ?I$ 
    have  $\langle \text{finite } ?I \rangle$ 
    by auto
    have  $\langle \neg \text{learn } (f\ ?i) (f (Suc\ ?i)) \wedge \neg \text{forget}_{NOT} (f\ ?i) (f (Suc\ ?i)) \rangle$ 
    using Min-in[OF finite ?I] ?I ≠ {} by auto
    moreover have  $\langle \forall k < ?i. \text{learn-or-forget } (f\ k) (f (Suc\ k)) \rangle$ 
    using Min.coboundedI[of {i. i ≤ i0 ∧ ¬ learn (f i) (f (Suc i)) ∧ ¬ forgetNOT (f i) (f (Suc i))}, simplified]
    by (meson  $\langle \neg \text{learn } (f\ i_0) (f (Suc\ i_0)) \wedge \neg \text{forget}_{NOT} (f\ i_0) (f (Suc\ i_0)) \rangle$  less-imp-le
      dual-order.trans not-le)
    ultimately show ?thesis using that by blast
  qed
  define g where  $\langle g = (\lambda n. f\ (n + Suc\ i)) \rangle$ 
  have  $\langle \text{dpll-bj } (f\ i) (g\ 0) \rangle$ 
  using i-learn-forget cdclNOT cdclNOT.cases unfolding g-def by auto
  {
    fix j
    assume  $\langle j \leq i \rangle$ 
    then have  $\langle \text{learn-or-forget}^{**} (f\ 0) (f\ j) \rangle$ 
    apply (induction j)
    apply simp
    by (metis (no-types, lifting) Suc-leD Suc-le-lessD rtranclp.simps
       $\langle \forall k < i. \text{learn } (f\ k) (f (Suc\ k)) \vee \text{forget}_{NOT} (f\ k) (f (Suc\ k)) \rangle$ )
  }
  then have  $\langle \text{learn-or-forget}^{**} (f\ 0) (f\ i) \rangle$  by blast

```

then have $\langle (2 + \text{card}(\text{atms-of-ms } A)) \wedge (1 + \text{card}(\text{atms-of-ms } A))$
 $- \mu_C (1 + \text{card}(\text{atms-of-ms } A)) (2 + \text{card}(\text{atms-of-ms } A)) (\text{trail-weight } (g \ 0))$
 $< (2 + \text{card}(\text{atms-of-ms } A)) \wedge (1 + \text{card}(\text{atms-of-ms } A))$
 $- \mu_C (1 + \text{card}(\text{atms-of-ms } A)) (2 + \text{card}(\text{atms-of-ms } A)) (\text{trail-weight } (f \ 0)) \rangle$
using *learn-or-forget-dpll- μ_C* [*of* $\langle f \ 0 \rangle \langle f \ i \rangle \langle g \ 0 \rangle A$] *inv* $\langle \text{dpll-bj } (f \ i) (g \ 0) \rangle$
unfolding *cdcl_{NOT}-NOT-all-inv-def* **by** *linarith*

moreover have *cdcl_{NOT}-i*: $\langle \text{cdcl}_{NOT}^{**} (f \ 0) (g \ 0) \rangle$
using *rtrancp-learn-or-forget-cdcl_{NOT}*[*of* $\langle f \ 0 \rangle \langle f \ i \rangle \langle \text{learn-or-forget}^{**} (f \ 0) (f \ i) \rangle$
 $\text{cdcl}_{NOT}[\text{of } i]$ **unfolding** *g-def* **by** *auto*

moreover have $\langle \bigwedge i. \text{cdcl}_{NOT} (g \ i) (g \ (\text{Suc } i)) \rangle$
using *cdcl_{NOT} g-def* **by** *auto*

moreover have $\langle \text{cdcl}_{NOT}\text{-NOT-all-inv } A (g \ 0) \rangle$
using *inv cdcl_{NOT}-i rtrancp-cdcl_{NOT}-trail-clauses-bound g-def cdcl_{NOT}-NOT-all-inv* **by** *auto*

ultimately obtain *j* **where** *j*: $\langle \bigwedge i. i \geq j \implies \text{learn-or-forget } (g \ i) (g \ (\text{Suc } i)) \rangle$
using *IH* **unfolding** $\mu[\text{symmetric}]$ **by** *presburger*

show *?thesis*

proof

$\{$
 $\quad \text{fix } k$
 $\quad \text{assume } \langle k \geq j + \text{Suc } i \rangle$
 $\quad \text{then have } \langle \text{learn-or-forget } (f \ k) (f \ (\text{Suc } k)) \rangle$
 $\quad \quad \text{using } j[\text{of } \langle k - \text{Suc } i \rangle]$ **unfolding** *g-def* **by** *auto*
 $\quad \}$

then show $\langle \forall k \geq j + \text{Suc } i. \text{learn-or-forget } (f \ k) (f \ (\text{Suc } k)) \rangle$
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 $\neg ?\text{case}$

then have *j*: $\langle \exists i. \neg \text{learn } (f \ i) (f \ (\text{Suc } i)) \wedge \neg \text{forget}_{NOT} (f \ i) (f \ (\text{Suc } i)) \rangle$
by *blast*

obtain *i* **where**
 $\langle \neg \text{learn } (f \ i) (f \ (\text{Suc } i)) \wedge \neg \text{forget}_{NOT} (f \ i) (f \ (\text{Suc } i)) \rangle$ **and**
 $\langle \forall k < i. \text{learn-or-forget } (f \ k) (f \ (\text{Suc } k)) \rangle$

proof –

obtain *i*₀ **where** $\langle \neg \text{learn } (f \ i_0) (f \ (\text{Suc } i_0)) \wedge \neg \text{forget}_{NOT} (f \ i_0) (f \ (\text{Suc } i_0)) \rangle$
using *j* **by** *auto*

then have $\langle \{i. i \leq i_0 \wedge \neg \text{learn } (f \ i) (f \ (\text{Suc } i)) \wedge \neg \text{forget}_{NOT} (f \ i) (f \ (\text{Suc } i))\} \neq \{\} \rangle$
by *auto*

let *?I* = $\langle \{i. i \leq i_0 \wedge \neg \text{learn } (f \ i) (f \ (\text{Suc } i)) \wedge \neg \text{forget}_{NOT} (f \ i) (f \ (\text{Suc } i))\} \rangle$

let *?i* = *Min ?I*

have $\langle \text{finite } ?I \rangle$
by *auto*

have $\langle \neg \text{learn } (f \ ?i) (f \ (\text{Suc } ?i)) \wedge \neg \text{forget}_{NOT} (f \ ?i) (f \ (\text{Suc } ?i)) \rangle$
using *Min-in[OF finite ?I] ?I ≠ {}* **by** *auto*

moreover have $\langle \forall k < ?i. \text{learn-or-forget } (f \ k) (f \ (\text{Suc } k)) \rangle$
using *Min.coboundedI*[*of* $\langle \{i. i \leq i_0 \wedge \neg \text{learn } (f \ i) (f \ (\text{Suc } i)) \wedge \neg \text{forget}_{NOT} (f \ i) (f \ (\text{Suc } i))\} \rangle$, *simplified*]

by (*meson* $\langle \neg \text{learn } (f \ i_0) (f \ (\text{Suc } i_0)) \wedge \neg \text{forget}_{NOT} (f \ i_0) (f \ (\text{Suc } i_0)) \rangle$ *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)) ∧ ¬ forgetNOT (f i) (f (Suc i))⟩ cdclNOT cdclNOT.cases
  by blast
{
  fix j
  assume ⟨j ≤ i⟩
  then have ⟨learn-or-forget** (f 0) (f j)⟩
    apply (induction j)
    apply simp
    by (metis (no-types, lifting) Suc-leD Suc-le-lessD rtrancpl.simps
      ⟨∀ k < i. learn (f k) (f (Suc k)) ∨ forgetNOT (f k) (f (Suc k))⟩)
  }
then have ⟨learn-or-forget** (f 0) (f i)⟩ by blast

then show False
  using learn-or-forget-dpll-μC[of ⟨f 0⟩ ⟨f i⟩ ⟨f (Suc i)⟩ A] inv 0
  ⟨dpll-bj (f i) (f (Suc i))⟩ unfolding cdclNOT-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). cdclNOT S T ∧ cdclNOT-NOT-all-inv A S}⟩
  (is ⟨wf {(T, S). cdclNOT S T ∧ ?inv S}⟩)
unfolding wf-iff-no-infinite-down-chain
proof (rule ccontr)
  assume ⟨¬ ¬ (∃ f. ∀ i. (f (Suc i), f i) ∈ {(T, S). cdclNOT S T ∧ ?inv S})⟩
  then obtain f where
    ⟨∀ i. cdclNOT (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-cdclNOT-exists-learn-and-forget-infinite-chain[of f] by meson
  then show False using no-infinite-lf by blast
qed

```

lemma *inv-and-trancpl-cdcl_{NOT}-trancpl-cdcl_{NOT}-and-inv:*

```

⟨cdclNOT++ S T ∧ cdclNOT-NOT-all-inv A S ⟷ (λS T. cdclNOT S T ∧ cdclNOT-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: trancpl.r-into-trancpl)
  by (subst trancpl.simps) (auto intro: cdclNOT-NOT-all-inv trancpl-into-rtrancpl)
next
assume ?B
then have ?A by induction auto
moreover have ?I using ⟨?B⟩ trancplD by fastforce
ultimately show ⟨?A ∧ ?I⟩ by blast
qed

```

lemma *wf-trancpl-cdcl_{NOT}-no-learn-and-forget-infinite-chain:*

assumes

```

  no-infinite-lf:  $\langle \bigwedge f j. \neg (\forall i \geq j. \text{learn-or-forget } (f i) (f (\text{Suc } i))) \rangle$ 
shows  $\langle \text{wf } \{(T, S). \text{cdcl}_{NOT}^{++} S T \wedge \text{cdcl}_{NOT-NOT-all-inv} A S\} \rangle$ 
using wf-trancl[OF wf-cdclNOT-no-learn-and-forget-infinite-chain[OF no-infinite-lf]]
apply (rule wf-subset)
by (auto simp: trancl-set-tranclp inv-and-tranclp-cdclNOT-tranclp-cdclNOT-and-inv)

lemma cdclNOT-final-state:
assumes
  n-s:  $\langle \text{no-step cdcl}_{NOT} S \rangle$  and
  inv:  $\langle \text{cdcl}_{NOT-NOT-all-inv} A S \rangle$  and
  decomp:  $\langle \text{all-decomposition-implies-m } (\text{clauses}_{NOT} S) (\text{get-all-ann-decomposition } (\text{trail } S)) \rangle$ 
shows  $\langle \text{unsatisfiable } (\text{set-mset } (\text{clauses}_{NOT} S)) \rangle$ 
   $\vee (\text{trail } S \models_{asm} \text{clauses}_{NOT} S \wedge \text{satisfiable } (\text{set-mset } (\text{clauses}_{NOT} S))) \rangle$ 
proof –
  have n-s':  $\langle \text{no-step dpll-bj } S \rangle$ 
    using n-s by (auto simp: cdclNOT.simps)
  show ?thesis
    apply (rule dpll-backjump-final-state[of S A])
    using inv decomp n-s' unfolding cdclNOT-NOT-all-inv-def by auto
qed

lemma full-cdclNOT-final-state:
assumes
  full:  $\langle \text{full cdcl}_{NOT} S T \rangle$  and
  inv:  $\langle \text{cdcl}_{NOT-NOT-all-inv} A S \rangle$  and
  n-d:  $\langle \text{no-dup } (\text{trail } S) \rangle$  and
  decomp:  $\langle \text{all-decomposition-implies-m } (\text{clauses}_{NOT} S) (\text{get-all-ann-decomposition } (\text{trail } S)) \rangle$ 
shows  $\langle \text{unsatisfiable } (\text{set-mset } (\text{clauses}_{NOT} T)) \rangle$ 
   $\vee (\text{trail } T \models_{asm} \text{clauses}_{NOT} T \wedge \text{satisfiable } (\text{set-mset } (\text{clauses}_{NOT} T))) \rangle$ 
proof –
  have st:  $\langle \text{cdcl}_{NOT}^{**} S T \rangle$  and n-s:  $\langle \text{no-step cdcl}_{NOT} T \rangle$ 
    using full unfolding full-def by blast+
  have n-s':  $\langle \text{cdcl}_{NOT-NOT-all-inv} A T \rangle$ 
    using cdclNOT-NOT-all-inv inv st by blast
  moreover have  $\langle \text{all-decomposition-implies-m } (\text{clauses}_{NOT} T) (\text{get-all-ann-decomposition } (\text{trail } T)) \rangle$ 
    using cdclNOT-NOT-all-inv-def decomp inv rtranclp-cdclNOT-all-decomposition-implies st by auto
  ultimately show ?thesis
    using cdclNOT-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-cl_{NOT} remove-cl_{NOT} +
 conflict-driven-clause-learning trail clauses_{NOT} prepend-trail tl-trail add-cl_{NOT} remove-cl_{NOT}

```

    inv decide-conds backjump-conds propagate-conds
  ⟨λC S. distinct-mset C ∧ ¬tautology C ∧ learn-restrictions C S ∧
    (∃ F K d F' C' L. trail S = F' @ Decided K # F ∧ C = add-mset L C' ∧ F ⊨as CNot C'
      ∧ add-mset L C' ⊄# clausesNOT S)⟩
  ⟨λC S. ¬(∃ F' F K d L. trail S = F' @ Decided K # F ∧ F ⊨as CNot (remove1-mset L C))
    ∧ forget-restrictions C S⟩
  for
    trail :: ⟨'st ⇒ ('v, unit) ann-lits⟩ and
    clausesNOT :: ⟨'st ⇒ 'v clauses⟩ and
    prepend-trail :: ⟨('v, unit) ann-lit ⇒ 'st ⇒ 'st⟩ and
    tl-trail :: ⟨'st ⇒ 'st⟩ and
    add-clNOT :: ⟨'v clause ⇒ 'st ⇒ 'st⟩ and
    remove-clNOT :: ⟨'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 cdclNOT-learn-all-induct[consumes 1, case-names dpll-bj learn forgetNOT]:
  fixes S T :: ⟨'st⟩
  assumes ⟨cdclNOT S T⟩ and
  dpll: ⟨λT. dpll-bj S T ⇒ P S T⟩ and
  learning:
    ⟨λC F K F' C' L T. clausesNOT S ⊨pm C ⇒
      atms-of C ⊆ atms-of-mm (clausesNOT S) ∪ atm-of ' (lits-of-l (trail S)) ⇒
      distinct-mset C ⇒
      ¬ tautology C ⇒
      learn-restrictions C S ⇒
      trail S = F' @ Decided K # F ⇒
      C = add-mset L C' ⇒
      F ⊨as CNot C' ⇒
      add-mset L C' ⊄# clausesNOT S ⇒
      T ~ add-clNOT C S ⇒
      P S T⟩ and
  forgetting: ⟨λC T. removeAll-mset C (clausesNOT S) ⊨pm C ⇒
    C ∈# clausesNOT S ⇒
    ¬(∃ F' F K L. trail S = F' @ Decided K # F ∧ F ⊨as CNot (C - {#L#})) ⇒
    T ~ remove-clNOT C S ⇒
    forget-restrictions C S ⇒
    P S T⟩
  shows ⟨P S T⟩
using assms(1)
apply (induction rule: cdclNOT.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.forgetNOT.cases[OF forget-ops-axioms] dest!: assms(4))
done

```

```

lemma rtranclp-cdclNOT-inv:
  ⟨cdclNOT** S T ⇒ inv S ⇒ inv T⟩
  apply (induction rule: rtranclp-induct)
  apply simp
  using cdclNOT-inv unfolding conflict-driven-clause-learning-def
  conflict-driven-clause-learning-axioms-def by blast

```

lemma *learn-always-simple-clauses*:

assumes

learn: $\langle \text{learn } S \ T \rangle$ **and**

n-d: $\langle \text{no-dup } (\text{trail } S) \rangle$

shows $\langle \text{set-mset } (\text{clauses}_{NOT} \ T - \text{clauses}_{NOT} \ S) \rangle$

$\subseteq \text{simple-clss } (\text{atms-of-mm } (\text{clauses}_{NOT} \ S) \cup \text{atm-of } ' \text{ lits-of-l } (\text{trail } S)) \rangle$

proof

fix *C* **assume** *C*: $\langle C \in \text{set-mset } (\text{clauses}_{NOT} \ T - \text{clauses}_{NOT} \ S) \rangle$

have $\langle \text{distinct-mset } C \rangle \langle \neg \text{tautology } C \rangle$ **using** *learn* *C* *n-d* **by** $(\text{elim } \text{learn}_{NOT} E; \text{auto}) +$

then have $\langle C \in \text{simple-clss } (\text{atms-of } C) \rangle$

using *distinct-mset-not-tautology-implies-in-simple-clss* **by** *blast*

moreover have $\langle \text{atms-of } C \subseteq \text{atms-of-mm } (\text{clauses}_{NOT} \ S) \cup \text{atm-of } ' \text{ lits-of-l } (\text{trail } S) \rangle$

using *learn* *C* *n-d* **by** $(\text{elim } \text{learn}_{NOT} E) (\text{auto } \text{simp: } \text{atms-of-ms-def } \text{atms-of-def } \text{image-Un } \text{true-annots-CNot-all-atms-defined})$

moreover have $\langle \text{finite } (\text{atms-of-mm } (\text{clauses}_{NOT} \ S) \cup \text{atm-of } ' \text{ lits-of-l } (\text{trail } S)) \rangle$

by *auto*

ultimately show $\langle C \in \text{simple-clss } (\text{atms-of-mm } (\text{clauses}_{NOT} \ S) \cup \text{atm-of } ' \text{ lits-of-l } (\text{trail } S)) \rangle$

using *simple-clss-mono* **by** $(\text{metis } (\text{no-types}) \text{insert-subset } \text{mk-disjoint-insert})$

qed

definition $\langle \text{conflicting-bj-clss } S \equiv$

$\{C + \{\#L\# \} \mid C \ L. \ C + \{\#L\# \} \in \# \text{ clauses}_{NOT} \ S \wedge \text{distinct-mset } (C + \{\#L\# \})$

$\wedge \neg \text{tautology } (C + \{\#L\# \})$

$\wedge (\exists F' \ K \ F. \ \text{trail } S = F' @ \text{Decided } K \ \# \ F \wedge F \models_{as} CNot \ C) \rangle$

lemma *conflicting-bj-clss-remove-clss_{NOT}[simp]*:

$\langle \text{conflicting-bj-clss } (\text{remove-clss}_{NOT} \ C \ S) = \text{conflicting-bj-clss } S - \{C\} \rangle$

unfolding *conflicting-bj-clss-def* **by** *fastforce*

lemma *conflicting-bj-clss-remove-clss'_{NOT}[simp]*:

$\langle T \sim \text{remove-clss}_{NOT} \ C \ S \implies \text{conflicting-bj-clss } T = \text{conflicting-bj-clss } S - \{C\} \rangle$

unfolding *conflicting-bj-clss-def* **by** *fastforce*

lemma *conflicting-bj-clss-add-clss_{NOT}-state-eq*:

assumes

T: $\langle T \sim \text{add-clss}_{NOT} \ C' \ S \rangle$ **and**

n-d: $\langle \text{no-dup } (\text{trail } S) \rangle$

shows $\langle \text{conflicting-bj-clss } T$

$= \text{conflicting-bj-clss } S$

$\cup (\text{if } \exists C \ L. \ C' = \text{add-mset } L \ C \wedge \text{distinct-mset } (\text{add-mset } L \ C) \wedge \neg \text{tautology } (\text{add-mset } L \ C)$

$\wedge (\exists F' \ K \ d \ F. \ \text{trail } S = F' @ \text{Decided } K \ \# \ F \wedge F \models_{as} CNot \ C)$

$\text{then } \{C'\} \text{ else } \{\}) \rangle$

proof –

define *P* **where** $\langle P = (\lambda C \ L \ T. \ \text{distinct-mset } (\text{add-mset } L \ C) \wedge \neg \text{tautology } (\text{add-mset } L \ C) \wedge$

$(\exists F' \ K \ F. \ \text{trail } T = F' @ \text{Decided } K \ \# \ F \wedge F \models_{as} CNot \ C)) \rangle$

have *conf*: $\langle \bigwedge T. \ \text{conflicting-bj-clss } T = \{ \text{add-mset } L \ C \mid C \ L. \ \text{add-mset } L \ C \in \# \text{ clauses}_{NOT} \ T \wedge P \ C \ L \ T \} \rangle$

unfolding *conflicting-bj-clss-def* *P-def* **by** *auto*

have *P-S-T*: $\langle \bigwedge C \ L. \ P \ C \ L \ T = P \ C \ L \ S \rangle$

using *T* *n-d* **unfolding** *P-def* **by** *auto*

have *P*: $\langle \text{conflicting-bj-clss } T = \{ \text{add-mset } L \ C \mid C \ L. \ \text{add-mset } L \ C \in \# \text{ clauses}_{NOT} \ S \wedge P \ C \ L \ T \} \cup$

$\{ \text{add-mset } L \ C \mid C \ L. \ \text{add-mset } L \ C \in \# \{ \#C'\# \} \wedge P \ C \ L \ T \} \rangle$

using *T* *n-d* **unfolding** *conf* **by** *auto*

moreover have $\langle \{ \text{add-mset } L \ C \mid C \ L. \ \text{add-mset } L \ C \in \# \text{ clauses}_{NOT} \ S \wedge P \ C \ L \ T \} = \text{conflicting-bj-clss } S \rangle$

using T n -d unfolding P -def *conflicting-bj-clss-def* by *auto*
 moreover have $\langle \{ \text{add-mset } L \ C \mid C \ L. \ \text{add-mset } L \ C \in \# \ \{ \#C' \# \} \wedge P \ C \ L \ T \} =$
 $\langle \text{if } \exists C \ L. \ C' = \text{add-mset } L \ C \wedge P \ C \ L \ S \text{ then } \{ C' \} \text{ else } \{ \} \rangle$
 using n -d T by (*force simp: P-S-T*)
 ultimately show ?thesis unfolding P -def by *presburger*
 qed

lemma *conflicting-bj-clss-add-clss_{NOT}*:
 $\langle \text{no-dup } (\text{trail } S) \implies$
 $\text{conflicting-bj-clss } (\text{add-clss}_{NOT} \ C' \ S)$
 $= \text{conflicting-bj-clss } S$
 $\cup \langle \text{if } \exists C \ L. \ C' = C + \{ \#L \# \} \wedge \text{distinct-mset } (C + \{ \#L \# \}) \wedge \neg \text{tautology } (C + \{ \#L \# \})$
 $\wedge (\exists F' \ K \ d \ F. \ \text{trail } S = F' @ \text{Decided } K \ \# \ F \wedge F \models_{as} C \text{Not } C)$
 $\text{then } \{ C' \} \text{ else } \{ \} \rangle$
 using *conflicting-bj-clss-add-clss_{NOT}-state-eq* by *auto*

lemma *conflicting-bj-clss-incl-clauses*:
 $\langle \text{conflicting-bj-clss } S \subseteq \text{set-mset } (\text{clauses}_{NOT} \ S) \rangle$
 unfolding *conflicting-bj-clss-def* by *auto*

lemma *finite-conflicting-bj-clss[simp]*:
 $\langle \text{finite } (\text{conflicting-bj-clss } S) \rangle$
 using *conflicting-bj-clss-incl-clauses*[of S] *rev-finite-subset* by *blast*

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

abbreviation $\langle \text{conflicting-bj-clss-yet } b \ S \equiv$
 $\exists \wedge b - \text{card } (\text{conflicting-bj-clss } S) \rangle$

abbreviation $\mu_L :: \langle \text{nat} \Rightarrow 'st \Rightarrow \text{nat} \times \text{nat} \rangle$ **where**
 $\langle \mu_L \ b \ S \equiv (\text{conflicting-bj-clss-yet } b \ S, \text{card } (\text{set-mset } (\text{clauses}_{NOT} \ S))) \rangle$

lemma *do-not-forget-before-backtrack-rule-clause-learned-clause-untouched*:
 assumes $\langle \text{forget}_{NOT} \ S \ T \rangle$
 shows $\langle \text{conflicting-bj-clss } S = \text{conflicting-bj-clss } T \rangle$
 using *assms* apply (*elim forget_{NOT}E*)
 apply *rule*
 apply (*subst conflicting-bj-clss-remove-clss_{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}*: $\langle \text{forget}_{NOT} \ S \ T \rangle$
 shows $\langle (\mu_L \ b \ T, \mu_L \ b \ S) \in \text{less-than } < *lex* > \text{ less-than} \rangle$
proof –
 have $\langle \text{card } (\text{set-mset } (\text{clauses}_{NOT} \ S)) > 0 \rangle$
 using *forget_{NOT}* by (*elim forget_{NOT}E*) (*auto simp: size-mset-removeAll-mset-le-iff card-gt-0-iff*)
 then have $\langle \text{card } (\text{set-mset } (\text{clauses}_{NOT} \ T)) < \text{card } (\text{set-mset } (\text{clauses}_{NOT} \ S)) \rangle$
 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*:

$\langle \{a. (a = b \vee Q a) \wedge S a\} = (if\ S\ b\ then\ \{b\}\ else\ \{\}) \cup \{a. Q\ a \wedge S\ a\} \rangle$
by *auto*

lemma *set-insert-neg*:

$\langle A \neq insert\ a\ A \longleftrightarrow a \notin A \rangle$
by *auto*

lemma *learn- μ_L -decrease*:

assumes *learnST*: $\langle learn\ S\ T \rangle$ **and** *n-d*: $\langle no_dup\ (trail\ S) \rangle$ **and**
A: $\langle atms_of_mm\ (clauses_{NOT}\ S) \cup atm_of\ 'lits_of_l\ (trail\ S) \subseteq A \rangle$ **and**
fin-A: $\langle finite\ A \rangle$
shows $\langle (\mu_L\ (card\ A)\ T, \mu_L\ (card\ A)\ S) \in less_than\ <*\textit{lex}*\>\ less_than \rangle$

proof –

have [*simp*]: $\langle (atms_of_mm\ (clauses_{NOT}\ T) \cup atm_of\ 'lits_of_l\ (trail\ T))$
 $= (atms_of_mm\ (clauses_{NOT}\ S) \cup atm_of\ 'lits_of_l\ (trail\ S)) \rangle$
using *learnST n-d* **by** (*elim learn_{NOT}E*) *auto*

then have $\langle card\ (atms_of_mm\ (clauses_{NOT}\ T) \cup atm_of\ 'lits_of_l\ (trail\ T))$
 $= card\ (atms_of_mm\ (clauses_{NOT}\ S) \cup atm_of\ 'lits_of_l\ (trail\ S)) \rangle$
by (*auto intro!*: *card-mono*)

then have $3: \langle 3::nat \rangle \wedge card\ (atms_of_mm\ (clauses_{NOT}\ T) \cup atm_of\ 'lits_of_l\ (trail\ T))$
 $= 3 \wedge card\ (atms_of_mm\ (clauses_{NOT}\ S) \cup atm_of\ 'lits_of_l\ (trail\ S)) \rangle$
by (*auto intro*: *power-mono*)

moreover have $\langle conflicting_bj_clss\ S \subseteq conflicting_bj_clss\ T \rangle$
using *learnST n-d* **by** (*simp add*: *learn-conflicting-increasing*)

moreover have $\langle conflicting_bj_clss\ S \neq conflicting_bj_clss\ T \rangle$
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: $\langle trail\ S = F' @ Decided\ K \# F \rangle$ **and**

C: $\langle C = add_mset\ L\ C' \rangle$ **and**

F: $\langle F \models_{as}\ CNot\ C' \rangle$ **and**

C-S: $\langle add_mset\ L\ C' \notin clauses_{NOT}\ S \rangle$

by *blast*

moreover have $\langle distinct_mset\ C \rangle \wedge \neg tautology\ C$ **using** *inv* **by** *blast+*

ultimately have $\langle add_mset\ L\ C' \in conflicting_bj_clss\ T \rangle$

using *T n-d unfolding conflicting-bj-clss-def* **by** *fastforce*

moreover have $\langle add_mset\ L\ C' \notin conflicting_bj_clss\ S \rangle$

using *C-S unfolding conflicting-bj-clss-def* **by** *auto*

ultimately show *?case* **by** *blast*

qed

moreover have *fin-T*: $\langle finite\ (conflicting_bj_clss\ T) \rangle$

using *learnST* **by** *induction* (*auto simp add*: *conflicting-bj-clss-add-clss_{NOT}*)

ultimately have $\langle card\ (conflicting_bj_clss\ T) \geq card\ (conflicting_bj_clss\ S) \rangle$

using *card-mono* **by** *blast*

moreover

have *fin'*: $\langle finite\ (atms_of_mm\ (clauses_{NOT}\ T) \cup atm_of\ 'lits_of_l\ (trail\ T)) \rangle$
by *auto*

have 1: $\langle atms_of_ms\ (conflicting_bj_clss\ T) \subseteq atms_of_mm\ (clauses_{NOT}\ T) \rangle$
unfolding *conflicting-bj-clss-def atms-of-ms-def* **by** *auto*

have 2: $\langle \bigwedge x. x \in conflicting_bj_clss\ T \implies \neg tautology\ x \wedge distinct_mset\ x \rangle$

```

unfolding conflicting-bj-clss-def by auto
have  $T$ :  $\langle \text{conflicting-bj-clss } T \rangle$ 
 $\subseteq \text{simple-clss } (\text{atms-of-mm } (\text{clauses}_{NOT} T) \cup \text{atm-of } \langle \text{lits-of-l } (\text{trail } T) \rangle)$ 
by standard (meson 1 2 fin'  $\langle \text{finite } (\text{conflicting-bj-clss } T) \rangle$  simple-clss-mono
distinct-mset-set-def simplified-in-simple-clss subsetCE sup.coboundedI1)
moreover
then have #:  $\langle 3 \wedge \text{card } (\text{atms-of-mm } (\text{clauses}_{NOT} T) \cup \text{atm-of } \langle \text{lits-of-l } (\text{trail } T) \rangle) \rangle$ 
 $\geq \text{card } (\text{conflicting-bj-clss } T)$ 
by (meson Nat.le-trans simple-clss-card simple-clss-finite card-mono fin')
have  $\langle \text{atms-of-mm } (\text{clauses}_{NOT} T) \cup \text{atm-of } \langle \text{lits-of-l } (\text{trail } T) \rangle \subseteq A \rangle$ 
using learnNOTE[OF learnST] A by simp
then have  $\langle 3 \wedge (\text{card } A) \geq \text{card } (\text{conflicting-bj-clss } T) \rangle$ 
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  $\langle \text{conflicting-bj-clss } S \neq \text{conflicting-bj-clss } T \rangle$ 
 $\langle \text{conflicting-bj-clss } S \subseteq \text{conflicting-bj-clss } T \rangle$ 
diff-less-mono2 le-less-trans not-le psubsetI)
qed

```

We have to assume the following:

- *inv S*: the invariant holds in the initial state.
- *A* is a (finite *finite A*) superset of the literals in the trail *atm-of* $\langle \text{lits-of-l } (\text{trail } S) \rangle \subseteq \text{atms-of-ms } A$ and in the clauses $\text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A$. This can be 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**

$\langle \mu_{CDCL} A T \equiv ((2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A)))$
 $- \mu_C (1 + \text{card } (\text{atms-of-ms } A)) (2 + \text{card } (\text{atms-of-ms } A)) (\text{trail-weight } T),$
 $\text{conflicting-bj-clss-yet } (\text{card } (\text{atms-of-ms } A)) T, \text{card } (\text{set-mset } (\text{clauses}_{NOT} T))) \rangle$

lemma *cdcl_{NOT}-decreasing-measure*:

assumes

$\langle \text{cdcl}_{NOT} S T \rangle$ **and**

inv: $\langle \text{inv } S \rangle$ **and**

atm-clss: $\langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A \rangle$ **and**

atm-lits: $\langle \text{atm-of } \langle \text{lits-of-l } (\text{trail } S) \rangle \subseteq \text{atms-of-ms } A \rangle$ **and**

n-d: $\langle \text{no-dup } (\text{trail } S) \rangle$ **and**

fin-A: $\langle \text{finite } A \rangle$

shows $\langle (\mu_{CDCL} A T, \mu_{CDCL} A S)$
 $\in \text{less-than } \langle *lex* \rangle (\text{less-than } \langle *lex* \rangle \text{less-than}) \rangle$

using *assms*(1)

proof *induction*

case $\langle \text{c-dpll-bj } T \rangle$

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 $\langle \text{c-learn } T \rangle$ **note** *learn* = *this*(1)

then have *S*: $\langle \text{trail } S = \text{trail } T \rangle$

```

  using inv atm-clss atm-lits n-d fin-A
  by (elim learnNOTE) auto
show ?case
  using learn- $\mu_L$ -decrease[OF learn n-d, of ⟨atms-of-ms A⟩] atm-clss atm-lits fin-A n-d
  unfolding S  $\mu_{CDCL}$ -def by auto
next
case (c-forgetNOT T) note forgetNOT = this(1)
have ⟨trail S = trail T⟩ using forgetNOT by induction auto
then show ?case
  using forget- $\mu_L$ -decrease[OF forgetNOT] unfolding  $\mu_{CDCL}$ -def by auto
qed

```

lemma wf-cdcl_{NOT}-restricted-learning:

```

  assumes ⟨finite A⟩
  shows ⟨wf {(T, S).
    (atms-of-mm (clausesNOT S) ⊆ atms-of-ms A ∧ atm-of ‘ lits-of-l (trail S) ⊆ atms-of-ms A
    ∧ no-dup (trail S)
    ∧ inv S)
    ∧ cdclNOT S T }⟩
  by (rule wf-wf-if-measure'[of ⟨less-than <*> (less-than <*> less-than)⟩])
    (auto intro: cdclNOT-decreasing-measure[OF - - - - - assms])

```

definition $\mu_C' :: \langle 'v \text{ clause set} \Rightarrow 'st \Rightarrow nat \rangle$ **where**

$\langle \mu_C' A T \equiv \mu_C (1 + \text{card } (\text{atms-of-ms } A)) (2 + \text{card } (\text{atms-of-ms } A)) (\text{trail-weight } T) \rangle$

definition $\mu_{CDCL}' :: \langle 'v \text{ clause set} \Rightarrow 'st \Rightarrow nat \rangle$ **where**

$\langle \mu_{CDCL}' A T \equiv$
 $((2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A)) - \mu_C' A T) * (1 + 3^{\text{card } (\text{atms-of-ms } A)}) *$
 2
 $+ \text{conflicting-bj-clss-yet } (\text{card } (\text{atms-of-ms } A)) T * 2$
 $+ \text{card } (\text{set-mset } (\text{clauses}_{NOT} T)) \rangle$

lemma cdcl_{NOT}-decreasing-measure':

```

  assumes
    ⟨cdclNOT S T⟩ and
    inv: ⟨inv S⟩ and
    atm-clss: ⟨atms-of-mm (clausesNOT 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 $\langle \mu_{CDCL}' A T < \mu_{CDCL}' A S \rangle$

using assms(1)

proof (induction rule: cdcl_{NOT}-learn-all-induct)

case (dpll-bj T)

then have $\langle (2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A)) - \mu_C' A T$
 $< (2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A)) - \mu_C' A S \rangle$

using dpll-bj-trail-mes-decreasing-prop fin-A inv n-d atm-clss atm-trail

unfolding μ_C' -def **by** blast

then have XX: $\langle ((2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A)) - \mu_C' A T) + 1$
 $\leq (2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A)) - \mu_C' A S \rangle$

by auto

from mult-le-mono1[OF this, of $\langle 1 + 3^{\text{card } (\text{atms-of-ms } A)} \rangle$]

have $\langle ((2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A)) - \mu_C' A T) *$
 $(1 + 3^{\text{card } (\text{atms-of-ms } A)}) + (1 + 3^{\text{card } (\text{atms-of-ms } A)})$
 $\leq ((2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A)) - \mu_C' A S)$
 $* (1 + 3^{\text{card } (\text{atms-of-ms } A)}) \rangle$


```

unfolding Nat.add-mult-distrib
by presburger
moreover
have cl-T-S:  $\langle \text{clauses}_{NOT} T = \text{clauses}_{NOT} S \rangle$ 
  using dpll-bj.hyps inv dpll-bj-clauses by auto
have  $\langle \text{conflicting-bj-clss-yet} (\text{card} (\text{atms-of-ms } A)) S < 1 + 3 \wedge \text{card} (\text{atms-of-ms } A) \rangle$ 
  by simp
ultimately have  $\langle ((2 + \text{card} (\text{atms-of-ms } A)) \wedge (1 + \text{card} (\text{atms-of-ms } A)) - \mu_C' A T) \cdot (1 + 3 \wedge \text{card} (\text{atms-of-ms } A)) + \text{conflicting-bj-clss-yet} (\text{card} (\text{atms-of-ms } A)) T \rangle$ 
  <  $\langle ((2 + \text{card} (\text{atms-of-ms } A)) \wedge (1 + \text{card} (\text{atms-of-ms } A)) - \mu_C' A S) \cdot (1 + 3 \wedge \text{card} (\text{atms-of-ms } A)) \rangle$ 
  by linarith
then have  $\langle ((2 + \text{card} (\text{atms-of-ms } A)) \wedge (1 + \text{card} (\text{atms-of-ms } A)) - \mu_C' A T) \cdot (1 + 3 \wedge \text{card} (\text{atms-of-ms } A)) + \text{conflicting-bj-clss-yet} (\text{card} (\text{atms-of-ms } A)) T \rangle$ 
  <  $\langle ((2 + \text{card} (\text{atms-of-ms } A)) \wedge (1 + \text{card} (\text{atms-of-ms } A)) - \mu_C' A S) \cdot (1 + 3 \wedge \text{card} (\text{atms-of-ms } A)) + \text{conflicting-bj-clss-yet} (\text{card} (\text{atms-of-ms } A)) S \rangle$ 
  by linarith
then have  $\langle ((2 + \text{card} (\text{atms-of-ms } A)) \wedge (1 + \text{card} (\text{atms-of-ms } A)) - \mu_C' A T) \cdot (1 + 3 \wedge \text{card} (\text{atms-of-ms } A)) \cdot 2 + \text{conflicting-bj-clss-yet} (\text{card} (\text{atms-of-ms } A)) T \cdot 2 \rangle$ 
  <  $\langle ((2 + \text{card} (\text{atms-of-ms } A)) \wedge (1 + \text{card} (\text{atms-of-ms } A)) - \mu_C' A S) \cdot (1 + 3 \wedge \text{card} (\text{atms-of-ms } A)) \cdot 2 + \text{conflicting-bj-clss-yet} (\text{card} (\text{atms-of-ms } A)) S \cdot 2 \rangle$ 
  by linarith
then show ?case unfolding  $\mu_{CDCL}'$ -def cl-T-S by presburger
next
case (learn C F' K F C' L T) note clss-S-C = this(1) and atms-C = this(2) and dist = this(3)
  and tauto = this(4) and learn-restr = this(5) and tr-S = this(6) and C' = this(7) and
  F-C = this(8) and C-new = this(9) and T = this(10)
have  $\langle \text{insert } C (\text{conflicting-bj-clss } S) \subseteq \text{simple-clss} (\text{atms-of-ms } A) \rangle$ 
  proof -
    have  $\langle C \in \text{simple-clss} (\text{atms-of-ms } A) \rangle$ 
      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  $\langle \text{conflicting-bj-clss } S \subseteq \text{simple-clss} (\text{atms-of-ms } A) \rangle$ 
      proof
        fix x :: 'v clause
        assume  $\langle x \in \text{conflicting-bj-clss } S \rangle$ 
        then have  $\langle x \in \# \text{clauses}_{NOT} S \wedge \text{distinct-mset } x \wedge \neg \text{tautology } x \rangle$ 
          unfolding conflicting-bj-clss-def by blast
        then show  $\langle x \in \text{simple-clss} (\text{atms-of-ms } A) \rangle$ 
          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
    qed
  ultimately show ?thesis
    by auto
  qed
then have  $\langle \text{card} (\text{insert } C (\text{conflicting-bj-clss } S)) \leq 3 \wedge (\text{card} (\text{atms-of-ms } A)) \rangle$ 
  by (meson Nat.le-trans atms-of-ms-finite simple-clss-card simple-clss-finite
    card-mono fin-A)
moreover have [simp]:  $\langle \text{card} (\text{insert } C (\text{conflicting-bj-clss } S)) \rangle$ 

```

$= \text{Suc } (\text{card } ((\text{conflicting-bj-clss } S)))$
by (*metis* (*no-types*) C' C -new *card-insert-if conflicting-bj-clss-incl-clauses contra-subsetD* *finite-conflicting-bj-clss*)
moreover have [*simp*]: $\langle \text{conflicting-bj-clss } (\text{add-cl}_\text{NOT } C S) = \text{conflicting-bj-clss } S \cup \{C\} \rangle$
using *dist tauto F-C* **by** (*subst conflicting-bj-clss-add-cl}_\text{NOT } [OF n-d]*) (*force simp: C' tr-S n-d*)
ultimately have [*simp*]: $\langle \text{conflicting-bj-clss-yet } (\text{card } (\text{atms-of-ms } A)) S$
 $= \text{Suc } (\text{conflicting-bj-clss-yet } (\text{card } (\text{atms-of-ms } A)) (\text{add-cl}_\text{NOT } C S)) \rangle$
by *simp*
have 1: $\langle \text{clauses}_\text{NOT } T = \text{clauses}_\text{NOT } (\text{add-cl}_\text{NOT } C S) \rangle$ **using** T **by** *auto*
have 2: $\langle \text{conflicting-bj-clss-yet } (\text{card } (\text{atms-of-ms } A)) T$
 $= \text{conflicting-bj-clss-yet } (\text{card } (\text{atms-of-ms } A)) (\text{add-cl}_\text{NOT } C S) \rangle$
using T **unfolding** *conflicting-bj-clss-def* **by** *auto*
have 3: $\langle \mu_{C'} A T = \mu_{C'} A (\text{add-cl}_\text{NOT } C S) \rangle$
using T **unfolding** $\mu_{C'}$ -def **by** *auto*
have $\langle ((2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A)) - \mu_{C'} A (\text{add-cl}_\text{NOT } C S))$
 $* (1 + 3 \wedge \text{card } (\text{atms-of-ms } A)) * 2$
 $= ((2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A)) - \mu_{C'} A S)$
 $* (1 + 3 \wedge \text{card } (\text{atms-of-ms } A)) * 2 \rangle$
using n -d **unfolding** $\mu_{C'}$ -def **by** *auto*
moreover
have $\langle \text{conflicting-bj-clss-yet } (\text{card } (\text{atms-of-ms } A)) (\text{add-cl}_\text{NOT } C S)$
 $* 2$
 $+ \text{card } (\text{set-mset } (\text{clauses}_\text{NOT } (\text{add-cl}_\text{NOT } C S)))$
 $< \text{conflicting-bj-clss-yet } (\text{card } (\text{atms-of-ms } A)) S * 2$
 $+ \text{card } (\text{set-mset } (\text{clauses}_\text{NOT } S)) \rangle$
by (*simp add: C' C-new n-d*)
ultimately show ?*case* **unfolding** μ_{CDCL} '-def 1 2 3 **by** *presburger*
next
case (*forget*_{NOT} $C T$) **note** $T = \text{this}(4)$
have [*simp*]: $\langle \mu_{C'} A (\text{remove-cl}_\text{NOT } C S) = \mu_{C'} A S \rangle$
unfolding $\mu_{C'}$ -def **by** *auto*
have $\langle \text{forget}_\text{NOT } S T \rangle$
apply (*rule forget*_{NOT}.*intros*) **using** *forget*_{NOT} **by** *auto*
then have $\langle \text{conflicting-bj-clss } T = \text{conflicting-bj-clss } S \rangle$
using *do-not-forget-before-backtrack-rule-clause-learned-clause-untouched* **by** *blast*
moreover have $\langle \text{card } (\text{set-mset } (\text{clauses}_\text{NOT } T)) < \text{card } (\text{set-mset } (\text{clauses}_\text{NOT } S)) \rangle$
by (*metis T card-Diff1-less clauses-remove-cl}_\text{NOT } finite-set-mset forget*_{NOT}.*hyps*(2)
order-refl set-mset-minus-replicate-mset(1) *state-eq*_{NOT}-*clauses*)
ultimately show ?*case* **unfolding** μ_{CDCL} '-def
using T $\langle \mu_{C'} A (\text{remove-cl}_\text{NOT } C S) = \mu_{C'} A S \rangle$ **by** (*metis* (*no-types*) *add-le-cancel-left*
 $\mu_{C'}$ -def *not-le state-eq*_{NOT}-*trail*)
qed

lemma *cdcl*_{NOT}-*clauses-bound*:

assumes
 $\langle \text{cdcl}_\text{NOT } S T \rangle$ **and**
 $\langle \text{inv } S \rangle$ **and**
 $\langle \text{atms-of-mm } (\text{clauses}_\text{NOT } S) \subseteq A \rangle$ **and**
 $\langle \text{atm-of } (\text{lits-of-l } (\text{trail } S)) \subseteq A \rangle$ **and**
 n -d: $\langle \text{no-dup } (\text{trail } S) \rangle$ **and**
 $\text{fin-}A[\text{simp}]$: $\langle \text{finite } A \rangle$
shows $\langle \text{set-mset } (\text{clauses}_\text{NOT } T) \subseteq \text{set-mset } (\text{clauses}_\text{NOT } S) \cup \text{simple-clss } A \rangle$
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 $\langle \text{atms-of } C \subseteq A \rangle$
 using *atms-C atms-clss-S atms-trail-S* by fast
 then have $\langle \text{simple-clss } (\text{atms-of } C) \subseteq \text{simple-clss } A \rangle$
 by (*simp add: simple-clss-mono*)
 then have $\langle C \in \text{simple-clss } A \rangle$
 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 *rtrancpl-cdcl_{NOT}-clauses-bound*:

assumes
 $\langle \text{cdcl}_{\text{NOT}}^{**} S T \rangle$ and
 $\langle \text{inv } S \rangle$ and
 $\langle \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \subseteq A \rangle$ and
 $\langle \text{atm-of } '(\text{lits-of-l } (\text{trail } S)) \subseteq A \rangle$ and
n-d: $\langle \text{no-dup } (\text{trail } S) \rangle$ and
finite: $\langle \text{finite } A \rangle$
 shows $\langle \text{set-mset } (\text{clauses}_{\text{NOT}} T) \subseteq \text{set-mset } (\text{clauses}_{\text{NOT}} S) \cup \text{simple-clss } A \rangle$
 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-clss-S = this(7)*
 have $\langle \text{inv } T \rangle$
 using *rtrancpl-cdcl_{NOT}-inv st inv* by blast
 moreover have $\langle \text{atms-of-mm } (\text{clauses}_{\text{NOT}} T) \subseteq A \rangle$ and $\langle \text{atm-of } ' \text{lits-of-l } (\text{trail } T) \subseteq A \rangle$
 using *rtrancpl-cdcl_{NOT}-trail-clauses-bound[OF st] inv atms-clss-S atms-trail-S n-d* by auto
 moreover have $\langle \text{no-dup } (\text{trail } T) \rangle$
 using *rtrancpl-cdcl_{NOT}-no-dup[OF st inv S n-d]* by *simp*
 ultimately have $\langle \text{set-mset } (\text{clauses}_{\text{NOT}} U) \subseteq \text{set-mset } (\text{clauses}_{\text{NOT}} T) \cup \text{simple-clss } A \rangle$
 using *cdcl_{NOT} finite n-d* by (*auto simp: cdcl_{NOT}-clauses-bound*)
 then show ?case using *IH* by auto
qed

lemma *rtrancpl-cdcl_{NOT}-card-clauses-bound*:

assumes
 $\langle \text{cdcl}_{\text{NOT}}^{**} S T \rangle$ and
 $\langle \text{inv } S \rangle$ and
 $\langle \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \subseteq A \rangle$ and
 $\langle \text{atm-of } '(\text{lits-of-l } (\text{trail } S)) \subseteq A \rangle$ and
n-d: $\langle \text{no-dup } (\text{trail } S) \rangle$ and
finite: $\langle \text{finite } A \rangle$
 shows $\langle \text{card } (\text{set-mset } (\text{clauses}_{\text{NOT}} T)) \leq \text{card } (\text{set-mset } (\text{clauses}_{\text{NOT}} S)) + 3 \wedge (\text{card } A) \rangle$
 using *rtrancpl-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 *rtrancpl-cdcl_{NOT}-card-clauses-bound'*:

assumes
 $\langle \text{cdcl}_{NOT}^{**} S T \rangle$ **and**
 $\langle \text{inv } S \rangle$ **and**
 $\langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq A \rangle$ **and**
 $\langle \text{atm-of } '(\text{lits-of-l } (\text{trail } S)) \subseteq A \rangle$ **and**
 $n\text{-d: } \langle \text{no-dup } (\text{trail } S) \rangle$ **and**
 $\text{finite: } \langle \text{finite } A \rangle$
shows $\langle \text{card } \{C \mid C. C \in \# \text{ clauses}_{NOT} T \wedge (\text{tautology } C \vee \neg \text{distinct-mset } C)\} \leq \text{card } \{C \mid C. C \in \# \text{ clauses}_{NOT} S \wedge (\text{tautology } C \vee \neg \text{distinct-mset } C)\} + 3 \wedge (\text{card } A) \rangle$
(is $\langle \text{card } ?T \leq \text{card } ?S + \neg \rangle$
using $\text{rtrancpl-cdcl}_{NOT}\text{-clauses-bound}[OF \text{ assms}] \text{ finite}$
proof –
have $\langle ?T \subseteq ?S \cup \text{simple-clss } A \rangle$
using $\text{rtrancpl-cdcl}_{NOT}\text{-clauses-bound}[OF \text{ assms}]$ **by force**
then have $\langle \text{card } ?T \leq \text{card } (?S \cup \text{simple-clss } A) \rangle$
using finite **by** $(\text{simp add: assms}(5) \text{ simple-clss-finite card-mono})$
then show $?thesis$
by $(\text{meson le-trans simple-clss-card card-Un-le local.finite nat-add-left-cancel-le})$
qed

lemma $\text{rtrancpl-cdcl}_{NOT}\text{-card-simple-clauses-bound}$:

assumes
 $\langle \text{cdcl}_{NOT}^{**} S T \rangle$ **and**
 $\langle \text{inv } S \rangle$ **and**
 $NA: \langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq A \rangle$ **and**
 $MA: \langle \text{atm-of } '(\text{lits-of-l } (\text{trail } S)) \subseteq A \rangle$ **and**
 $n\text{-d: } \langle \text{no-dup } (\text{trail } S) \rangle$ **and**
 $\text{finite: } \langle \text{finite } A \rangle$
shows $\langle \text{card } (\text{set-mset } (\text{clauses}_{NOT} T)) \leq \text{card } \{C. C \in \# \text{ clauses}_{NOT} S \wedge (\text{tautology } C \vee \neg \text{distinct-mset } C)\} + 3 \wedge (\text{card } A) \rangle$
(is $\langle \text{card } ?T \leq \text{card } ?S + \neg \rangle$
using $\text{rtrancpl-cdcl}_{NOT}\text{-clauses-bound}[OF \text{ assms}] \text{ finite}$
proof –
have $\langle \bigwedge x. x \in \# \text{ clauses}_{NOT} T \implies \neg \text{tautology } x \implies \text{distinct-mset } x \implies x \in \text{simple-clss } A \rangle$
using $\text{rtrancpl-cdcl}_{NOT}\text{-clauses-bound}[OF \text{ assms}]$ **by** $(\text{metis } (\text{no-types, hide-lams}) \text{ Un-iff } NA \text{ atms-of-atms-of-ms-mono simple-clss-mono contra-subsetD subset-trans distinct-mset-not-tautology-implies-in-simple-clss})$
then have $\langle \text{set-mset } (\text{clauses}_{NOT} T) \subseteq ?S \cup \text{simple-clss } A \rangle$
using $\text{rtrancpl-cdcl}_{NOT}\text{-clauses-bound}[OF \text{ assms}]$ **by auto**
then have $\langle \text{card } (\text{set-mset } (\text{clauses}_{NOT} T)) \leq \text{card } (?S \cup \text{simple-clss } A) \rangle$
using finite **by** $(\text{simp add: assms}(5) \text{ simple-clss-finite card-mono})$
then show $?thesis$
by $(\text{meson le-trans simple-clss-card card-Un-le local.finite nat-add-left-cancel-le})$
qed

definition $\mu_{CDCL}'\text{-bound} :: \langle 'v \text{ clause set} \Rightarrow 'st \Rightarrow \text{nat} \rangle$ **where**

$\mu_{CDCL}'\text{-bound } A S =$
 $((2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A))) * (1 + 3 \wedge \text{card } (\text{atms-of-ms } A)) * 2$
 $+ 2 * 3 \wedge (\text{card } (\text{atms-of-ms } A))$
 $+ \text{card } \{C. C \in \# \text{ clauses}_{NOT} S \wedge (\text{tautology } C \vee \neg \text{distinct-mset } C)\} + 3 \wedge (\text{card } (\text{atms-of-ms } A))$

lemma $\mu_{CDCL}'\text{-bound-reduce-trail-to}_{NOT}[\text{simp}]$:

$\langle \mu_{CDCL}'\text{-bound } A (\text{reduce-trail-to}_{NOT} M S) = \mu_{CDCL}'\text{-bound } A S \rangle$
unfolding $\mu_{CDCL}'\text{-bound-def}$ **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 ⟨*μ_{CDCL}' A U ≤ μ_{CDCL}'-bound A S*⟩

proof –

have ⟨ $((2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A)) - \mu_C' A U)$
 $\leq (2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A))$ ⟩

by *auto*

then have ⟨ $((2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A)) - \mu_C' A U)$
 $* (1 + 3 \wedge \text{card } (\text{atms-of-ms } A)) * 2$

$\leq (2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A)) * (1 + 3 \wedge \text{card } (\text{atms-of-ms } A)) * 2$ ⟩

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))*⟩

$\leq \text{card } \{C. C \in \# \text{ clauses}_{NOT} S \wedge (\text{tautology } C \vee \neg \text{distinct-mset } C)\} + 3 \wedge \text{card } (\text{atms-of-ms } A)$ ⟩

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 ⟨*μ_{CDCL}' A T ≤ μ_{CDCL}'-bound A S*⟩

proof –

have ⟨*μ_{CDCL}' A (reduce-trail-to_{NOT} (trail T) T) = μ_{CDCL}' A T*⟩

unfolding *μ_{CDCL}'-def μ_C'-def conflicting-bj-clss-def* **by** *auto*

then show *?thesis* **using** *rtranclp-cdcl_{NOT}-μ_{CDCL}'-bound-reduce-trail-to_{NOT}[OF assms, of - ⟨trail T⟩]*

state-eq_{NOT}-ref **by** *fastforce*

qed

lemma *rtranclp-μ_{CDCL}'-bound-decreasing*:

assumes

⟨*cdcl_{NOT}** S T*⟩ **and**
 ⟨*inv S*⟩ **and**
 ⟨*atms-of-mm (clauses_{NOT} S) ⊆ 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 ⟨ $\{C. C \in \# \text{ clauses}_{NOT} T \wedge (\text{tautology } C \vee \neg \text{distinct-mset } C)\}$ ⟩

```

 $\subseteq \{C. C \in \# \text{ clauses}_{NOT} S \wedge (\text{tautology } C \vee \neg \text{distinct-mset } C)\}$  (is  $\langle ?T \subseteq ?S \rangle$ )
proof (rule Set.subsetI)
  fix  $C$  assume  $\langle C \in ?T \rangle$ 
  then have  $C-T: \langle C \in \# \text{ clauses}_{NOT} T \rangle$  and  $t-d: \langle \text{tautology } C \vee \neg \text{distinct-mset } C \rangle$ 
    by auto
  then have  $\langle C \notin \text{simple-clss } (\text{atms-of-ms } A) \rangle$ 
    by (auto dest: simple-clssE)
  then show  $\langle C \in ?S \rangle$ 
    using  $C-T$  rtrancp-cdclNOT-clauses-bound[OF assms]  $t-d$  by force
qed
then have  $\langle \text{card } \{C. C \in \# \text{ clauses}_{NOT} T \wedge (\text{tautology } C \vee \neg \text{distinct-mset } C)\} \leq$ 
   $\text{card } \{C. C \in \# \text{ clauses}_{NOT} S \wedge (\text{tautology } C \vee \neg \text{distinct-mset } C)\} \rangle$ 
  by (simp add: card-mono)
then show  $?thesis$ 
  unfolding  $\mu_{CDCL}'\text{-bound-def}$  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
     $cdcl_{NOT} :: \langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$  and
     $restart :: \langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ 
  begin
inductive  $cdcl_{NOT}\text{-raw-restart} :: \langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$  where
   $\langle cdcl_{NOT} S T \Longrightarrow cdcl_{NOT}\text{-raw-restart } S T \rangle \mid$ 
   $\langle restart S T \Longrightarrow cdcl_{NOT}\text{-raw-restart } S T \rangle$ 
end

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

lemma  $cdcl_{NOT}\text{-iff-}cdcl_{NOT}\text{-raw-restart-no-restarts}$ :
   $\langle cdcl_{NOT} S T \longleftrightarrow restart\text{-ops}.cdcl_{NOT}\text{-raw-restart } cdcl_{NOT} (\lambda - . \text{False}) S T \rangle$ 
  (is  $\langle ?C S T \longleftrightarrow ?R S T \rangle$ )
proof
  fix  $S T$ 

```

```

assume  $\langle ?C \ S \ T \rangle$ 
then show  $\langle ?R \ S \ T \rangle$  by (simp add: restart-ops.cdclNOT-raw-restart.intros(1))
next
fix  $S \ T$ 
assume  $\langle ?R \ S \ T \rangle$ 
then show  $\langle ?C \ S \ T \rangle$ 
  apply (cases rule: restart-ops.cdclNOT-raw-restart.cases)
  using  $\langle ?R \ S \ T \rangle$  by fast+
qed

```

```

lemma cdclNOT-cdclNOT-raw-restart:
   $\langle cdcl_{NOT} \ S \ T \implies restart-ops.cdcl_{NOT}-raw-restart \ cdcl_{NOT} \ restart \ S \ T \rangle$ 
  by (simp add: restart-ops.cdclNOT-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 assumptions 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 $\mu-bound$ taking the same parameter as μ and the initial state of the considered $cdcl_{NOT}$ chain.

```

locale cdclNOT-increasing-restarts-ops =
  restart-ops cdclNOT restart for
  restart ::  $\langle 'st \Rightarrow 'st \Rightarrow bool \rangle$  and
  cdclNOT ::  $\langle 'st \Rightarrow 'st \Rightarrow bool \rangle +$ 
fixes
  f ::  $\langle nat \Rightarrow nat \rangle$  and
  bound-inv ::  $\langle 'bound \Rightarrow 'st \Rightarrow bool \rangle$  and
   $\mu :: \langle 'bound \Rightarrow 'st \Rightarrow nat \rangle$  and
  cdclNOT-inv ::  $\langle 'st \Rightarrow bool \rangle$  and
   $\mu-bound :: \langle 'bound \Rightarrow 'st \Rightarrow nat \rangle$ 
assumes
  f:  $\langle unbounded \ f \rangle$  and
  f-ge-1:  $\langle \bigwedge n. n \geq 1 \implies f \ n \neq 0 \rangle$  and
  bound-inv:  $\langle \bigwedge A \ S \ T. cdcl_{NOT}-inv \ S \implies bound-inv \ A \ S \implies cdcl_{NOT} \ S \ T \implies bound-inv \ A \ T \rangle$  and
  cdclNOT-measure:  $\langle \bigwedge A \ S \ T. cdcl_{NOT}-inv \ S \implies bound-inv \ A \ S \implies cdcl_{NOT} \ S \ T \implies \mu \ A \ T < \mu \ A \ S \rangle$  and

```

measure-bound2: $\langle \bigwedge A \ T \ U. \text{cdcl}_{NOT}\text{-inv } T \implies \text{bound-inv } A \ T \implies \text{cdcl}_{NOT}^{**} \ T \ U \implies \mu \ A \ U \leq \mu\text{-bound } A \ T \rangle$ **and**
measure-bound4: $\langle \bigwedge A \ T \ U. \text{cdcl}_{NOT}\text{-inv } T \implies \text{bound-inv } A \ T \implies \text{cdcl}_{NOT}^{**} \ T \ U \implies \mu\text{-bound } A \ U \leq \mu\text{-bound } A \ T \rangle$ **and**
cdcl_{NOT}-restart-inv: $\langle \bigwedge A \ U \ V. \text{cdcl}_{NOT}\text{-inv } U \implies \text{restart } U \ V \implies \text{bound-inv } A \ U \implies \text{bound-inv } A \ V \rangle$
and
exists-bound: $\langle \bigwedge R \ S. \text{cdcl}_{NOT}\text{-inv } R \implies \text{restart } R \ S \implies \exists A. \text{bound-inv } A \ S \rangle$ **and**
cdcl_{NOT}-inv: $\langle \bigwedge S \ T. \text{cdcl}_{NOT}\text{-inv } S \implies \text{cdcl}_{NOT} \ S \ T \implies \text{cdcl}_{NOT}\text{-inv } T \rangle$ **and**
cdcl_{NOT}-inv-restart: $\langle \bigwedge S \ T. \text{cdcl}_{NOT}\text{-inv } S \implies \text{restart } S \ T \implies \text{cdcl}_{NOT}\text{-inv } T \rangle$
begin

lemma *cdcl_{NOT}-cdcl_{NOT}-inv*:

assumes

$\langle (\text{cdcl}_{NOT} \rightsquigarrow n) \ S \ T \rangle$ **and**

$\langle \text{cdcl}_{NOT}\text{-inv } S \rangle$

shows $\langle \text{cdcl}_{NOT}\text{-inv } T \rangle$

using *assms* **by** (*induction n arbitrary: T*) (*auto intro:bound-inv cdcl_{NOT}-inv*)

lemma *cdcl_{NOT}-bound-inv*:

assumes

$\langle (\text{cdcl}_{NOT} \rightsquigarrow n) \ S \ T \rangle$ **and**

$\langle \text{cdcl}_{NOT}\text{-inv } S \rangle$

$\langle \text{bound-inv } A \ S \rangle$

shows $\langle \text{bound-inv } A \ T \rangle$

using *assms* **by** (*induction n arbitrary: T*) (*auto intro:bound-inv cdcl_{NOT}-cdcl_{NOT}-inv*)

lemma *rtrancpl-cdcl_{NOT}-cdcl_{NOT}-inv*:

assumes

$\langle \text{cdcl}_{NOT}^{**} \ S \ T \rangle$ **and**

$\langle \text{cdcl}_{NOT}\text{-inv } S \rangle$

shows $\langle \text{cdcl}_{NOT}\text{-inv } T \rangle$

using *assms* **by** *induction* (*auto intro: cdcl_{NOT}-inv*)

lemma *rtrancpl-cdcl_{NOT}-bound-inv*:

assumes

$\langle \text{cdcl}_{NOT}^{**} \ S \ T \rangle$ **and**

$\langle \text{bound-inv } A \ S \rangle$ **and**

$\langle \text{cdcl}_{NOT}\text{-inv } S \rangle$

shows $\langle \text{bound-inv } A \ T \rangle$

using *assms* **by** *induction* (*auto intro:bound-inv rtrancpl-cdcl_{NOT}-cdcl_{NOT}-inv*)

lemma *cdcl_{NOT}-comp-n-le*:

assumes

$\langle (\text{cdcl}_{NOT} \rightsquigarrow (\text{Suc } n)) \ S \ T \rangle$ **and**

$\langle \text{bound-inv } A \ S \rangle$

$\langle \text{cdcl}_{NOT}\text{-inv } S \rangle$

shows $\langle \mu \ A \ T < \mu \ A \ S - n \rangle$

using *assms*

proof (*induction n arbitrary: T*)

case 0

then show ?*case* **using** *cdcl_{NOT}-measure* **by** *auto*

next

case (*Suc n*) **note** *IH* = *this(1)[OF - this(3) this(4)]* **and** *S-T* = *this(2)* **and** *b-inv* = *this(3)* **and**

c-inv = *this(4)*

obtain *U* :: '*st* **where** *S-U*: $\langle (\text{cdcl}_{NOT} \rightsquigarrow (\text{Suc } n)) \ S \ U \rangle$ **and** *U-T*: $\langle \text{cdcl}_{NOT} \ U \ T \rangle$ **using** *S-T* **by**

auto
then have $\langle \mu A \ U < \mu A \ S - n \rangle$ **using** $IH[of \ U]$ **by** *simp*
moreover
have $\langle bound_inv \ A \ U \rangle$
using $S-U \ b_inv \ cdcl_{NOT}\text{-}bound_inv \ c_inv$ **by** *blast*
then have $\langle \mu A \ T < \mu A \ U \rangle$ **using** $cdcl_{NOT}\text{-}measure[OF \ - \ - \ U-T] \ S-U \ c_inv \ cdcl_{NOT}\text{-}cdcl_{NOT}\text{-}inv$
by *auto*
ultimately show *?case* **by** *linarith*
qed

lemma *wf-cdcl_{NOT}*:
 $\langle wf \ \{ (T, S). \ cdcl_{NOT} \ S \ T \wedge \ cdcl_{NOT}\text{-}inv \ S \wedge \ bound_inv \ A \ S \} \rangle$ **(is** $\langle wf \ ?A \rangle$ **)**
apply $(rule \ wfP\text{-}if\text{-}measure2[of \ - \ - \ \langle \mu \ A \rangle])$
using $cdcl_{NOT}\text{-}comp\text{-}n\text{-}le[of \ 0 \ - \ - \ A]$ **by** *auto*

lemma *rtrancpl-cdcl_{NOT}-measure*:
assumes
 $\langle cdcl_{NOT}^{**} \ S \ T \rangle$ **and**
 $\langle bound_inv \ A \ S \rangle$ **and**
 $\langle cdcl_{NOT}\text{-}inv \ S \rangle$
shows $\langle \mu A \ T \leq \mu A \ S \rangle$
using *assms*
proof (*induction rule: rtrancpl-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 $\langle bound_inv \ A \ T \rangle$
by (*meson* $cdcl_{NOT}\text{-}bound_inv \ rtrancpl\text{-}imp\text{-}relpowp \ st \ step.prem$)
moreover have $\langle cdcl_{NOT}\text{-}inv \ T \rangle$
using $c_inv \ rtrancpl\text{-}cdcl_{NOT}\text{-}cdcl_{NOT}\text{-}inv \ st$ **by** *blast*
ultimately have $\langle \mu A \ U < \mu A \ T \rangle$ **using** $cdcl_{NOT}\text{-}measure[OF \ - \ - \ cdcl_{NOT}]$ **by** *auto*
then show *?case* **using** IH **by** *linarith*
qed

lemma *cdcl_{NOT}-comp-bounded*:
assumes
 $\langle bound_inv \ A \ S \rangle$ **and** $\langle cdcl_{NOT}\text{-}inv \ S \rangle$ **and** $\langle m \geq 1 + \mu A \ S \rangle$
shows $\langle \neg (cdcl_{NOT} \ \widetilde{\sim} \ m) \ S \ T \rangle$
using *assms* $cdcl_{NOT}\text{-}comp\text{-}n\text{-}le[of \ \langle m-1 \rangle \ S \ T \ A]$ **by** *fastforce*

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

inductive *cdcl_{NOT}-restart* **where**
restart-step: $\langle (cdcl_{NOT} \ \widetilde{\sim} \ m) \ S \ T \implies m \geq f \ n \implies restart \ T \ U \implies cdcl_{NOT}\text{-}restart \ (S, n) \ (U, Suc \ n) \rangle \mid$
restart-full: $\langle full1 \ cdcl_{NOT} \ S \ T \implies cdcl_{NOT}\text{-}restart \ (S, n) \ (T, Suc \ n) \rangle$

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

lemma *cdcl_{NOT}-restart-cdcl_{NOT}-raw-restart*:
 $\langle cdcl_{NOT}\text{-restart} \ S \ T \implies cdcl_{NOT}\text{-raw-restart}^{**} \ (fst \ S) \ (fst \ T) \rangle$

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

lemma *cdcl_{NOT}-with-restart-bound-inv:*

assumes
 $\langle \text{cdcl}_{\text{NOT}}\text{-restart} S T \rangle$ **and**
 $\langle \text{bound-inv} A (\text{fst } S) \rangle$ **and**
 $\langle \text{cdcl}_{\text{NOT}}\text{-inv} (\text{fst } S) \rangle$
shows $\langle \text{bound-inv} A (\text{fst } T) \rangle$
using *assms* **apply** (*induction rule: cdcl_{NOT}-restart.induct*)
prefer 2 **apply** (*metis rtrancpl-unfold fstI full1-def rtrancpl-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
 $\langle \text{cdcl}_{\text{NOT}}\text{-restart} S T \rangle$ **and**
 $\langle \text{cdcl}_{\text{NOT}}\text{-inv} (\text{fst } S) \rangle$
shows $\langle \text{cdcl}_{\text{NOT}}\text{-inv} (\text{fst } T) \rangle$
using *assms* **apply** *induction*
apply (*metis cdcl_{NOT}-cdcl_{NOT}-inv cdcl_{NOT}-inv-restart fst-conv*)
apply (*metis fstI full-def full-unfold rtrancpl-cdcl_{NOT}-cdcl_{NOT}-inv*)
done

lemma *rtrancpl-cdcl_{NOT}-with-restart-cdcl_{NOT}-inv:*

assumes
 $\langle \text{cdcl}_{\text{NOT}}\text{-restart}^{**} S T \rangle$ **and**
 $\langle \text{cdcl}_{\text{NOT}}\text{-inv} (\text{fst } S) \rangle$
shows $\langle \text{cdcl}_{\text{NOT}}\text{-inv} (\text{fst } T) \rangle$
using *assms* **by** *induction (auto intro: cdcl_{NOT}-with-restart-cdcl_{NOT}-inv)*

lemma *rtrancpl-cdcl_{NOT}-with-restart-bound-inv:*

assumes
 $\langle \text{cdcl}_{\text{NOT}}\text{-restart}^{**} S T \rangle$ **and**
 $\langle \text{cdcl}_{\text{NOT}}\text{-inv} (\text{fst } S) \rangle$ **and**
 $\langle \text{bound-inv} A (\text{fst } S) \rangle$
shows $\langle \text{bound-inv} A (\text{fst } T) \rangle$
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 rtrancpl-cdcl_{NOT}-with-restart-cdcl_{NOT}-inv* **by** *blast*

lemma *cdcl_{NOT}-with-restart-increasing-number:*

$\langle \text{cdcl}_{\text{NOT}}\text{-restart} S T \rangle \implies \text{snd } T = 1 + \text{snd } S$
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-cl_{NOT} remove-cl_{NOT}
for
 trail :: $\langle 'st \Rightarrow ('v, \text{unit}) \text{ ann-lits} \rangle$ **and**
 clauses_{NOT} :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**
 prepend-trail :: $\langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
 tl-trail :: $\langle 'st \Rightarrow 'st \rangle$ **and**
 add-cl_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
 remove-cl_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
 f :: $\langle \text{nat} \Rightarrow \text{nat} \rangle$ **and**
 restart :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **and**
 bound-inv :: $\langle 'bound \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **and**
 μ :: $\langle 'bound \Rightarrow 'st \Rightarrow \text{nat} \rangle$ **and**
cdcl_{NOT} :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **and**
cdcl_{NOT}-inv :: $\langle 'st \Rightarrow \text{bool} \rangle$ **and**
 μ -bound :: $\langle 'bound \Rightarrow 'st \Rightarrow \text{nat} \rangle$ +
assumes
 measure-bound: $\langle \bigwedge A \ T \ V \ n. \text{cdcl}_{NOT}\text{-inv } T \Longrightarrow \text{bound-inv } A \ T$
 $\Longrightarrow \text{cdcl}_{NOT}\text{-restart } (T, n) \ (V, \text{Suc } n) \Longrightarrow \mu \ A \ V \leq \mu\text{-bound } A \ T \rangle$ **and**
cdcl_{NOT}-raw-restart- μ -bound:
 $\langle \text{cdcl}_{NOT}\text{-restart } (T, a) \ (V, b) \Longrightarrow \text{cdcl}_{NOT}\text{-inv } T \Longrightarrow \text{bound-inv } A \ T$
 $\Longrightarrow \mu\text{-bound } A \ V \leq \mu\text{-bound } A \ T \rangle$
begin

lemma *rtrancp-cdcl_{NOT}-raw-restart- μ -bound*:
 $\langle \text{cdcl}_{NOT}\text{-restart}^{**} (T, a) \ (V, b) \Longrightarrow \text{cdcl}_{NOT}\text{-inv } T \Longrightarrow \text{bound-inv } A \ T$
 $\Longrightarrow \mu\text{-bound } A \ V \leq \mu\text{-bound } A \ T \rangle$
apply (induction rule: *rtrancp-induct2*)
apply *simp*
by (metis *cdcl_{NOT}-raw-restart- μ -bound* dual-order.trans *fst-conv*
rtrancp-cdcl_{NOT}-with-restart-bound-inv *rtrancp-cdcl_{NOT}-with-restart-cdcl_{NOT}-inv*)

lemma *cdcl_{NOT}-raw-restart-measure-bound*:
 $\langle \text{cdcl}_{NOT}\text{-restart } (T, a) \ (V, b) \Longrightarrow \text{cdcl}_{NOT}\text{-inv } T \Longrightarrow \text{bound-inv } A \ T$
 $\Longrightarrow \mu \ A \ V \leq \mu\text{-bound } A \ T \rangle$
apply (cases rule: *cdcl_{NOT}-restart.cases*)
apply *simp*
using *measure-bound* *relpowp-imp-rtrancp* **apply** *fastforce*
by (metis *full-def* *full-unfold* *measure-bound2* *prod.inject*)

lemma *rtrancp-cdcl_{NOT}-raw-restart-measure-bound*:
 $\langle \text{cdcl}_{NOT}\text{-restart}^{**} (T, a) \ (V, b) \Longrightarrow \text{cdcl}_{NOT}\text{-inv } T \Longrightarrow \text{bound-inv } A \ T$
 $\Longrightarrow \mu \ A \ V \leq \mu\text{-bound } A \ T \rangle$
apply (induction rule: *rtrancp-induct2*)
apply (*simp* add: *measure-bound2*)
by (metis *dual-order.trans* *fst-conv* *measure-bound2* *r-into-rtrancp* *rtrancp.rtrancp-refl*
rtrancp-cdcl_{NOT}-with-restart-bound-inv *rtrancp-cdcl_{NOT}-with-restart-cdcl_{NOT}-inv*
rtrancp-cdcl_{NOT}-raw-restart- μ -bound)

lemma *wf-cdcl_{NOT}-restart*:
 $\langle \text{wf } \{(T, S). \text{cdcl}_{NOT}\text{-restart } S \ T \wedge \text{cdcl}_{NOT}\text{-inv } (\text{fst } S)\} \rangle$ (is $\langle \text{wf } ?A \rangle$)
proof (rule *ccontr*)
assume $\langle \neg ?thesis \rangle$
then obtain *g* **where**

```

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

have snd-g: ⟨ $\bigwedge i. \text{snd } (g \ i) = i + \text{snd } (g \ 0)$ ⟩
  apply (induct-tac i)
  apply simp
  by (metis Suc-eq-plus1-left add commute add.left-commute
    cdclNOT-with-restart-increasing-number g)
then have snd-g-0: ⟨ $\bigwedge i. i > 0 \implies \text{snd } (g \ i) = i + \text{snd } (g \ 0)$ ⟩
  by blast
have unbounded-f-g: ⟨unbounded ( $\lambda i. f \ (\text{snd } (g \ i))$ )⟩
  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: ⟨ $\bigwedge T \ \text{Ta} \ m. (\text{cdcl}_{NOT} \ \sim m) \ T \ \text{Ta} \implies \text{no-step } \text{cdcl}_{NOT} \ T \implies m = 0$ ⟩
    apply (case-tac m) by simp (meson relpowp-E2)
  have ⟨ $\exists \ T \ m. (\text{cdcl}_{NOT} \ \sim m) \ (\text{fst } (g \ i)) \ T \wedge m \geq f \ (\text{snd } (g \ i))$ ⟩
    using g[of i] apply (cases rule: cdclNOT-restart.cases)
    apply auto[]
    using g[of ⟨Suc i⟩] f-ge-1 apply (cases rule: cdclNOT-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] cdclNOT-inv-g[of 0] apply (cases rule: cdclNOT-restart.cases)
  apply (metis One-nat-def cdclNOT-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 = ⟨ $\mu\text{-bound } A \ (\text{fst } (g \ 1)) + 1$ ⟩
obtain j where
  j: ⟨ $f \ (\text{snd } (g \ j)) > ?j$ ⟩ and ⟨ $j > 1$ ⟩
  using unbounded-f-g not-bounded-nat-exists-larger by blast
{
  fix i j
  have cdclNOT-with-restart: ⟨ $j \geq i \implies \text{cdcl}_{NOT}\text{-restart}^{**} \ (g \ i) \ (g \ j)$ ⟩
    apply (induction j)
    apply simp
    by (metis g le-Suc-eq rtranclp.rtrancl-into-rtrancl rtranclp.rtrancl-refl)
  } note cdclNOT-restart = this
have ⟨cdclNOT-inv (fst (g (Suc 0)))⟩
  by (simp add: cdclNOT-inv-g)
have ⟨cdclNOT-restart** (fst (g 1), snd (g 1)) (fst (g j), snd (g j))⟩
  using ⟨ $j > 1$ ⟩ by (simp add: cdclNOT-restart)
have ⟨ $\mu \ A \ (\text{fst } (g \ j)) \leq \mu\text{-bound } A \ (\text{fst } (g \ 1))$ ⟩
  apply (rule rtranclp-cdclNOT-raw-restart-measure-bound)
  using ⟨cdclNOT-restart** (fst (g 1), snd (g 1)) (fst (g j), snd (g j))⟩ apply blast
  apply (simp add: cdclNOT-inv-g)
  using ⟨bound-inv A (fst (g 1))⟩ apply simp
done
then have ⟨ $\mu \ A \ (\text{fst } (g \ j)) \leq ?j$ ⟩
  by auto
have inv: ⟨bound-inv A (fst (g j))⟩
  using ⟨bound-inv A (fst (g 1))⟩ ⟨cdclNOT-inv (fst (g (Suc 0)))⟩

```

```

  ⟨cdclNOT-restart** (fst (g 1), snd (g 1)) (fst (g j), snd (g j))⟩
  rtrancpl-cdclNOT-with-restart-bound-inv by auto
obtain T m where
  cdclNOT-m: ⟨cdclNOT  $\rightsquigarrow$  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))⟩
  cdclNOT-comp-bounded[OF inv cdclNOT-inv-g, of ] cdclNOT-inv-g cdclNOT-m
  ⟨?j < m⟩ by auto
qed

lemma cdclNOT-restart-steps-bigger-than-bound:
assumes
  ⟨cdclNOT-restart S T⟩ and
  ⟨bound-inv A (fst S)⟩ and
  ⟨cdclNOT-inv (fst S)⟩ and
  ⟨f (snd S) > μ-bound A (fst S)⟩
shows ⟨full1 cdclNOT (fst S) (fst T)⟩
using assms
proof (induction rule: cdclNOT-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
  cdclNOT-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 cdclNOT-inv μ m rtrancpl-cdclNOT-raw-restart-measure-bound by fastforce
then have False using cdclNOT-comp-n-le[of m' S T A] restart-step unfolding m by simp
then show ?case by fast
qed

lemma rtrancpl-cdclNOT-with-inv-inv-rtrancpl-cdclNOT:
assumes
  inv: ⟨cdclNOT-inv S⟩ and
  binv: ⟨bound-inv A S⟩
shows ⟨(λS T. cdclNOT S T ∧ cdclNOT-inv S ∧ bound-inv A S)** S T  $\longleftrightarrow$  cdclNOT** S T⟩
  (is ⟨?A** S T  $\longleftrightarrow$  ?B** S T⟩)
apply (rule iffI)
  using rtrancpl-mono[of ?A ?B] apply blast
apply (induction rule: rtrancpl-induct)
  using inv binv apply simp
by (metis (mono-tags, lifting) binv inv rtrancpl.simps rtrancpl-cdclNOT-bound-inv
  rtrancpl-cdclNOT-cdclNOT-inv)

lemma no-step-cdclNOT-restart-no-step-cdclNOT:
assumes
  n-s: ⟨no-step cdclNOT-restart S⟩ and
  inv: ⟨cdclNOT-inv (fst S)⟩ and
  binv: ⟨bound-inv A (fst S)⟩
shows ⟨no-step cdclNOT (fst S)⟩
proof (rule ccontr)

```

```

assume  $\neg ?thesis$ 
then obtain  $T$  where  $T: \langle cdcl_{NOT} (fst S) T \rangle$ 
  by blast
then obtain  $U$  where  $U: \langle full (\lambda S T. cdcl_{NOT} S T \wedge cdcl_{NOT-inv} S \wedge bound-inv A S) T U \rangle$ 
  using wf-exists-normal-form-full[OF wf-cdclNOT, of A T] by auto
moreover have  $inv-T: \langle cdcl_{NOT-inv} T \rangle$ 
  using  $\langle cdcl_{NOT} (fst S) T \rangle$   $cdcl_{NOT-inv} inv$  by blast
moreover have  $b-inv-T: \langle bound-inv A T \rangle$ 
  using  $\langle cdcl_{NOT} (fst S) T \rangle$   $binv bound-inv inv$  by blast
ultimately have  $\langle full cdcl_{NOT} T U \rangle$ 
  using  $rtrancp-cdcl_{NOT-with-inv-inv-rtrancp-cdcl_{NOT} rtrancp-cdcl_{NOT-bound-inv}$ 
     $rtrancp-cdcl_{NOT-cdcl_{NOT-inv}$  unfolding full-def by blast
then have  $\langle full1 cdcl_{NOT} (fst S) U \rangle$ 
  using  $T full-full1$  by metis
then show False by (metis n-s prod.collapse restart-full)
qed

end

```

2.2.6 Merging backjump and learning

```

locale cdclNOT-merge-bj-learn-ops =
  decide-ops trail clausesNOT prepend-trail tl-trail add-clsNOT remove-clsNOT decide-conds +
  forget-ops trail clausesNOT prepend-trail tl-trail add-clsNOT remove-clsNOT forget-conds +
  propagate-ops trail clausesNOT prepend-trail tl-trail add-clsNOT remove-clsNOT propagate-conds
for
  trail ::  $\langle 'st \Rightarrow ('v, unit) ann-lits \rangle$  and
  clausesNOT ::  $\langle 'st \Rightarrow 'v clauses \rangle$  and
  prepend-trail ::  $\langle ('v, unit) ann-lit \Rightarrow 'st \Rightarrow 'st \rangle$  and
  tl-trail ::  $\langle 'st \Rightarrow 'st \rangle$  and
  add-clsNOT ::  $\langle 'v clause \Rightarrow 'st \Rightarrow 'st \rangle$  and
  remove-clsNOT ::  $\langle 'v clause \Rightarrow 'st \Rightarrow 'st \rangle$  and
  decide-conds ::  $\langle 'st \Rightarrow 'st \Rightarrow bool \rangle$  and
  propagate-conds ::  $\langle ('v, unit) ann-lit \Rightarrow 'st \Rightarrow 'st \Rightarrow bool \rangle$  and
  forget-conds ::  $\langle 'v clause \Rightarrow 'st \Rightarrow bool \rangle$  +
  fixes backjump-l-cond ::  $\langle 'v clause \Rightarrow 'v clause \Rightarrow 'v literal \Rightarrow 'st \Rightarrow 'st \Rightarrow bool \rangle$ 
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:  $\langle trail S = F' @ Decided K \# F$ 
     $\Rightarrow T \sim prepend-trail (Propagated L ()) (reduce-trail-to_{NOT} F (add-cl_{NOT} C'' S))$ 
     $\Rightarrow C \in \# clauses_{NOT} S$ 
     $\Rightarrow trail S \models_{as} CNot C$ 
     $\Rightarrow undefined-lit F L$ 
     $\Rightarrow atm-of L \in atms-of-mm (clauses_{NOT} S) \cup atm-of ' (lits-of-l (trail S))$ 
     $\Rightarrow clauses_{NOT} S \models_{pm} add-mset L C'$ 
     $\Rightarrow C'' = add-mset L C'$ 
     $\Rightarrow F \models_{as} CNot C'$ 
     $\Rightarrow backjump-l-cond C C' L S T$ 
     $\Rightarrow backjump-l S T \rangle$ 

```

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

```

declare reduce-trail-toNOT-length-ne[simp del] Set.Un-iff[simp del] Set.insert-iff[simp del]

```

inductive-cases *backjump-LE*: $\langle \text{backjump-l } S \ T \rangle$

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* :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **for** *S* :: *'st* **where**

cdcl_{NOT}-merged-bj-learn-decide_{NOT}: $\langle \text{decide}_{\text{NOT}} S \ S' \Rightarrow \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn } S \ S' \rangle$ |
cdcl_{NOT}-merged-bj-learn-propagate_{NOT}: $\langle \text{propagate}_{\text{NOT}} S \ S' \Rightarrow \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn } S \ S' \rangle$ |
cdcl_{NOT}-merged-bj-learn-backjump-l: $\langle \text{backjump-l } S \ S' \Rightarrow \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn } S \ S' \rangle$ |
cdcl_{NOT}-merged-bj-learn-forget_{NOT}: $\langle \text{forget}_{\text{NOT}} S \ S' \Rightarrow \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn } S \ S' \rangle$

lemma *cdcl_{NOT}-merged-bj-learn-no-dup-inv*:

$\langle \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn } S \ T \Rightarrow \text{no-dup } (\text{trail } S) \Rightarrow \text{no-dup } (\text{trail } T) \rangle$

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-cl_{NOT}* *remove-cl_{NOT}*

decide-conds *propagate-conds* *forget-conds*

$\langle \lambda C \ C' \ L' \ S \ T. \text{backjump-l-cond } C \ C' \ L' \ S \ T$

$\wedge \text{distinct-mset } C' \wedge L' \notin \# \ C' \wedge \neg \text{tautology } (\text{add-mset } L' \ C') \rangle$

for

trail :: $\langle 'st \Rightarrow ('v, \text{unit}) \text{ ann-lits} \rangle$ **and**

clauses_{NOT} :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**

prepend-trail :: $\langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

tl-trail :: $\langle 'st \Rightarrow 'st \rangle$ **and**

add-cl_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

remove-cl_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

decide-conds :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **and**

propagate-conds :: $\langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **and**

forget-conds :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **and**

backjump-l-cond :: $\langle 'v \text{ clause} \Rightarrow 'v \text{ clause} \Rightarrow 'v \text{ literal} \Rightarrow 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ +

fixes

inv :: $\langle 'st \Rightarrow \text{bool} \rangle$

begin

abbreviation *backjump-conds* :: $\langle 'v \text{ clause} \Rightarrow 'v \text{ clause} \Rightarrow 'v \text{ literal} \Rightarrow 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$

where

$\langle \text{backjump-conds} \equiv \lambda C \ C' \ L' \ S \ T. \text{distinct-mset } C' \wedge L' \notin \# \ C' \wedge \neg \text{tautology } (\text{add-mset } L' \ C') \rangle$

sublocale *backjumping-ops* *trail* *clauses_{NOT}* *prepend-trail* *tl-trail* *add-cl_{NOT}* *remove-cl_{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-cl_{NOT}* *remove-cl_{NOT}*

decide-conds *propagate-conds* *forget-conds* *backjump-l-cond* *inv*

for

trail :: $\langle 'st \Rightarrow ('v, \text{unit}) \text{ ann-lits} \rangle$ **and**

clauses_{NOT} :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**

```

prepend-trail :: ⟨'v, unit⟩ ann-lit ⇒ 'st ⇒ 'st⟩ and
tl-trail :: ⟨'st ⇒ 'st⟩ and
add-clsNOT :: ⟨'v clause ⇒ 'st ⇒ 'st⟩ and
remove-clsNOT :: ⟨'v clause ⇒ 'st ⇒ 'st⟩ and
decide-cons :: ⟨'st ⇒ 'st ⇒ bool⟩ and
propagate-cons :: ⟨'v, unit⟩ ann-lit ⇒ 'st ⇒ 'st ⇒ bool⟩ and
forget-cons :: ⟨'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:
  ⟨ $\bigwedge S C F' K F L$ .
    inv S
    ⇒ trail S = F' @ Decided K # F
    ⇒ C ∈ # clausesNOT S
    ⇒ trail S ⊨as CNot C
    ⇒ undefined-lit F L
    ⇒ atm-of L ∈ atms-of-mm (clausesNOT S) ∪ atm-of ' (lits-of-l (F' @ Decided K # F))
    ⇒ clausesNOT S ⊨pm add-mset L C'
    ⇒ F ⊨as CNot C'
    ⇒ ¬no-step backjump-l S⟩ and
  cdcl-merged-inv: ⟨ $\bigwedge S T$ . cdclNOT-merged-bj-learn S T ⇒ inv S ⇒ inv T⟩ and
  can-propagate-or-decide-or-backjump-l:
  ⟨atm-of L ∈ atms-of-mm (clausesNOT S) ⇒
    undefined-lit (trail S) L ⇒
    inv S ⇒
    satisfiable (set-mset (clausesNOT S)) ⇒
    ∃ T. decideNOT S T ∨ propagateNOT 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 ∈ # clausesNOT S⟩ and
    T: ⟨T ∼ prepend-trail (Propagated L ()) (reduce-trail-toNOT F (add-clsNOT (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 (clausesNOT S) ∪ atm-of ' (lits-of-l (trail S))⟩ and
    S-C-L: ⟨clausesNOT 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 ∉ # 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' ∼ prepend-trail (Propagated L ()) (reduce-trail-toNOT F (add-clsNOT D S))⟩
    and
    tr-S: ⟨trail S = F' @ Decided K # F⟩ and
    C: ⟨C ∈ # clausesNOT 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 (clausesNOT S) ∪ atm-of ' (lits-of-l F' ∪ lits-of-l F))⟩
    and
    cls-S-C': ⟨clausesNOT 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 clausesNOT prepend-trail tl-trail add-clsNOT remove-clsNOT
  backjump-conds
  by unfold-locales
have ⟨∃ T. backjump S T⟩
  apply rule
  apply (rule backjump.intros)
    using tr-S apply simp
    apply (rule state-eqNOT-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*
 $\langle \lambda C \cdot \text{distinct-mset } C \wedge \neg \text{tautology } C \rangle$
forget-conds
by *unfold-locales*

lemma *backjump-l-learn-backjump:*

assumes *bt*: $\langle \text{backjump-l } S \ T \rangle$ **and** *inv*: $\langle \text{inv } S \rangle$

shows $\langle \exists C' L D. \text{learn } S \ (\text{add-cls}_{\text{NOT}} D \ S) \wedge D = \text{add-mset } L \ C' \wedge \text{backjump } (\text{add-cls}_{\text{NOT}} D \ S) \ T \wedge \text{atms-of } (\text{add-mset } L \ C') \subseteq \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \cup \text{atm-of } ' (\text{lits-of-l } (\text{trail } S)) \rangle$

proof –

obtain $C \ F' \ K \ F \ L \ l \ C' \ D$ **where**

tr-S: $\langle \text{trail } S = F' \ @ \text{Decided } K \ \# \ F \rangle$ **and**

T: $\langle T \sim \text{prepend-trail } (\text{Propagated } L \ l) \ (\text{reduce-trail-to}_{\text{NOT}} F \ (\text{add-cls}_{\text{NOT}} D \ S)) \rangle$ **and**

C-cls-S: $\langle C \in \# \text{clauses}_{\text{NOT}} S \rangle$ **and**

tr-S-CNot-C: $\langle \text{trail } S \models_{\text{as}} C \text{Not } C \rangle$ **and**

undef: $\langle \text{undefined-lit } F \ L \rangle$ **and**

atm-L: $\langle \text{atm-of } L \in \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \cup \text{atm-of } ' (\text{lits-of-l } (\text{trail } S)) \rangle$ **and**

clss-C: $\langle \text{clauses}_{\text{NOT}} S \models_{\text{pm}} D \rangle$ **and**

D: $\langle D = \text{add-mset } L \ C' \rangle$

$\langle F \models_{\text{as}} C \text{Not } C' \rangle$ **and**

distinct: $\langle \text{distinct-mset } D \rangle$ **and**

not-tauto: $\langle \neg \text{tautology } D \rangle$ **and**

cond: $\langle \text{backjump-l-cond } C \ C' \ L \ S \ T \rangle$

using *bt inv* **by** (*elim backjump-lE*) *simp*

have *atms-C'*: $\langle \text{atms-of } C' \subseteq \text{atm-of } ' (\text{lits-of-l } F) \rangle$

by (*metis D(2) atms-of-def image-subsetI true-annots-CNot-all-atms-defined*)

then have $\langle \text{atms-of } (\text{add-mset } L \ C') \subseteq \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \cup \text{atm-of } ' (\text{lits-of-l } (\text{trail } S)) \rangle$

using *atm-L tr-S* **by** *auto*

moreover have *learn*: $\langle \text{learn } S \ (\text{add-cls}_{\text{NOT}} D \ S) \rangle$

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*: $\langle \text{backjump } (\text{add-cls}_{\text{NOT}} D \ S) \ T \rangle$

apply (*rule backjump.intros[of - - - - L C C']*)

using $\langle F \models_{\text{as}} C \text{Not } C' \rangle$ *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*: $\langle \text{backjump-l } S \ T \rangle$ **and** *inv*: $\langle \text{inv } S \rangle$

shows $\langle \exists C' L D S'. \text{backjump } S \ S' \wedge \text{learn } S' \ T \wedge D = (\text{add-mset } L \ C') \wedge T \sim \text{add-cls}_{\text{NOT}} D \ S' \wedge \text{atms-of } (\text{add-mset } L \ C') \subseteq \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \cup \text{atm-of } ' (\text{lits-of-l } (\text{trail } S)) \wedge \text{clauses}_{\text{NOT}} S \models_{\text{pm}} D \rangle$

$\wedge \text{learn } S' \ T$

$\wedge D = (\text{add-mset } L \ C')$

$\wedge T \sim \text{add-cls}_{\text{NOT}} D \ S'$

$\wedge \text{atms-of } (\text{add-mset } L \ C') \subseteq \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \cup \text{atm-of } ' (\text{lits-of-l } (\text{trail } S))$

$\wedge \text{clauses}_{\text{NOT}} S \models_{\text{pm}} D$

proof –

obtain $C F' K F L \mid C' D$ where
 $tr-S$: $\langle trail\ S = F' @ Decided\ K \# F \rangle$ and
 T : $\langle T \sim prepend-trail\ (Propagated\ L\ l)\ (reduce-trail-to_{NOT}\ F\ (add-cl_{NOT}\ D\ S)) \rangle$ and
 $C-cl_{NOT}-S$: $\langle C \in \# clauses_{NOT}\ S \rangle$ and
 $tr-S-CNot-C$: $\langle trail\ S \models_{as}\ CNot\ C \rangle$ and
 $undef$: $\langle undefined-lit\ F\ L \rangle$ and
 $atm-L$: $\langle atm-of\ L \in atms-of-mm\ (clauses_{NOT}\ S) \cup atm-of\ ' (lits-of-l\ (trail\ S)) \rangle$ and
 $clss-C$: $\langle clauses_{NOT}\ S \models_{pm}\ D \rangle$ and
 D : $\langle D = add-mset\ L\ C' \rangle$
 $\langle F \models_{as}\ CNot\ C' \rangle$ and
 $distinct$: $\langle distinct-mset\ D \rangle$ and
 $not-tauto$: $\langle \neg\ tautology\ D \rangle$ and
 $cond$: $\langle backjump-l-cond\ C\ C'\ L\ S\ T \rangle$
using $bt\ inv$ by $(elim\ backjump-le)\ simp$
let $?S' = \langle prepend-trail\ (Propagated\ L\ ())\ (reduce-trail-to_{NOT}\ F\ S) \rangle$
have $atms-C'$: $\langle atms-of\ C' \subseteq atm-of\ ' (lits-of-l\ F) \rangle$
by $(metis\ D(2)\ atms-of-def\ image-subsetI\ true-annots-CNot-all-atms-defined)$
then have $\langle atms-of\ (add-mset\ L\ C') \subseteq atms-of-mm\ (clauses_{NOT}\ S) \cup atm-of\ ' (lits-of-l\ (trail\ S)) \rangle$
using $atm-L\ tr-S$ by $auto$
moreover have $learn$: $\langle learn\ ?S'\ T \rangle$
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 : $\langle backjump\ S\ (prepend-trail\ (Propagated\ L\ ())\ (reduce-trail-to_{NOT}\ F\ S)) \rangle$
apply $(rule\ backjump.intros[of\ S\ F'\ K\ F - L])$
using $\langle F \models_{as}\ CNot\ C' \rangle\ C-cl_{NOT}-S\ tr-S-CNot-C\ undef\ T\ distinct\ not-tauto\ D\ cond\ clss-C\ atm-L$
by $(auto\ simp:\ tr-S)$
moreover have $\langle T \sim (add-cl_{NOT}\ D\ ?S') \rangle$
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- $cdcl_{NOT}$:

$\langle cdcl_{NOT}$ -merged-bj-learn $S\ T \implies inv\ S \implies cdcl_{NOT}^{++}\ S\ T \rangle$

proof (induction rule: $cdcl_{NOT}$ -merged-bj-learn.induct)

case $(cdcl_{NOT}$ -merged-bj-learn-decide $_{NOT}\ T)$

then have $\langle cdcl_{NOT}\ S\ T \rangle$

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 $\langle cdcl_{NOT}\ S\ T \rangle$

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 $\langle cdcl_{NOT}\ S\ T \rangle$

using $c-forget_{NOT}$ by $blast$

then show $?case$ by $auto$

next

case $\langle \text{cdcl}_{NOT}\text{-merged-bj-learn-backjump-l } T \rangle$ **note** $bt = \text{this}(1)$ **and** $inv = \text{this}(2)$
obtain $C' :: \langle 'v \text{ clause} \rangle$ **and** $L :: \langle 'v \text{ literal} \rangle$ **and** $D :: \langle 'v \text{ clause} \rangle$ **where**
 $f3: \langle \text{learn } S \text{ (add-cl}_{NOT} D S) \wedge$
 $\text{backjump (add-cl}_{NOT} D S) T \wedge$
 $\text{atms-of (add-mset } L C') \subseteq \text{atms-of-mm (clauses}_{NOT} S) \cup \text{atm-of ' lits-of-l (trail } S) \rangle$ **and**
 $D: \langle D = \text{add-mset } L C' \rangle$
using $\text{backjump-l-learn-backjump}[OF bt inv]$ **by** blast
then have $f4: \langle \text{cdcl}_{NOT} S \text{ (add-cl}_{NOT} D S) \rangle$
using $c\text{-learn}$ **by** blast
have $\langle \text{cdcl}_{NOT} \text{ (add-cl}_{NOT} D S) T \rangle$
using $f3$ $\text{bj-backjump } c\text{-dpll-bj}$ **by** blast
then show $?case$
using $f4$ **by** $(\text{meson } \text{trancpl.r-into-trancpl } \text{trancpl.trancpl-into-trancpl})$

qed

lemma $\text{rtrancpl-cdcl}_{NOT}\text{-merged-bj-learn-is-rtrancpl-cdcl}_{NOT}\text{-and-inv}$:

$\langle \text{cdcl}_{NOT}\text{-merged-bj-learn}^{**} S T \implies inv S \implies \text{cdcl}_{NOT}^{**} S T \wedge inv T \rangle$

proof (induction rule: rtrancpl-induct)

case $base$

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

next

case $(\text{step } T U)$ **note** $st = \text{this}(1)$ **and** $\text{cdcl}_{NOT} = \text{this}(2)$ **and** $IH = \text{this}(3)[OF \text{this}(4-)]$ **and**
 $inv = \text{this}(4)$

have $\langle \text{cdcl}_{NOT}^{**} T U \rangle$

using $\text{cdcl}_{NOT}\text{-merged-bj-learn-is-trancpl-cdcl}_{NOT}[OF \text{cdcl}_{NOT}] IH$
 inv **by** $auto$

then have $\langle \text{cdcl}_{NOT}^{**} S U \rangle$ **using** IH **by** fastforce

moreover have $\langle inv U \rangle$ **using** IH cdcl_{NOT} $\text{cdcl-merged-inv } inv$ **by** blast

ultimately show $?case$ **using** st **by** fast

qed

lemma $\text{rtrancpl-cdcl}_{NOT}\text{-merged-bj-learn-is-rtrancpl-cdcl}_{NOT}$:

$\langle \text{cdcl}_{NOT}\text{-merged-bj-learn}^{**} S T \implies inv S \implies \text{cdcl}_{NOT}^{**} S T \rangle$

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

lemma $\text{rtrancpl-cdcl}_{NOT}\text{-merged-bj-learn-inv}$:

$\langle \text{cdcl}_{NOT}\text{-merged-bj-learn}^{**} S T \implies inv S \implies inv T \rangle$

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

lemma $\text{rtrancpl-cdcl}_{NOT}\text{-merged-bj-learn-no-dup-inv}$:

$\langle \text{cdcl}_{NOT}\text{-merged-bj-learn}^{**} S T \implies \text{no-dup (trail } S) \implies \text{no-dup (trail } T) \rangle$

by (induction rule: rtrancpl-induct) (auto simp: $\text{cdcl}_{NOT}\text{-merged-bj-learn-no-dup-inv}$)

definition $\mu_C' :: \langle 'v \text{ clause set} \Rightarrow 'st \Rightarrow nat \rangle$ **where**

$\langle \mu_C' A T \equiv \mu_C (1 + \text{card (atms-of-ms } A)) (2 + \text{card (atms-of-ms } A)) (\text{trail-weight } T) \rangle$

definition $\mu_{CDCL}'\text{-merged} :: \langle 'v \text{ clause set} \Rightarrow 'st \Rightarrow nat \rangle$ **where**

$\langle \mu_{CDCL}'\text{-merged } A T \equiv$
 $((2 + \text{card (atms-of-ms } A)) \wedge (1 + \text{card (atms-of-ms } A)) - \mu_C' A T) * 2 + \text{card (set-mset (clauses}_{NOT} T)) \rangle$

lemma $\text{cdcl}_{NOT}\text{-decreasing-measure}'$:

assumes

$\langle \text{cdcl}_{NOT}\text{-merged-bj-learn } S T \rangle$ **and**

$inv: \langle inv S \rangle$ **and**

$atm-clss$: $\langle atm\text{-}of\text{-}mm \ (clauses_{NOT} \ S) \subseteq atm\text{-}of\text{-}ms \ A \rangle$ and
 $atm-trail$: $\langle atm\text{-}of \ ' \ lits\text{-}of\text{-}l \ (trail \ S) \subseteq atm\text{-}of\text{-}ms \ A \rangle$ and
 $n\text{-}d$: $\langle no\text{-}dup \ (trail \ S) \rangle$ and
 $fin\text{-}A$: $\langle finite \ A \rangle$
shows $\langle \mu_{CDCL}'\text{-merged} \ A \ T < \mu_{CDCL}'\text{-merged} \ A \ S \rangle$
using $assms(1)$
proof *induction*
case $\langle cdcl_{NOT}\text{-merged}\text{-bj}\text{-learn}\text{-decide}_{NOT} \ T \rangle$
have $\langle clauses_{NOT} \ S = clauses_{NOT} \ T \rangle$
using $cdcl_{NOT}\text{-merged}\text{-bj}\text{-learn}\text{-decide}_{NOT}.hyps$ **by** *auto*
moreover **have**
 $\langle (2 + card \ (atms\text{-}of\text{-}ms \ A)) \wedge (1 + card \ (atms\text{-}of\text{-}ms \ A))$
 $\quad - \mu_C \ (1 + card \ (atms\text{-}of\text{-}ms \ A)) \ (2 + card \ (atms\text{-}of\text{-}ms \ A)) \ (trail\text{-}weight \ T)$
 $< (2 + card \ (atms\text{-}of\text{-}ms \ A)) \wedge (1 + card \ (atms\text{-}of\text{-}ms \ A))$
 $\quad - \mu_C \ (1 + card \ (atms\text{-}of\text{-}ms \ A)) \ (2 + card \ (atms\text{-}of\text{-}ms \ A)) \ (trail\text{-}weight \ S) \rangle$
apply $(rule \ dpll\text{-bj}\text{-trail}\text{-mes}\text{-decreasing}\text{-prop})$
using $cdcl_{NOT}\text{-merged}\text{-bj}\text{-learn}\text{-decide}_{NOT} \ fin\text{-}A \ atm\text{-}clss \ atm\text{-}trail \ n\text{-}d \ inv$
by $(simp\text{-}all \ add: \ bj\text{-decide}_{NOT} \ cdcl_{NOT}\text{-merged}\text{-bj}\text{-learn}\text{-decide}_{NOT}.hyps)$
ultimately **show** $?case$
unfolding $\mu_{CDCL}'\text{-merged}\text{-def} \ \mu_C'\text{-def}$ **by** *simp*
next
case $\langle cdcl_{NOT}\text{-merged}\text{-bj}\text{-learn}\text{-propagate}_{NOT} \ T \rangle$
have $\langle clauses_{NOT} \ S = clauses_{NOT} \ T \rangle$
using $cdcl_{NOT}\text{-merged}\text{-bj}\text{-learn}\text{-propagate}_{NOT}.hyps$
by $(simp \ add: \ bj\text{-propagate}_{NOT} \ inv \ dpll\text{-bj}\text{-clauses})$
moreover **have**
 $\langle (2 + card \ (atms\text{-}of\text{-}ms \ A)) \wedge (1 + card \ (atms\text{-}of\text{-}ms \ A))$
 $\quad - \mu_C \ (1 + card \ (atms\text{-}of\text{-}ms \ A)) \ (2 + card \ (atms\text{-}of\text{-}ms \ A)) \ (trail\text{-}weight \ T)$
 $< (2 + card \ (atms\text{-}of\text{-}ms \ A)) \wedge (1 + card \ (atms\text{-}of\text{-}ms \ A))$
 $\quad - \mu_C \ (1 + card \ (atms\text{-}of\text{-}ms \ A)) \ (2 + card \ (atms\text{-}of\text{-}ms \ A)) \ (trail\text{-}weight \ S) \rangle$
apply $(rule \ dpll\text{-bj}\text{-trail}\text{-mes}\text{-decreasing}\text{-prop})$
using $inv \ n\text{-}d \ atm\text{-}clss \ atm\text{-}trail \ fin\text{-}A$ **by** $(simp\text{-}all \ add: \ bj\text{-propagate}_{NOT} \ cdcl_{NOT}\text{-merged}\text{-bj}\text{-learn}\text{-propagate}_{NOT}.hyps)$
ultimately **show** $?case$
unfolding $\mu_{CDCL}'\text{-merged}\text{-def} \ \mu_C'\text{-def}$ **by** *simp*
next
case $\langle cdcl_{NOT}\text{-merged}\text{-bj}\text{-learn}\text{-forget}_{NOT} \ T \rangle$
have $\langle card \ (set\text{-mset} \ (clauses_{NOT} \ T)) < card \ (set\text{-mset} \ (clauses_{NOT} \ S)) \rangle$
using $\langle forget_{NOT} \ S \ T \rangle$ **by** $(metis \ card\text{-Diff1}\text{-less} \ clauses\text{-remove}\text{-cls}_{NOT} \ finite\text{-set}\text{-mset} \ forget_{NOT}.cases \ linear \ set\text{-mset}\text{-minus}\text{-replicate}\text{-mset}(1) \ state\text{-eq}_{NOT}\text{-def})$
moreover
have $\langle trail \ S = trail \ T \rangle$
using $\langle forget_{NOT} \ S \ T \rangle$ **by** $(auto \ elim: \ forget_{NOT}E)$
then **have**
 $\langle (2 + card \ (atms\text{-}of\text{-}ms \ A)) \wedge (1 + card \ (atms\text{-}of\text{-}ms \ A))$
 $\quad - \mu_C \ (1 + card \ (atms\text{-}of\text{-}ms \ A)) \ (2 + card \ (atms\text{-}of\text{-}ms \ A)) \ (trail\text{-}weight \ T)$
 $= (2 + card \ (atms\text{-}of\text{-}ms \ A)) \wedge (1 + card \ (atms\text{-}of\text{-}ms \ A))$
 $\quad - \mu_C \ (1 + card \ (atms\text{-}of\text{-}ms \ A)) \ (2 + card \ (atms\text{-}of\text{-}ms \ A)) \ (trail\text{-}weight \ S) \rangle$
by *auto*
ultimately **show** $?case$
unfolding $\mu_{CDCL}'\text{-merged}\text{-def} \ \mu_C'\text{-def}$ **by** *simp*
next
case $\langle cdcl_{NOT}\text{-merged}\text{-bj}\text{-learn}\text{-backjump}\text{-l} \ T \rangle$ **note** $bj\text{-l} = this(1)$
obtain $C' \ L \ D \ S'$ **where**
 $learn$: $\langle learn \ S' \ T \rangle$ **and**
 bj : $\langle backjump \ S \ S' \rangle$ **and**

$atms-C: \langle atms-of \ (add-mset \ L \ C') \subseteq atms-of-mm \ (clauses_{NOT} \ S) \cup atm-of \ ' \ (lits-of-l \ (trail \ S)) \rangle$
and
 $D: \langle D = add-mset \ L \ C' \rangle$ **and**
 $T: \langle T \sim add-cls_{NOT} \ D \ S' \rangle$
using $bj-l \ inv \ backjump-l-backjump-learn \ [of \ S] \ n-d \ atm-clss \ atm-trail \ by \ blast$
have $card-T-S: \langle card \ (set-mset \ (clauses_{NOT} \ T)) \leq 1 + card \ (set-mset \ (clauses_{NOT} \ S)) \rangle$
using $bj-l \ inv \ by \ (force \ elim!: \ backjump-lE \ simp: \ card-insert-if)$
have $tr-S-T: \langle trail-weight \ S' = trail-weight \ T \rangle$
using $T \ by \ auto$
have
 $((2 + card \ (atms-of-ms \ A)) \wedge (1 + card \ (atms-of-ms \ A))$
 $- \mu_C \ (1 + card \ (atms-of-ms \ A)) \ (2 + card \ (atms-of-ms \ A)) \ (trail-weight \ S'))$
 $< ((2 + card \ (atms-of-ms \ A)) \wedge (1 + card \ (atms-of-ms \ A))$
 $- \mu_C \ (1 + card \ (atms-of-ms \ A)) \ (2 + card \ (atms-of-ms \ A))$
 $\ (trail-weight \ S)) \rangle$
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 $card-T-S \ unfolding \ \mu_{CDCL}'-merged-def \ \mu_C'-def \ tr-S-T \ by \ linarith$
qed

lemma $wf-cdcl_{NOT}-merged-bj-learn:$

assumes

$fin-A: \langle finite \ A \rangle$

shows $\langle wf \ \{ (T, S). \$

$(inv \ S \wedge atms-of-mm \ (clauses_{NOT} \ S) \subseteq atms-of-ms \ A \wedge atm-of \ ' \ lits-of-l \ (trail \ S) \subseteq atms-of-ms \ A$
 $\wedge no-dup \ (trail \ S)) \rangle$

$\wedge cdcl_{NOT}-merged-bj-learn \ S \ T \rangle$

apply $(rule \ wfP-if-measure[of \ - \ - \ \mu_{CDCL}'-merged \ A])$

using $cdcl_{NOT}-decreasing-measure' \ fin-A \ by \ simp$

lemma $in-atms-neg-defined: \langle x \in atms-of \ C' \implies F \models_{as} CNot \ C' \implies x \in atm-of \ ' \ lits-of-l \ F \rangle$

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 $\langle cdcl_{NOT}-merged-bj-learn \ S \ T \rangle$ **and** $\langle inv \ S \rangle$

shows $\langle atms-of-mm \ (clauses_{NOT} \ T) \subseteq atms-of-mm \ (clauses_{NOT} \ S) \cup atm-of \ ' \ (lits-of-l \ (trail \ S)) \rangle$

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

$\langle cdcl_{NOT}-merged-bj-learn \ S \ T \rangle$ **and** $\langle inv \ S \rangle$ **and**

$\langle atms-of-mm \ (clauses_{NOT} \ S) \subseteq A \rangle$ **and**

$\langle atm-of \ ' \ (lits-of-l \ (trail \ S)) \subseteq A \rangle$

shows $\langle \text{atm-of } '(\text{lits-of-l } (\text{trail } T)) \subseteq A \rangle$
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: $\langle \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn}^{**} S T \rangle$ **and**
 inv: $\langle \text{inv } S \rangle$ **and**
 atms-clauses-S: $\langle \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \subseteq A \rangle$ **and**
 atms-trail-S: $\langle \text{atm-of } '(\text{lits-of-l } (\text{trail } S)) \subseteq A \rangle$
shows $\langle \text{atm-of } '(\text{lits-of-l } (\text{trail } T)) \subseteq A \wedge \text{atms-of-mm } (\text{clauses}_{\text{NOT}} T) \subseteq A \rangle$
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 $\langle \text{inv } T \rangle$ **using** *inv st rtranclp-cdcl_{NOT}-merged-bj-learn-is-rtranclp-cdcl_{NOT}-and-inv* **by** *blast*

then have $\langle \text{atms-of-mm } (\text{clauses}_{\text{NOT}} U) \subseteq A \rangle$

using *cdcl_{NOT}-merged-bj-learn-atms-of-ms-clauses-decreasing cdcl_{NOT} IH $\langle \text{inv } T \rangle$* **by** *fast*

moreover

have $\langle \text{atm-of } '(\text{lits-of-l } (\text{trail } U)) \subseteq A \rangle$

using *cdcl_{NOT}-merged-bj-learn-atms-in-trail-in-set[of - A] $\langle \text{inv } T \rangle$ cdcl_{NOT} step.IH* **by** *auto*

ultimately show ?*case* **by** *fast*

qed

lemma *cdcl_{NOT}-merged-bj-learn-trail-clauses-bound:*

assumes
 cdcl: $\langle \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn } S T \rangle$ **and**
 inv: $\langle \text{inv } S \rangle$ **and**
 atms-clauses-S: $\langle \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \subseteq A \rangle$ **and**
 atms-trail-S: $\langle \text{atm-of } '(\text{lits-of-l } (\text{trail } S)) \subseteq A \rangle$
shows $\langle \text{atm-of } '(\text{lits-of-l } (\text{trail } T)) \subseteq A \wedge \text{atms-of-mm } (\text{clauses}_{\text{NOT}} T) \subseteq A \rangle$
using *rtranclp-cdcl_{NOT}-merged-bj-learn-trail-clauses-bound[of S T] assms* **by** *auto*

lemma *tranclp-cdcl_{NOT}-cdcl_{NOT}-tranclp:*

assumes

$\langle \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn}^{++} S T \rangle$ **and**

inv: $\langle \text{inv } S \rangle$ **and**

atm-clss: $\langle \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \subseteq \text{atms-of-ms } A \rangle$ **and**

atm-trail: $\langle \text{atm-of } ' \text{lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A \rangle$ **and**

n-d: $\langle \text{no-dup } (\text{trail } S) \rangle$ **and**

fin-A[simp]: $\langle \text{finite } A \rangle$

shows $\langle (T, S) \in \{(T, S). \}$

$(\text{inv } S \wedge \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \subseteq \text{atms-of-ms } A \wedge \text{atm-of } ' \text{lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A$
 $\wedge \text{no-dup } (\text{trail } S))$

$\wedge \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn } S T \rangle^{+} \text{ (is } \langle - \in ?P^{+} \rangle)$

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 cdclNOT = this(2) and IH = this(3)
have st: ⟨cdclNOT-merged-bj-learn** S T⟩
  using [[simp-trace]]
  by (simp add: rtrancpl-unfold st)
have ⟨cdclNOT** S T⟩
  apply (rule rtrancpl-cdclNOT-merged-bj-learn-is-rtrancpl-cdclNOT)
  using st cdclNOT inv n-d atm-clss atm-trail inv by auto
have ⟨inv T⟩
  apply (rule rtrancpl-cdclNOT-merged-bj-learn-inv)
  using inv st cdclNOT n-d atm-clss atm-trail inv by auto
moreover have ⟨atms-of-mm (clausesNOT T) ⊆ atms-of-ms A⟩
  using rtrancpl-cdclNOT-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 rtrancpl-cdclNOT-merged-bj-learn-trail-clauses-bound[OF st inv atm-clss atm-trail]
  by fast
moreover have ⟨no-dup (trail T)⟩
  using rtrancpl-cdclNOT-merged-bj-learn-no-dup-inv[OF st n-d] by fast
ultimately have ⟨(U, T) ∈ ?P⟩
  using cdclNOT by auto
then show ?case using IH by (simp add: trancpl-into-trancpl2)
qed

```

lemma wf-trancpl-cdcl_{NOT}-merged-bj-learn:

```

assumes ⟨finite A⟩
shows ⟨wf {(T, S).
  (inv S ∧ atms-of-mm (clausesNOT S) ⊆ atms-of-ms A ∧ atm-of ‘ lits-of-l (trail S) ⊆ atms-of-ms A
  ∧ no-dup (trail S))
  ∧ cdclNOT-merged-bj-learn++ S T}⟩
apply (rule wf-subset)
apply (rule wf-trancpl[OF wf-cdclNOT-merged-bj-learn])
using assms apply simp
using trancpl-cdclNOT-cdclNOT-trancpl[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 cdclNOT-merged-bj-learn S⟩ and
  atms-S: ⟨atms-of-mm (clausesNOT 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 (clausesNOT S) (get-all-ann-decomposition (trail S))⟩
shows ⟨unsatisfiable (set-mset (clausesNOT S))
  ∨ (trail S ⊨asm clausesNOT S ∧ satisfiable (set-mset (clausesNOT S)))⟩

```

proof –

```

let ?N = ⟨set-mset (clausesNOT 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** $\langle C \in ?N \rangle$ **and** $\langle \neg ?M \models_a C \rangle$ **using** *M* **unfolding** *true-annots-def* **by** *auto*

obtain *I* **::** $\langle 'v \text{ literal set} \rangle$ **where**

$\langle I \models_s ?N \rangle$ **and**

cons: $\langle \text{consistent-interp } I \rangle$ **and**

tot: $\langle \text{total-over-m } I \ ?N \rangle$ **and**

atm-I-N: $\langle \text{atm-of } 'I \subseteq \text{atms-of-ms } ?N \rangle$

using *sat* **unfolding** *satisfiable-def-min* **by** *auto*

let *?I* = $\langle I \cup \{P \mid P \in \text{lits-of-l } ?M \wedge \text{atm-of } P \notin \text{atm-of } 'I \} \rangle$

let *?O* = $\langle \{ \text{unmark } L \mid L. \text{is-decided } L \wedge L \in \text{set } ?M \wedge \text{atm-of } (\text{lit-of } L) \notin \text{atms-of-ms } ?N \} \rangle$

have *cons-I'*: $\langle \text{consistent-interp } ?I \rangle$

using *cons* **using** $\langle \text{no-dup } ?M \rangle$ **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'*: $\langle \text{total-over-m } ?I \ (\ ?N \cup \text{unmark-l } ?M) \rangle$

using *tot* *atms-of-s-def* **unfolding** *total-over-m-def* *total-over-set-def*

by (*fastforce simp: image-iff*)

have $\langle \{P \mid P \in \text{lits-of-l } ?M \wedge \text{atm-of } P \notin \text{atm-of } 'I \} \models_s ?O \rangle$

using $\langle I \models_s ?N \rangle$ *atm-I-N* **by** (*auto simp add: atm-of-eq-atm-of true-clss-def lits-of-def*)

then have *I'-N*: $\langle ?I \models_s ?N \cup ?O \rangle$

using $\langle I \models_s ?N \rangle$ *true-clss-union-increase* **by** *force*

have *tot'*: $\langle \text{total-over-m } ?I \ (\ ?N \cup ?O) \rangle$

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*: $\langle \text{atms-of-ms } ?N \subseteq \text{atm-of } ' \text{lits-of-l } ?M \rangle$

proof (*rule ccontr*)

assume $\langle \neg ?thesis \rangle$

then obtain *l* **::** $\langle 'v \text{ where}$

l-N: $\langle l \in \text{atms-of-ms } ?N \rangle$ **and**

l-M: $\langle l \notin \text{atm-of } ' \text{lits-of-l } ?M \rangle$

by *auto*

have $\langle \text{undefined-lit } ?M \ (\text{Pos } l) \rangle$

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* $\langle \text{Pos } l \rangle$ *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 $\langle ?M \models_{as} C\text{Not } C \rangle$

apply (*rule all-variables-defined-not-imply-cnot*)

using *atms-N-M* $\langle C \in ?N \rangle$ $\langle \neg ?M \models_a C \rangle$ *atms-of-atms-of-ms-mono*[*OF* $\langle C \in ?N \rangle$]

by (*auto dest: atms-of-atms-of-ms-mono*)

have $\langle \exists l \in \text{set } ?M. \text{is-decided } l \rangle$

proof (*rule ccontr*)

let *?O* = $\langle \{ \text{unmark } L \mid L. \text{is-decided } L \wedge L \in \text{set } ?M \wedge \text{atm-of } (\text{lit-of } L) \notin \text{atms-of-ms } ?N \} \rangle$

have $\vartheta[\text{iff}]: \langle \bigwedge I. \text{total-over-m } I \ (\ ?N \cup ?O \cup \text{unmark-l } ?M) \rangle$

$\longleftrightarrow \text{total-over-m } I \ (\ ?N \cup \text{unmark-l } ?M) \rangle$

unfolding *total-over-set-def* *total-over-m-def* *atms-of-ms-def* **by** *blast*

assume $\langle \neg ?thesis \rangle$

then have [*simp*]: $\langle \{ \text{unmark } L \mid L. \text{is-decided } L \wedge L \in \text{set } ?M \} \rangle$

= $\langle \{ \text{unmark } L \mid L. \text{is-decided } L \wedge L \in \text{set } ?M \wedge \text{atm-of } (\text{lit-of } L) \notin \text{atms-of-ms } ?N \} \rangle$

```

    by auto
  then have  $\langle ?N \cup ?O \models_{ps} \text{unmark-l } ?M \rangle$ 
    using all-decomposition-implies-propagated-lits-are-implied[OF decomp] by auto

  then have  $\langle ?I \models_s \text{unmark-l } ?M \rangle$ 
    using cons-I' I'-N tot-I'  $\langle ?I \models_s ?N \cup ?O \rangle$  unfolding  $\vartheta$  true-clss-clss-def by blast
  then have  $\langle \text{lits-of-l } ?M \subseteq ?I \rangle$ 
    unfolding true-clss-def lits-of-def by auto
  then have  $\langle ?M \models_{as} ?N \rangle$ 
    using I'-N  $\langle C \in ?N \rangle \langle \neg ?M \models_a C \rangle$  cons-I' atms-N-M
    by (meson  $\langle \text{trail } S \models_{as} C\text{Not } C \rangle$  consistent-CNot-not rev-subsetD sup-ge1 true-annot-def
      true-annots-def true-clss-mono-set-mset-l true-clss-def)
  then show False using M by fast
qed

from List.split-list-first-propE[OF this] obtain  $K :: \langle 'v \text{ literal} \rangle$  and  $d :: \text{unit}$  and
   $F F' :: \langle ('v, \text{unit}) \text{ ann-lits} \rangle$  where
   $M\text{-}K: \langle ?M = F' @ \text{Decided } K \# F \rangle$  and
   $nm: \langle \forall f \in \text{set } F'. \neg \text{is-decided } f \rangle$ 
  by (metis (full-types) is-decided-ex-Decided old.unit.exhaust)
let  $?K = \langle \text{Decided } K :: ('v, \text{unit}) \text{ ann-lit} \rangle$ 
have  $\langle ?K \in \text{set } ?M \rangle$ 
  unfolding M-K by auto
let  $?C = \langle \text{image-mset lit-of } \{ \#L \in \#mset ?M. \text{is-decided } L \wedge L \neq ?K \# \} :: 'v \text{ clause} \rangle$ 
let  $?C' = \langle \text{set-mset (image-mset } (\lambda L :: 'v \text{ literal}. \{ \#L \# \}) (?C + \text{unmark } ?K)) \rangle$ 
have  $\langle ?N \cup \{ \text{unmark } L \mid L. \text{is-decided } L \wedge L \in \text{set } ?M \} \models_{ps} \text{unmark-l } ?M \rangle$ 
  using all-decomposition-implies-propagated-lits-are-implied[OF decomp] .
moreover have  $C': \langle ?C' = \{ \text{unmark } L \mid L. \text{is-decided } L \wedge L \in \text{set } ?M \} \rangle$ 
  unfolding M-K apply standard
  apply force
  by auto
ultimately have  $N\text{-}C\text{-}M: \langle ?N \cup ?C' \models_{ps} \text{unmark-l } ?M \rangle$ 
  by auto
have  $N\text{-}M\text{-}False: \langle ?N \cup (\lambda L. \text{unmark } L) ' (\text{set } ?M) \models_{ps} \{ \{ \# \} \} \rangle$ 
  unfolding true-clss-clss-def true-annots-def Ball-def true-annot-def
proof (intro allI impI)
  fix  $LL :: 'v \text{ literal set}$ 
  assume
     $tot: \langle \text{total-over-m } LL (\text{set-mset (clauses}_{NOT} S) \cup \text{unmark-l (trail } S) \cup \{ \{ \# \} \}) \rangle$  and
     $cons: \langle \text{consistent-interp } LL \rangle$  and
     $LL: \langle LL \models_s \text{set-mset (clauses}_{NOT} S) \cup \text{unmark-l (trail } S) \rangle$ 
  have  $\langle \text{total-over-m } LL (C\text{Not } C) \rangle$ 
    by (metis  $\langle C \in \# \text{clauses}_{NOT} S \rangle$  insert-absorb tot total-over-m-CNot-toal-over-m
      total-over-m-insert total-over-m-union)
  then have  $\text{total-over-m } LL (\text{unmark-l (trail } S) \cup C\text{Not } C)$ 
    using tot by force
  then show  $LL \models_s \{ \{ \# \} \}$ 
    using tot cons LL
    by (metis (no-types)  $\langle C \in \# \text{clauses}_{NOT} S \rangle \langle \text{trail } S \models_{as} C\text{Not } C \rangle$  consistent-CNot-not
      true-annots-true-clss-clss true-clss-clss-def true-clss-def true-clss-union)
qed

have  $\langle \text{undefined-lit } F K \rangle$  using  $\langle \text{no-dup } ?M \rangle$  unfolding M-K by (auto simp: defined-lit-map)
moreover {
  have  $\langle ?N \cup ?C' \models_{ps} \{ \{ \# \} \} \rangle$ 
  proof -
    have  $A: \langle ?N \cup ?C' \cup \text{unmark-l } ?M = ?N \cup \text{unmark-l } ?M \rangle$ 

```

```

    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  $\langle ?N \models_p \text{image-mset } \text{uminus } ?C + \{\#\text{-}K\# \} \rangle$ 
  unfolding true-clss-clss-def true-clss-clss-def total-over-m-def
  proof (intro allI impI)
    fix I
    assume
      tot:  $\langle \text{total-over-set } I \text{ (atms-of-ms } (?N \cup \{\text{image-mset } \text{uminus } ?C + \{\#\text{-}K\# \}\}) \rangle$  and
      cons:  $\langle \text{consistent-interp } I \rangle$  and
       $\langle I \models_s ?N \rangle$ 
    have  $\langle (K \in I \wedge -K \notin I) \vee (-K \in I \wedge K \notin I) \rangle$ 
      using cons tot unfolding consistent-interp-def by (cases K) auto
    have  $\langle \{a \in \text{set } (\text{trail } S). \text{is-decided } a \wedge a \neq \text{Decided } K\} =$ 
       $\text{set } (\text{trail } S) \cap \{L. \text{is-decided } L \wedge L \neq \text{Decided } K\} \rangle$ 
      by auto
    then have tot':  $\langle \text{total-over-set } I$ 
       $(\text{atm-of 'lit-of ' (set } ?M \cap \{L. \text{is-decided } L \wedge L \neq \text{Decided } K\})) \rangle$ 
      using tot by (auto simp add: atms-of-uminus-lit-atm-of-lit-of)
    { fix x ::  $\langle ('v, \text{unit}) \text{ann-lit} \rangle$ 
      assume
        a3:  $\langle \text{lit-of } x \notin I \rangle$  and
        a1:  $\langle x \in \text{set } ?M \rangle$  and
        a4:  $\langle \text{is-decided } x \rangle$  and
        a5:  $\langle x \neq \text{Decided } K \rangle$ 
      then have  $\langle \text{Pos } (\text{atm-of } (\text{lit-of } x)) \in I \vee \text{Neg } (\text{atm-of } (\text{lit-of } x)) \in I \rangle$ 
        using a5 a4 tot' a1 unfolding total-over-set-def atms-of-s-def by blast
      moreover have f6:  $\langle \text{Neg } (\text{atm-of } (\text{lit-of } x)) = - \text{Pos } (\text{atm-of } (\text{lit-of } x)) \rangle$ 
        by simp
      ultimately have  $\langle - \text{lit-of } x \in I \rangle$ 
        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  $\langle \neg I \models_s ?C' \rangle$ 
    using  $\langle ?N \cup ?C' \models_{ps} \{\{\#\}\} \rangle$  tot cons  $\langle I \models_s ?N \rangle$ 
    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  $\langle I \models \text{image-mset } \text{uminus } ?C + \{\#\text{-}K\# \} \rangle$ 
    unfolding true-clss-def true-clss-def Bex-def
    using  $\langle (K \in I \wedge -K \notin I) \vee (-K \in I \wedge K \notin I) \rangle$ 
    by (auto dest!: H)
qed }
moreover have  $\langle F \models_{as} \text{CNot } (\text{image-mset } \text{uminus } ?C) \rangle$ 
  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  $\neg K$ ]
   $\langle \text{image-mset } \text{uminus } (\text{image-mset lit-of } \{\#\text{ } L : \#\text{ mset } ?M. \text{is-decided } L \wedge L \neq \text{Decided } K\# \}) \rangle$ 
   $\langle C \in ?N \rangle$  n-s  $\langle ?M \models_{as} \text{CNot } C \rangle$  bj-backjump inv sat unfolding M-K
  by (auto simp: cdclNOT-merged-bj-learn.simps)
then show ?thesis by fast
qed auto
qed

```

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

assumes $\langle \text{cdcl}_{NOT}\text{-merged-bj-learn } S \ T \rangle$ **and** $\text{inv}: \langle \text{inv } S \rangle$
 $\langle \text{all-decomposition-implies-m } (\text{clauses}_{NOT} \ S) \ (\text{get-all-ann-decomposition } (\text{trail } S)) \rangle$
shows
 $\langle \text{all-decomposition-implies-m } (\text{clauses}_{NOT} \ T) \ (\text{get-all-ann-decomposition } (\text{trail } T)) \rangle$
using *assms*
proof (*induction rule: cdcl_{NOT}-merged-bj-learn.induct*)
case $\langle \text{cdcl}_{NOT}\text{-merged-bj-learn-backjump-l } T \rangle$ **note** $\text{bj-l} = \text{this}(1)$
obtain $C' \ L \ D \ S'$ **where**
 $\text{learn}: \langle \text{learn } S' \ T \rangle$ **and**
 $\text{bj}: \langle \text{backjump } S \ S' \rangle$ **and**
 $\text{atms-C}: \langle \text{atms-of } (\text{add-mset } L \ C') \subseteq \text{atms-of-mm } (\text{clauses}_{NOT} \ S) \cup \text{atm-of } ' \ (\text{lits-of-l } (\text{trail } S)) \rangle$
and
 $D: \langle D = \text{add-mset } L \ C' \rangle$ **and**
 $T: \langle T \sim \text{add-cl}_{NOT} \ D \ S' \rangle$
using $\text{bj-l inv backjump-l-backjump-learn [of } S \text{] by blast}$
have $\langle \text{all-decomposition-implies-m } (\text{clauses}_{NOT} \ S') \ (\text{get-all-ann-decomposition } (\text{trail } S')) \rangle$
using $\text{bj bj-backjump dpll-bj-clauses inv}(1) \ \text{inv}(2)$
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*
 $\text{dest!}: \text{dpll-bj.intros cdcl}_{NOT}.\text{intros}$)

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

assumes $\langle \text{cdcl}_{NOT}\text{-merged-bj-learn}^{**} \ S \ T \rangle$ **and** $\text{inv}: \langle \text{inv } S \rangle$
 $\langle \text{all-decomposition-implies-m } (\text{clauses}_{NOT} \ S) \ (\text{get-all-ann-decomposition } (\text{trail } S)) \rangle$
shows
 $\langle \text{all-decomposition-implies-m } (\text{clauses}_{NOT} \ T) \ (\text{get-all-ann-decomposition } (\text{trail } T)) \rangle$
using *assms*
apply (*induction rule: rtranclp-induct*)
apply *simp*
using *cdcl_{NOT}-merged-bj-learn-all-decomposition-implies*
 $\text{rtranclp-cdcl}_{NOT}\text{-merged-bj-learn-is-rtranclp-cdcl}_{NOT}\text{-and-inv}$ **by** *blast*

lemma *full-cdcl_{NOT}-merged-bj-learn-final-state:*

fixes $A :: \langle 'v \text{ clause set} \rangle$ **and** $S \ T :: \langle 'st \rangle$
assumes
 $\text{full}: \langle \text{full cdcl}_{NOT}\text{-merged-bj-learn } S \ T \rangle$ **and**
 $\text{atms-S}: \langle \text{atms-of-mm } (\text{clauses}_{NOT} \ S) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $\text{atms-trail}: \langle \text{atm-of } ' \ \text{lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $n\text{-d}: \langle \text{no-dup } (\text{trail } S) \rangle$ **and**
 $\langle \text{finite } A \rangle$ **and**
 $\text{inv}: \langle \text{inv } S \rangle$ **and**
 $\text{decomp}: \langle \text{all-decomposition-implies-m } (\text{clauses}_{NOT} \ S) \ (\text{get-all-ann-decomposition } (\text{trail } S)) \rangle$
shows $\langle \text{unsatisfiable } (\text{set-mset } (\text{clauses}_{NOT} \ T)) \rangle$
 $\vee (\text{trail } T \models_{\text{asm}} \text{clauses}_{NOT} \ T \wedge \text{satisfiable } (\text{set-mset } (\text{clauses}_{NOT} \ T)))$

proof –

have $\text{st}: \langle \text{cdcl}_{NOT}\text{-merged-bj-learn}^{**} \ S \ T \rangle$ **and** $n\text{-s}: \langle \text{no-step cdcl}_{NOT}\text{-merged-bj-learn } T \rangle$
using *full unfolding full-def* **by** *blast+*
then have $\text{st}': \langle \text{cdcl}_{NOT}^{**} \ S \ T \rangle$
using $\text{inv rtranclp-cdcl}_{NOT}\text{-merged-bj-learn-is-rtranclp-cdcl}_{NOT}\text{-and-inv } n\text{-d}$ **by** *auto*
have $\langle \text{atms-of-mm } (\text{clauses}_{NOT} \ T) \subseteq \text{atms-of-ms } A \rangle$ **and** $\langle \text{atm-of } ' \ \text{lits-of-l } (\text{trail } T) \subseteq \text{atms-of-ms } A \rangle$
using *rtranclp-cdcl_{NOT}-merged-bj-learn-trail-clauses-bound[OF st inv atms-S atms-trail]* **by** *blast+*
moreover have $\langle \text{no-dup } (\text{trail } T) \rangle$
using *rtranclp-cdcl_{NOT}-merged-bj-learn-no-dup-inv inv n-d st* **by** *blast*
moreover have $\langle \text{inv } T \rangle$

```

    using rtrancpl-cdclNOT-merged-bj-learn-inv inv st by blast
  moreover have ⟨all-decomposition-implies-m (clausesNOT T) (get-all-ann-decomposition (trail T))⟩
    using rtrancpl-cdclNOT-merged-bj-learn-all-decomposition-implies inv st decomp n-d by blast
  ultimately show ?thesis
    using cdclNOT-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 cdclNOT-with-backtrack-and-restarts =
  conflict-driven-clause-learning-learning-before-backjump-only-distinct-learnt
  trail clausesNOT prepend-trail tl-trail add-clNOT remove-clNOT
  inv decide-conds backjump-conds propagate-conds learn-restrictions forget-restrictions
for
  trail :: ⟨'st ⇒ ('v, unit) ann-lits⟩ and
  clausesNOT :: ⟨'st ⇒ 'v clauses⟩ and
  prepend-trail :: ⟨('v, unit) ann-lit ⇒ 'st ⇒ 'st⟩ and
  tl-trail :: ⟨'st ⇒ 'st⟩ and
  add-clNOT :: ⟨'v clause ⇒ 'st ⇒ 'st⟩ and
  remove-clNOT :: ⟨'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: ⟨ $\bigwedge n. n \geq 1 \implies f\ n \geq 1$ ⟩ and
  inv-restart: ⟨ $\bigwedge S\ T. inv\ S \implies T \sim reduce\_trail\_to_{NOT} (\llbracket :: 'a\ list \rrbracket S \implies inv\ T)$ ⟩
begin

```

lemma bound-inv-inv:

```

assumes
  ⟨inv S⟩ and
  n-d: ⟨no-dup (trail S)⟩ and
  atms-clss-S-A: ⟨atms-of-mm (clausesNOT 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
  cdclNOT: ⟨cdclNOT S T⟩
shows
  ⟨atms-of-mm (clausesNOT T) ⊆ atms-of-ms A⟩ and
  ⟨atm-of ' lits-of-l (trail T) ⊆ atms-of-ms A⟩ and
  ⟨finite A⟩
proof –
  have ⟨cdclNOT S T⟩
    using ⟨inv S⟩ cdclNOT by linarith
  then have ⟨atms-of-mm (clausesNOT T) ⊆ atms-of-mm (clausesNOT S) ∪ atm-of ' lits-of-l (trail S)⟩
    using ⟨inv S⟩
  by (meson conflict-driven-clause-learning-ops.cdclNOT-atms-of-ms-clauses-decreasing
    conflict-driven-clause-learning-ops-axioms n-d)

```

```

then show  $\langle \text{atms-of-mm } (\text{clauses}_{NOT} T) \subseteq \text{atms-of-ms } A \rangle$ 
  using  $\text{atms-clss-}S\text{-}A$   $\text{atms-trail-}S\text{-}A$  by blast
next
  show  $\langle \text{atm-of } ' \text{ lits-of-l } (\text{trail } T) \subseteq \text{atms-of-ms } A \rangle$ 
    by  $(\text{meson } \langle \text{inv } S \rangle \text{atms-clss-}S\text{-}A \text{atms-trail-}S\text{-}A \text{cdcl}_{NOT} \text{cdcl}_{NOT}\text{-atms-in-trail-in-set } n\text{-}d)$ 
next
  show  $\langle \text{finite } A \rangle$ 
    using  $\langle \text{finite } A \rangle$  by simp
qed

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

lemma  $\text{cdcl}_{NOT}\text{-with-restart-}\mu_{CDCL}'\text{-le-}\mu_{CDCL}'\text{-bound}$ :
assumes
   $\text{cdcl}_{NOT}$ :  $\langle \text{cdcl}_{NOT}\text{-restart } (T, a) (V, b) \rangle$  and
   $\text{cdcl}_{NOT}\text{-inv}$ :
     $\langle \text{inv } T \rangle$ 
     $\langle \text{no-dup } (\text{trail } T) \rangle$  and
   $\text{bound-inv}$ :
     $\langle \text{atms-of-mm } (\text{clauses}_{NOT} T) \subseteq \text{atms-of-ms } A \rangle$ 
     $\langle \text{atm-of } ' \text{ lits-of-l } (\text{trail } T) \subseteq \text{atms-of-ms } A \rangle$ 
     $\langle \text{finite } A \rangle$ 
shows  $\langle \mu_{CDCL}' A V \leq \mu_{CDCL}'\text{-bound } A T \rangle$ 
using  $\text{cdcl}_{NOT}\text{-inv}$   $\text{bound-inv}$ 
proof (induction rule: cdclNOT-with-restart-induct[OF cdclNOT])
case  $(1 m S T n U)$  note  $U = \text{this}(3)$ 
show ?case
  apply  $(\text{rule } \text{rtranclp-cdcl}_{NOT}\text{-}\mu_{CDCL}'\text{-bound-reduce-trail-to}_{NOT}[\text{of } S T])$ 
    using  $\langle (\text{cdcl}_{NOT} \rightsquigarrow m) S T \rangle$  apply  $(\text{fastforce dest!: relpowp-imp-rtranclp})$ 
    using 1 by auto
next
case  $(2 S T n)$  note  $\text{full} = \text{this}(2)$ 
show ?case
  apply  $(\text{rule } \text{rtranclp-cdcl}_{NOT}\text{-}\mu_{CDCL}'\text{-bound})$ 
    using full 2 unfolding full1-def by force+
qed

lemma  $\text{cdcl}_{NOT}\text{-with-restart-}\mu_{CDCL}'\text{-bound-le-}\mu_{CDCL}'\text{-bound}$ :
assumes
   $\text{cdcl}_{NOT}$ :  $\langle \text{cdcl}_{NOT}\text{-restart } (T, a) (V, b) \rangle$  and

```

$cdcl_{NOT}\text{-inv}$:
 $\langle inv\ T \rangle$
 $\langle no\text{-}dup\ (trail\ T) \rangle$ **and**
 $bound\text{-}inv$:
 $\langle atms\text{-}of\text{-}mm\ (clauses_{NOT}\ T) \subseteq atms\text{-}of\text{-}ms\ A \rangle$
 $\langle atm\text{-}of\ 'lits\text{-}of\text{-}l\ (trail\ T) \subseteq atms\text{-}of\text{-}ms\ A \rangle$
 $\langle finite\ A \rangle$
shows $\langle \mu_{CDCL}'\text{-}bound\ A\ V \leq \mu_{CDCL}'\text{-}bound\ A\ T \rangle$
using $cdcl_{NOT}\text{-inv}\ bound\text{-}inv$
proof (*induction rule: $cdcl_{NOT}\text{-with-restart-induct}[OF\ cdcl_{NOT}]$*)
case $(1\ m\ S\ T\ n\ U)$ **note** $U = this(3)$
have $\langle \mu_{CDCL}'\text{-}bound\ A\ T \leq \mu_{CDCL}'\text{-}bound\ A\ S \rangle$
apply (*rule $rtrancp\text{-}\mu_{CDCL}'\text{-}bound\text{-}decreasing$*)
using $\langle (cdcl_{NOT} \rightsquigarrow m)\ S\ T \rangle$ **apply** (*fastforce dest: relpowp-imp-rtrancp*)
using 1 **by** *auto*
then show ?case **using** U **unfolding** $\mu_{CDCL}'\text{-}bound\text{-}def$ **by** *auto*
next
case $(2\ S\ T\ n)$ **note** $full = this(2)$
show ?case
apply (*rule $rtrancp\text{-}\mu_{CDCL}'\text{-}bound\text{-}decreasing$*)
using $full\ 2$ **unfolding** $full1\text{-}def$ **by** *force+*
qed

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

lemma $cdcl_{NOT}\text{-restart-all-decomposition-implies}$:
assumes $\langle cdcl_{NOT}\text{-restart}\ S\ T \rangle$ **and**
 $\langle inv\ (fst\ S) \rangle$ **and**
 $\langle no\text{-}dup\ (trail\ (fst\ S)) \rangle$
 $\langle all\text{-}decomposition\text{-}implies\text{-}m\ (clauses_{NOT}\ (fst\ S))\ (get\text{-}all\text{-}ann\text{-}decomposition\ (trail\ (fst\ S))) \rangle$
shows
 $\langle all\text{-}decomposition\text{-}implies\text{-}m\ (clauses_{NOT}\ (fst\ T))\ (get\text{-}all\text{-}ann\text{-}decomposition\ (trail\ (fst\ T))) \rangle$
using *assms* **apply** (*induction*)
using $rtrancp\text{-}cdcl_{NOT}\text{-all-decomposition-implies}$ **by** (*auto dest!: trancp-into-rtrancp*
 $simp: full1\text{-}def$)

lemma $rtrancp\text{-}cdcl_{NOT}\text{-restart-all-decomposition-implies}$:
assumes $\langle cdcl_{NOT}\text{-restart}^{**}\ S\ T \rangle$ **and**
 $inv: \langle inv\ (fst\ S) \rangle$ **and**
 $n\text{-}d: \langle no\text{-}dup\ (trail\ (fst\ S)) \rangle$ **and**
 $decomp:$
 $\langle all\text{-}decomposition\text{-}implies\text{-}m\ (clauses_{NOT}\ (fst\ S))\ (get\text{-}all\text{-}ann\text{-}decomposition\ (trail\ (fst\ S))) \rangle$
shows
 $\langle all\text{-}decomposition\text{-}implies\text{-}m\ (clauses_{NOT}\ (fst\ T))\ (get\text{-}all\text{-}ann\text{-}decomposition\ (trail\ (fst\ T))) \rangle$
using *assms*(1)

```

proof (induction rule: rtrancpl-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  $\langle \text{inv } (\text{fst } T) \rangle$ 
    using rtrancpl-cdclNOT-with-restart-cdclNOT-inv[OF st] inv n-d by blast
  moreover have  $\langle \text{no-dup } (\text{trail } (\text{fst } T)) \rangle$ 
    using rtrancpl-cdclNOT-with-restart-cdclNOT-inv[OF st] inv n-d by blast
  ultimately show ?case
    using cdclNOT-restart-all-decomposition-implies r IH n-d by fast
qed

lemma cdclNOT-restart-sat-ext-iff:
  assumes
    st:  $\langle \text{cdcl}_{\text{NOT}}\text{-restart } S \ T \rangle$  and
    n-d:  $\langle \text{no-dup } (\text{trail } (\text{fst } S)) \rangle$  and
    inv:  $\langle \text{inv } (\text{fst } S) \rangle$ 
  shows  $\langle I \models_{\text{sextm}} \text{clauses}_{\text{NOT}} (\text{fst } S) \longleftrightarrow I \models_{\text{sextm}} \text{clauses}_{\text{NOT}} (\text{fst } T) \rangle$ 
  using assms
proof (induction)
  case (restart-step m S T n U)
  then show ?case
    using rtrancpl-cdclNOT-bj-sat-ext-iff n-d by (fastforce dest!: relpowp-imp-rtrancpl)
next
  case restart-full
  then show ?case using rtrancpl-cdclNOT-bj-sat-ext-iff unfolding full1-def
    by (fastforce dest!: trancpl-into-rtrancpl)
qed

lemma rtrancpl-cdclNOT-restart-sat-ext-iff:
  fixes S T ::  $\langle 'st \times \text{nat} \rangle$ 
  assumes
    st:  $\langle \text{cdcl}_{\text{NOT}}\text{-restart}^{**} S \ T \rangle$  and
    n-d:  $\langle \text{no-dup } (\text{trail } (\text{fst } S)) \rangle$  and
    inv:  $\langle \text{inv } (\text{fst } S) \rangle$ 
  shows  $\langle I \models_{\text{sextm}} \text{clauses}_{\text{NOT}} (\text{fst } S) \longleftrightarrow I \models_{\text{sextm}} \text{clauses}_{\text{NOT}} (\text{fst } T) \rangle$ 
  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  $\langle \text{inv } (\text{fst } T) \rangle$ 
    using rtrancpl-cdclNOT-with-restart-cdclNOT-inv[OF st] inv n-d by blast+
  moreover have  $\langle \text{no-dup } (\text{trail } (\text{fst } T)) \rangle$ 
    using rtrancpl-cdclNOT-with-restart-cdclNOT-inv rtrancpl-cdclNOT-no-dup st inv n-d by blast
  ultimately show ?case
    using cdclNOT-restart-sat-ext-iff[OF r] IH by blast
qed

theorem full-cdclNOT-restart-backjump-final-state:
  fixes A ::  $\langle 'v \text{ clause set} \rangle$  and S T ::  $\langle 'st \rangle$ 
  assumes
    full:  $\langle \text{full } \text{cdcl}_{\text{NOT}}\text{-restart } (S, n) \ (T, m) \rangle$  and
    atms-S:  $\langle \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \subseteq \text{atms-of-ms } A \rangle$  and

```


atms-trail: $\langle \text{atm-of } \text{' lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A \rangle$ **and**
n-d: $\langle \text{no-dup } (\text{trail } S) \rangle$ **and**
fin-A[simp]: $\langle \text{finite } A \rangle$ **and**
inv: $\langle \text{inv } S \rangle$ **and**
decomp: $\langle \text{all-decomposition-implies-m } (\text{clauses}_{NOT} S) (\text{get-all-ann-decomposition } (\text{trail } S)) \rangle$
shows $\langle \text{unsatisfiable } (\text{set-mset } (\text{clauses}_{NOT} S)) \rangle$
 $\vee (\text{lits-of-l } (\text{trail } T) \models_{\text{sextm}} \text{clauses}_{NOT} S \wedge \text{satisfiable } (\text{set-mset } (\text{clauses}_{NOT} S)))$
proof —
have *st*: $\langle \text{cdcl}_{NOT}\text{-restart}^{**} (S, n) (T, m) \rangle$ **and**
n-s: $\langle \text{no-step cdcl}_{NOT}\text{-restart } (T, m) \rangle$
using *full unfolding full-def by fast+*
have *binv-T*: $\langle \text{atms-of-mm } (\text{clauses}_{NOT} T) \subseteq \text{atms-of-ms } A \rangle$
 $\langle \text{atm-of } \text{' lits-of-l } (\text{trail } T) \subseteq \text{atms-of-ms } A \rangle$
using *rtrancpl-cdcl_{NOT}-with-restart-bound-inv[OF st, of A] inv n-d atms-S atms-trail*
by *auto*
moreover have *inv-T*: $\langle \text{no-dup } (\text{trail } T) \rangle \langle \text{inv } T \rangle$
using *rtrancpl-cdcl_{NOT}-with-restart-cdcl_{NOT}-inv[OF st] inv n-d by auto*
moreover have $\langle \text{all-decomposition-implies-m } (\text{clauses}_{NOT} T) (\text{get-all-ann-decomposition } (\text{trail } T)) \rangle$
using *rtrancpl-cdcl_{NOT}-restart-all-decomposition-implies[OF st] inv n-d*
decomp by auto
ultimately have *T*: $\langle \text{unsatisfiable } (\text{set-mset } (\text{clauses}_{NOT} T)) \rangle$
 $\vee (\text{trail } T \models_{\text{asm}} \text{clauses}_{NOT} T \wedge \text{satisfiable } (\text{set-mset } (\text{clauses}_{NOT} T)))$
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*: $\langle \bigwedge I. I \models_{\text{sextm}} \text{clauses}_{NOT} S \longleftrightarrow I \models_{\text{sextm}} \text{clauses}_{NOT} T \rangle$
using *rtrancpl-cdcl_{NOT}-restart-sat-ext-iff[OF st] inv n-d atms-S*
atms-trail by auto
have *cons-T*: $\langle \text{consistent-interp } (\text{lits-of-l } (\text{trail } T)) \rangle$
using *inv-T(1) distinct-consistent-interp by blast*
consider
 $(\text{unsat}) \langle \text{unsatisfiable } (\text{set-mset } (\text{clauses}_{NOT} T)) \rangle$
 $| (\text{sat}) \langle \text{trail } T \models_{\text{asm}} \text{clauses}_{NOT} T \rangle$ **and** $\langle \text{satisfiable } (\text{set-mset } (\text{clauses}_{NOT} T)) \rangle$
using *T by blast*
then show *?thesis*
proof *cases*
case *unsat*
then have $\langle \text{unsatisfiable } (\text{set-mset } (\text{clauses}_{NOT} S)) \rangle$
using *eq-sat-S-T consistent-true-clss-ext-satisfiable true-clss-imp-true-clss-ext*
unfolding satisfiable-def by blast
then show *?thesis by fast*
next
case *sat*
then have $\langle \text{lits-of-l } (\text{trail } T) \models_{\text{sextm}} \text{clauses}_{NOT} S \rangle$
using *rtrancpl-cdcl_{NOT}-restart-sat-ext-iff[OF st] inv n-d atms-S*
atms-trail by (auto simp: true-clss-imp-true-clss-ext true-annots-true-clss)
moreover then have $\langle \text{satisfiable } (\text{set-mset } (\text{clauses}_{NOT} S)) \rangle$
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-cl_{NOT} remove-cl_{NOT}

```

    decide-conds propagate-conds forget-conds
    ⟨ΛC C' L' S T. distinct-mset C' ∧ L' ∉# C' ∧ backjump-l-cond C C' L' S T⟩ inv
for
  trail :: ⟨'st ⇒ ('v, unit) ann-lits⟩ and
  clausesNOT :: ⟨'st ⇒ 'v clauses⟩ and
  prepend-trail :: ⟨('v, unit) ann-lit ⇒ 'st ⇒ 'st⟩ and
  tl-trail :: ⟨'st ⇒ 'st⟩ and
  add-clNOT :: ⟨'v clause ⇒ 'st ⇒ 'st⟩ and
  remove-clNOT :: ⟨'v clause ⇒ 'st ⇒ 'st⟩ and
  decide-conds :: ⟨'st ⇒ 'st ⇒ bool⟩ and
  propagate-conds :: ⟨('v, unit) ann-lit ⇒ 'st ⇒ 'st ⇒ bool⟩ and
  inv :: ⟨'st ⇒ bool⟩ and
  forget-conds :: ⟨'v clause ⇒ 'st ⇒ bool⟩ and
  backjump-l-cond :: ⟨'v clause ⇒ 'v clause ⇒ 'v literal ⇒ 'st ⇒ 'st ⇒ bool⟩
+
fixes f :: ⟨nat ⇒ nat⟩
assumes
  unbounded: ⟨unbounded f⟩ and f-ge-1: ⟨Λn. n ≥ 1 ⇒ f n ≥ 1⟩ and
  inv-restart: ⟨ΛS T. inv S ⇒ T ∼ reduce-trail-toNOT [] S ⇒ inv T⟩
begin

definition not-simplified-cls :: ⟨'b clause multiset ⇒ 'b clauses⟩
where
  not-simplified-cls A ≡ {#C ∈# A. C ∉ simple-clss (atms-of-mm A)#}

lemma not-simplified-cls-tautology-distinct-mset:
  not-simplified-cls A = {#C ∈# A. tautology C ∨ ¬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 (clausesNOT S) ⊆ atms-of-ms A⟩ and
  ⟨x ∈# clausesNOT S⟩ and ⟨finite A⟩
  shows ⟨x ∈ simple-clss (atms-of-ms A) ∨ x ∈# not-simplified-cls (clausesNOT S)⟩
proof –
  consider
    (simpl) ⟨¬tautology x⟩ and ⟨distinct-mset x⟩
  | (n-simp) ⟨tautology x ∨ ¬distinct-mset x⟩
  by auto
  then show ?thesis
  proof cases
    case simpl
    then have ⟨x ∈ 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 ∈# not-simplified-cls (clausesNOT S)⟩
    using ⟨x ∈# clausesNOT S⟩ unfolding not-simplified-cls-tautology-distinct-mset by auto
    then show ?thesis by blast
  qed
qed

lemma cdclNOT-merged-bj-learn-clauses-bound:
  assumes

```

$\langle \text{cdcl}_{NOT}\text{-merged-bj-learn } S \ T \rangle$ **and**
 $\text{inv: } \langle \text{inv } S \rangle$ **and**
 $\text{atms-clss: } \langle \text{atms-of-mm } (\text{clauses}_{NOT} \ S) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $\text{atms-trail: } \langle \text{atm-of } '(\text{lits-of-l } (\text{trail } S)) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $\text{fin-}A[\text{simp}]: \langle \text{finite } A \rangle$
shows $\langle \text{set-mset } (\text{clauses}_{NOT} \ T) \subseteq \text{set-mset } (\text{not-simplified-cls } (\text{clauses}_{NOT} \ S))$
 $\cup \text{simple-clss } (\text{atms-of-ms } A) \rangle$
using $\text{assms}(1-4)$
proof ($\text{induction rule: cdcl}_{NOT}\text{-merged-bj-learn.induct}$)
case $\text{cdcl}_{NOT}\text{-merged-bj-learn-decide}_{NOT}$
then show $?case$ **using** dpll-bj-clauses **by** ($\text{force dest!: simple-clss-or-not-simplified-cls}$)
next
case $\text{cdcl}_{NOT}\text{-merged-bj-learn-propagate}_{NOT}$
then show $?case$ **using** dpll-bj-clauses **by** ($\text{force dest!: simple-clss-or-not-simplified-cls}$)
next
case $\text{cdcl}_{NOT}\text{-merged-bj-learn-forget}_{NOT}$
then show $?case$ **using** $\text{clauses-remove-cl}_{NOT}$ **unfolding** $\text{state-eq}_{NOT}\text{-def}$
by ($\text{force elim!: forget}_{NOT}E$ $\text{dest: simple-clss-or-not-simplified-cls}$)
next
case ($\text{cdcl}_{NOT}\text{-merged-bj-learn-backjump-l } T$) **note** $\text{bj} = \text{this}(1)$ **and** $\text{inv} = \text{this}(2)$ **and**
 $\text{atms-clss} = \text{this}(3)$ **and** $\text{atms-trail} = \text{this}(4)$

have $\text{st: } \langle \text{cdcl}_{NOT}\text{-merged-bj-learn}^{**} \ S \ T \rangle$
using $\text{bj inv cdcl}_{NOT}\text{-merged-bj-learn.simps}$ **by** blast+
have $\langle \text{atm-of } '(\text{lits-of-l } (\text{trail } T)) \subseteq \text{atms-of-ms } A \rangle$ **and** $\langle \text{atms-of-mm } (\text{clauses}_{NOT} \ T) \subseteq \text{atms-of-ms}$
 $A \rangle$
using $\text{rtranclp-cdcl}_{NOT}\text{-merged-bj-learn-trail-clauses-bound}[OF \ \text{st}]$ $\text{inv atms-trail atms-clss}$
by auto

obtain $F' \ K \ F \ L \ l \ C' \ C \ D$ **where**
 $\text{tr-}S: \langle \text{trail } S = F' @ \text{Decided } K \ \# \ F \rangle$ **and**
 $T: \langle T \sim \text{prepend-trail } (\text{Propagated } L \ l) \ (\text{reduce-trail-to}_{NOT} \ F \ (\text{add-cl}_{NOT} \ D \ S)) \rangle$ **and**
 $\langle C \in \# \ \text{clauses}_{NOT} \ S \rangle$ **and**
 $\langle \text{trail } S \models_{as} CNot \ C \rangle$ **and**
 $\text{undef: } \langle \text{undefined-lit } F \ L \rangle$ **and**
 $\langle \text{clauses}_{NOT} \ S \models_{pm} \text{add-mset } L \ C' \rangle$ **and**
 $\langle F \models_{as} CNot \ C' \rangle$ **and**
 $D: \langle D = \text{add-mset } L \ C' \rangle$ **and**
 $\text{dist: } \langle \text{distinct-mset } (\text{add-mset } L \ C') \rangle$ **and**
 $\text{tauto: } \langle \neg \text{tautology } (\text{add-mset } L \ C') \rangle$ **and**
 $\langle \text{backjump-l-cond } C \ C' \ L \ S \ T \rangle$
using $\langle \text{backjump-l } S \ T \rangle$ **apply** (elim backjump-lE) **by** auto

have $\langle \text{atms-of } C' \subseteq \text{atm-of } '(\text{lits-of-l } F) \rangle$
using $\langle F \models_{as} CNot \ C' \rangle$ **by** ($\text{simp add: atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set}$
 $\text{atms-of-def image-subset-iff in-CNot-implies-uminus}(2))$
then have $\langle \text{atms-of } (C' + \{\#L\}) \subseteq \text{atms-of-ms } A \rangle$
using T $\langle \text{atm-of } ' \text{lits-of-l } (\text{trail } T) \subseteq \text{atms-of-ms } A \rangle$ $\text{tr-}S$ undef **by** auto
then have $\langle \text{simple-clss } (\text{atms-of } (\text{add-mset } L \ C')) \subseteq \text{simple-clss } (\text{atms-of-ms } A) \rangle$
apply $-$ **by** ($\text{rule simple-clss-mono}$) (simp-all)
then have $\langle \text{add-mset } L \ C' \in \text{simple-clss } (\text{atms-of-ms } A) \rangle$
using $\text{distinct-mset-not-tautology-implies-in-simple-clss}[OF \ \text{dist tauto}]$
by auto
then show $?case$
using T $\text{inv atms-clss undef tr-}S \ D$ **by** ($\text{force dest!: simple-clss-or-not-simplified-cls}$)
qed

lemma *cdcl_{NOT}-merged-bj-learn-not-simplified-decreasing*:
assumes $\langle \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn } S \ T \rangle$
shows $\langle \text{not-simplified-cls } (\text{clauses}_{\text{NOT}} \ T) \subseteq \# \text{ not-simplified-cls } (\text{clauses}_{\text{NOT}} \ S) \rangle$
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 *rtrancpl-cdcl_{NOT}-merged-bj-learn-not-simplified-decreasing*:
assumes $\langle \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn}^{**} \ S \ T \rangle$
shows $\langle \text{not-simplified-cls } (\text{clauses}_{\text{NOT}} \ T) \subseteq \# \text{ not-simplified-cls } (\text{clauses}_{\text{NOT}} \ S) \rangle$
using *assms apply induction*
apply *simp*
by (*drule cdcl_{NOT}-merged-bj-learn-not-simplified-decreasing*) *auto*

lemma *rtrancpl-cdcl_{NOT}-merged-bj-learn-clauses-bound*:
assumes
 $\langle \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn}^{**} \ S \ T \rangle$ **and**
 $\langle \text{inv } S \rangle$ **and**
 $\langle \text{atms-of-mm } (\text{clauses}_{\text{NOT}} \ S) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $\langle \text{atm-of } (\text{lits-of-l } (\text{trail } S)) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $\text{finite[simp]: } \langle \text{finite } A \rangle$
shows $\langle \text{set-mset } (\text{clauses}_{\text{NOT}} \ T) \subseteq \text{set-mset } (\text{not-simplified-cls } (\text{clauses}_{\text{NOT}} \ S)) \cup \text{simple-clss } (\text{atms-of-ms } A) \rangle$
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** $st = \text{this}(1)$ **and** $\text{cdcl}_{\text{NOT}} = \text{this}(2)$ **and** $IH = \text{this}(3)[\text{OF } \text{this}(4-6)]$ **and**
 $\text{inv} = \text{this}(4)$ **and** $\text{atms-clss-}S = \text{this}(5)$ **and** $\text{atms-trail-}S = \text{this}(6)$
have st' : $\langle \text{cdcl}_{\text{NOT}}^{**} \ S \ T \rangle$
using *inv rtrancpl-cdcl_{NOT}-merged-bj-learn-is-rtrancpl-cdcl_{NOT}-and-inv st* **by** *blast*
have $\langle \text{inv } T \rangle$
using *inv rtrancpl-cdcl_{NOT}-merged-bj-learn-inv st* **by** *blast*
moreover
have $\langle \text{atms-of-mm } (\text{clauses}_{\text{NOT}} \ T) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $\langle \text{atm-of } (\text{lits-of-l } (\text{trail } T)) \subseteq \text{atms-of-ms } A \rangle$
using *rtrancpl-cdcl_{NOT}-merged-bj-learn-trail-clauses-bound[OF st] inv atms-clss-S*
 $\text{atms-trail-}S$ **by** *blast+*
ultimately have $\langle \text{set-mset } (\text{clauses}_{\text{NOT}} \ U) \subseteq \text{set-mset } (\text{not-simplified-cls } (\text{clauses}_{\text{NOT}} \ T)) \cup \text{simple-clss } (\text{atms-of-ms } A) \rangle$
using *cdcl_{NOT} finite cdcl_{NOT}-merged-bj-learn-clauses-bound*
by (*auto intro!: cdcl_{NOT}-merged-bj-learn-clauses-bound*)
moreover have $\langle \text{set-mset } (\text{not-simplified-cls } (\text{clauses}_{\text{NOT}} \ T)) \subseteq \text{set-mset } (\text{not-simplified-cls } (\text{clauses}_{\text{NOT}} \ S)) \rangle$
using *rtrancpl-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 $\mu_{\text{CDCL}}\text{'-bound}$ **where**
 $\mu_{\text{CDCL}}\text{'-bound } A \ T \equiv ((2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A))) * 2$
 $+ \text{card } (\text{set-mset } (\text{not-simplified-cls}(\text{clauses}_{\text{NOT}} \ T)))$
 $+ 3 \wedge \text{card } (\text{atms-of-ms } A)$

lemma *rtrancpl-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 *rtrancpl-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* ⟨*λS T. T ~ reduce-trail-to_{NOT} ([::'a list) S*

cdcl_{NOT}-merged-bj-learn f

⟨*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

⟨*λS. 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 *rtrancpl-cdcl_{NOT}-merged-bj-learn-clauses-bound-card* **apply** *blast*

apply (*drule rtrancpl-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**

$\langle \text{finite } A \rangle$
shows $\langle \mu_{CDCL}'\text{-merged } A \text{ (fst } V) \leq \mu_{CDCL}'\text{-bound } A \text{ (fst } T) \rangle$
using *assms*
proof *induction*
case $\langle \text{restart-full } S \text{ } T \text{ } n \rangle$
show $?_{\text{case}}$
unfolding *fst-conv*
apply $\langle \text{rule } r\text{trancpl-cdcl}_{NOT}\text{-merged-bj-learn-clauses-bound-card} \rangle$
using *restart-full* **unfolding** *full1-def* **by** $\langle \text{force } \text{dest!} : \text{trancpl-into-rtrancpl} \rangle +$
next
case $\langle \text{restart-step } m \text{ } S \text{ } T \text{ } n \text{ } U \rangle$ **note** $st = \text{this}(1)$ **and** $U = \text{this}(3)$ **and** $inv = \text{this}(4)$ **and**
 $n\text{-d} = \text{this}(5)$ **and** $\text{atms-clss} = \text{this}(6)$ **and** $\text{atms-trail} = \text{this}(7)$ **and** $\text{finite} = \text{this}(8)$
then have $st' : \langle \text{cdcl}_{NOT}\text{-merged-bj-learn}^{**} S \text{ } T \rangle$
by $\langle \text{blast } \text{dest} : \text{relpowp-imp-rtrancpl} \rangle$
then have $st'' : \langle \text{cdcl}_{NOT}^{**} S \text{ } T \rangle$
using $inv \text{ } n\text{-d}$ **apply** $-$ **by** $\langle \text{rule } r\text{trancpl-cdcl}_{NOT}\text{-merged-bj-learn-is-rtrancpl-cdcl}_{NOT} \rangle$ *auto*
have $\langle inv \text{ } T \rangle$
apply $\langle \text{rule } r\text{trancpl-cdcl}_{NOT}\text{-merged-bj-learn-inv} \rangle$
using $inv \text{ } st' \text{ } n\text{-d}$ **by** *auto*
then have $\langle inv \text{ } U \rangle$
using U **by** $\langle \text{auto } \text{simp} : \text{inv-restart} \rangle$
have $\langle \text{atms-of-mm } (\text{clauses}_{NOT} \text{ } T) \subseteq \text{atms-of-ms } A \rangle$
using $r\text{trancpl-cdcl}_{NOT}\text{-merged-bj-learn-trail-clauses-bound}[OF \text{ } st'] \text{ } inv \text{ } \text{atms-clss} \text{ } \text{atms-trail} \text{ } n\text{-d}$
by *simp*
then have $\langle \text{atms-of-mm } (\text{clauses}_{NOT} \text{ } U) \subseteq \text{atms-of-ms } A \rangle$
using U **by** *simp*
have $\langle \text{not-simplified-cls } (\text{clauses}_{NOT} \text{ } U) \subseteq \# \text{ not-simplified-cls } (\text{clauses}_{NOT} \text{ } T) \rangle$
using $\langle U \sim \text{reduce-trail-to}_{NOT} [] \text{ } T \rangle$ **by** *auto*
moreover have $\langle \text{not-simplified-cls } (\text{clauses}_{NOT} \text{ } T) \subseteq \# \text{ not-simplified-cls } (\text{clauses}_{NOT} \text{ } S) \rangle$
apply $\langle \text{rule } r\text{trancpl-cdcl}_{NOT}\text{-merged-bj-learn-not-simplified-decreasing} \rangle$
using $\langle (\text{cdcl}_{NOT}\text{-merged-bj-learn} \text{ } \widetilde{m}) S \text{ } T \rangle$ **by** $\langle \text{auto } \text{dest!} : \text{relpowp-imp-rtrancpl} \rangle$
ultimately have $U\text{-}S : \langle \text{not-simplified-cls } (\text{clauses}_{NOT} \text{ } U) \subseteq \# \text{ not-simplified-cls } (\text{clauses}_{NOT} \text{ } S) \rangle$
by *auto*

have $\langle (\text{set-mset } (\text{clauses}_{NOT} \text{ } U))$
 $\subseteq \text{set-mset } (\text{not-simplified-cls } (\text{clauses}_{NOT} \text{ } U)) \cup \text{simple-clss } (\text{atms-of-ms } A) \rangle$
apply $\langle \text{rule } r\text{trancpl-cdcl}_{NOT}\text{-merged-bj-learn-clauses-bound} \rangle$
apply *simp*
using $inv \text{ } U$ **apply** *simp*
using $\langle \text{atms-of-mm } (\text{clauses}_{NOT} \text{ } U) \subseteq \text{atms-of-ms } A \rangle$ **apply** *simp*
using U **apply** *simp*
using *finite* **apply** *simp*
done
then have $f1 : \langle \text{card } (\text{set-mset } (\text{clauses}_{NOT} \text{ } U)) \leq \text{card } (\text{set-mset } (\text{not-simplified-cls } (\text{clauses}_{NOT} \text{ } U))$
 $\cup \text{simple-clss } (\text{atms-of-ms } A)) \rangle$
by $\langle \text{simp add} : \text{simple-clss-finite card-mono local.finite} \rangle$

moreover have $\langle \text{set-mset } (\text{not-simplified-cls } (\text{clauses}_{NOT} \text{ } U)) \cup \text{simple-clss } (\text{atms-of-ms } A)$
 $\subseteq \text{set-mset } (\text{not-simplified-cls } (\text{clauses}_{NOT} \text{ } S)) \cup \text{simple-clss } (\text{atms-of-ms } A) \rangle$
using $U\text{-}S$ **by** *auto*
then have $f2 :$
 $\langle \text{card } (\text{set-mset } (\text{not-simplified-cls } (\text{clauses}_{NOT} \text{ } U)) \cup \text{simple-clss } (\text{atms-of-ms } A))$
 $\leq \text{card } (\text{set-mset } (\text{not-simplified-cls } (\text{clauses}_{NOT} \text{ } S)) \cup \text{simple-clss } (\text{atms-of-ms } A)) \rangle$
by $\langle \text{simp add} : \text{simple-clss-finite card-mono local.finite} \rangle$

moreover have $\langle \text{card } (\text{set-mset } (\text{not-simplified-cls } (\text{clauses}_{NOT} \text{ } S))$

$\cup \text{simple-clss} (\text{atms-of-ms } A)$
 $\leq \text{card} (\text{set-mset} (\text{not-simplified-cls} (\text{clauses}_{NOT} S))) + \text{card} (\text{simple-clss} (\text{atms-of-ms } A))$
using *card-Un-le* **by** *blast*
moreover have $\langle \text{card} (\text{simple-clss} (\text{atms-of-ms } A)) \leq 3 \wedge \text{card} (\text{atms-of-ms } A) \rangle$
using *atms-of-ms-finite simple-clss-card local.finite* **by** *blast*
ultimately have $\langle \text{card} (\text{set-mset} (\text{clauses}_{NOT} U))$
 $\leq \text{card} (\text{set-mset} (\text{not-simplified-cls} (\text{clauses}_{NOT} S))) + 3 \wedge \text{card} (\text{atms-of-ms } A) \rangle$
by *linarith*
then show ?case **unfolding** $\mu_{CDCL}'\text{-merged-def}$ **by** *auto*
qed

lemma *cdcl_{NOT}-restart- μ_{CDCL}' -bound-le- μ_{CDCL}' -bound:*

assumes
 $\langle \text{cdcl}_{NOT}\text{-restart } T \ V \rangle$ **and**
 $\langle \text{no-dup} (\text{trail} (\text{fst } T)) \rangle$ **and**
 $\langle \text{inv} (\text{fst } T) \rangle$ **and**
fin: $\langle \text{finite } A \rangle$
shows $\langle \mu_{CDCL}'\text{-bound } A (\text{fst } V) \leq \mu_{CDCL}'\text{-bound } A (\text{fst } T) \rangle$
using *assms(1-3)*
proof *induction*
case (*restart-full* $S \ T \ n$)
have $\langle \text{not-simplified-cls} (\text{clauses}_{NOT} T) \subseteq \# \text{not-simplified-cls} (\text{clauses}_{NOT} S) \rangle$
apply (*rule rtranclp-cdcl_{NOT}-merged-bj-learn-not-simplified-decreasing*)
using $\langle \text{full1 } \text{cdcl}_{NOT}\text{-merged-bj-learn } S \ T \rangle$ **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** $st = \text{this}(1)$ **and** $U = \text{this}(3)$ **and** $n\text{-d} = \text{this}(4)$ **and**
 $inv = \text{this}(5)$
then have $st': \langle \text{cdcl}_{NOT}\text{-merged-bj-learn}^{**} S \ T \rangle$
by (*blast dest: relpowp-imp-rtranclp*)
then have $st'': \langle \text{cdcl}_{NOT}^{**} S \ T \rangle$
using $inv \ n\text{-d}$ **apply** – **by** (*rule rtranclp-cdcl_{NOT}-merged-bj-learn-is-rtranclp-cdcl_{NOT}*) *auto*
have $\langle inv \ T \rangle$
apply (*rule rtranclp-cdcl_{NOT}-merged-bj-learn-inv*)
using $inv \ st' \ n\text{-d}$ **by** *auto*
then have $\langle inv \ U \rangle$
using U **by** (*auto simp: inv-restart*)
have $\langle \text{not-simplified-cls} (\text{clauses}_{NOT} U) \subseteq \# \text{not-simplified-cls} (\text{clauses}_{NOT} T) \rangle$
using $\langle U \sim \text{reduce-trail-to}_{NOT} [] \ T \rangle$ **by** *auto*
moreover have $\langle \text{not-simplified-cls} (\text{clauses}_{NOT} T) \subseteq \# \text{not-simplified-cls} (\text{clauses}_{NOT} S) \rangle$
apply (*rule rtranclp-cdcl_{NOT}-merged-bj-learn-not-simplified-decreasing*)
using $\langle (\text{cdcl}_{NOT}\text{-merged-bj-learn} \ \widetilde{\sim} \ m) \ S \ T \rangle$ **by** (*auto dest!: relpowp-imp-rtranclp*)
ultimately have $U\text{-S: } \langle \text{not-simplified-cls} (\text{clauses}_{NOT} U) \subseteq \# \text{not-simplified-cls} (\text{clauses}_{NOT} S) \rangle$
by *auto*
then show ?case **by** (*auto simp: card-mono set-mset-mono*)
qed

sublocale *cdcl_{NOT}-increasing-restarts - - - - f*

$\langle \lambda S \ T. T \sim \text{reduce-trail-to}_{NOT} ([]::'a \text{ list}) \ S \rangle$
 $\langle \lambda A \ S. \text{atms-of-mm} (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A$
 $\wedge \text{atm-of } ' \text{ lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A \wedge \text{finite } A \rangle$
 $\mu_{CDCL}'\text{-merged } \text{cdcl}_{NOT}\text{-merged-bj-learn}$
 $\langle \lambda S. inv \ S \wedge \text{no-dup} (\text{trail } S) \rangle$
 $\langle \lambda A \ T. ((2 + \text{card} (\text{atms-of-ms } A)) \wedge (1 + \text{card} (\text{atms-of-ms } A))) * 2$

```

+ card (set-mset (not-simplified-cls(cldclNOT T)))
+ 3 ^ card (atms-of-ms A)
apply unfold-locales
  using cdclNOT-restart-μCDCL'-merged-le-μCDCL'-bound apply force
  using cdclNOT-restart-μCDCL'-bound-le-μCDCL'-bound by fastforce

lemma true-clss-ext-decrease-right-insert:  $\langle I \models_{\text{sext}} \text{insert } C \text{ (set-mset } M) \rangle \implies I \models_{\text{sextm}} M \rangle$ 
by (metis Diff-insert-absorb insert-absorb true-clss-ext-decrease-right-remove-r)

lemma true-clss-ext-decrease-add-implied:
  assumes  $\langle M \models_{\text{pm}} C \rangle$ 
  shows  $\langle I \models_{\text{sext}} \text{insert } C \text{ (set-mset } M) \rangle \longleftrightarrow I \models_{\text{sextm}} M \rangle$ 
proof -
  { fix J
    assume
       $\langle I \models_{\text{sextm}} M \rangle$  and
       $\langle I \subseteq J \rangle$  and
      tot:  $\langle \text{total-over-m } J \text{ (set-mset } (\{\#C\} + M)) \rangle$  and
      cons:  $\langle \text{consistent-interp } J \rangle$ 
    then have  $\langle J \models_{\text{sm}} M \rangle$  unfolding true-clss-ext-def by auto

    moreover
      with  $\langle M \models_{\text{pm}} C \rangle$  have  $\langle J \models C \rangle$ 
      using tot cons unfolding true-clss-cls-def by auto
      ultimately have  $\langle J \models_{\text{sm}} \{\#C\} + M \rangle$  by auto
    }
  then have H:  $\langle I \models_{\text{sextm}} M \implies I \models_{\text{sext}} \text{insert } C \text{ (set-mset } M) \rangle$ 
  unfolding true-clss-ext-def by auto
  then show ?thesis
  by (auto simp: true-clss-ext-decrease-right-insert)
qed

lemma cdclNOT-merged-bj-learn-bj-sat-ext-iff:
  assumes  $\langle \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn } S \text{ } T \rangle$  and inv:  $\langle \text{inv } S \rangle$ 
  shows  $\langle I \models_{\text{sextm}} \text{clauses}_{\text{NOT}} S \rangle \longleftrightarrow I \models_{\text{sextm}} \text{clauses}_{\text{NOT}} T \rangle$ 
  using assms
proof (induction rule: cdclNOT-merged-bj-learn.induct)
  case (cdclNOT-merged-bj-learn-backjump-l T) note bj-l = this(1)
  obtain C' L D S' where
    learn:  $\langle \text{learn } S' \text{ } T \rangle$  and
    bj:  $\langle \text{backjump } S \text{ } S' \rangle$  and
    atms-C:  $\langle \text{atms-of } (\text{add-mset } L \text{ } C') \subseteq \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \cup \text{atm-of } ' (\text{lits-of-l } (\text{trail } S)) \rangle$ 
  and
    D:  $\langle D = \text{add-mset } L \text{ } C' \rangle$  and
    T:  $\langle T \sim \text{add-cl}_{\text{NOT}} D \text{ } S' \rangle$  and
    clss-D:  $\langle \text{clauses}_{\text{NOT}} S \models_{\text{pm}} D \rangle$ 
  using bj-l inv backjump-l-backjump-learn [of S] by blast
  have [simp]:  $\langle \text{clauses}_{\text{NOT}} S' = \text{clauses}_{\text{NOT}} S \rangle$ 
  using bj by (auto elim: backjumpE)
  have  $\langle (I \models_{\text{sextm}} \text{clauses}_{\text{NOT}} S) \longleftrightarrow (I \models_{\text{sextm}} \text{clauses}_{\text{NOT}} S') \rangle$ 
  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: cdclNOT-bj-sat-ext-iff
  dest!:: dpll-bj.intros cdclNOT.intros)

```


lemma *rtrancpl-cdcl_{NOT}-merged-bj-learn-bj-sat-ext-iff*:
assumes $\langle \text{cdcl}_{NOT}\text{-merged-bj-learn}^{**} S T \rangle$ **and** $\langle \text{inv } S \rangle$
shows $\langle I \models_{\text{sextm}} \text{clauses}_{NOT} S \longleftrightarrow I \models_{\text{sextm}} \text{clauses}_{NOT} T \rangle$
using *assms apply (induction rule: rtrancpl-induct)*
apply *simp*
using *cdcl_{NOT}-merged-bj-learn-bj-sat-ext-iff*
rtrancpl-cdcl_{NOT}-merged-bj-learn-is-rtrancpl-cdcl_{NOT}-and-inv **by** *blast*

lemma *cdcl_{NOT}-restart-eq-sat-iff*:
assumes
 $\langle \text{cdcl}_{NOT}\text{-restart } S T \rangle$ **and**
 $\text{inv: } \langle \text{inv } (\text{fst } S) \rangle$
shows $\langle I \models_{\text{sextm}} \text{clauses}_{NOT} (\text{fst } S) \longleftrightarrow I \models_{\text{sextm}} \text{clauses}_{NOT} (\text{fst } T) \rangle$
using *assms*

proof (*induction rule: cdcl_{NOT}-restart.induct*)
case (*restart-full* $S T n$)
then have $\langle \text{cdcl}_{NOT}\text{-merged-bj-learn}^{**} S T \rangle$
by (*simp add: trancpl-into-rtrancpl full1-def*)
then show *?case*
using *rtrancpl-cdcl_{NOT}-merged-bj-learn-bj-sat-ext-iff restart-full.prem*s **by** *auto*

next
case (*restart-step* $m S T n U$)
then have $\langle \text{cdcl}_{NOT}\text{-merged-bj-learn}^{**} S T \rangle$
by (*auto simp: trancpl-into-rtrancpl full1-def dest!: relpowp-imp-rtrancpl*)
then have $\langle I \models_{\text{sextm}} \text{clauses}_{NOT} S \longleftrightarrow I \models_{\text{sextm}} \text{clauses}_{NOT} T \rangle$
using *rtrancpl-cdcl_{NOT}-merged-bj-learn-bj-sat-ext-iff restart-step.prem*s **by** *auto*
moreover have $\langle I \models_{\text{sextm}} \text{clauses}_{NOT} T \longleftrightarrow I \models_{\text{sextm}} \text{clauses}_{NOT} U \rangle$
using *restart-step.hyps(3)* **by** *auto*
ultimately show *?case* **by** *auto*

qed

lemma *rtrancpl-cdcl_{NOT}-restart-eq-sat-iff*:
assumes
 $\langle \text{cdcl}_{NOT}\text{-restart}^{**} S T \rangle$ **and**
 $\text{inv: } \langle \text{inv } (\text{fst } S) \rangle$ **and** $n\text{-d: } \langle \text{no-dup}(\text{trail } (\text{fst } S)) \rangle$
shows $\langle I \models_{\text{sextm}} \text{clauses}_{NOT} (\text{fst } S) \longleftrightarrow I \models_{\text{sextm}} \text{clauses}_{NOT} (\text{fst } T) \rangle$
using *assms(1)*

proof (*induction rule: rtrancpl-induct*)
case *base*
then show *?case* **by** *simp*

next
case (*step* $T U$) **note** $st = \text{this}(1)$ **and** $cdcl = \text{this}(2)$ **and** $IH = \text{this}(3)$
have $\langle \text{inv } (\text{fst } T) \rangle$ **and** $\langle \text{no-dup } (\text{trail } (\text{fst } T)) \rangle$
using *rtrancpl-cdcl_{NOT}-with-restart-cdcl_{NOT}-inv* **using** st inv $n\text{-d}$ **by** *blast+*
then have $\langle I \models_{\text{sextm}} \text{clauses}_{NOT} (\text{fst } T) \longleftrightarrow I \models_{\text{sextm}} \text{clauses}_{NOT} (\text{fst } U) \rangle$
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
 $\langle \text{cdcl}_{NOT}\text{-restart } S T \rangle$ **and**
 $\text{inv: } \langle \text{inv } (\text{fst } S) \rangle$ **and** $n\text{-d: } \langle \text{no-dup}(\text{trail } (\text{fst } S)) \rangle$ **and**
 $\langle \text{all-decomposition-implies-m } (\text{clauses}_{NOT} (\text{fst } S)) \rangle$
 $\langle \text{get-all-ann-decomposition } (\text{trail } (\text{fst } S)) \rangle$
shows $\langle \text{all-decomposition-implies-m } (\text{clauses}_{NOT} (\text{fst } T)) \rangle$

```

    (get-all-ann-decomposition (trail (fst T)))
  using assms
proof induction
  case (restart-full S T n) note full = this(1) and inv = this(2) and n-d = this(3) and
    decomp = this(4)
  have st: ⟨cdclNOT-merged-bj-learn** S T⟩ and
    n-s: ⟨no-step cdclNOT-merged-bj-learn T⟩
    using full unfolding full1-def by (fast dest: tranclp-into-rtranclp)+
  have st': ⟨cdclNOT** S T⟩
    using inv rtranclp-cdclNOT-merged-bj-learn-is-rtranclp-cdclNOT-and-inv st n-d by auto
  have ⟨inv T⟩
    using rtranclp-cdclNOT-cdclNOT-inv[OF st] inv n-d by auto
  then show ?case
    using rtranclp-cdclNOT-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
    ⟨cdclNOT-restart** S T⟩ and
    inv: ⟨inv (fst S)⟩ and n-d: ⟨no-dup(trail (fst S))⟩ and
    decomp: ⟨all-decomposition-implies-m (clausesNOT (fst S))
      (get-all-ann-decomposition (trail (fst S)))⟩
  shows ⟨all-decomposition-implies-m (clausesNOT (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-cdclNOT-with-restart-cdclNOT-inv using st inv n-d by blast+
  then show ?case
    using cdclNOT-restart-all-decomposition-implies-m[OF cdcl] IH by auto
qed

```

lemma *full-cdcl_{NOT}-restart-normal-form:*

```

  assumes
    full: ⟨full cdclNOT-restart S T⟩ and
    inv: ⟨inv (fst S)⟩ and n-d: ⟨no-dup(trail (fst S))⟩ and
    decomp: ⟨all-decomposition-implies-m (clausesNOT (fst S))
      (get-all-ann-decomposition (trail (fst S)))⟩ and
    atms-cls: ⟨atms-of-mm (clausesNOT (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 (clausesNOT (fst S)))
    ∨ lits-of-l (trail (fst T)) ⊨ sextm clausesNOT (fst S) ∧
    satisfiable (set-mset (clausesNOT (fst S)))⟩
proof -
  have inv-T: ⟨inv (fst T)⟩ and n-d-T: ⟨no-dup (trail (fst T))⟩
    using rtranclp-cdclNOT-with-restart-cdclNOT-inv using full inv n-d unfolding full-def by blast+

```

moreover have
atms-clb-T: $\langle \text{atms-of-mm } (\text{clauses}_{NOT} \text{ (fst } T)) \subseteq \text{atms-of-ms } A \rangle$ **and**
atms-trail-T: $\langle \text{atm-of } \text{ ' lits-of-l } (\text{trail } (\text{fst } T)) \subseteq \text{atms-of-ms } A \rangle$
using *rtrancpl-cdcl_{NOT}-with-restart-bound-inv*[of *S T A*] *full atms-clb atms-trail fin inv n-d*
unfolding full-def by blast+
ultimately have $\langle \text{no-step cdcl}_{NOT}\text{-merged-bj-learn } (\text{fst } T) \rangle$
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 $\langle \text{all-decomposition-implies-m } (\text{clauses}_{NOT} \text{ (fst } T))$
 $(\text{get-all-ann-decomposition } (\text{trail } (\text{fst } T))) \rangle$
using *rtrancpl-cdcl_{NOT}-restart-all-decomposition-implies-m*[of *S T*] *inv n-d decomp*
full unfolding full-def by auto
ultimately have $\langle \text{unsatisfiable } (\text{set-mset } (\text{clauses}_{NOT} \text{ (fst } T)))$
 $\vee \text{trail } (\text{fst } T) \models_{asm} \text{clauses}_{NOT} \text{ (fst } T) \wedge \text{satisfiable } (\text{set-mset } (\text{clauses}_{NOT} \text{ (fst } T))) \rangle$
apply –
apply (*rule cdcl_{NOT}-merged-bj-learn-final-state*)
using atms-clb-T atms-trail-T fin n-d-T fin inv-T by blast+
then consider
 $(\text{unsat}) \langle \text{unsatisfiable } (\text{set-mset } (\text{clauses}_{NOT} \text{ (fst } T))) \rangle$
 $| (\text{sat}) \langle \text{trail } (\text{fst } T) \models_{asm} \text{clauses}_{NOT} \text{ (fst } T) \rangle$ **and** $\langle \text{satisfiable } (\text{set-mset } (\text{clauses}_{NOT} \text{ (fst } T))) \rangle$
by auto
then show $\langle \text{unsatisfiable } (\text{set-mset } (\text{clauses}_{NOT} \text{ (fst } S)))$
 $\vee \text{lits-of-l } (\text{trail } (\text{fst } T)) \models_{sextm} \text{clauses}_{NOT} \text{ (fst } S) \wedge$
 $\text{satisfiable } (\text{set-mset } (\text{clauses}_{NOT} \text{ (fst } S))) \rangle$
proof cases
case unsat
then have $\langle \text{unsatisfiable } (\text{set-mset } (\text{clauses}_{NOT} \text{ (fst } S))) \rangle$
unfolding satisfiable-def apply auto
using *rtrancpl-cdcl_{NOT}-restart-eq-sat-iff*[of *S T*] *full inv n-d*
consistent-true-clss-ext-satisfiable true-clss-imp-true-clb-ext
unfolding satisfiable-def full-def by blast
then show ?thesis by blast
next
case sat
then have $\langle \text{lits-of-l } (\text{trail } (\text{fst } T)) \models_{sextm} \text{clauses}_{NOT} \text{ (fst } T) \rangle$
using true-clss-imp-true-clb-ext by (auto simp: true-annots-true-clb)
then have $\langle \text{lits-of-l } (\text{trail } (\text{fst } T)) \models_{sextm} \text{clauses}_{NOT} \text{ (fst } S) \rangle$
using *rtrancpl-cdcl_{NOT}-restart-eq-sat-iff*[of *S T*] *full inv n-d* **unfolding full-def by blast**
moreover then have $\langle \text{satisfiable } (\text{set-mset } (\text{clauses}_{NOT} \text{ (fst } S))) \rangle$
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: $\langle \text{trail } S = [] \rangle \langle \text{clauses}_{NOT} S = N \rangle$ **and**

full: $\langle \text{full cdcl}_{NOT}\text{-restart } (S, 0) T \rangle$ **and**

inv: $\langle \text{inv } S \rangle$

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

$\vee \text{lits-of-l } (\text{trail } (\text{fst } T)) \models_{sextm} N \wedge \text{satisfiable } (\text{set-mset } N) \rangle$

using *full-cdcl_{NOT}-restart-normal-form*[of $\langle (S, 0) T \rangle$] *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 \equiv *map convert-ann-lit-from-NOT*

lemma *undefined-lit-convert-trail-from-NOT*[*simp*]:
undefined-lit (*convert-trail-from-NOT F*) *L* \longleftrightarrow *undefined-lit F L*
by (*induction F rule: ann-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* \equiv *convert-trail-from-W* (*fst S*)

lemma *undefined-lit-convert-trail-from-W*[*iff*]:
undefined-lit (*convert-trail-from-W M*) *L* \longleftrightarrow *undefined-lit M L*
by (*auto simp: defined-lit-map image-comp*)

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

sublocale *state*_W \subseteq *dpll-state-ops* **where**
trail = $\lambda S.$ *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 = $\lambda S.$ *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*
by *unfold-locales*

sublocale *state*_W \subseteq *dpll-state* **where**
trail = $\lambda S.$ *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 = $\lambda S.$ *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*
by *unfold-locales* (*auto simp: map-tl o-def*)

context *state*_W
begin
declare *state-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 = $\lambda S. \text{convert-trail-from-}W \text{ (trail } S) \text{ and}$
clauses_{NOT} = *clauses* **and**
prepend-trail = $\lambda L S. \text{cons-trail (convert-ann-lit-from-NOT } L) S \text{ and}$
tl-trail = $\lambda S. \text{tl-trail } S \text{ and}$
add-cls_{NOT} = $\lambda C S. \text{add-learned-cls } C S \text{ and}$
remove-cls_{NOT} = $\lambda C S. \text{remove-cls } C S \text{ and}$
decide-conds = $\lambda - -. \text{True and}$
propagate-conds = $\lambda - -. \text{True and}$
forget-conds = $\lambda - S. \text{conflicting } S = \text{None and}$
backjump-l-cond = $\lambda C C' L' S T. \text{backjump-l-cond } C C' L' S T$
 $\wedge \text{distinct-mset } C' \wedge L' \notin \# C' \wedge \neg \text{tautology (add-mset } L' C')$
by *unfold-locales*

sublocale *conflict-driven-clause-learning_W* \subseteq *cdcl_{NOT}-merge-bj-learn-proxy* **where**
trail = $\lambda S. \text{convert-trail-from-}W \text{ (trail } S) \text{ and}$
clauses_{NOT} = *clauses* **and**
prepend-trail = $\lambda L S. \text{cons-trail (convert-ann-lit-from-NOT } L) S \text{ and}$
tl-trail = $\lambda S. \text{tl-trail } S \text{ and}$
add-cls_{NOT} = $\lambda C S. \text{add-learned-cls } C S \text{ and}$
remove-cls_{NOT} = $\lambda C S. \text{remove-cls } C S \text{ and}$
decide-conds = $\lambda - -. \text{True and}$
propagate-conds = $\lambda - -. \text{True and}$
forget-conds = $\lambda - S. \text{conflicting } S = \text{None and}$
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 = $\lambda S. \text{convert-trail-from-}W \text{ (trail } S) \text{ and}$
clauses_{NOT} = *clauses* **and**
prepend-trail = $\lambda L S. \text{cons-trail (convert-ann-lit-from-NOT } L) S \text{ and}$
tl-trail = $\lambda S. \text{tl-trail } S \text{ and}$
add-cls_{NOT} = $\lambda C S. \text{add-learned-cls } C S \text{ and}$
remove-cls_{NOT} = $\lambda C S. \text{remove-cls } C S \text{ and}$
decide-conds = $\lambda - -. \text{True and}$
propagate-conds = $\lambda - -. \text{True and}$
forget-conds = $\lambda - S. \text{conflicting } S = \text{None and}$
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 $\langle F \models_{as} C \text{Not } C' \rangle \langle \text{undefined-lit } F L \rangle \text{Decided-Propagated-in-iff-in-lits-of-l}$
in-CNot-implies-uminus(2) **by** *fast*

then have *dist: distinct-mset ?C' L* $\notin \# C'$

by *simp-all*

have *no-dup F*

using $\langle \text{inv}_{NOT} S \rangle \langle \text{convert-trail-from-}W \text{ (trail } S) = F' @ \text{Decided } K \# F \rangle$

```

    unfolding invNOT-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  $\neg$  tautology C'
    using  $\langle F \models_{as} CNot\ C' \rangle$  consistent-CNot-not-tautology true-annots-true-cls by blast
  then have taut:  $\neg$  tautology (add-mset L ?C')
    using  $\langle F \models_{as} CNot\ C' \rangle$   $\langle undefined-lit\ F\ L \rangle$  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  $\langle inv_{NOT}\ S \rangle$  unfolding invNOT-def by (simp add: o-def)
  have f3: atm-of L  $\in$  atms-of-mm (clauses S)
     $\cup$  atm-of 'lits-of-l (convert-trail-from-W (trail S))
    using  $\langle convert-trail-from-W\ (trail\ S) = F' @ Decided\ K \# F \rangle$ 
       $\langle atm-of\ L \in atms-of-mm\ (clauses\ S) \cup atm-of\ 'lits-of-l\ (F' @ Decided\ K \# F) \rangle$  by auto
  have f4: clauses S  $\models_{pm}$  add-mset L ?C'
    by (metis 1(7) dist(2) remdups-mset-singleton-sum true-clss-cls-remdups-mset)
  have F  $\models_{as}$  CNot ?C'
    by (simp add:  $\langle F \models_{as} CNot\ C' \rangle$ )
  have Ex (backjump-l S)
    apply standard
    apply (rule backjump-l.intros[of - - - - L add-mset L ?C' - ?C'])
    using f4 f3 f2  $\langle \neg$  tautology (add-mset L ?C')
      1 taut dist  $\langle F \models_{as} CNot\ (remdups-mset\ C') \rangle$ 
      state-eqNOT-ref unfolding backjump-l-cond-def set-mset-remdups-mset by blast+
  then show ?case
    by blast
next
case (3 L S)
then show  $\exists T. decide_{NOT}\ S\ T \vee propagate_{NOT}\ S\ T \vee backjump-l\ S\ T$ 
  using decideNOT.intros[of S L] by auto
qed

```

context *conflict-driven-clause-learning_W*
begin

Notations are lost while proving locale inclusion:

notation *state-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*:
 $trail\ S = trail\ T \implies trail\ (reduce-trail-to_{NOT}\ F\ S) = trail\ (reduce-trail-to_{NOT}\ F\ T)$
proof (*induction* F S *arbitrary*: T *rule*: reduce-trail-to_{NOT}.induct)
case (1 F S T) **note** IH = *this*(1) **and** tr = *this*(2)
then have $\square = convert-trail-from-W\ (trail\ S)$
 $\vee length\ F = length\ (convert-trail-from-W\ (trail\ S))$
 $\vee trail\ (reduce-trail-to_{NOT}\ F\ (tl-trail\ S)) = trail\ (reduce-trail-to_{NOT}\ F\ (tl-trail\ T))$
using IH **by** (metis (no-types) trail-tl-trail)
then show $trail\ (reduce-trail-to_{NOT}\ F\ S) = trail\ (reduce-trail-to_{NOT}\ F\ T)$
using tr **by** (metis (no-types) reduce-trail-to_{NOT}.elims)
qed

lemma *trail-reduce-trail-to_{NOT}-add-learned-cls*:

no-dup (*trail S*) \implies
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
 $\exists M. \text{trail } S = M @ \text{trail } T \wedge (\forall m \in \text{set } M. \neg \text{is-decided } m)$
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: rtrancpl-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  $S\ T$ 
   $\vee (no\text{-}step\ cdcl_W\text{-}merge\ T \wedge conflicting\ T \neq None)$ 
using  $cdcl_W$ -restart  $inv$ 
proof induction
  case ( $fw$ -propagate  $S\ T$ ) note  $propa = this(1)$ 
  then obtain  $M\ N\ U\ L\ C$  where
     $H$ :  $state\text{-}butlast\ S = (M, N, U, None)$  and
     $CL$ :  $C + \{\#L\#\} \in \# \text{ clauses } S$  and
     $M\text{-}C$ :  $M \models_{as} CNot\ C$  and
     $undef$ :  $undefined\text{-}lit\ (trail\ S)\ L$  and
     $T$ :  $state\text{-}butlast\ T = (Propagated\ L\ (C + \{\#L\#\}) \# M, N, U, None)$ 
    by ( $auto\ elim$ :  $propagate\text{-}high\text{-}levelE$ )
  have  $propagate_{NOT}\ S\ T$ 
    using  $H\ CL\ T\ undef\ M\text{-}C$  by ( $auto\ simp$ :  $state\text{-}eq_{NOT}\text{-}def\ clauses\text{-}def\ simp\ del$ :  $state\text{-}simp$ )
  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
     $undef\text{-}L$ :  $undefined\text{-}lit\ (trail\ S)\ L$  and
     $atm\text{-}L$ :  $atm\text{-}of\ L \in atm\text{-}of\text{-}mm\ (init\text{-}clss\ S)$  and
     $T$ :  $T \sim cons\text{-}trail\ (Decided\ L)\ S$ 
    by ( $auto\ elim$ :  $decideE$ )
  have  $decide_{NOT}\ S\ T$ 
    apply ( $rule\ decide_{NOT}.decide_{NOT}$ )
    using  $undef\text{-}L$  apply ( $simp$ ; fail)
    using  $atm\text{-}L\ inv$  apply ( $auto\ simp$ :  $cdcl_W$ -all-struct-inv-def no-strange-atm-def clauses-def; fail)
    using  $T\ undef\text{-}L$  unfolding  $state\text{-}eq_{NOT}\text{-}def$  by ( $auto\ simp$ :  $clauses\text{-}def$ )
  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
     $C\text{-}le$ :  $C \in \# \text{ learned-clss } S$  and
     $\neg(trail\ S) \models_{asm} \text{ clauses } S$  and
     $C \notin set\ (get\text{-}all\text{-}mark\text{-}of\text{-}propagated\ (trail\ S))$  and
     $C\text{-}init$ :  $C \notin \# \text{ init-clss } S$  and
     $T$ :  $T \sim remove\text{-}cls\ C\ S$  and
     $S\text{-}C$ :  $(removeAll\text{-}mset\ C\ (\text{clauses } S) \models_{pm} C)$ 
    by ( $auto\ elim$ :  $forgetE$ )
  have  $forget_{NOT}\ S\ T$ 
    apply ( $rule\ forget_{NOT}.forget_{NOT}$ )
    using  $S\text{-}C$  apply blast
    using  $S$  apply  $simp$ 
    using  $C\text{-}init\ C\text{-}le$  apply ( $simp\ add$ :  $clauses\text{-}def$ )
    using  $T\ C\text{-}le\ C\text{-}init$  by ( $auto\ simp$ :  $Un\text{-}Diff\ state\text{-}eq_{NOT}\text{-}def\ clauses\text{-}def\ ac\text{-}sims$ )
  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
     $confl\text{-}T$ :  $conflicting\ T = Some\ CT$  and
     $CT$ :  $CT = C_S$  and
     $C_S$ :  $C_S \in \# \text{ clauses } S$  and
     $tr\text{-}S\text{-}C_S$ :  $trail\ S \models_{as} CNot\ C_S$ 
    using  $confl$  by ( $elim\ conflictE$ )  $auto$ 
  have  $inv\text{-}T$ :  $cdcl_W$ -all-struct-inv  $T$ 

```

```

    using cdclW-restart.simps cdclW-all-struct-inv-inv confl inv by blast
then have cdclW-M-level-inv T
  unfolding cdclW-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 rtrancpl-cdclW-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: rtrancpl-induct)
    (auto simp: skip-or-resolve.simps elim!: rulesE)
  moreover then have no-step cdclW-merge U
    by (auto simp: cdclW-merge.simps elim: rulesE)
  ultimately show ?thesis by blast
next
  case bt note s-or-r = this(1) and bt = this(2)
  have cdclW-restart** T T'
    using s-or-r mono-rtrancpl[of skip-or-resolve cdclW-restart]
    rtrancpl-skip-or-resolve-rtrancpl-cdclW-restart
    by blast
  then have cdclW-M-level-inv T'
    using rtrancpl-cdclW-restart-consistent-inv ⟨cdclW-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 cdclW-restart** T T'
    using rtrancpl-skip-or-resolve-rtrancpl-cdclW-restart s-or-r by blast
  have inv-T': cdclW-all-struct-inv T'
    using ⟨cdclW-restart** T T'⟩ inv-T rtrancpl-cdclW-all-struct-inv-inv by blast

```

```

have inv-U: cdclW-all-struct-inv U
  using cdclW-merge-restart-cdclW-restart confl fw-r-conflict inv local.bj
  rtrancp-cdclW-all-struct-inv-inv by blast

have [simp]: init-clss S = init-clss T'
  using ⟨cdclW-restart** T T'⟩ cdclW-restart-init-clss confl cdclW-all-struct-inv-def conflict
  inv by (metis rtrancp-cdclW-restart-init-clss)
then have atm-L: atm-of L ∈ atms-of-mm (clauses S)
  using inv-T' confl-T' unfolding cdclW-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 cdclW-all-struct-inv-def cdclW-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 cdclW-all-struct-inv-def cdclW-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-toNOT M1 S) = M1
  using M1-M2 confl by (subst reduce-trail-toNOT-reduce-trail-convert)
  (auto simp: comp-def elim: rulesE)
have every-mark-is-a-conflict U
  using inv-U unfolding cdclW-all-struct-inv-def cdclW-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 cdclW-all-struct-inv-def cdclW-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 cdclW-all-struct-inv-def
  cdclW-M-level-inv-def apply (auto simp: state-eqNOT-def
    trail-reduce-trail-toNOT-add-learned-cls; fail)[]
  using CS apply (auto; fail)[]
  using tr-S-CS apply (simp; fail)

  using undef-L apply (auto; fail)[]
  using atm-L apply (simp add: trail-reduce-trail-toNOT-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 \equiv *restart-ops.cdcl_{NOT}-raw-restart cdcl_{NOT} restart*

lemma *cdcl_W-merge-restart-is-cdcl_{NOT}-merged-bj-learn-restart-no-step*:
assumes
inv: cdcl_W-all-struct-inv S and
cdcl_W-restart:cdcl_W-merge-restart S T
shows *cdcl_{NOT}-restart** S T \vee (no-step cdcl_W-merge T \wedge conflicting T \neq None)*
proof –
consider
(fw) cdcl_W-merge S T |
(fw-r) restart S T
using *cdcl_W-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 \vee (no-step cdcl_W-merge T \wedge conflicting T \neq None)*
using *inv cdcl_W-merge-is-cdcl_{NOT}-merged-bj-learn* **by** *blast*
have *invS: inv_{NOT} S*
using *inv unfolding cdcl_W-all-struct-inv-def cdcl_W-M-level-inv-def* **by** *auto*
have *ff2: cdcl_{NOT}⁺⁺ S T \longrightarrow cdcl_{NOT}^{**} S T*
by (*meson tranclp-into-rtranclp*)
have *ff3: no-dup (convert-trail-from-W (trail S))*
using *invS* **by** (*simp add: comp-def*)
have *cdcl_{NOT} \leq cdcl_{NOT}-restart*
by (*auto simp: restart-ops.cdcl_{NOT}-raw-restart.simps*)
then show *?thesis*
using *ff3 ff2 IH cdcl_{NOT}-merged-bj-learn-is-tranclp-cdcl_{NOT}*
rtranclp-mono[of cdcl_{NOT} cdcl_{NOT}-restart] invS predicate2D **by** *blast*
next
case *fw-r*
then show *?thesis* **by** (*blast intro: restart-ops.cdcl_{NOT}-raw-restart.intros*)
qed
qed

abbreviation $\mu_{FW} :: 'st \Rightarrow nat$ **where**
 $\mu_{FW} S \equiv$ (*if no-step cdcl_W-merge S then 0 else 1 + μ_{CDCL} '-merged (set-mset (init-cls 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 $\mu_{FW} T < \mu_{FW} S$
proof –
let *?A = init-cls S*

```

have atm-clauses:  $\text{atms-of-mm } (\text{clauses } S) \subseteq \text{atms-of-mm } ?A$ 
  using inv unfolding cdclW-all-struct-inv-def no-strange-atm-def clauses-def by auto
have atm-trail:  $\text{atm-of } \text{‘lits-of-l } (\text{trail } S) \subseteq \text{atms-of-mm } ?A$ 
  using inv unfolding cdclW-all-struct-inv-def no-strange-atm-def clauses-def by auto
have n-d: no-dup (trail S)
  using inv unfolding cdclW-all-struct-inv-def by (auto simp: cdclW-M-level-inv-decomp)
have [simp]:  $\neg \text{no-step cdcl}_W\text{-merge } S$ 
  using fw by auto
have [simp]:  $\text{init-clss } S = \text{init-clss } T$ 
  using cdclW-merge-restart-cdclW-restart[of S T] inv rtrancpl-cdclW-restart-init-clss
  unfolding cdclW-all-struct-inv-def
  by (meson cdclW-merge.simps cdclW-merge-restart.simps cdclW-rf.simps fw)
consider
  (merged) cdclNOT-merged-bj-learn S T |
  (n-s) no-step cdclW-merge T
  using cdclW-merge-is-cdclNOT-merged-bj-learn inv fw by blast
then show ?thesis
proof cases
case merged
  then show ?thesis
    using cdclNOT-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-cdclW-merge: wf  $\{(T, S). \text{cdcl}_W\text{-all-struct-inv } S \wedge \text{cdcl}_W\text{-merge } S T\}$ 
  apply (rule wfP-if-measure[of - -  $\mu_{FW}$ ])
  using cdclW-merge- $\mu_{FW}$ -decreasing by blast

```

```

lemma trancpl-cdclW-merge-cdclW-merge-trancpl:
 $\{(T, S). \text{cdcl}_W\text{-all-struct-inv } S \wedge \text{cdcl}_W\text{-merge}^{++} S T\}$ 
 $\subseteq \{(T, S). \text{cdcl}_W\text{-all-struct-inv } S \wedge \text{cdcl}_W\text{-merge } S T\}^+$ 
proof -
have  $(T, S) \in \{(T, S). \text{cdcl}_W\text{-all-struct-inv } S \wedge \text{cdcl}_W\text{-merge } S T\}^+$ 
  if inv:  $\text{cdcl}_W\text{-all-struct-inv } S$  and  $\text{cdcl}_W\text{-merge}^{++} S T$ 
  for S T :: 'st
  using that(2)
  proof (induction rule: trancpl-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  $\text{cdcl}_W\text{-all-struct-inv } T$ 
    using st by (meson inv rtrancpl-cdclW-all-struct-inv-inv
      rtrancpl-cdclW-merge-rtrancpl-cdclW-restart trancpl-into-rtrancpl)
  then have  $(U, T) \in \{(T, S). \text{cdcl}_W\text{-all-struct-inv } S \wedge \text{cdcl}_W\text{-merge } S T\}^+$ 
    using s by auto
  then show ?case using IH by auto
  qed
then show ?thesis by auto
qed

```

```

lemma wf-trancpl-cdclW-merge: wf  $\{(T, S). \text{cdcl}_W\text{-all-struct-inv } S \wedge \text{cdcl}_W\text{-merge}^{++} S T\}$ 

```

```

apply (rule wf-subset)
apply (rule wf-trancl)
using wf-cdclW-merge apply simp
using tranclp-cdclW-merge-cdclW-merge-trancl by simp

lemma wf-cdclW-bj-all-struct: wf {(T, S). cdclW-all-struct-inv S ∧ cdclW-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 cdclW-bj-measure by (simp add: cdclW-all-struct-inv-def)

lemma cdclW-conflicting-true-cdclW-merge-restart:
  assumes cdclW S V and confl: conflicting S = None
  shows (cdclW-merge S V ∧ conflicting V = None) ∨ (conflicting V ≠ None ∧ conflict S V)
  using assms
proof (induction rule: cdclW.induct)
  case W-propagate
  then show ?case by (auto intro: cdclW-merge.intros elim: rulesE)
next
  case (W-conflict S')
  then show ?case by (auto intro: cdclW-merge.intros elim: rulesE)
next
  case W-other
  then show ?case
  proof cases
    case decide
    then show ?thesis
    by (auto intro: cdclW-merge.intros elim: rulesE)
  next
    case bj
    then show ?thesis
    using confl by (auto simp: cdclW-bj.simps elim: rulesE)
  qed
qed

lemma trancl-cdclW-conflicting-true-cdclW-merge-restart:
  assumes cdclW++ S V and inv: cdclW-M-level-inv S and conflicting S = None
  shows (cdclW-merge++ S V ∧ conflicting V = None)
    ∨ (∃ T U. cdclW-merge** S T ∧ conflicting V ≠ None ∧ conflict T U ∧ cdclW-bj** U V)
  using assms
proof induction
  case base
  then show ?case using cdclW-conflicting-true-cdclW-merge-restart by blast
next
  case (step U V) note st = this(1) and cdclW = this(2) and IH = this(3)[OF this(4-)] and
    confl[simp] = this(5) and inv = this(4)
  from cdclW
  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 cdclW-merge.fw-propagate[of U V] by auto
  next
    case W-conflict
    moreover have confl-U: conflicting U = None and confl-V: conflicting V ≠ None
    using W-conflict by (auto elim!: conflictE)

```

```

moreover have cdclW-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 cdclW-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: cdclW-bj.simps)
then show ?thesis
proof cases
  case s-or-r
    have f1: cdclW-bj++ U V
      by (simp add: local.bj tranclp.r-into-trancl)
    obtain T T' :: 'st where
      f2: cdclW-merge++ S U
        ∨ cdclW-merge** S T ∧ conflicting U ≠ None
        ∧ conflict T T' ∧ cdclW-bj** T' U
      using IH confl by (meson bj rtranclp.intros(1)
        rtranclp-cdclW-merge-restart-no-step-cdclW-bj
        rtranclp-cdclW-merge-tranclp-cdclW-merge-restart)
    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
  cdclW-merge** S T and
  conflicting U ≠ None and
  conflict T T' and
  cdclW-bj** T' U
  using IH confl by (meson bj rtranclp.intros(1)
    rtranclp-cdclW-merge-restart-no-step-cdclW-bj
    rtranclp-cdclW-merge-tranclp-cdclW-merge-restart)
have invU: cdclW-M-level-inv U
  using inv rtranclp-cdclW-restart-consistent-inv step.hyps(1)
  by (meson (cdclW-bj** T' U) (cdclW-merge** S T) (conflict T T')
    cdclW-restart-consistent-inv conflict rtranclp-cdclW-bj-rtranclp-cdclW-restart
    rtranclp-cdclW-merge-rtranclp-cdclW-restart)
then have conflicting V = None
  using (backtrack U V) inv by (auto elim: backtrackE
    simp: cdclW-M-level-inv-decomp)
have full cdclW-bj T' V
  apply (rule rtranclp-fullI[of cdclW-bj T' U V])
  using (cdclW-bj** T' U) apply fast
  using (backtrack U V) backtrack-is-full1-cdclW-bj invU unfolding full1-def full-def
  by blast

```

```

    then show ?thesis
    using cdclW-merge.fw-conflict[of T T' V] ⟨conflict T T'⟩
    ⟨cdclW-merge** S T⟩ ⟨conflicting V = None⟩ by auto
  qed
qed
qed
qed

lemma wf-cdclW: wf {(T, S). cdclW-all-struct-inv S ∧ cdclW S T}
  unfolding wf-iff-no-infinite-down-chain
proof clarify
  fix f :: nat ⇒ 'st
  assume ∀ i. (f (Suc i), f i) ∈ {(T, S). cdclW-all-struct-inv S ∧ cdclW S T}
  then have f: ∧ i. (f (Suc i), f i) ∈ {(T, S). cdclW-all-struct-inv S ∧ cdclW S T}
    by blast
  {
    fix f :: nat ⇒ 'st
    assume
      f: (f (Suc i), f i) ∈ {(T, S). cdclW-all-struct-inv S ∧ cdclW S T} and
      confl: conflicting (f i) ≠ None for i
    have (f (Suc i), f i) ∈ {(T, S). cdclW-all-struct-inv S ∧ cdclW-bj S T} for i
      using f[of i] confl[of i] by (auto simp: cdclW.simps cdclW-o.simps cdclW-rf.simps
        elim!: rulesE)
    then have False
      using wf-cdclW-bj-all-struct unfolding wf-iff-no-infinite-down-chain by blast
  } note no-infinite-conflict = this

have st: cdclW++ (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 ⇒ cdclW++ (f i) (f j) for i j :: nat
  using st[of i j - i - 1] by auto

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

define i0 where i0: i0 = Max {i0. ∀ i < i0. conflicting (f i) ≠ None}
have finite {i0. ∀ i < i0. conflicting (f i) ≠ None}
proof -
  have {i0. ∀ i < i0. conflicting (f i) ≠ None} ⊆ {0..ib}
    using ib 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 {i0. ∀ i < i0. conflicting (f i) ≠ None} ≠ {}
  by auto
ultimately have i0 ∈ {i0. ∀ i < i0. conflicting (f i) ≠ None}
  using Max-in[of {i0. ∀ i < i0. conflicting (f i) ≠ None}] unfolding i0 by fast
then have confl-i0: conflicting (f i0) = None
proof -

```



```

have f1:  $\forall n < i_0. \text{conflicting } (f\ n) \neq \text{None}$ 
  using  $\langle i_0 \in \{i_0. \forall i < i_0. \text{conflicting } (f\ i) \neq \text{None}\} \rangle$  by blast
have Suc i0  $\notin \{n. \forall na < n. \text{conflicting } (f\ na) \neq \text{None}\}$ 
  by (metis (lifting) Max-ge  $\langle \text{finite } \{i_0. \forall i < i_0. \text{conflicting } (f\ i) \neq \text{None}\} \rangle$ 
    i0 lessI not-le)
then have  $\exists n < \text{Suc } i_0. \text{conflicting } (f\ n) = \text{None}$ 
  by fastforce
then show  $\text{conflicting } (f\ i_0) = \text{None}$ 
  using f1 by (metis le-less less-Suc-eq-le)
qed
have  $\forall i < i_0. \text{conflicting } (f\ i) \neq \text{None}$ 
  using  $\langle i_0 \in \{i_0. \forall i < i_0. \text{conflicting } (f\ i) \neq \text{None}\} \rangle$  by blast

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

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

  then have (?f (Suc j), ?f j)  $\in \{(T, S). \text{cdcl}_W\text{-all-struct-inv } S \wedge \text{cdcl}_W\text{-bj } S\ T\}$ 
    for j :: nat
    using f[of i+j+1] by auto
  then show False
    using wf-cdclW-bj-all-struct unfolding wf-iff-no-infinite-down-chain by fast
qed

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

define g where g:  $g = \text{rec-nat } i_0\ (\lambda\ i. \text{LEAST } j. j > i \wedge \text{conflicting } (f\ j) = \text{None})$ 
have g-0:  $g\ 0 = i_0$ 
  unfolding g by auto
have g-Suc:  $g\ (\text{Suc } i) = (\text{LEAST } j. j > g\ i \wedge \text{conflicting } (f\ j) = \text{None})$  for i
  unfolding g by auto
have g-prop:  $g\ (\text{Suc } i) > g\ i \wedge \text{conflicting } (f\ (g\ (\text{Suc } i))) = \text{None}$  for i

```

```

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

lemma wf-cdclW-stgy:
   $\langle \text{wf } \{(T, S). \text{cdcl}_W\text{-all-struct-inv } S \wedge \text{cdcl}_W\text{-stgy } S T\} \rangle$ 
  by (rule wf-subset[OF wf-cdclW]) (auto dest:  $\text{cdcl}_W\text{-stgy-cdcl}_W$ )

end

```

2.3.5 Inclusion of Weidendenbch's CDCL with Strategy

```

context conflict-driven-clause-learningW
begin
abbreviation propagate-conds where

```

propagate-conds $\equiv \lambda -. \text{propagate}$

abbreviation (*input*) *decide-conds* **where**

decide-conds $S\ T \equiv \text{decide } S\ T \wedge \text{no-step conflict } S \wedge \text{no-step propagate } S$

abbreviation *backjump-l-conds-stgy* $:: 'v \text{ clause} \Rightarrow 'v \text{ clause} \Rightarrow 'v \text{ literal} \Rightarrow 'st \Rightarrow 'st \Rightarrow \text{bool}$ **where**

backjump-l-conds-stgy $C\ C'\ L\ S\ V \equiv$

$(\exists T\ U. \text{conflict } S\ T \wedge \text{full skip-or-resolve } T\ U \wedge \text{conflicting } T = \text{Some } C \wedge$
 $\text{mark-of } (\text{hd-trail } V) = \text{add-mset } L\ C' \wedge \text{backtrack } U\ V)$

abbreviation *inv_{NOT}-stgy* **where**

inv_{NOT}-stgy $S \equiv \text{conflicting } S = \text{None} \wedge \text{cdcl}_W\text{-all-struct-inv } S \wedge \text{no-smaller-propa } S \wedge$
 $\text{cdcl}_W\text{-stgy-invariant } S \wedge \text{propagated-clauses-clauses } S$

interpretation *cdcl_W-with-strategy*; *cdcl_{NOT}-merge-bj-learn-ops* **where**

trail $= \lambda S. \text{convert-trail-from-}W\ (\text{trail } S)$ **and**

clauses_{NOT} $= \text{clauses}$ **and**

prepend-trail $= \lambda L\ S. \text{cons-trail } (\text{convert-ann-lit-from-NOT } L)\ S$ **and**

tl-trail $= \lambda S. \text{tl-trail } S$ **and**

add-cl_{NOT} $= \lambda C\ S. \text{add-learned-cls } C\ S$ **and**

remove-cl_{NOT} $= \lambda C\ S. \text{remove-cls } C\ S$ **and**

decide-conds $= \text{decide-conds}$ **and**

propagate-conds $= \text{propagate-conds}$ **and**

forget-conds $= \lambda -. \text{False}$ **and**

backjump-l-cond $= \lambda C\ C'\ L'\ S\ T. \text{backjump-l-conds-stgy } C\ C'\ L'\ S\ T$
 $\wedge \text{distinct-mset } C' \wedge L' \notin \# C' \wedge \neg \text{tautology } (\text{add-mset } L'\ C')$

by *unfold-locales*

interpretation *cdcl_W-with-strategy*; *cdcl_{NOT}-merge-bj-learn-proxy* **where**

trail $= \lambda S. \text{convert-trail-from-}W\ (\text{trail } S)$ **and**

clauses_{NOT} $= \text{clauses}$ **and**

prepend-trail $= \lambda L\ S. \text{cons-trail } (\text{convert-ann-lit-from-NOT } L)\ S$ **and**

tl-trail $= \lambda S. \text{tl-trail } S$ **and**

add-cl_{NOT} $= \lambda C\ S. \text{add-learned-cls } C\ S$ **and**

remove-cl_{NOT} $= \lambda C\ S. \text{remove-cls } C\ S$ **and**

decide-conds $= \text{decide-conds}$ **and**

propagate-conds $= \text{propagate-conds}$ **and**

forget-conds $= \lambda -. \text{False}$ **and**

backjump-l-cond $= \text{backjump-l-conds-stgy}$ **and**

inv $= \text{inv}_{\text{NOT-stgy}}$

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 $S = \text{None}$

shows

conflicting $T = \text{None}$

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-rtrancpl[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-rtrancpl-cdcl_W-stgy[of T V]* **by** *blast*
then show *?thesis*
using *confl cdcl_W-stgy.conflict'[of S T]* **by** *auto*
qed

lemma *rtrancpl-transition-function*:
 $\langle R^{**} a b \implies \exists f j. (\forall i < j. R (f i) (f (Suc i))) \wedge f 0 = a \wedge f j = b \rangle$

proof (*induction rule: rtrancpl-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: $\langle \forall i < j. R (f i) (f (Suc i)) \rangle$ **and**
a: $\langle f 0 = a \rangle$ **and**
b: $\langle f j = b \rangle$
by *auto*
let *?f = $\langle f (Suc j := c) \rangle$*

```

have
  i:  $\langle \forall i < \text{Suc } j. R \ (?f \ i) \ ( ?f \ (\text{Suc } i)) \rangle$  and
  a:  $\langle ?f \ 0 = a \rangle$  and
  b:  $\langle ?f \ (\text{Suc } j) = c \rangle$ 
  using i a b R by auto
then show ?case by blast
qed

lemma cdclW-bj-cdclW-stgy:  $\langle \text{cdcl}_W\text{-bj } S \ T \implies \text{cdcl}_W\text{-stgy } S \ T \rangle$ 
  by (rule cdclW-stgy.other') (auto simp: cdclW-bj.simps cdclW-o.simps elim!: rulesE)

lemma cdclW-restart-propagated-clauses-clauses:
   $\langle \text{cdcl}_W\text{-restart } S \ T \implies \text{propagated-clauses-clauses } S \implies \text{propagated-clauses-clauses } T \rangle$ 
  by (induction rule: cdclW-restart-all-induct) (auto simp: propagated-clauses-clauses-def
    in-get-all-mark-of-propagated-in-trail simp: state-prop)

lemma rtrancp-cdclW-restart-propagated-clauses-clauses:
   $\langle \text{cdcl}_W\text{-restart}^{**} S \ T \implies \text{propagated-clauses-clauses } S \implies \text{propagated-clauses-clauses } T \rangle$ 
  by (induction rule: rtrancp-induct) (auto simp: cdclW-restart-propagated-clauses-clauses)

lemma rtrancp-cdclW-stgy-propagated-clauses-clauses:
   $\langle \text{cdcl}_W\text{-stgy}^{**} S \ T \implies \text{propagated-clauses-clauses } S \implies \text{propagated-clauses-clauses } T \rangle$ 
  using rtrancp-cdclW-restart-propagated-clauses-clauses[of S T]
  rtrancp-cdclW-stgy-rtrancp-cdclW-restart by blast

lemma conflicting-clause-bt-lvl-gt-0-backjump:
  assumes
    inv:  $\langle \text{inv}_{NOT}\text{-stgy } S \rangle$  and
    C:  $\langle C \in \# \text{ clauses } S \rangle$  and
    tr-C:  $\langle \text{trail } S \models_{as} C \text{Not } C \rangle$  and
    bt:  $\langle \text{backtrack-lvl } S > 0 \rangle$ 
  shows  $\langle \exists T \ U \ V. \text{conflict } S \ T \wedge \text{full skip-or-resolve } T \ U \wedge \text{backtrack } U \ V \rangle$ 
proof -
  let ?T = update-conflicting (Some C) S
  have confl-S-T:  $\text{conflict } S \ ?T$ 
    using C tr-C inv by (auto intro!: conflict-rule)
  have count:  $\text{count-decided } (\text{trail } S) > 0$ 
    using inv bt unfolding cdclW-stgy-invariant-def cdclW-all-struct-inv-def cdclW-M-level-inv-def
    by auto
  have  $\langle (\exists K \ M'. \text{trail } S = M' @ \text{Decided } K \ \# \ M) \implies D \in \# \text{ clauses } S \implies \neg M \models_{as} C \text{Not } D \rangle$  for M
  D
    using inv C tr-C unfolding cdclW-stgy-invariant-def no-smaller-conf-def
    by auto
  from this[OF - C] have C-ne:  $\langle C \neq \{\#\} \rangle$ 
    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:  $\langle \text{full skip-or-resolve } ?T \ U \rangle$ 
    by (meson wf-exists-normal-form-full wf-skip-or-resolve)
  then have s-o-r:  $\text{skip-or-resolve}^{**} ?T \ U$ 
    unfolding full-def by blast
  then obtain C' where C':  $\langle \text{conflicting } U = \text{Some } C' \rangle$ 
    by (induction rule: rtrancp-induct) (auto simp: skip-or-resolve.simps elim!: rulesE)
  have  $\langle \text{cdcl}_W\text{-stgy}^{**} ?T \ U \rangle$ 
    using s-o-r by induction

```

```

(auto simp: skip-or-resolve.simps dest!: cdclW-bj.intros cdclW-bj-cdclW-stgy)
then have ⟨cdclW-stgy** S U⟩
  using confl-S-T by (auto dest!: cdclW-stgy.intros)
then have
  inv-U: ⟨cdclW-all-struct-inv U⟩ and
  no-smaller-U: ⟨no-smaller-propa U⟩ and
  inv-stgy-U: ⟨cdclW-stgy-invariant U⟩
  using inv rtrancp-cdclW-stgy-cdclW-all-struct-inv rtrancp-cdclW-stgy-no-smaller-propa
  rtrancp-cdclW-stgy-cdclW-stgy-invariant by blast+
show ?thesis
proof (cases C')
case (add L D)
then obtain V where ⟨cdclW-stgy U V⟩
  using conflicting-no-false-can-do-step[of U C'] C' inv-U inv-stgy-U
  unfolding cdclW-all-struct-inv-def cdclW-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: cdclW-stgy.simps cdclW-o.simps cdclW-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 ⟹ skip-or-resolve (f i) (f (Suc i))⟩ and
  f-0: ⟨f 0 = ?T⟩ and
  f-j: ⟨f j = U⟩ for i
  using rtrancp-transition-function[OF f-s-o-r] by blast
have j-0: ⟨j ≠ 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 ≤ 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: ⟨cdclW-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!: cdclW-bj.intros cdclW-bj-cdclW-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: ⟨cdclW-stgy** S (f k)⟩ if ⟨k < j⟩ for k
  using confl-S-T that st-f-T[of k] by (auto dest!: cdclW-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..W-all-struct-inv (f l)⟩ and
  no-smaller-f-l: ⟨no-smaller-propa (f l)⟩ and
  inv-stgy-f-l: ⟨cdclW-stgy-invariant (f l)⟩ and
  propa-clf-f-l: ⟨propagated-clauses-clauses (f l)⟩
  using inv st-f-s-k[OF ⟨l < j⟩] rtrancp-cdclW-stgy-cdclW-all-struct-inv[of S f l]
    rtrancp-cdclW-stgy-no-smaller-propa[of S f l]
    rtrancp-cdclW-stgy-cdclW-stgy-invariant[of S f l]
    rtrancp-cdclW-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 cdclW-all-struct-inv-def cdclW-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-clf-f-l no-smaller-f-l hd-T' inv-f-l
  unfolding cdclW-all-struct-inv-def cdclW-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 cdclW-all-struct-inv-def cdclW-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: ⟨cdclW-all-struct-inv S⟩
shows ⟨cdclW-with-strategy.backjump-l S V⟩
proof –
have inv-U: ⟨cdclW-all-struct-inv U⟩
  by (metis cdclW-stgy.conflict' cdclW-stgy-cdclW-all-struct-inv
    conf full full-def inv rtranclp-cdclW-all-struct-inv-inv
    rtranclp-skip-or-resolve-rtranclp-cdclW-restart)
then have inv-V: ⟨cdclW-all-struct-inv V⟩
  by (metis backtrack bt cdclW-bj-cdclW-stgy cdclW-stgy-cdclW-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'  $\equiv$  i and
lev-K-U: get-level (trail U) K = i + 1 and
V: V  $\sim$  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':  $\langle$ clauses U  $\models_{pm}$  add-mset L D' $\rangle$  and
D-D':  $\langle$ D'  $\subseteq \#$  D $\rangle$ 
using bt by (auto elim!: rulesE)
let ?D' =  $\langle$ add-mset L D' $\rangle$ 
obtain M' where M':  $\langle$ trail U = M' @ M2 @ Decided K # M1 $\rangle$ 
  using decomp by auto
have  $\langle$ clauses V = {#?D'#} + clauses U $\rangle$ 
  using V by auto
moreover have  $\langle$ trail V = (Propagated L ?D') # trail (reduce-trail-to M1 U) $\rangle$ 
  using V T M tr-T-S[symmetric] M' clss-T-U[symmetric] unfolding state-eqNOT-def
  by (auto simp del: state-simp dest!: state-simp(1))
ultimately have V':  $\langle$ V  $\sim_{NOT}$ 
  cons-trail (Propagated L dummy-cls) (reduce-trail-toNOT M1 (add-learned-cls ?D' S)) $\rangle$ 
  using V T M tr-T-S[symmetric] M' clss-T-U[symmetric] unfolding state-eqNOT-def
  by (auto simp del: state-simp
    simp: trail-reduce-trail-toNOT-drop drop-map drop-tl clss-T-S)
have  $\langle$ no-dup (trail V) $\rangle$ 
  using inv-V V unfolding cdclW-all-struct-inv-def cdclW-M-level-inv-def by blast
then have undef-L:  $\langle$ undefined-lit M1 L $\rangle$ 
  using V decomp by (auto simp: defined-lit-map)

have  $\langle$ atm-of L  $\in$  atms-of-mm (init-clss V) $\rangle$ 
  using inv-V V decomp unfolding cdclW-all-struct-inv-def no-strange-atm-def by auto
moreover have init-clss-VU-S:  $\langle$ init-clss V = init-clss S $\rangle$ 
   $\langle$ init-clss U = init-clss S $\rangle$   $\langle$ learned-clss U = learned-clss S $\rangle$ 
  using T V init-T-U learned-T-U by auto
ultimately have atm-L:  $\langle$ atm-of L  $\in$  atms-of-mm (clauses S) $\rangle$ 
  by (auto simp: clauses-def)

have  $\langle$ distinct-mset ?D' $\rangle$  and  $\langle$  $\neg$  tautology ?D' $\rangle$ 
  using inv-U confl-D decomp D-D' unfolding cdclW-all-struct-inv-def distinct-cdclW-state-def
  apply simp-all
  using inv-V V not-tautology-mono[OF D-D'] distinct-mset-mono[OF D-D']
  unfolding cdclW-all-struct-inv-def
  apply (auto simp add: tautology-add-mset)
  done
have  $\langle$ L  $\notin \#$  D' $\rangle$ 
  using  $\langle$ distinct-mset ?D' $\rangle$  by (auto simp: not-in-iff)
have bj:  $\langle$ backjump-l-conds-stgy C D' L S V $\rangle$ 
  apply (rule exI[of - T], rule exI[of - U])
  using  $\langle$ distinct-mset ?D' $\rangle$   $\langle$  $\neg$  tautology ?D' $\rangle$  conf full bt confl-D
   $\langle$ L  $\notin \#$  D' $\rangle$  V T
  by (auto)

have M1-D': M1  $\models_{as}$  CNot D'
  using backtrack-M1-CNot-D'[of U D'  $\langle$ i $\rangle$  K M1 M2 L  $\langle$ add-mset L D $\rangle$  V  $\langle$ Propagated L (add-mset L
```

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

lemma *is-decided-o-convert-ann-lit-from-W* [*simp*]:
 $\langle is-decided \text{ o } convert-ann-lit-from-W = is-decided \rangle$
apply (*rule ext*)
apply (*rename-tac x, case-tac x*)
apply (*auto simp: comp-def*)
done

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

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

$add_cls_{NOT} = \lambda C S. add_learned_cls C S$ and
 $remove_cls_{NOT} = \lambda C S. remove_cls C S$ and
 $decide_conds = decide_conds$ and
 $propagate_conds = propagate_conds$ and
 $forget_conds = \lambda -. False$ 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 $cdcl_W-with-strategy-cdcl_{NOT}-merged-bj-learn-cdcl_W-stgy[of S T]$
 $cdcl_W-with-strategy-cdcl_{NOT}-merged-bj-learn-conflict[of S T]$
 $rtrancpl-cdcl_W-stgy-cdcl_W-all-struct-inv$ $rtrancpl-cdcl_W-stgy-no-smaller-propa$
 $rtrancpl-cdcl_W-stgy-cdcl_W-stgy-invariant$ $rtrancpl-cdcl_W-stgy-propagated-clauses-clauses$
by blast
next
case (1 $C' S C F' K F L$)
have $\langle count-decided (convert-trail-from-W (trail S)) > 0 \rangle$
unfolding $\langle convert-trail-from-W (trail S) = F' @ Decided K \# F \rangle$ **by** simp
then have $\langle count-decided (trail S) > 0 \rangle$
by simp
then have $\langle backtrack-lvl S > 0 \rangle$
using $\langle inv_{NOT_stgy} S \rangle$ **unfolding** $cdcl_W-all-struct-inv-def$ $cdcl_W-M-level-inv-def$ **by** auto
have $\exists T U V. conflict S T \wedge full\ skip-or-resolve\ T\ U \wedge backtrack\ U\ V$
apply (*rule conflicting-clause-bt-lvl-gt-0-backjump*)
using $\langle inv_{NOT_stgy} S \rangle$ **apply** (*auto; fail*)[]
using $\langle C \in \# clauses S \rangle$ **apply** (*simp; fail*)
using $\langle convert-trail-from-W (trail S) \models_{as} CNot C \rangle$ **apply** (*simp; fail*)
using $\langle backtrack-lvl S > 0 \rangle$ **by** (*simp; fail*)
then show ?case
using *conflict-full-skip-or-resolve-backtrack-backjump-l* $\langle inv_{NOT_stgy} S \rangle$ **by** blast
next
case (3 $L S$) **note** $atm = this(1,2)$ **and** $inv = this(3)$ **and** $sat = this(4)$
moreover have $\langle Ex(cdcl_W-with-strategy.backjump-l S) \rangle$ **if** $\langle conflict S T \rangle$ **for** T
proof –
have $\langle \exists C. C \in \# clauses S \wedge trail S \models_{as} CNot C \rangle$
using *that* **by** (*auto elim: rulesE*)
then obtain C **where** $\langle C \in \# clauses S \rangle$ **and** $\langle trail S \models_{as} CNot C \rangle$ **by** blast
have $\langle backtrack-lvl S > 0 \rangle$
proof (*rule ccontr*)
assume $\neg ?thesis$
then have $\langle backtrack-lvl S = 0 \rangle$
by simp
then have $\langle count-decided (trail S) = 0 \rangle$
using inv **unfolding** $cdcl_W-all-struct-inv-def$ $cdcl_W-M-level-inv-def$ **by** simp
then have $\langle get-all-ann-decomposition (trail S) = [([], trail S)] \rangle$
by (*auto simp: filter-empty-conv no-decision-get-all-ann-decomposition count-decided-0-iff*)
then have $\langle set-mset (clauses S) \models_{ps} unmark-l (trail S) \rangle$
using 3(3) **unfolding** $cdcl_W-all-struct-inv-def$ **by** auto
obtain I **where**
 $consistent: \langle consistent-interp I \rangle$ **and**
 $I-S: \langle I \models_m clauses S \rangle$ **and**
 $tot: \langle total-over-m I (set-mset (clauses S)) \rangle$
using sat **by** (*auto simp: satisfiable-def*)
have $\langle total-over-m I (set-mset (clauses S)) \wedge total-over-m I (unmark-l (trail S)) \rangle$
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  $\langle I \models_s \text{unmark-l } (\text{trail } S) \rangle$ 
    using (set-mset (clauses S)  $\models_{ps}$  unmark-l (trail S)) consistent I-S
    unfolding true-clss-clss-def clauses-def
    by auto

  have  $\langle I \models_s \text{CNot } C \rangle$ 
    by (meson  $\langle \text{trail } S \models_{as} \text{CNot } C \rangle \langle I \models_s \text{unmark-l } (\text{trail } S) \rangle$  set-mp true-annots-true-clss
        true-clss-def true-clss-def true-clss-singleton-lit-of-implies-incl true-lit-def)
  moreover have  $\langle I \models C \rangle$ 
    using  $\langle C \in \# \text{ clauses } S \rangle$  and  $\langle I \models_m \text{ clauses } S \rangle$  unfolding true-clss-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]
     $\langle C \in \# \text{ clauses } S \rangle \langle \text{trail } S \models_{as} \text{CNot } C \rangle$  inv by fast
qed
moreover {
  have atm:  $\langle \text{atms-of-mm } (\text{clauses } S) = \text{atms-of-mm } (\text{init-clss } S) \rangle$ 
    using 3(3) unfolding cdclW-all-struct-inv-def no-strange-atm-def
    by (auto simp: clauses-def)
  have  $\langle \text{decide } S (\text{cons-trail } (\text{Decided } L) S) \rangle$ 
    apply (rule decide-rule)
    using 3 by (auto simp: atm) }
  moreover have  $\langle \text{cons-trail } (\text{Decided } L) S \sim_{NOT} \text{cons-trail } (\text{Decided } L) S \rangle$ 
    by (simp add: state-eqNOT-def del: state-simp)
  ultimately show  $\exists T. \text{cdcl}_W\text{-with-strategy.decide}_{NOT} S T \vee$ 
     $\text{cdcl}_W\text{-with-strategy.propagate}_{NOT} S T \vee$ 
     $\text{cdcl}_W\text{-with-strategy.backjump-l } S T$ 
    using  $\text{cdcl}_W\text{-with-strategy.decide}_{NOT}.\text{intros}$ [of S L  $\text{cons-trail } (\text{Decided } L) S$ ]
    by auto
qed

thm cdclW-with-strategy.full-cdclNOT-merged-bj-learn-final-state

end

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

context conflict-driven-clause-learningW
begin
lemma rtrancpl-cdclW-merge-stgy-distinct-mset-clauses:
  assumes
    invR:  $\text{cdcl}_W\text{-all-struct-inv } R$  and
    st:  $\text{cdcl}_W\text{-s}^{**} R S$  and
    smaller:  $\langle \text{no-smaller-propa } R \rangle$  and
    dist:  $\text{distinct-mset } (\text{clauses } R)$ 
  shows  $\text{distinct-mset } (\text{clauses } S)$ 
  using rtrancpl-cdclW-stgy-distinct-mset-clauses[OF - invR dist smaller]
    invR st rtrancpl-mono[of  $\text{cdcl}_W\text{-s}' \text{cdcl}_W\text{-stgy}^{**}$ ]  $\text{cdcl}_W\text{-s}'\text{-is-rtrancpl-cdcl}_W\text{-stgy}$ 
    by (auto dest!:  $\text{cdcl}_W\text{-s}'\text{-is-rtrancpl-cdcl}_W\text{-stgy}$ )

```

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** $S\ T :: \langle 'st \times nat \rangle$ **where**
 $\langle cdcl_W\text{-stgy}\ (fst\ S)\ (fst\ T) \implies snd\ S = snd\ T \implies cdcl_W\text{-restart-stgy}\ S\ T \rangle \mid$
 $\langle restart\ (fst\ S)\ (fst\ T) \implies snd\ T = Suc\ (snd\ S) \implies cdcl_W\text{-restart-stgy}\ S\ T \rangle$

lemma *cdcl_W-stgy-cdcl_W-restart*: $\langle cdcl_W\text{-stgy}\ S\ S' \implies cdcl_W\text{-restart}\ S\ S' \rangle$
by (*induction rule*: *cdcl_W-stgy.induct*) *auto*

lemma *cdcl_W-restart-stgy-cdcl_W-restart*:
 $\langle cdcl_W\text{-restart-stgy}\ S\ T \implies cdcl_W\text{-restart}\ (fst\ S)\ (fst\ T) \rangle$
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 *rtrancpl-cdcl_W-restart-stgy-cdcl_W-restart*:
 $\langle cdcl_W\text{-restart-stgy}^{**}\ S\ T \implies cdcl_W\text{-restart}^{**}\ (fst\ S)\ (fst\ T) \rangle$
by (*induction rule*: *rtrancpl-induct*)
(*auto dest*: *cdcl_W-restart-stgy-cdcl_W-restart*)

lemma *cdcl_W-stgy-cdcl_W-restart-stgy*:
 $\langle cdcl_W\text{-stgy}\ S\ T \implies cdcl_W\text{-restart-stgy}\ (S, n)\ (T, n) \rangle$
using *cdcl_W-restart-stgy.intros* [*of* $\langle (S, n) \rangle \langle (T, n) \rangle$]
by *auto*

lemma *rtrancpl-cdcl_W-stgy-cdcl_W-restart-stgy*:
 $\langle cdcl_W\text{-stgy}^{**}\ S\ T \implies cdcl_W\text{-restart-stgy}^{**}\ (S, n)\ (T, n) \rangle$
apply (*induction rule*: *rtrancpl-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*:
 $\langle cdcl_W\text{-restart-stgy}\ S\ T \implies cdcl_W\text{-all-struct-inv}\ (fst\ S) \implies cdcl_W\text{-all-struct-inv}\ (fst\ T) \rangle$

using *cdcl_W-all-struct-inv-inv*[*OF cdcl_W-restart-stgy-cdcl_W-restart*] .

lemma *rtrancp-cdcl_W-restart-dcl_W-all-struct-inv*:
 $\langle \text{cdcl}_W\text{-restart-stgy}^{**} S T \implies \text{cdcl}_W\text{-all-struct-inv (fst } S) \implies \text{cdcl}_W\text{-all-struct-inv (fst } T) \rangle$
by (*induction rule*: *rtrancp-induct*)
(auto intro: cdcl_W-restart-dcl_W-all-struct-inv)

lemma *restart-cdcl_W-stgy-invariant*:
 $\langle \text{restart } S T \implies \text{cdcl}_W\text{-stgy-invariant } T \rangle$
by (*auto simp: restart.simps cdcl_W-stgy-invariant-def state-prop no-smaller-conf-def*)

lemma *cdcl_W-restart-dcl_W-stgy-invariant*:
 $\langle \text{cdcl}_W\text{-restart-stgy } S T \implies \text{cdcl}_W\text{-all-struct-inv (fst } S) \implies \text{cdcl}_W\text{-stgy-invariant (fst } S) \implies \text{cdcl}_W\text{-stgy-invariant (fst } T) \rangle$
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 *rtrancp-cdcl_W-restart-dcl_W-stgy-invariant*:
 $\langle \text{cdcl}_W\text{-restart-stgy}^{**} S T \implies \text{cdcl}_W\text{-all-struct-inv (fst } S) \implies \text{cdcl}_W\text{-stgy-invariant (fst } S) \implies \text{cdcl}_W\text{-stgy-invariant (fst } T) \rangle$
apply (*induction rule: rtrancp-induct*)
subgoal by *auto*
subgoal by (*auto simp: rtrancp-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 :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ (**infix** ~ 50) **and**
state :: $\langle 'st \Rightarrow ('v, 'v \text{ clause}) \text{ ann-lits} \times 'v \text{ clauses} \times 'v \text{ clauses} \times 'v \text{ clause option} \times 'b \rangle$ **and**
trail :: $\langle 'st \Rightarrow ('v, 'v \text{ clause}) \text{ ann-lits} \rangle$ **and**
init-clss :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**
learned-clss :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**
conflicting :: $\langle 'st \Rightarrow 'v \text{ clause option} \rangle$ **and**

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


```

  init-state :: ⟨'v clauses ⇒ 'st⟩ +
fixes
  f :: ⟨nat ⇒ nat⟩

locale cdclW-restart-restart =
  cdclW-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 $cdcl_W\text{-stgy}^{+\downarrow} S T$: With a $cdcl_W\text{-stgy}^\downarrow S T$, this rules could be applied one after the other, doing nothing each time.

```

context cdclW-restart-restart-ops

```

```

begin

```

```

inductive cdclW-merge-with-restart where

```

```

  restart-step:

```

```

  ⟨(cdclW-stgy~(card (set-mset (learned-clss T)) - card (set-mset (learned-clss S)))) S T
  ⇒ card (set-mset (learned-clss T)) - card (set-mset (learned-clss S)) > f n
  ⇒ restart T U ⇒ cdclW-merge-with-restart (S, n) (U, Suc n)⟩ |

```

```

  restart-full: ⟨full1 cdclW-stgy S T ⇒ cdclW-merge-with-restart (S, n) (T, Suc n)⟩

```

```

lemma cdclW-merge-with-restart-rtranclp-cdclW-restart:

```

```

  ⟨cdclW-merge-with-restart S T ⇒ cdclW-restart** (fst S) (fst T)⟩

```

```

  by (induction rule: cdclW-merge-with-restart.induct)

```

```

  (auto dest!: relpowp-imp-rtranclp rtranclp-cdclW-stgy-rtranclp-cdclW-restart cdclW-restart.rf
  cdclW-rf.restart tranclp-into-rtranclp simp: full1-def)

```

```

lemma cdclW-merge-with-restart-increasing-number:

```

```

  ⟨cdclW-merge-with-restart S T ⇒ snd T = 1 + snd S⟩

```

```

  by (induction rule: cdclW-merge-with-restart.induct) auto

```

```

lemma ⟨full1 cdclW-stgy S T ⇒ cdclW-merge-with-restart (S, n) (T, Suc n)⟩

```

```

  using restart-full by blast

```

```

lemma cdclW-all-struct-inv-learned-clss-bound:

```

```

  assumes inv: ⟨cdclW-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 cdclW-all-struct-inv-def distinct-cdclW-state-def distinct-mset-set-def
    by auto

```

```

  moreover have ⟨¬tautology C⟩

```

```

    using C inv unfolding cdclW-all-struct-inv-def cdclW-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 cdclW-all-struct-inv-def no-strange-atm-def by force

```

```

  moreover have ⟨finite (atms-of-mm (init-clss S))⟩

```

```

    using inv unfolding cdclW-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 cdclW-merge-with-restart-init-clss:
  ⟨cdclW-merge-with-restart  $S$   $T$   $\implies$  cdclW-M-level-inv (fst  $S$ )  $\implies$ 
  init-clss (fst  $S$ ) = init-clss (fst  $T$ )⟩
  using cdclW-merge-with-restart-rtrancpl-cdclW-restart rtrancpl-cdclW-restart-init-clss by blast

```

```

lemma (in cdclW-restart-restart)
  ⟨wf {( $T$ ,  $S$ ). cdclW-all-struct-inv (fst  $S$ )  $\wedge$  cdclW-merge-with-restart  $S$   $T$ }⟩

```

```

proof (rule ccontr)
  assume ⟨ $\neg$  ?thesis⟩
  then obtain  $g$  where
     $g$ : ⟨ $\bigwedge i$ . cdclW-merge-with-restart ( $g$   $i$ ) ( $g$  (Suc  $i$ ))⟩ and
     $inv$ : ⟨ $\bigwedge i$ . cdclW-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 cdclW-all-struct-inv-def by (metis cdclW-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 cdclW-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 cdclW-merge-with-restart-increasing-number  $g$ )
  then have snd- $g$ -0: ⟨ $\bigwedge i$ .  $i > 0 \implies$  snd ( $g$   $i$ ) =  $i$  + snd ( $g$  0)⟩
    by blast
  have unbounded- $f$ - $g$ : ⟨unbounded ( $\lambda i$ .  $f$  (snd ( $g$   $i$ )))⟩
    using  $f$  unfolding bounded-def by (metis add.commute  $f$  less-or-eq-imp-le snd- $g$ 
      not-bounded-nat-exists-larger not-le le-iff-add)

  obtain  $k$  where
     $f$ - $g$ - $k$ : ⟨ $f$  (snd ( $g$   $k$ )) > card (simple-clss (atms-of-mm (init-clss (fst ? $S$ )))⟩ and
    ⟨ $k$  > card (simple-clss (atms-of-mm (init-clss (fst ? $S$ )))⟩
    using not-bounded-nat-exists-larger[OF unbounded- $f$ - $g$ ] by blast

```

The following does not hold anymore with the non-strict version of cardinality in the definition.

```

{ fix  $i$ 
  assume ⟨no-step cdclW-stgy (fst ( $g$   $i$ ))⟩
  with  $g$ [of  $i$ ]
  have False
  proof (induction rule: cdclW-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 ⟨cdclW-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: trancplD)
  qed
} note  $H$  = this

```

obtain $m \ T$ **where**
 m : $\langle m = \text{card} (\text{set-mset} (\text{learned-clss } T)) - \text{card} (\text{set-mset} (\text{learned-clss} (\text{fst } (g \ k)))) \rangle$ **and**
 $\langle m > f (\text{snd } (g \ k)) \rangle$ **and**
 $\langle \text{restart } T (\text{fst } (g \ (k+1))) \rangle$ **and**
 $\text{cdcl}_W\text{-stgy}$: $\langle (\text{cdcl}_W\text{-stgy} \ \widehat{\sim} \ m) (\text{fst } (g \ k)) \ T \rangle$
using $g[\text{of } k] \ H[\text{of } \langle \text{Suc } k \rangle]$ **by** $(\text{force simp: cdcl}_W\text{-merge-with-restart.simps full1-def})$
have $\langle \text{cdcl}_W\text{-stgy}^{**} (\text{fst } (g \ k)) \ T \rangle$
using $\text{cdcl}_W\text{-stgy relpowp-imp-rtrancpl}$ **by** *metis*
then have $\langle \text{cdcl}_W\text{-all-struct-inv } T \rangle$
using $\text{inv}[\text{of } k] \ \text{rtrancpl-cdcl}_W\text{-all-struct-inv-inv rtrancpl-cdcl}_W\text{-stgy-rtrancpl-cdcl}_W\text{-restart}$
by *blast*
moreover have $\langle \text{card} (\text{set-mset} (\text{learned-clss } T)) - \text{card} (\text{set-mset} (\text{learned-clss} (\text{fst } (g \ k)))) \rangle$
 $> \text{card} (\text{simple-clss} (\text{atms-of-mm} (\text{init-clss} (\text{fst } ?S)))) \rangle$
unfolding $m[\text{symmetric}]$ **using** $\langle m > f (\text{snd } (g \ k)) \rangle \ f\text{-}g\text{-}k$ **by** *linarith*
then have $\langle \text{card} (\text{set-mset} (\text{learned-clss } T)) \rangle$
 $> \text{card} (\text{simple-clss} (\text{atms-of-mm} (\text{init-clss} (\text{fst } ?S)))) \rangle$
by *linarith*
moreover
have $\langle \text{init-clss} (\text{fst } (g \ k)) = \text{init-clss } T \rangle$
using $\langle \text{cdcl}_W\text{-stgy}^{**} (\text{fst } (g \ k)) \ T \rangle \ \text{rtrancpl-cdcl}_W\text{-stgy-rtrancpl-cdcl}_W\text{-restart}$
 $\ \text{rtrancpl-cdcl}_W\text{-restart-init-clss inv unfolding cdcl}_W\text{-all-struct-inv-def}$ **by** *blast*
then have $\langle \text{init-clss} (\text{fst } ?S) = \text{init-clss } T \rangle$
using $\text{init-g}[\text{of } k]$ **by** *auto*
ultimately show *False*
using $\text{cdcl}_W\text{-all-struct-inv-learned-clss-bound}$
by $(\text{simp add: } \langle \text{finite} (\text{atms-of-mm} (\text{init-clss} (\text{fst } (g \ 0)))) \rangle \ \text{simple-clss-finite}$
 $\ \text{card-mono leD})$

qed

lemma $\text{cdcl}_W\text{-merge-with-restart-distinct-mset-clauses}$:

assumes invR : $\langle \text{cdcl}_W\text{-all-struct-inv} (\text{fst } R) \rangle$ **and**
 st : $\langle \text{cdcl}_W\text{-merge-with-restart } R \ S \rangle$ **and**
 dist : $\langle \text{distinct-mset} (\text{clauses} (\text{fst } R)) \rangle$ **and**
 R : $\langle \text{no-smaller-propa} (\text{fst } R) \rangle$
shows $\langle \text{distinct-mset} (\text{clauses} (\text{fst } S)) \rangle$
using $\text{assms}(2,1,3,4)$

proof *induction*

case $(\text{restart-full } S \ T)$
then show $?case$ **using** $\text{rtrancpl-cdcl}_W\text{-stgy-distinct-mset-clauses}[\text{of } S \ T]$ **unfolding** full1-def
by $(\text{auto dest: trancpl-into-rtrancpl})$

next

case $(\text{restart-step } T \ S \ n \ U)$
then have $\langle \text{distinct-mset} (\text{clauses } T) \rangle$
using $\text{rtrancpl-cdcl}_W\text{-stgy-distinct-mset-clauses}[\text{of } S \ T]$ **unfolding** full1-def
by $(\text{auto dest: relpowp-imp-rtrancpl})$
then show $?case$ **using** $\langle \text{restart } T \ U \rangle$ **unfolding** clauses-def
by $(\text{metis distinct-mset-union fstI restartE subset-mset.le-iff-add union-assoc})$

qed

inductive $\text{cdcl}_W\text{-restart-with-restart}$ **where**

restart-step:

$\langle \text{cdcl}_W\text{-stgy}^{**} \ S \ T \implies$
 $\text{card} (\text{set-mset} (\text{learned-clss } T)) - \text{card} (\text{set-mset} (\text{learned-clss } S)) > f \ n \implies$
 $\text{restart } T \ U \implies$

$\text{cdcl}_W\text{-restart-with-restart } (S, n) \ (U, \text{Suc } n) \mid$

restart-full: $\langle \text{full1 cdcl}_W\text{-stgy } S \ T \implies \text{cdcl}_W\text{-restart-with-restart } (S, n) \ (T, \text{Suc } n) \rangle$

```

lemma cdclW-restart-with-restart-rtrancpl-cdclW-restart:
  ⟨cdclW-restart-with-restart S T  $\implies$  cdclW-restart** (fst S) (fst T)⟩
  apply (induction rule: cdclW-restart-with-restart.induct)
  by (auto dest!: relpoup-imp-rtrancpl trancpl-into-rtrancpl cdclW-restart.rf
    cdclW-rf.restart rtrancpl-cdclW-stgy-rtrancpl-cdclW-restart
    simp: full1-def)

lemma cdclW-restart-with-restart-increasing-number:
  ⟨cdclW-restart-with-restart S T  $\implies$  snd T = 1 + snd S⟩
  by (induction rule: cdclW-restart-with-restart.induct) auto

lemma ⟨full1 cdclW-stgy S T  $\implies$  cdclW-restart-with-restart (S, n) (T, Suc n)⟩
  using restart-full by blast

lemma cdclW-restart-with-restart-init-clss:
  ⟨cdclW-restart-with-restart S T  $\implies$  cdclW-M-level-inv (fst S)  $\implies$ 
    init-clss (fst S) = init-clss (fst T)⟩
  using cdclW-restart-with-restart-rtrancpl-cdclW-restart rtrancpl-cdclW-restart-init-clss by blast

theorem (in cdclW-restart-restart)
  ⟨wf {(T, S). cdclW-all-struct-inv (fst S)  $\wedge$  cdclW-restart-with-restart S T}⟩
proof (rule ccontr)
  assume ⟨ $\neg$  ?thesis⟩
  then obtain g where
    g: ⟨ $\bigwedge i$ . cdclW-restart-with-restart (g i) (g (Suc i))⟩ and
    inv: ⟨ $\bigwedge i$ . cdclW-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 cdclW-all-struct-inv-def by (metis cdclW-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 cdclW-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 cdclW-restart-with-restart-increasing-number g)
  then have snd-g-0: ⟨ $\bigwedge i$ . i > 0  $\implies$  snd (g i) = i + snd (g 0)⟩
    by blast
  have unbounded-f-g: ⟨unbounded ( $\lambda i$ . f (snd (g i)))⟩
    using f unfolding bounded-def by (metis add.commute f less-or-eq-imp-le snd-g
      not-bounded-nat-exists-larger not-le le-iff-add)

  obtain k where
    f-g-k: ⟨f (snd (g k)) > card (simple-clss (atms-of-mm (init-clss (fst ?S))))⟩ and
    ⟨k > card (simple-clss (atms-of-mm (init-clss (fst ?S))))⟩
    using not-bounded-nat-exists-larger[OF unbounded-f-g] by blast

```

The following does not hold anymore with the non-strict version of cardinality in the definition.

```

have H: False if ⟨no-step cdclW-stgy (fst (g i))⟩ for i
  using g[of i] that
proof (induction rule: cdclW-restart-with-restart.induct)

```

```

    case (restart-step S T n) note H = this(1) and c = this(2) and n-s = this(4)
  obtain S' where ⟨cdclW-stgy S S'⟩
    using H c by (subst (asm) rtrancpl-unfold) (auto dest!: trancplD)
    then show False using n-s by auto
next
  case (restart-full S T)
    then show False unfolding full1-def by (auto dest: trancplD)
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
  cdclW-stgy: ⟨cdclW-stgy** (fst (g k)) T⟩
  using g[of k] H[of ⟨Suc k⟩] by (force simp: cdclW-restart-with-restart.simps full1-def)
have ⟨cdclW-all-struct-inv T⟩
  using inv[of k] rtrancpl-cdclW-all-struct-inv-inv rtrancpl-cdclW-stgy-rtrancpl-cdclW-restart
    cdclW-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 ⟨cdclW-stgy** (fst (g k)) T⟩ rtrancpl-cdclW-stgy-rtrancpl-cdclW-restart rtrancpl-cdclW-restart-init-clss
    inv unfolding cdclW-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 cdclW-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: ⟨cdclW-all-struct-inv (fst R)⟩ and
  st: ⟨cdclW-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 rtrancpl-cdclW-stgy-distinct-mset-clauses[of S T] unfolding full1-def
    by (auto dest: trancpl-into-rtrancpl)
next
  case (restart-step S T n U)
  then have ⟨distinct-mset (clauses T)⟩ using rtrancpl-cdclW-stgy-distinct-mset-clauses[of S T]
    unfolding full1-def by (auto dest: relpowp-imp-rtrancpl)
  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 $i = (2::'a)^k - (1::'a)$
- $\text{luby-sequence-core } (i - 2^{k-1} + 1)$, if $(2::'a)^{k-1} \leq i$ and $i \leq (2::'a)^k - (1::'a)$

Then the sequence is then scaled by a constant unit run (called *ur* here), strictly positive.

```

function luby-sequence-core :: ⟨nat ⇒ nat⟩ where
  ⟨luby-sequence-core i =
    (if ∃ k. i = 2^k - 1
      then 2^((SOME k. i = 2^k - 1) - 1)
      else luby-sequence-core (i - 2^((SOME k. 2^(k-1) ≤ i ∧ i < 2^k - 1) - 1) + 1))⟩
by auto

```

termination

proof (*relation less-than*, *goal-cases*)

case 1

then show ?case by auto

next

case (2 i)

let ?k = $\langle \text{SOME } k. 2 \wedge (k - 1) \leq i \wedge i < 2 \wedge k - 1 \rangle$

have $\langle 2 \wedge (?k - 1) \leq i \wedge i < 2 \wedge ?k - 1 \rangle$

by (rule someI-ex) (use 2 exists-luby-decomp in blast)

then show ?case

proof –

have $\langle \forall n \text{ na. } \neg (1::\text{nat}) \leq n \vee 1 \leq n \wedge \text{na} \rangle$

by (meson one-le-power)

then have f1: $\langle (1::\text{nat}) \leq 2 \wedge (?k - 1) \rangle$

using one-le-numeral by blast

have f2: $\langle i - 2 \wedge (?k - 1) + 2 \wedge (?k - 1) = i \rangle$

using $\langle 2 \wedge (?k - 1) \leq i \wedge i < 2 \wedge ?k - 1 \rangle$ le-add-diff-inverse2 by blast

have f3: $\langle 2 \wedge ?k - 1 \neq \text{Suc } 0 \rangle$

using f1 $\langle 2 \wedge (?k - 1) \leq i \wedge i < 2 \wedge ?k - 1 \rangle$ by linarith

have $\langle 2 \wedge ?k - (1::\text{nat}) \neq 0 \rangle$

using $\langle 2 \wedge (?k - 1) \leq i \wedge i < 2 \wedge ?k - 1 \rangle$ gr-implies-not0 by blast

then have f4: $\langle 2 \wedge ?k \neq (1::\text{nat}) \rangle$

by linarith

have f5: $\langle \forall n \text{ na. if na = 0 then } (n::\text{nat}) \wedge \text{na} = 1 \text{ else } n \wedge \text{na} = n * n \wedge (\text{na} - 1) \rangle$

by (simp add: power-eq-if)

then have $\langle ?k \neq 0 \rangle$

using f4 by meson

then have $\langle 2 \wedge (?k - 1) \neq \text{Suc } 0 \rangle$

using f5 f3 by presburger

then have $\langle \text{Suc } 0 < 2 \wedge (?k - 1) \rangle$

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: $\langle (2::\text{nat}) \wedge (k::\text{nat}) - 1 = 2 \wedge k' - 1 \rangle$

shows $\langle k' = k \rangle$

proof –

have $\langle (2::\text{nat}) \wedge (k::\text{nat}) = 2 \wedge k' \rangle$

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:

$\langle \text{luby-sequence-core } (2^k - 1) = 2^{(k-1)} \rangle$ (is $\langle ?L = ?K \rangle$)

proof –

have decomp: $\langle \exists ka. 2 \wedge k - 1 = 2 \wedge ka - 1 \rangle$

by auto

have $\langle ?L = 2^{((\text{SOME } k'. (2::\text{nat}) \wedge k - 1 = 2 \wedge k' - 1) - 1)} \rangle$

apply (subst luby-sequence-core.simps, subst decomp)

by simp

moreover have $\langle (SOME\ k'.\ (2::nat)^\wedge k - 1 = 2^\wedge k' - 1) = k \rangle$
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
 $H: \langle 2^\wedge (k - Suc\ 0) \leq i \rangle$ **and**
 $k': \langle i < 2^\wedge k' - Suc\ 0 \rangle$ **and**
 $k-k': \langle k > k' \rangle$
shows $\langle False \rangle$

proof $-$

have $\langle 2^\wedge k' - Suc\ 0 < 2^\wedge (k - Suc\ 0) \rangle$
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
 $k-i: \langle 2^\wedge (k - 1) \leq i \rangle$ **and**
 $i-k: \langle i < 2^\wedge k - 1 \rangle$
shows $\langle luby-sequence-core\ i = luby-sequence-core\ (i - 2^\wedge (k - 1) + 1) \rangle$

proof $-$

have $H: \langle \neg (\exists ka.\ i = 2^\wedge ka - 1) \rangle$
proof *(rule ccontr)*
assume $\langle \neg ?thesis \rangle$
then obtain $k':nat$ **where** $k': \langle i = 2^\wedge k' - 1 \rangle$ **by** *blast*
have $\langle (2::nat)^\wedge k' - 1 < 2^\wedge k - 1 \rangle$
using *i-k unfolding k'*.
then have $\langle (2::nat)^\wedge k' < 2^\wedge k \rangle$
by *linarith*
then have $\langle k' < k \rangle$
by *simp*
have $\langle 2^\wedge (k - 1) \leq 2^\wedge k' - (1::nat) \rangle$
using *k-i unfolding k'*.
then have $\langle (2::nat)^\wedge (k-1) < 2^\wedge k' \rangle$
by *(metis Suc-diff-1 not-le not-less-eq zero-less-numeral zero-less-power)*
then have $\langle k-1 < k' \rangle$
by *simp*

show *False* **using** $\langle k' < k \rangle \langle k-1 < k' \rangle$ **by** *linarith*

qed

have $\langle \bigwedge k\ k'.\ 2^\wedge (k - Suc\ 0) \leq i \implies i < 2^\wedge k - Suc\ 0 \implies 2^\wedge (k' - Suc\ 0) \leq i \implies i < 2^\wedge k' - Suc\ 0 \implies k = k' \rangle$

by *(meson different-luby-decomposition-false linorder-neqE-nat)*

then have $k: \langle (SOME\ k.\ 2^\wedge (k - Suc\ 0) \leq i \wedge i < 2^\wedge k - Suc\ 0) = k \rangle$

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  $\langle \exists b. \forall n. \text{luby-sequence-core } n \leq b \rangle$ 
    then obtain  $b$  where  $b$ :  $\langle \bigwedge n. \text{luby-sequence-core } n \leq b \rangle$ 
      by metis
    have  $\langle \text{luby-sequence-core } (2^{b+1} - 1) = 2^b \rangle$ 
      using luby-sequence-core-two-power-minus-one[of  $b+1$ ] by simp
    moreover have  $\langle (2::nat)^b > b \rangle$ 
      by (induction b) auto
    ultimately show False using  $b$ [of  $\langle 2^{b+1} - 1 \rangle$ ] by linarith
  qed

  abbreviation luby-sequence ::  $\langle nat \Rightarrow nat \rangle$  where
     $\langle \text{luby-sequence } n \equiv ur * \text{luby-sequence-core } n \rangle$ 

  lemma bounded-luby-sequence:  $\langle \text{unbounded luby-sequence} \rangle$ 
    using bounded-const-product[of ur] luby-sequence-axioms
    luby-sequence-def unbounded-luby-sequence-core by blast

  lemma luby-sequence-core-0:  $\langle \text{luby-sequence-core } 0 = 1 \rangle$ 
  proof -
    have  $0$ :  $\langle (0::nat) = 2^0 - 1 \rangle$ 
      by auto
    show ?thesis
      by (subst 0, subst luby-sequence-core-two-power-minus-one) simp
  qed

  lemma  $\langle \text{luby-sequence-core } n \geq 1 \rangle$ 
  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  $\langle 2^{k-1} \leq \text{Suc } n \rangle$  and  $\langle \text{Suc } n < 2^k - 1 \rangle$  |
      (pow2)  $k$  where  $\langle \text{Suc } n = 2^k - \text{Suc } 0 \rangle$ 
    using exists-luby-decomp[of  $\langle \text{Suc } n \rangle$ ] 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  $n$ :  $\langle \text{Suc } n - 2^{k-1} + 1 < \text{Suc } n \rangle$ 
        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-learningW
  — 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 cdclW-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 stateW-adding-init-clause-no-state =
  stateW-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* ⇒ *'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

add-init-cls :: *'v clause* ⇒ *'st* ⇒ *'st*

assumes

add-init-cls:

state st = (*M*, *N*, *U*, *S'*) ⇒

state (*add-init-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* ⇒ *'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  $\Rightarrow$  'st and
  add-init-cls :: 'v clause  $\Rightarrow$  'st  $\Rightarrow$  'st +
assumes
  state-prop[simp]:
     $\langle$ state  $S = (\text{trail } S, \text{init-clss } S, \text{learned-clss } S, \text{conflicting } S, \text{additional-info } S)$  $\rangle$ 

locale stateW-adding-init-clause =
  stateW-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  $\Rightarrow$  'st  $\Rightarrow$  bool (infix  $\sim$  50) and
  state :: 'st  $\Rightarrow$  ('v, 'v clause) ann-lits  $\times$  'v clauses  $\times$  'v clauses  $\times$  'v clause option  $\times$ 
    'b and
  trail :: 'st  $\Rightarrow$  ('v, 'v clause) ann-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-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 and
  add-init-cls :: 'v clause  $\Rightarrow$  'st  $\Rightarrow$  'st
begin

sublocale stateW
  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) =  $\{\#C\# \} + \text{init-clss } st$ 
    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  $N$   $S$ ) =  $\{\#N\# \} + \text{init-clss } S + \text{learned-clss } S$ 
  unfolding clauses-def by auto

```

```

lemma reduce-trail-to-add-init-cl[simp]:
  trail (reduce-trail-to F (add-init-cl C S)) = trail (reduce-trail-to F S)
  by (rule trail-eq-reduce-trail-to-eq) auto

lemma conflicting-add-init-cl-iff-conflicting[simp]:
  conflicting (add-init-cl C S) = None  $\longleftrightarrow$  conflicting S = None
  by fastforce+
end

locale conflict-driven-clause-learning-with-adding-init-clauseW =
  stateW-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  $\Rightarrow$  'st  $\Rightarrow$  bool (infix  $\sim$  50) and
  state :: 'st  $\Rightarrow$  ('v, 'v clause) ann-lits  $\times$  'v clauses  $\times$  'v clauses  $\times$  'v clause option  $\times$ 
    'b and
  trail :: 'st  $\Rightarrow$  ('v, 'v clause) ann-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-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 and
  add-init-cls :: 'v clause  $\Rightarrow$  'st  $\Rightarrow$  'st
begin

sublocale conflict-driven-clause-learningW
  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 the 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  $\neg L \in \#$  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 \Rightarrow 'st \Rightarrow 'st **where**
 add-new-clause-and-update $C S =$
 (if trail $S \models_{as} C \text{Not } C$
 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:
 $\exists M. \text{ trail } S = M @ \text{ trail } (\text{cut-trail-wrt-clause } C (\text{trail } S) S)$

proof (induction trail S arbitrary: S rule: ann-lit-list-induct)
 case Nil
 then show ?case **by** simp
 next
 case (Decided $L M$) **note** $IH = \text{this}(1)[\text{of tl-trail } S]$ **and** $M = \text{this}(2)[\text{symmetric}]$
 then show ?case **using** Cons-eq-appendI **by** fastforce+
 next
 case (Propagated $L l M$) **note** $IH = \text{this}(1)[\text{of tl-trail } S]$ **and** $M = \text{this}(2)[\text{symmetric}]$
 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
 $M: \text{ trail } T = M @ \text{ trail } (\text{cut-trail-wrt-clause } C (\text{trail } T) T)$
 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 $T = \text{count-decided } (\text{trail } T)$
 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  $\models_{as}$  CNot C
  shows
    (trail ((cut-trail-wrt-clause C (trail T) T)))  $\models_{as}$  CNot C
  using assms
proof (induction trail T arbitrary: T rule: ann-lit-list-induct)
  case Nil
  then show ?case by simp
next
  case (Decided L M) note IH = this(1)[of 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  $\in$   $\{\{\#- l\# \mid l. l \in \# C\} \rightarrow M \models_a mma\} \rightarrow M \models_{as} \{\{\#- l\# \mid l. l \in \# C\}$ 
    using true-annots-def by blast
  have mma  $\in$   $\{\{\#- l\# \mid l. l \in \# C\} \rightarrow \text{trail } T \models_a mma$ 
    using CNot-def M bt by (metis (no-types) true-annots-def)
  then have M  $\models_{as} \{\{\#- l\# \mid l. l \in \# C\}$ 
    using f6 True M bt by (force 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  $\in$   $\{\{\#- l\# \mid l. l \in \# C\} \rightarrow M \models_a mma\} \rightarrow M \models_{as} \{\{\#- l\# \mid l. l \in \# C\}$ 
    using true-annots-def by blast
  have mma  $\in$   $\{\{\#- l\# \mid l. l \in \# C\} \rightarrow \text{trail } T \models_a mma$ 
    using CNot-def M bt by (metis (no-types) true-annots-def)
  then have M  $\models_{as} \{\{\#- l\# \mid l. l \in \# C\}$ 
    using f6 True M bt by (force 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:
  (( $\forall L \in \# C. -L \notin \text{lits-of-l (trail T)}$ )  $\wedge$  trail (cut-trail-wrt-clause C (trail T) T) = [])
   $\vee$  ( $-\text{lit-of (hd (trail (cut-trail-wrt-clause C (trail T) T)))} \in \# C$ 
     $\wedge$  length (trail (cut-trail-wrt-clause C (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 \Rightarrow 'st \Rightarrow bool **for** *S* **where**

add-conflict:

```

trail S  $\models_{asm}$  init-clss S  $\implies$  distinct-mset C  $\implies$  conflicting S = None  $\implies$ 
trail S  $\models_{as}$  CNot C  $\implies$ 
full cdclW-stgy
  (update-conflicting (Some C)
    (add-init-cls C (cut-trail-wrt-clause C (trail S) S))) T  $\implies$ 
incremental-cdclW S T |

```

add-no-conflict:

```

trail S  $\models_{asm}$  init-clss S  $\implies$  distinct-mset C  $\implies$  conflicting S = None  $\implies$ 
 $\neg$  trail S  $\models_{as}$  CNot C  $\implies$ 
full cdclW-stgy (add-init-cls C S) T  $\implies$ 
incremental-cdclW S T

```

lemma cdcl_W-all-struct-inv-add-new-clause-and-update-cdcl_W-all-struct-inv:

assumes

```

  inv-T: cdclW-all-struct-inv T and
  tr-T-N[simp]: trail T  $\models_{asm}$  N and
  tr-C[simp]: trail T  $\models_{as}$  CNot C and
  [simp]: distinct-mset C

```

shows cdcl_W-all-struct-inv (add-new-clause-and-update C T) (is cdcl_W-all-struct-inv ?T')

proof –

```

let ?T = update-conflicting (Some C)
  (add-init-cls C (cut-trail-wrt-clause C (trail T) T))

```

obtain *M* **where**

```

M: trail T = M @ trail (cut-trail-wrt-clause C (trail T) T)

```

```

  using trail-cut-trail-wrt-clause[of T C] by blast

```

```

have H[dest]:  $\bigwedge x. x \in \text{lits-of-l (trail (cut-trail-wrt-clause C (trail T) T))} \implies$ 
  x  $\in$  lits-of-l (trail T)

```

```

  using inv-T arg-cong[OF M, of lits-of-l] by auto

```

```

have H'[dest]:  $\bigwedge x. x \in \text{set (trail (cut-trail-wrt-clause C (trail T) T))} \implies$ 
  x  $\in$  set (trail T)

```

```

  using inv-T arg-cong[OF M, of set] by auto

```

```

have H-proped:  $\bigwedge x. x \in \text{set (get-all-mark-of-propagated (trail (cut-trail-wrt-clause C$ 

```



```

\implies x \in \text{set } (\text{get-all-mark-of-propagated } (\text{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 cdclW-all-struct-inv-def no-strange-atm-def add-new-clause-and-update-def
  cdclW-M-level-inv-def by (auto 20 1)
have M-lev: cdclW-M-level-inv T
  using inv-T unfolding cdclW-all-struct-inv-def by blast
then have no-dup (M @ trail (cut-trail-wrt-clause C (trail T) T))
  unfolding cdclW-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 cdclW-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]: cdclW-M-level-inv ?T
  using M-lev unfolding cdclW-M-level-inv-def
  by (auto simp: M-lev cdclW-M-level-inv-def cut-trail-wrt-clause-backtrack-lvl-length-decided)

have [simp]:  $\bigwedge s. s \in \# \text{ learned-clss } T \implies \neg \text{tautology } s$ 
  using inv-T unfolding cdclW-all-struct-inv-def by auto

have distinct-cdclW-state T
  using inv-T unfolding cdclW-all-struct-inv-def by auto
then have [simp]: distinct-cdclW-state ?T
  unfolding distinct-cdclW-state-def by auto

have cdclW-conflicting T
  using inv-T unfolding cdclW-all-struct-inv-def by auto
have trail ?T  $\models_{as}$  CNot C
  by (simp add: cut-trail-wrt-clause-CNot-trail)
then have [simp]: cdclW-conflicting ?T
  unfolding cdclW-conflicting-def apply simp
  by (metis M  $\langle$ cdclW-conflicting T $\rangle$  append-assoc cdclW-conflicting-decomp(2))

have
  decomp-T: all-decomposition-implies-m (clauses T) (get-all-ann-decomposition (trail T))
  using inv-T unfolding cdclW-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  $(a, b) \in \text{set } (\text{get-all-ann-decomposition } (\text{trail } ?T))$ 
  from in-get-all-ann-decomposition-in-get-all-ann-decomposition-prepend[OF this, of M]
  obtain b' where
     $(a, b' @ b) \in \text{set } (\text{get-all-ann-decomposition } (\text{trail } T))$ 
    using M by auto
  then have unmark-l a  $\cup$  set-mset (clauses T)  $\models_{ps}$  unmark-l (b' @ b)
    using decomp-T unfolding all-decomposition-implies-def by fastforce
  then have unmark-l a  $\cup$  set-mset (clauses ?T)  $\models_{ps}$  unmark-l (b' @ b)
    by (simp add: clauses-def)
  then show unmark-l a  $\cup$  set-mset (clauses ?T)  $\models_{ps}$  unmark-l b

```

```

    by (auto simp: image-Un)
qed

have [simp]: cdclW-learned-clause ?T
  using inv-T unfolding cdclW-all-struct-inv-def cdclW-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 cdclW-all-struct-inv-def by (auto simp: add-new-clause-and-update-def)
qed

lemma cdclW-all-struct-inv-add-new-clause-and-update-cdclW-stgy-inv:
  assumes
    inv-s: cdclW-stgy-invariant T and
    inv: cdclW-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 cdclW-stgy-invariant (add-new-clause-and-update C T)
    (is cdclW-stgy-invariant ?T')
proof -
  have cdclW-all-struct-inv ?T'
    using cdclW-all-struct-inv-add-new-clause-and-update-cdclW-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 cdclW-M-level-inv-decomp(2) cdclW-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) ⊨as CNot C
    by (simp add: add-new-clause-and-update-def cut-trail-wrt-clause-CNot-trail
      cdclW-M-level-inv-def cdclW-all-struct-inv-def)
  obtain MT where
    MT: trail T = MT @ trail (cut-trail-wrt-clause C (trail T) T)
    using trail-cut-trail-wrt-clause by blast
  consider
    (false) ∀ L ∈ #C. - L ∉ lits-of-l (trail T) and
    trail (cut-trail-wrt-clause C (trail T) T) = [] |
    (not-false)
    - lit-of (hd (trail (cut-trail-wrt-clause C (trail T) T))) ∈ # C and
    1 ≤ length (trail (cut-trail-wrt-clause C (trail T) T))
  using cut-trail-wrt-clause-hd-trail-in-or-empty-trail[of C T] by auto
  then show ?thesis
  proof cases
    case false note C = this(1) and empty-tr = this(2)
    then have [simp]: C = {#}
      by (simp add: in-CNot-implies-uminus(2) multiset-eqI)
    show ?thesis
      using empty-tr unfolding cdclW-stgy-invariant-def no-smaller-confl-def
      cdclW-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 <cdclW-all-struct-inv ?T'> unfolding cdclW-all-struct-inv-def cdclW-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 cdclW-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 cdclW-all-struct-inv-def
          cdclW-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 <cdclW-all-struct-inv ?T'> unfolding cdclW-all-struct-inv-def
        cdclW-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 cdclW-stgy-invariant-def cdclW-all-struct-inv-def
  by (auto simp: add-new-clause-and-update-def get-level-def count-decided-def intro: rev-beI)
qed
qed

```

lemma *incremental-cdcl_W-inv:*

```

assumes
  inc: incremental-cdclW S T and
  inv: cdclW-all-struct-inv S and
  s-inv: cdclW-stgy-invariant S and
  learned-entailed: ⟨cdclW-learned-clauses-entailed-by-init S⟩
shows
  cdclW-all-struct-inv T and
  cdclW-stgy-invariant T and
  learned-entailed: ⟨cdclW-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': cdclW-all-struct-inv ?T and inv-s-T: cdclW-stgy-invariant ?T
  using add-confl.hyps(1,2,4) add-new-clause-and-update-def
  cdclW-all-struct-inv-add-new-clause-and-update-cdclW-all-struct-inv inv apply auto[1]
  using add-confl.hyps(1,2,4) add-new-clause-and-update-def
  cdclW-all-struct-inv-add-new-clause-and-update-cdclW-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
    cdclW-all-struct-inv-add-new-clause-and-update-cdclW-all-struct-inv
    rtranclp-cdclW-all-struct-inv-inv rtranclp-cdclW-stgy-rtranclp-cdclW-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
    cdclW-all-struct-inv-add-new-clause-and-update-cdclW-all-struct-inv full-def inv
    rtranclp-cdclW-stgy-cdclW-stgy-invariant)

case 3 show ?case
  using learned-entailed rtranclp-cdclW-learned-clauses-entailed[of ?T T] add-confl inv'
  unfolding cdclW-all-struct-inv-def full-def
  by (auto simp: cdclW-learned-clauses-entailed-by-init-def
    dest!: rtranclp-cdclW-stgy-rtranclp-cdclW-restart)
next
case (add-no-confl C T)
have inv': cdclW-all-struct-inv (add-init-cls C S)
  using inv ⟨distinct-mset C⟩ unfolding cdclW-all-struct-inv-def no-strange-atm-def
  cdclW-M-level-inv-def distinct-cdclW-state-def cdclW-conflicting-def cdclW-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-cdclW-stgy-cdclW-all-struct-inv)

case 2
have nc: ∀ M. (∃ K i M'. trail S = M' @ Decided K # M) ⟶ ¬ M ⊨as CNot C
  using ⟨¬ trail S ⊨as CNot C⟩
  by (auto simp: true-annots-true-cls-def-iff-negation-in-model)

have cdclW-stgy-invariant (add-init-cls C S)
  using s-inv ⟨¬ trail S ⊨as CNot C⟩ inv unfolding cdclW-stgy-invariant-def
  no-smaller-confl-def eq-commute[of - trail -] cdclW-M-level-inv-def cdclW-all-struct-inv-def
  by (auto simp: clauses-def nc)
then show ?case
  by (metis ⟨cdclW-all-struct-inv (add-init-cls C S)⟩ add-no-confl.hyps(5) full-def

```

```

    rtrancpl-cdclW-stgy-cdclW-stgy-invariant)

case 3
have ⟨cdclW-learned-clauses-entailed-by-init (add-init-cls C S)⟩
  using learned-entailed by (auto simp: cdclW-learned-clauses-entailed-by-init-def)
then show ?case
  using add-no-confl(5) learned-entailed rtrancpl-cdclW-learned-clauses-entailed[of - T] add-confl inv'
  unfolding cdclW-all-struct-inv-def full-def
  by (auto simp: cdclW-learned-clauses-entailed-by-init-def
    dest!: rtrancpl-cdclW-stgy-rtrancpl-cdclW-restart)
qed

lemma rtrancpl-incremental-cdclW-inv:
  assumes
    inc: incremental-cdclW** S T and
    inv: cdclW-all-struct-inv S and
    s-inv: cdclW-stgy-invariant S and
    learned-entailed: ⟨cdclW-learned-clauses-entailed-by-init S⟩
  shows
    cdclW-all-struct-inv T and
    cdclW-stgy-invariant T and
    ⟨cdclW-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-cdclW-inv by blast+

lemma incremental-conclusive-state:
  assumes
    inc: incremental-cdclW S T and
    inv: cdclW-all-struct-inv S and
    s-inv: cdclW-stgy-invariant S and
    learned-entailed: ⟨cdclW-learned-clauses-entailed-by-init S⟩
  shows conflicting T = Some {#} ∧ unsatisfiable (set-mset (init-cls T))
    ∨ conflicting T = None ∧ trail T ⊨asm init-cls T ∧ satisfiable (set-mset (init-cls 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 cdclW-stgy T T
    using full unfolding full-def by auto
  then show ?case
    using C conf dist full incremental-cdclW.add-confl incremental-cdclW-inv
    incremental-cdclW-inv inv learned-entailed
    ⟨full cdclW-stgy T T⟩ full-cdclW-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 cdclW-stgy T T
    using full unfolding full-def by auto
  then show ?case
    using ⟨full cdclW-stgy T T⟩ full-cdclW-stgy-inv-normal-form C conf dist full

```

```

    incremental-cdclW.add-no-confl incremental-cdclW-inv inv learned-entailed
    s-inv tr by blast
qed

lemma tranclp-incremental-correct:
  assumes
    inc: incremental-cdclW++ S T and
    inv: cdclW-all-struct-inv S and
    s-inv: cdclW-stgy-invariant S and
    learned-entailed: ⟨cdclW-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-cdclW-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 and simply iterate over-and-over on lists.

4.1.1 Common Rules

Propagation

The following theorem holds:

lemma *lits-of-l-unfold*:

$$(\forall c \in \text{set } C. -c \in \text{lits-of-l } Ms) \longleftrightarrow Ms \models_{as} CNot (mset C)$$

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 ($\lambda a. \text{atm-of } a \notin \text{atm-of ' lits-of-l } M$) l of
 a # [] \Rightarrow if M \models_{as} CNot (mset l - {#a#}) then Some a else None
 | - \Rightarrow None)

definition *is-unit-clause-code* :: 'a literal list \Rightarrow ('a, 'b) ann-lits

\Rightarrow 'a literal option **where**

is-unit-clause-code l M =

(case List.filter ($\lambda a. \text{atm-of } a \notin \text{atm-of ' lits-of-l } M$) l of
 a # [] \Rightarrow if ($\forall c \in \text{set } (\text{remove1 } a \text{ l}). -c \in \text{lits-of-l } M$) then Some a else None
 | - \Rightarrow None)

lemma *is-unit-clause-is-unit-clause-code*[code]:
is-unit-clause l $M = \text{is-unit-clause-code } l$ M
proof –
 have 1: $\bigwedge a. (\forall c \in \text{set } (\text{remove1 } a \ l). - c \in \text{lits-of-}l \ M) \longleftrightarrow M \models_{\text{as}} \text{CNot } (\text{mset } l - \{\#a\# \})$
 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 $M = \text{Some } a$
 shows *undefined-lit* M a
proof –
 have (case [$a \leftarrow l$. *atm-of* $a \notin \text{atm-of ' lits-of-}l \ M$] of [] \Rightarrow *None*
 | [a] \Rightarrow if $M \models_{\text{as}} \text{CNot } (\text{mset } l - \{\#a\# \})$ then *Some* a else *None*
 | $a \# ab \# xa \Rightarrow \text{Map.empty } xa) = \text{Some } a$
 using *assms* **unfolding** *is-unit-clause-def* .
 then have $a \in \text{set } [a \leftarrow l . \text{atm-of } a \notin \text{atm-of ' lits-of-}l \ M]$
 apply (cases [$a \leftarrow l$. *atm-of* $a \notin \text{atm-of ' lits-of-}l \ M$])
 apply *simp*
 apply (rename-tac *aa list*; case-tac *list*) **by** (auto split: if-split-asm)
 then have *atm-of* $a \notin \text{atm-of ' lits-of-}l \ M$ **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 $M = \text{Some } a \implies M \models_{\text{as}} \text{CNot } (\text{mset } l - \{\#a\# \})$
unfolding *is-unit-clause-def*
proof –
 assume (case [$a \leftarrow l$. *atm-of* $a \notin \text{atm-of ' lits-of-}l \ M$] of [] \Rightarrow *None*
 | [a] \Rightarrow if $M \models_{\text{as}} \text{CNot } (\text{mset } l - \{\#a\# \})$ then *Some* a else *None*
 | $a \# ab \# xa \Rightarrow \text{Map.empty } xa) = \text{Some } a$
 then show ?thesis
 apply (cases [$a \leftarrow l$. *atm-of* $a \notin \text{atm-of ' lits-of-}l \ M$], *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 $M = \text{Some } a \implies a \in \text{set } l$
unfolding *is-unit-clause-def*
proof –
 assume (case [$a \leftarrow l$. *atm-of* $a \notin \text{atm-of ' lits-of-}l \ M$] of [] \Rightarrow *None*
 | [a] \Rightarrow if $M \models_{\text{as}} \text{CNot } (\text{mset } l - \{\#a\# \})$ then *Some* a else *None*
 | $a \# ab \# xa \Rightarrow \text{Map.empty } xa) = \text{Some } a$
 then show $a \in \text{set } l$
 by (cases [$a \leftarrow l$. *atm-of* $a \notin \text{atm-of ' lits-of-}l \ M$])
 (fastforce dest: *filter-eq-ConsD* split: if-split-asm split: list.splits)+
qed

lemma *is-unit-clause-Nil*[*simp*]: *is-unit-clause* [] $M = \text{None}$
unfolding *is-unit-clause-def* **by** *auto*

Unit propagation for all clauses

Finding the first clause to propagate

fun *find-first-unit-clause* :: 'a literal list list \Rightarrow ('a, 'b) ann-lits

\Rightarrow ('a literal \times 'a literal list) option **where**

find-first-unit-clause (a # l) M =

(case *is-unit-clause* a M of

None \Rightarrow *find-first-unit-clause* l M

| Some L \Rightarrow Some (L, a) |

find-first-unit-clause [] - = None

lemma *find-first-unit-clause-some*:

find-first-unit-clause l M = Some (a, c)

$\implies c \in \text{set } l \wedge M \models_{\text{as}} \text{CNot } (\text{mset } c - \{\#a\# \}) \wedge \text{undefined-lit } M a \wedge a \in \text{set } c$

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 $\models_{\text{as}} \text{CNot } (\text{mset } c - \{\#a\# \})$ **and**

undef: *undefined-lit* M a **and**

ac: a $\in \text{set } c$

shows *is-unit-clause* c M \neq None

proof -

have [a \leftarrow c . atm-of a \notin 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: $\text{mset } c + \{\#ac\# \} - \{\#a\# \} = \text{mset } c - \{\#a\# \} + \{\#ac\# \}$

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 $\in \text{set } l \implies M \models_{\text{as}} \text{CNot } (\text{mset } c - \{\#a\# \}) \implies \text{undefined-lit } M a \implies a \in \text{set } c$

$\implies \text{find-first-unit-clause } l M \neq \text{None}$

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 \Rightarrow 'a literal set \Rightarrow 'a literal option **where**
find-first-unused-var (a # l) M =
 (case List.find ($\lambda lit. lit \notin M \wedge -lit \notin M$) a of
 None \Rightarrow *find-first-unused-var* l M
 | Some a \Rightarrow Some a) |
find-first-unused-var [] - = None

lemma *find-none[iff]*:
 List.find ($\lambda lit. lit \notin M \wedge -lit \notin M$) a = None \longleftrightarrow atm-of ' set a \subseteq atm-of ' M
apply (induct a)
using atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set
by (force simp add: atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set)+

lemma *find-some*: List.find ($\lambda lit. lit \notin M \wedge -lit \notin M$) a = Some b $\implies b \in \text{set } a \wedge b \notin M \wedge -b \notin M$
unfolding *find-Some-iff* **by** (metis nth-mem)

lemma *find-first-unused-var-None[iff]*:
find-first-unused-var l M = None $\longleftrightarrow (\forall a \in \text{set } l. \text{atm-of ' set } a \subseteq \text{atm-of ' } M)$
by (induct l)
 (*auto split: option.splits dest!: find-some*
 simp add: image-subset-iff atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set)

lemma *find-first-unused-var-Some-not-all-incl*:
assumes *find-first-unused-var* l M = Some c
shows $\neg(\forall a \in \text{set } l. \text{atm-of ' set } a \subseteq \text{atm-of ' } M)$
proof –
have *find-first-unused-var* l M \neq None
using *assms* **by** (cases *find-first-unused-var* l M) auto
then show $\neg(\forall a \in \text{set } l. \text{atm-of ' set } a \subseteq \text{atm-of ' } M)$ **by** auto
qed

lemma *find-first-unused-var-Some*:
find-first-unused-var l M = Some a $\implies (\exists m \in \text{set } l. a \in \text{set } m \wedge a \notin M \wedge -a \notin M)$
by (induct l) (*auto split: option.splits dest: find-some*)

lemma *find-first-unused-var-undefined*:
find-first-unused-var l (lits-of-l Ms) = Some a $\implies \text{undefined-lit } Ms a$
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 \Rightarrow ('a, 'b) ann-lits \Rightarrow nat
where
maximum-level-code [] - = 0 |
maximum-level-code (L # Ls) M = max (get-level M L) (*maximum-level-code* Ls M)

lemma *maximum-level-code-eq-get-maximum-level[simp]*:
maximum-level-code D M = get-maximum-level M (mset D)
by (induction D) (*auto simp add: get-maximum-level-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 & j < i then Some (L, j) else find-level-decomp M Ls (L#D) k
    )

```

```

lemma find-level-decomp-some:
  assumes find-level-decomp M Ls D k = Some (L, j)
  shows L ∈ set Ls ∧ get-maximum-level M (mset (remove1 L (Ls @ D))) = j ∧ get-level M L = k
  using assms

```

```

proof (induction Ls arbitrary: D)

```

```

  case Nil
  then show ?case by simp

```

```

next

```

```

  case (Cons L' Ls) note IH = this(1) and H = this(2)

```

```

  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 ∈ set Ls ∧ get-maximum-level M (mset Ls + mset D - {#L#}) = j ∧ get-level M L = k

```

```

  using IH by simp

```

```

  have a2: find = Some (L, j)

```

```

  using H unfolding find-def by (auto split: if-split-asm)

```

```

  { assume Some (L', get-maximum-level M (mset D + mset Ls)) ≠ find

```

```

    then have f3: L ∈ set Ls and get-maximum-level M (mset Ls + mset (L' # D) - {#L#}) = j

```

```

      using a1 IH a2 unfolding find-def by meson+

```

```

    moreover then have mset Ls + mset D - {#L#} + {#L'#} = {#L'#} + mset D + (mset Ls
- {#L#})

```

```

      by (auto simp: ac-simps multiset-eq-iff Suc-leI)

```

```

    ultimately have f4: get-maximum-level M (mset Ls + mset D - {#L#} + {#L'#}) = j

```

```

      by 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 ≠ L' → L ∈ set Ls ∧ get-maximum-level M (mset Ls + mset D - {#L#} + {#L'#}) = j ∧
get-level M L = k

```

```

  using a2 a1[of L' # D] unfolding find-def

```

```

  apply (metis add.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  $M$   $Ls$   $E$   $k = \text{None}$  and  $\text{mset } (L \# D) = \text{mset } (Ls @ E)$ 
  shows  $\neg (L \in \text{set } Ls \wedge \text{get-maximum-level } M (\text{mset } D) < k \wedge k = \text{get-level } M L)$ 
  using assms
proof (induction  $Ls$  arbitrary:  $E$   $L$   $D$ )
  case Nil
  then show ?case by simp
next
  case (Cons  $L' Ls$ ) note  $IH = \text{this}(1)$  and  $\text{find-none} = \text{this}(2)$  and  $LD = \text{this}(3)$ 
  have  $\text{mset } D + \{\#L'\# \} = \text{mset } E + (\text{mset } Ls + \{\#L'\# \}) \implies \text{mset } D = \text{mset } E + \text{mset } Ls$ 
  by (metis add-right-imp-eq union-assoc)
  then show ?case
  using find-none  $IH[\text{of } L' \# E L D]$   $LD$  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  $i$   $M = \text{Some } M'$ 
  shows  $\exists K M2 M1. M = M2 @ M' \wedge M' = \text{Decided } K \# M1 \wedge \text{get-level } M K = (i+1)$ 
  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  $M = M2 @ \text{Decided } K \# M'$  and  $\text{get-level } M K = (i+1)$ 
  shows bt-cut  $i$   $M \neq \text{None}$ 
  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:
   $\exists N. (\text{Decided } K \# M', N) \in \text{set } (\text{get-all-ann-decomposition } (M2 @ \text{Decided } K \# M'))$ 
  apply (induction  $M2$  rule: ann-lit-list-induct)
  apply auto[2]
  by (rename-tac  $L$   $m$   $xs$ , case-tac get-all-ann-decomposition ( $xs @ \text{Decided } K \# M'$ ))
  auto

lemma bt-cut-in-get-all-ann-decomposition:
  assumes no-dup  $M$  and bt-cut  $i$   $M = \text{Some } M'$ 
  shows  $\exists M2. (M', M2) \in \text{set } (\text{get-all-ann-decomposition } M)$ 
  using bt-cut-some-decomp[OF assms] by (auto simp add: get-all-ann-decomposition-ex)

fun do-backtrack-step where
do-backtrack-step ( $M, N, U, \text{Some } D$ ) =
  (case find-level-decomp  $M$   $D$  [] (count-decided  $M$ ) of
    None  $\Rightarrow (M, N, U, \text{Some } D)$ 
  | Some ( $L, j$ )  $\Rightarrow$ 
    (case bt-cut  $j$   $M$  of
      Some (Decided -  $\# Ls$ )  $\Rightarrow$  (Propagated  $L$   $D \# Ls, N, D \# U, \text{None}$ )
    | -  $\Rightarrow (M, N, U, \text{Some } D)$ 
    ) |
  )
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-step* :: *int dpll_W-ann-lits* × *int literal list list*
 \Rightarrow *int dpll_W-ann-lits* × *int literal list list* **where**
DPLL-step = ($\lambda(Ms, N)$.
 (case *find-first-unit-clause* *N* *Ms* of
 Some (*L*, -) \Rightarrow (*Propagated* *L* ()) # *Ms*, *N*)
 | - \Rightarrow
 if $\exists C \in \text{set } N. (\forall c \in \text{set } C. -c \in \text{lits-of-l } Ms)$
 then
 (case *backtrack-split* *Ms* of
 (-, *L* # *M*) \Rightarrow (*Propagated* (- (*lit-of* *L*)) ()) # *M*, *N*)
 | (-, -) \Rightarrow (*Ms*, *N*)
)
 else
 (case *find-first-unused-var* *N* (*lits-of-l* *Ms*) of
 Some *a* \Rightarrow (*Decided* *a* # *Ms*, *N*)
 | None \Rightarrow (*Ms*, *N*))))

Example of propagation:

value *DPLL-step* ([*Decided* (*Neg* 1)], [[*Pos* (1::int), *Neg* 2]])

We define the conversion function between the states as defined in *Prop-DPLL* (with multisets) and here (with lists).

abbreviation *toS* $\equiv \lambda(Ms::(\text{int}, \text{unit}) \text{ ann-lits})$
 $(N:: \text{int literal list list}). (Ms, \text{mset } (\text{map mset } N))$

abbreviation *toS'* $\equiv \lambda(Ms::(\text{int}, \text{unit}) \text{ ann-lits},$
 $N:: \text{int literal list list}). (Ms, \text{mset } (\text{map mset } N))$

Proof of correctness of *DPLL-step*

lemma *DPLL-step-is-a-dpll_W-step*:

assumes *step*: (*Ms'*, *N'*) = *DPLL-step* (*Ms*, *N*)

and *neq*: (*Ms*, *N*) \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-first-unit-clause* *N* *Ms* = *Some* (*L*, *E*)

then have *Ms'N*: (*Ms'*, *N'*) = (*Propagated* *L* ()) # *Ms*, *N*)

using *step* **unfolding** *DPLL-step-def* **by** *auto*

obtain *C* **where**

C: *C* $\in \text{set } N$ **and**

Ms: *Ms* $\models_{\text{as}} C \text{Not } (\text{mset } C - \{\#L\# \})$ **and**

undef: *undefined-lit* *Ms* *L* **and**

L $\in \text{set } C$ **using** *find-first-unit-clause-some*[*OF unit*] **by** *metis*

have *dpll_W* (*Ms*, *mset* (*map mset* *N*))

(*Propagated* *L* ()) # *fst* (*Ms*, *mset* (*map mset* *N*)), *snd* (*Ms*, *mset* (*map mset* *N*)))

apply (*rule dpll_W.propagate*)

using *Ms undef C* (*L* $\in \text{set } C$) **by** (*auto simp add: C*)

then have ?*thesis* **using** *Ms'N* **by** *auto*

```

}
moreover
{ assume unit: find-first-unit-clause N Ms = None
  assume exC:  $\exists C \in \text{set } N. Ms \models_{\text{as}} C\text{Not } (\text{mset } C)$ 
  then obtain C where C:  $C \in \text{set } N$  and Ms:  $Ms \models_{\text{as}} C\text{Not } (\text{mset } C)$  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: dpllW (Ms, mset (map mset N))
    (Propagated ( $\neg$  lit-of L) () # M, snd (Ms, mset (map mset N)))
    apply (rule dpllW.backtrack[OF - (is-decided L), of ])
    using C Ms bt by auto
  moreover have (Ms', N') = (Propagated ( $\neg$  (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:  $\neg (\exists C \in \text{set } N. Ms \models_{\text{as}} C\text{Not } (\text{mset } C))$ 
  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 dpllW (Ms, mset (map mset N))
    (Decided L # fst (Ms, mset (map mset N)), snd (Ms, mset (map mset N)))
    apply (rule dpllW.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: $(Ms, N) = \text{DPLL-step } (Ms, N)$
 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: $\exists C \in \text{set } N. Ms \models_{\text{as}} C\text{Not } (\text{mset } C)$
 then have Ms: $(Ms, N) = (\text{case backtrack-split Ms of } (x, []) \Rightarrow (Ms, N) \mid (x, L \# M) \Rightarrow (\text{Propagated } (\neg \text{lit-of } L) () \# M, N))$
 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-dpllW-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
      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-dpllW-state-def by blast
}
ultimately show ?thesis by blast
qed

```

Adding invariants

Invariant tested in the function `function DPLL-ci :: int dpllW-ann-lits ⇒ int literal list list ⇒ int dpllW-ann-lits × int literal list list where`

`DPLL-ci Ms N =`

`(if ¬dpllW-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). dpllW-all-inv S ∧ dpllW S S'}})`

`show wf {(S', S).(toS' S', toS' S) ∈ {(S', S). dpllW-all-inv S ∧ dpllW S S'}}`

`using wf-if-measure-f[OF wf-dpllW, of toS'] by auto`

next

`fix Ms :: int dpllW-ann-lits and N x xa y`

`assume ¬ ¬ dpllW-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). dpllW-all-inv S ∧ dpllW 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-implicS-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.premis *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.premis *DPLL-part.psimps* *Ms'*

\langle *DPLL-ci* *Ms' N* = *DPLL-part* *Ms' N* \wedge *DPLL-part-dom* (*Ms'*, *N*) \rangle \langle *DPLL-part-dom* (*Ms*, *N*) \rangle 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 \neg *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 dpllW-all-inv (toS Ms N)
    and (S1, S2) ≠ (Ms, N)
    moreover obtain S1' S2' where DPLL-ci S1 N = (S1', S2') by (cases DPLL-ci S1 N) auto
    moreover have DPLL-ci Ms N = DPLL-ci S1 N using DPLL-ci.simps[of Ms N] calculation
    proof -
      have (case (S1, S2) 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 (S1, S2) = (Ms, N) then (Ms, N) else DPLL-ci S1 N) = DPLL-ci Ms N
        by fastforce
      then show ?thesis
        using calculation(2) by presburger
    qed
    ultimately have dpllW** (toS S1' N) (toS Ms' N) using IH[of (S1, S2) S1 S2] S step by simp

    moreover have dpllW (toS Ms N) (toS S1 N)
      by (metis DPLL-step-is-a-dpllW-step S ⟨(S1, S2) ≠ (Ms, N)⟩ prod.sel(2) snd-DPLL-step)
    ultimately have ?case by (metis (mono-tags, hide-lams) IH S ⟨(S1, S2) ≠ (Ms, N)⟩
      ⟨DPLL-ci Ms N = DPLL-ci S1 N⟩ ⟨dpllW-all-inv (toS Ms N)⟩ converse-rtranclp-into-rtranclp
      local.step)
  }
  ultimately show ?case by blast
qed

lemma dpllW-all-inv-dpllW-tranclp-irrefl:
  assumes dpllW-all-inv (Ms, N)
  and dpllW++ (Ms, N) (Ms, N)
  shows False
proof -
  have 1: wf {(S', S). dpllW-all-inv S ∧ dpllW++ S S'} using wf-dpllW-tranclp by auto
  have ((Ms, N), (Ms, N)) ∈ {(S', S). dpllW-all-inv S ∧ dpllW++ 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: dpllW-all-inv (toS Ms N)
  shows conclusive-dpllW-state (toS Ms N)
proof -
  have st: dpllW** (toS Ms N) (toS Ms N) using DPLL-ci-dpllW-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 dpllW++ (toS Ms N) (toS Ms N)
      by (metis DPLL-ci-dpllW-rtranclp DPLL-step-is-a-dpllW-step Ms'N ⟨DPLL-step (Ms, N) ≠ (Ms,
N)⟩
        prod.sel(2) rtranclp-into-tranclp2 snd-DPLL-step)
    then show False using dpllW-all-inv-dpllW-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** $(Ms', N) = DPLL\text{-}step\ (Ms, N)$
 unfolding *DPLL-step-def* **by** (*metis* (*no-types*, *lifting*) *DPLL-step-def prod.collapse snd-DPLL-step*)

lemma *DPLL-ci-obtains*:

obtains Ms' **where** $(Ms', N) = DPLL\text{-}ci\ Ms\ N$

proof (*induct rule: DPLL-ci.induct*)

case $(1\ Ms\ N)$ **note** $IH = this(1)$ **and** $that = this(2)$

obtain S **where** $SN: (S, N) = DPLL\text{-}step\ (Ms, N)$ **using** *DPLL-step-obtains* **by** *metis*

{ **assume** $\neg dpll_W\text{-}all\text{-}inv\ (toS\ Ms\ N)$

then have $?case$ **using** *that* **by** *auto*

}

moreover {

assume $n: (S, N) \neq (Ms, N)$

and $inv: dpll_W\text{-}all\text{-}inv\ (toS\ Ms\ N)$

have $\exists ms. DPLL\text{-}step\ (Ms, N) = (ms, N)$

by (*metis* $\langle \bigwedge thesisa. (\bigwedge S. (S, N) = DPLL\text{-}step\ (Ms, N) \implies thesisa) \implies thesisa \rangle$)

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*: $DPLL\text{-}ci\ Ms\ N = (Ms', N')$

shows $DPLL\text{-}ci\ Ms'\ N' = (Ms', N')$

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** $S: (S_1, N) = DPLL\text{-}step\ (Ms, N)$ **using** *DPLL-step-obtains* **by** *auto*

{ **assume** $\neg dpll_W\text{-}all\text{-}inv\ (toS\ Ms\ N)$

then have $?case$ **using** *step* **by** *auto*

}

moreover {

assume $dpll_W\text{-}all\text{-}inv\ (toS\ Ms\ N)$

and $(S_1, N) = (Ms, N)$

then have $?case$ **using** *S step* **by** *auto*

}

moreover

{ **assume** $inv: dpll_W\text{-}all\text{-}inv\ (toS\ Ms\ N)$

assume $n: (S_1, N) \neq (Ms, N)$

obtain S_1' **where** $SS: (S_1', N) = DPLL\text{-}ci\ S_1\ N$ **using** *DPLL-ci-obtains* **by** *blast*

moreover have $DPLL\text{-}ci\ Ms\ N = DPLL\text{-}ci\ S_1\ N$

proof –

have $(case\ (S_1, N)\ of\ (ms, lss) \Rightarrow if\ (ms, lss) = (Ms, N)\ then\ (Ms, N)\ else\ DPLL\text{-}ci\ ms\ N)$
 $= DPLL\text{-}ci\ Ms\ N$

using *S DPLL-ci.simps[of Ms N] calculation inv* **by** *presburger*

then have $(if\ (S_1, N) = (Ms, N)\ then\ (Ms, N)\ else\ DPLL\text{-}ci\ S_1\ N) = DPLL\text{-}ci\ Ms\ N$

by *fastforce*

```

    then show ?thesis
    using calculation n by presburger
  qed
moreover
  have  $DPLL\text{-}ci\ S_1' N = (S_1', N)$  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-dpllW-all-inv-final:
  fixes M Ms': (int, unit) ann-lits and
    N :: int literal list list
  assumes inv: dpllW-all-inv (Ms, mset (map mset N))
  and MsN: DPLL-part Ms N = (Ms', N)
  shows conclusive-dpllW-state (toS Ms' N) ∧ dpllW** (toS Ms N) (toS Ms' N)
proof -
  have 2: DPLL-ci Ms N = DPLL-part Ms N using inv dpllW-all-inv-implieS-2-eq3-and-dom by blast
  then have star: dpllW** (toS Ms N) (toS Ms' N) unfolding MsN using DPLL-ci-dpllW-rtrancp
  by blast
  then have inv': dpllW-all-inv (toS Ms' N) using inv rtrancp-dpllW-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 dpllW-state =
  {(M::(int, unit) ann-lits, N::int literal list list).
    dpllW-all-inv (toS M N)}
morphisms rough-state-of state-of
proof
  show ([],[]) ∈ {(M, N). dpllW-all-inv (toS M N)} by (auto simp add: dpllW-all-inv-def)
qed

```

```

lemma
  DPLL-part-dom ([], N)
  using dpllW-all-inv-implieS-2-eq3-and-dom[of [] N] by (simp add: dpllW-all-inv-def)

```

```

Some type classes instantiation dpllW-state :: equal
begin
definition equal-dpllW-state :: dpllW-state ⇒ dpllW-state ⇒ bool where
  equal-dpllW-state S S' = (rough-state-of S = rough-state-of S')
instance
  by standard (simp add: rough-state-of-inject equal-dpllW-state-def)
end

```

```

DPLL definition DPLL-step' :: dpllW-state ⇒ dpllW-state where
  DPLL-step' S = state-of (DPLL-step (rough-state-of S))

```

```

declare rough-state-of-inverse[simp]

```

```

lemma DPLL-step-dpllW-conc-inv:
  DPLL-step (rough-state-of S) ∈ {(M, N). dpllW-all-inv (toS M N)}

```

proof –

obtain $M\ N$ **where**
 S : $\langle \text{rough-state-of } S = (M, N) \rangle$
by $(\text{cases } \langle \text{rough-state-of } S \rangle)$
obtain $M'\ N'$ **where**
 S' : $\langle \text{DPLL-step } (\text{rough-state-of } S) = (M', N') \rangle$
by $(\text{cases } \langle \text{DPLL-step } (\text{rough-state-of } S) \rangle)$
have $\langle \text{dpll}_W^{**} \text{ (toS } M\ N) \text{ (toS } M'\ N') \rangle$
by $(\text{metis } \text{DPLL-step-is-a-dpll}_W\text{-step } S\ S'\ \text{fst-conv } r\text{-into-rtrancpl } r\text{trancpl.rtrancpl-refl } \text{snd-conv})$
then show $?thesis$
using $\text{rough-state-of}[of\ S]$ **unfolding** S' **unfolding** S **by** $(\text{auto intro: } r\text{trancpl-dpll}_W\text{-all-inv})$
qed

lemma $\text{rough-state-of-DPLL-step'-DPLL-step[simp]}$:
 $\text{rough-state-of } (\text{DPLL-step}'\ S) = \text{DPLL-step } (\text{rough-state-of } S)$
using $\text{DPLL-step-dpll}_W\text{-conc-inv } \text{DPLL-step'-def } \text{state-of-inverse}$ **by** auto

function $\text{DPLL-tot}:: \text{dpll}_W\text{-state} \Rightarrow \text{dpll}_W\text{-state}$ **where**

$\text{DPLL-tot } S =$
 $(\text{let } S' = \text{DPLL-step}'\ S \text{ in}$
 $\text{if } S' = S \text{ then } S \text{ else } \text{DPLL-tot } S')$
by fast+

termination

proof $(\text{relation } \{(T', T)\})$.

$(\text{rough-state-of } T', \text{rough-state-of } T)$
 $\in \{(S', S). (\text{toS}'\ S', \text{toS}'\ S)$
 $\in \{(S', S). \text{dpll}_W\text{-all-inv } S \wedge \text{dpll}_W\ S\ S'\}\}$

show $\text{wf } \{(b, a)\}$.

$(\text{rough-state-of } b, \text{rough-state-of } a)$
 $\in \{(b, a). (\text{toS}'\ b, \text{toS}'\ a)$
 $\in \{(b, a). \text{dpll}_W\text{-all-inv } a \wedge \text{dpll}_W\ a\ b\}\}$

using $\text{wf-if-measure-f}[OF\ \text{wf-if-measure-f}[OF\ \text{wf-dpll}_W, \text{of toS}'], \text{of rough-state-of}]$.

next

fix $S\ x$

assume $x: x = \text{DPLL-step}'\ S$

and $x \neq S$

have $\text{dpll}_W\text{-all-inv } (\text{case } \text{rough-state-of } S \text{ of } (Ms, N) \Rightarrow (Ms, \text{mset } (\text{map } \text{mset } N)))$

by $(\text{metis } (\text{no-types, lifting}) \text{ case-prodE mem-Collect-eq old.prod.case rough-state-of})$

moreover have $\text{dpll}_W\ (\text{case } \text{rough-state-of } S \text{ of } (Ms, N) \Rightarrow (Ms, \text{mset } (\text{map } \text{mset } N)))$

$(\text{case } \text{rough-state-of } (\text{DPLL-step}'\ S) \text{ of } (Ms, N) \Rightarrow (Ms, \text{mset } (\text{map } \text{mset } N)))$

proof –

obtain $Ms\ N$ **where** $Ms: (Ms, N) = \text{rough-state-of } S$ **by** $(\text{cases } \text{rough-state-of } S)$ **auto**

have $\text{dpll}_W\text{-all-inv } (\text{toS}'\ (Ms, N))$ **using** calculation **unfolding** Ms **by** blast

moreover obtain $Ms'\ N'$ **where** $Ms': (Ms', N') = \text{rough-state-of } (\text{DPLL-step}'\ S)$

by $(\text{cases } \text{rough-state-of } (\text{DPLL-step}'\ S))$ **auto**

ultimately have $\text{dpll}_W\text{-all-inv } (\text{toS}'\ (Ms', N'))$ **unfolding** Ms'

by $(\text{metis } (\text{no-types, lifting}) \text{ case-prod-unfold mem-Collect-eq rough-state-of})$

have $\text{dpll}_W\ (\text{toS } Ms\ N) (\text{toS } Ms'\ N')$

apply $(\text{rule } \text{DPLL-step-is-a-dpll}_W\text{-step}[of\ Ms'\ N'\ Ms\ N])$

unfolding $Ms\ Ms'$ **using** $\langle x \neq S \rangle$ $\text{rough-state-of-inject } x$ **by** fastforce+

then show $?thesis$ **unfolding** $Ms[\text{symmetric}]$ $Ms'[\text{symmetric}]$ **by** auto

qed

ultimately show $(x, S) \in \{(T', T). (\text{rough-state-of } T', \text{rough-state-of } T)$

$\in \{(S', S). (\text{toS}'\ S', \text{toS}'\ S) \in \{(S', S). \text{dpll}_W\text{-all-inv } S \wedge \text{dpll}_W\ S\ S'\}\}$

by $(\text{auto simp add: } x)$

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 λS. 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.prem 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'* \models_{asm} *N''* \longleftrightarrow *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* \times *int literal list list*

\Rightarrow *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* ($\forall A \in \text{set } N. (\exists a \in \text{set } A. a \in \text{lits-of-l } M), M$))

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 × 'b × 'c × 'd ⇒ 'a **where**

raw-trail ≡ (λ(M, -). M)

abbreviation *raw-cons-trail* :: 'a ⇒ 'a *list* × 'b × 'c × 'd ⇒ 'a *list* × 'b × 'c × 'd **where**

raw-cons-trail ≡ (λL (M, S). (L#M, S))

abbreviation *raw-tl-trail* :: 'a *list* × 'b × 'c × 'd ⇒ 'a *list* × 'b × 'c × 'd **where**

raw-tl-trail ≡ (λ(M, S). (tl M, S))

abbreviation *raw-init-clss* :: 'a × 'b × 'c × 'd ⇒ 'b **where**

raw-init-clss ≡ λ(M, N, -). N

abbreviation *raw-learned-clss* :: 'a × 'b × 'c × 'd ⇒ 'c **where**

raw-learned-clss ≡ λ(M, N, U, -). U

abbreviation *raw-conflicting* :: 'a × 'b × 'c × 'd ⇒ 'd **where**

raw-conflicting ≡ λ(M, N, U, D). D

abbreviation *raw-update-conflicting* :: 'd ⇒ 'a × 'b × 'c × 'd ⇒ 'a × 'b × 'c × '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-trail* *S*, *raw-init-clss* *S*, *raw-learned-clss* *S*, *raw-conflicting* *S*)
by (*cases* *S*) *auto*

definition *state where*

⟨*state* *S* = (*raw-trail* *S*, *raw-init-clss* *S*, *raw-learned-clss* *S*, *raw-conflicting* *S*, ())⟩

interpretation *state_W*

(=)
state
raw-trail *raw-init-clss* *raw-learned-clss* *raw-conflicting*
 $\lambda L (M, S). (L \# M, S)$
 $\lambda (M, S). (tl\ M, S)$
 $\lambda C (M, N, U, S). (M, N, add\text{-}mset\ C\ U, S)$
 $\lambda C (M, N, U, S). (M, removeAll\text{-}mset\ C\ N, removeAll\text{-}mset\ C\ U, S)$
 $\lambda D (M, N, U, -). (M, N, U, D)$
 $\lambda N. ([], N, \{\#\}, None)$
by *unfold-locales* (*auto simp: state-def*)

declare *state-simp*[*simp del*]

interpretation *conflict-driven-clause-learning_W*

(=) *state*
raw-trail *raw-init-clss* *raw-learned-clss*
raw-conflicting
 $\lambda L (M, S). (L \# M, S)$
 $\lambda (M, S). (tl\ M, S)$
 $\lambda C (M, N, U, S). (M, N, add\text{-}mset\ C\ U, S)$
 $\lambda C (M, N, U, S). (M, removeAll\text{-}mset\ C\ N, removeAll\text{-}mset\ C\ U, S)$
 $\lambda D (M, N, U, -). (M, N, U, D)$
 $\lambda N. ([], N, \{\#\}, None)$
by *unfold-locales auto*

declare *clauses-def*[*simp*]

lemma *reduce-trail-to-empty-trail*[*simp*]:

reduce-trail-to *F* ([], *aa*, *ab*, *b*) = ([], *aa*, *ab*, *b*)
using *reduce-trail-to.simps* **by** *auto*

lemma *reduce-trail-to'*:

reduce-trail-to *F* *S* =
 ((if *length* (*raw-trail* *S*) \geq *length* *F*
 then drop (*length* (*raw-trail* *S*) − *length* *F*) (*raw-trail* *S*)
 else []), *raw-init-clss* *S*, *raw-learned-clss* *S*, *raw-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]*:
 $I \models s \text{ (mset } \circ \text{ remdups) } 'N \longleftrightarrow I \models s \text{ mset } 'N$
by (*simp add: true-clss-def*)

lemma *true-clss-raw-remdups-mset-mset[simp]*:
 $\langle I \models s (\lambda L. \text{remdups-mset (mset L)}) 'N' \longleftrightarrow I \models s \text{ mset } 'N \rangle$
by (*simp add: true-clss-def*)

declare *satisfiable-carac*[*iff del*]
lemma *satisfiable-mset-remdups[simp]*:
 $\text{satisfiable ((mset } \circ \text{ remdups) } 'N) \longleftrightarrow \text{satisfiable (mset } 'N)$
 $\text{satisfiable ((}\lambda L. \text{remdups-mset (mset L)}) 'N') \longleftrightarrow \text{satisfiable (mset } 'N')$
unfolding *satisfiable-carac*[*symmetric*] **by** *simp-all*

type-synonym *'v cdcl_W-restart-state-inv-st* = (*'v, 'v literal list*) *ann-lit list* \times
'v literal list list \times *'v literal list list* \times *'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* \Rightarrow (*'a, 'c multiset*) *ann-lit* **where**
convert (Propagated L C) = Propagated L (mset C) |
convert (Decided K) = Decided K

abbreviation *convertC* :: *'a list option* \Rightarrow *'a multiset option* **where**
convertC \equiv *map-option mset*

lemma *convert-Propagated[elim!]*:
 $\text{convert } z = \text{Propagated } L \ C \Longrightarrow (\exists C'. z = \text{Propagated } L \ C' \wedge C = \text{mset } C')$
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]*: $\langle (\text{is-decided } \circ \text{convert}) = (\text{is-decided}) \rangle$
by *auto*

lemma *get-level-map-convert[simp]*:
 $\text{get-level (map convert M) } x = \text{get-level M } x$
by (*induction M rule: ann-lit-list-induct*) (*auto simp: comp-def get-level-def*)

lemma *get-maximum-level-map-convert[simp]*:
 $\text{get-maximum-level (map convert M) } D = \text{get-maximum-level M } D$
by (*induction D*) (*auto simp add: get-maximum-level-add-mset*)

lemma *count-decided-convert[simp]*:
 $\langle \text{count-decided (map convert M) } = \text{count-decided M} \rangle$
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]:
 $\langle \text{no-dup } (\text{map convert } M) = \text{no-dup } M \rangle$
by (*auto simp: no-dup-def image-image comp-def*)

Conversion function

fun *toS* :: '*v* *cdcl_W-restart-state-inv-st* \Rightarrow '*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 ($\square, \square, \square, \text{None}$) $\in \{S. \text{cdcl}_W\text{-all-struct-inv (toS S)}\}$
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* \Rightarrow
'*v* *cdcl_W-restart-state-inv* \Rightarrow *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 \longleftrightarrow undefined-lit M L
by (*auto simp add: defined-lit-map image-image*)

lemma *true-annot-map-convert*[simp]: *map convert M \models_a N \longleftrightarrow M \models_a N*
by (*simp-all add: true-annot-def image-image lits-of-def*)

lemma *true-annots-map-convert*[simp]: *map convert M \models_{as} N \longleftrightarrow M \models_{as} N*
unfolding *true-annots-def* **by** *auto*

lemmas *propagateE*

lemma *find-first-unit-clause-some-is-propagate*:

assumes *H*: *find-first-unit-clause (N @ U) M = Some (L, C)*
shows *propagate (toS (M, N, U, 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* \Rightarrow '*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 ≠ 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 (∀ c ∈ set N. ¬c ∈ lits-of-l M) then Some N else find-conflict M Ns)

```

lemma find-conflict-Some:

```

find-conflict M Ns = Some N ⇒ N ∈ set Ns ∧ M ⊨as CNot (mset N)
by (induction Ns rule: find-conflict.induct)
  (auto split: if-split-asm simp: lits-of-l-unfold)

```

lemma find-conflict-None:

```

find-conflict M Ns = None ⇔ (∀ N ∈ set Ns. ¬M ⊨as CNot (mset N))
by (induction Ns) (auto simp: lits-of-l-unfold)

```

lemma find-conflict-None-no-conf:

```

find-conflict M (N@U) = None ⇔ no-step conflict (toS (M, N, U, None))
by (auto simp add: find-conflict-None conflict.simps)

```

definition do-conflict-step :: 'v cdc_W-restart-state-inv-st ⇒ 'v cdc_W-restart-state-inv-st where
do-conflict-step S =

```

(case S of
  (M, N, U, None) ⇒
    (case find-conflict M (N @ U) of
      Some a ⇒ (M, N, U, Some a)
    | None ⇒ (M, N, U, None))
| S ⇒ S)

```

lemma do-conflict-step:

```

do-conflict-step S ≠ S ⇒ 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 ⇒ no-step conflict (toS S)
apply (cases S, cases raw-conflicting S)
unfolding do-conflict-step-def
using find-conflict-None-no-conf[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 ≠ None ⇒ 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 ≠ S ⇒ raw-conflicting (do-conflict-step S) ≠ 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* \circ *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) \implies *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* \Rightarrow '*v* *cdcl_W-restart-state-inv-st* **where**
do-skip-step (*Propagated* L C # Ls , N , U , *Some* D) =
 (if $-L \notin \text{set } D \wedge D \neq []$
 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 $S \neq S \implies \text{skip}$ (*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 $S = S \implies \text{no-step skip}$ (*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 $S = (a, b, c, \text{None}) \longleftrightarrow S = (a, b, c, \text{None})$
by (*cases* S rule: *do-skip-step.cases*) *auto*

Resolve **fun** *maximum-level-code*:: '*a* *literal list* \Rightarrow ('*a*, '*a* *literal list*) *ann-lit list* \Rightarrow *nat*
where
maximum-level-code [] = 0 |
maximum-level-code (L # Ls) $M = \max$ (*get-level* M L) (*maximum-level-code* Ls M)

lemma *maximum-level-code-eq-get-maximum-level*[*code, simp*]:
maximum-level-code D $M = \text{get-maximum-level}$ 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* \Rightarrow '*v* *cdcl_W-restart-state-inv-st* **where**
do-resolve-step (*Propagated* L C # Ls , N , U , *Some* D) =
 (if $-L \in \text{set } D \wedge \text{maximum-level-code}$ (*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) \implies *do-resolve-step* $S \neq S$
 $\implies \text{resolve}$ (*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
 - $L \in \text{set } D$ **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 cdclW-all-struct-inv-def cdclW-conflicting-def by fast
then have L ∈ 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 ∈ 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 ∈ 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 ⟨cdclW-all-struct-inv (toS (Propagated L C # M, N, U, Some D))⟩
    unfolding cdclW-all-struct-inv-def distinct-cdclW-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-remdups-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 ∈ 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  $\implies$  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]:
  cdclW-all-struct-inv (toS S)  $\implies$ 
  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 $S = (a, b, c, \text{None}) \longleftrightarrow S = (a, b, c, \text{None})$
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 ($\lambda(a, b). (\text{map convert } a, \text{map convert } b)$) (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** $S = \text{this}(1)$ **and** $\text{confl} = \text{this}(2)$
have $E: E = \text{Some } C$ **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

$L \in \text{set } C$ **and**
 j : get-maximum-level M (mset (remove1 $L C$)) = j **and**
 levL: get-level $M L = \text{count-decided } M$
using find-level-decomp-some[OF fd] **by** auto

obtain C' **where** $C: \text{mset } C = \text{add-mset } L (\text{mset } C')$
using $\langle L \in \text{set } C \rangle$ **by** (metis ex-mset in-multiset-in-set insert-DiffM)

obtain $M2$ **where** $M2: \text{bt-cut } j M = \text{Some } M2$
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**

$M1: M2 = \text{Decided } K \# M1$ **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 $j \leq \text{count-decided } M$ **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 $\langle L \in \text{set } C \rangle$ 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: cdclW-M-level-inv (toS S)
  using inv unfolding cdclW-all-struct-inv-def cdclW-learned-clause-alt-def S E
    cdclW-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 backtrackW-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 cdclW-all-struct-inv-def S by fast
  subgoal using inv unfolding cdclW-all-struct-inv-def S by fast
  subgoal using inv unfolding cdclW-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  $\longleftrightarrow$  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: cdclW-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 cdclW-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 cdclW-all-struct-inv-def cdclW-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  $\neg ?thesis$ 
  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  $get\_level\ M\ L' \leq get\_maximum\_level\ M\ D$ 
    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:  $bt\_cut\ j\ M \neq None$ 
  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:  $cdcl_W\text{-all-struct-inv}\ (toS\ S)$ 
  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  $do\_backtrack\_step\ S \in \{S.\ cdcl_W\text{-all-struct-inv}\ (toS\ S)\}$ 
proof cases
  case 0
  thus ?thesis using inv by simp
next
  case step
  then show ?thesis
    using inv
    by (auto dest!:  $cdcl_W\text{-restart.other}\ cdcl_W\text{-o.bj}\ cdcl_W\text{-bj.backtrack}\ intro: cdcl_W\text{-all-struct-inv-inv}$ )
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  $\Rightarrow$  (M, N, U, None)
    | Some L  $\Rightarrow$  (Decided L # M, N, U, None)) |

```

do-decide-step $S = S$

lemma *do-decide-step*:

do-decide-step $S \neq S \implies \text{decide } (\text{toS } S) (\text{toS } (\text{do-decide-step } S))$

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 $a :: ('a, 'a \text{ literal list}) \text{ ann-lit list and}$

$b :: 'a \text{ literal list list and } c :: 'a \text{ literal list list and}$

$e :: 'a \text{ literal list option}$

{

fix $a :: ('a, 'a \text{ literal list}) \text{ ann-lit list and}$

$b :: 'a \text{ literal list list and } c :: 'a \text{ literal list list and}$

$x2 :: 'a \text{ literal and } m :: 'a \text{ literal list}$

assume $a1: m \in \text{set } b$

assume $x2 \in \text{set } m$

then have $f2: \text{atm-of } x2 \in \text{atms-of } (\text{mset } m)$

by *simp*

have $\bigwedge f. (f m :: 'a \text{ literal multiset}) \in f \text{ ' set } b$

using $a1$ **by** *blast*

then have $\bigwedge f. (\text{atms-of } (f m) :: 'a \text{ set}) \subseteq \text{atms-of-ms } (f \text{ ' set } b)$

using *atms-of-atms-of-ms-mono* **by** *blast*

then have $\bigwedge n f. (n :: 'a) \in \text{atms-of-ms } (f \text{ ' set } b) \vee n \notin \text{atms-of } (f m)$

by (*meson contra-subsetD*)

then have $\text{atm-of } x2 \in \text{atms-of-ms } (\text{mset ' set } b)$

using $f2$ **by** *blast*

} **note** $H = \text{this}$

{

fix $m :: 'a \text{ literal list and } x2$

have $m \in \text{set } b \implies x2 \in \text{set } m \implies x2 \notin \text{lits-of-l } a \implies \neg x2 \notin \text{lits-of-l } a \implies$

$\exists aa \in \text{set } b. \neg \text{atm-of ' set } aa \subseteq \text{atm-of ' lits-of-l } a$

by (*meson atm-of-in-atm-of-set-in-uminus contra-subsetD rev-image-eqI*)

} **note** $H' = \text{this}$

assume *do-decide-step* $S \neq S$ **and**

$S = (a, b, c, e)$ **and**

raw-conflicting $S = \text{None}$

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 $S = S \implies \text{no-step decide } (\text{toS } S)$

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*]:

$cdcl_W\text{-all-struct-inv } (toS S) \implies \text{rough-state-of } (state\text{-of } (do\text{-decide-step } S)) = do\text{-decide-step } S$
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*]:
 $cdcl_W\text{-all-struct-inv } (toS S) \implies \text{rough-state-of } (state\text{-of } (do\text{-skip-step } S)) = do\text{-skip-step } S$
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* =
 $\{S :: 'v \text{ cdcl}_W\text{-restart-state-inv-st. } cdcl_W\text{-all-struct-inv } (toS S)$
 $\wedge cdcl_W\text{-stgy}^{**} (S0\text{-cdcl}_W\text{-restart (raw-init-clss (toS S))) (toS S)}\}$
morphisms *rough-state-from-init-state-of state-from-init-state-of*
proof
show ($[], [], [], None$) $\in \{S. cdcl_W\text{-all-struct-inv } (toS S)$
 $\wedge cdcl_W\text{-stgy}^{**} (S0\text{-cdcl}_W\text{-restart (raw-init-clss (toS S))) (toS S)}\}$
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))
 $\wedge cdcl_W\text{-stgy}^{**} (S0\text{-cdcl}_W\text{-restart (raw-init-clss (toS S))) (toS S)$ *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* \Rightarrow ' 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*:
 $c \in \text{set } (\text{raw-init-clss } (\text{rough-state-of } S) @ \text{raw-learned-clss } (\text{rough-state-of } S)) \implies \text{distinct } c$
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 $T \neq S$ 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* $S \neq S$
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* \Rightarrow '*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 \neq 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) \neq 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) \neq None \longleftrightarrow raw-conflicting S \neq None
by (*cases S*) *auto*

lemma *raw-trail-toS-neq-imp-raw-trail-neq*:
raw-trail (toS S) \neq raw-trail (toS S') \implies raw-trail S \neq raw-trail S'
by (*cases S, cases S'*) *auto*

lemma *do-cp-step-neq-raw-trail-increase*:
 $\exists c. \text{raw-trail (do-cp-step S)} = c @ \text{raw-trail S} \wedge (\forall m \in \text{set } c. \neg \text{is-decided } m)$
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) \neq None \implies 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

$\text{raw-conflicting } S = \text{None}$ **and**
 $\text{do-decide-step } S \neq S$
shows $\text{Suc } (\text{length } (\text{filter is-decided } (\text{raw-trail } S)))$
 $= \text{length } (\text{filter is-decided } (\text{raw-trail } (\text{do-decide-step } S)))$
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 $\text{raw-conflicting } S \neq \text{None}$ **and**
 $\text{do-decide-step } S \neq S$
shows $\text{length } (\text{filter is-decided } (\text{raw-trail } S)) < \text{length } (\text{filter is-decided } (\text{raw-trail } (\text{do-decide-step } S)))$
using *assms* **by** (*cases* S , *cases* $\text{raw-conflicting } S$)
(auto simp add: Let-def split: if-split-asm option.splits)

lemma *count-decided-raw-trail-toS:*
 $\text{count-decided } (\text{raw-trail } (\text{toS } S)) = \text{count-decided } (\text{raw-trail } S)$
by (*cases* S) (*auto simp: comp-def*)

lemma *rough-state-of-state-of-do-skip-step-rough-state-of[simp]:*
 $\text{rough-state-of } (\text{state-of } (\text{do-skip-step } (\text{rough-state-of } S))) = \text{do-skip-step } (\text{rough-state-of } S)$
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]:*
 $\text{raw-conflicting } (\text{do-resolve-step } S) = \text{None} \longleftrightarrow \text{raw-conflicting } S = \text{None}$
by (*cases* S *rule: do-resolve-step.cases*)
(auto simp add: Let-def split: option.splits)

lemma *raw-conflicting-do-skip-step-iff[iff]:*
 $\text{raw-conflicting } (\text{do-skip-step } S) = \text{None} \longleftrightarrow \text{raw-conflicting } S = \text{None}$
by (*cases* S *rule: do-skip-step.cases*)
(auto simp add: Let-def split: option.splits)

lemma *raw-conflicting-do-decide-step-iff[iff]:*
 $\text{raw-conflicting } (\text{do-decide-step } S) = \text{None} \longleftrightarrow \text{raw-conflicting } S = \text{None}$
by (*cases* S *rule: do-decide-step.cases*)
(auto simp add: Let-def split: option.splits)

lemma *raw-conflicting-do-backtrack-step-imp[simp]:*
 $\text{do-backtrack-step } S \neq S \implies \text{raw-conflicting } (\text{do-backtrack-step } S) = \text{None}$
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:*
 $\text{do-skip-step } S = S \longleftrightarrow \text{raw-trail } (\text{do-skip-step } S) = \text{raw-trail } S$
by (*cases* S *rule: do-skip-step.cases*) *auto*

lemma *do-decide-step-eq-iff-raw-trail-eq:*
 $\text{do-decide-step } S = S \longleftrightarrow \text{raw-trail } (\text{do-decide-step } S) = \text{raw-trail } S$
by (*cases* S *rule: do-decide-step.cases*) (*auto split: option.split*)

lemma *do-backtrack-step-eq-iff-raw-trail-eq:*
assumes *no-dup* ($\text{raw-trail } S$)
shows $\text{do-backtrack-step } S = S \longleftrightarrow \text{raw-trail } (\text{do-backtrack-step } S) = \text{raw-trail } S$

```

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  $\longleftrightarrow$  raw-trail (do-resolve-step S) = raw-trail S
by (cases S rule: do-resolve-step.cases) auto

```

```

lemma do-cdclW-stgy-step-no:
  assumes S: do-cdclW-stgy-step S = S
  shows no-step cdclW-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-cdclW-stgy-step)
  then show ?thesis
    using cdclW-all-struct-inv-rough-state cdclW-stgy-cdclW 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 cdclW-stgy-is-rtrancp-cdclW-restart:
  cdclW-stgy S T  $\implies$  cdclW-restart** S T
by (simp add: cdclW-stgy-trancp-cdclW-restart rtrancp-unfold)

```

```

lemma cdclW-stgy-init-raw-init-clss:
  cdclW-stgy S T  $\implies$  cdclW-M-level-inv S  $\implies$  raw-init-clss S = raw-init-clss T
using cdclW-stgy-no-more-init-clss by blast

```

```

lemma clauses-toS-rough-state-of-do-cdclW-stgy-step[simp]:
  raw-init-clss (toS (rough-state-of (do-cdclW-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-cdclW-stgy-step (state-of ?S) = state-of ?S)
  apply simp
by (metis cdclW-stgy-no-more-init-clss do-cdclW-stgy-step
  toS-rough-state-of-state-of-rough-state-from-init-state-of)

```

```

lemma rough-state-from-init-state-of-do-cdclW-stgy-step'[code abstract]:
  rough-state-from-init-state-of (do-cdclW-stgy-step' S) =
    rough-state-of (do-cdclW-stgy-step (id-of-I-to S))
proof —
  let ?S = (rough-state-from-init-state-of S)
  have cdclW-stgy** (S0-cdclW-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 cdclW-stgy**
    (toS (rough-state-from-init-state-of S))
    (toS (rough-state-of (do-cdclW-stgy-step
      (state-of (rough-state-from-init-state-of S)))))
    using do-cdclW-stgy-step[of state-of ?S]

```


by (cases do-cdcl_W-stgy-step (state-of ?S) = state-of ?S) auto
 ultimately show ?thesis
 unfolding do-cdcl_W-stgy-step'-def id-of-I-to-def
 by (auto intro!: state-from-init-state-of-inverse)
 qed

All rules together function do-all-cdcl_W-stgy where

do-all-cdcl_W-stgy S =

(let T = do-cdcl_W-stgy-step' S in
 if T = S then S else do-all-cdcl_W-stgy T)

by fast+

termination

proof (relation {(T, S).

(cdcl_W-restart-measure (toS (rough-state-from-init-state-of T)),
 cdcl_W-restart-measure (toS (rough-state-from-init-state-of S)))
 ∈ le_{rn} less-than 3}, goal-cases)

case 1

show ?case by (rule wf-if-measure-f) (auto intro!: wf-le_{rn} wf-less)

next

case (2 S T) note T = this(1) and ST = this(2)

let ?S = rough-state-from-init-state-of S

have S: cdcl_W-stgy** (S0-cdcl_W-restart (raw-init-clss (toS ?S))) (toS ?S)

using rough-state-from-init-state-of[of S] by auto

moreover have cdcl_W-stgy (toS (rough-state-from-init-state-of S))

(toS (rough-state-from-init-state-of T))

proof –

have ∧ c. rough-state-of (state-of (rough-state-from-init-state-of c)) =
 rough-state-from-init-state-of c

using rough-state-from-init-state-of state-of-inverse by fastforce

then have diff: do-cdcl_W-stgy-step (state-of (rough-state-from-init-state-of S))
 ≠ state-of (rough-state-from-init-state-of S)

using ST T by (metis (no-types) id-of-I-to-def rough-state-from-init-state-of-inject
 rough-state-from-init-state-of-do-cdcl_W-stgy-step')

have rough-state-of (do-cdcl_W-stgy-step (state-of (rough-state-from-init-state-of S)))
 = rough-state-from-init-state-of (do-cdcl_W-stgy-step' S)

by (simp add: id-of-I-to-def rough-state-from-init-state-of-do-cdcl_W-stgy-step')

then show ?thesis

using do-cdcl_W-stgy-step T diff unfolding id-of-I-to-def do-cdcl_W-stgy-step by fastforce

qed

moreover have invs: cdcl_W-all-struct-inv (toS (rough-state-from-init-state-of S))

using rough-state-from-init-state-of[of S] by auto

moreover {

have cdcl_W-all-struct-inv (S0-cdcl_W-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-all-struct-inv-def distinct-cdcl_W-state-def)

then have <no-smaller-propa (toS (rough-state-from-init-state-of S))>

using rtranclp-cdcl_W-stgy-no-smaller-propa[OF S]

by (auto simp: empty-trail-no-smaller-propa) }

ultimately show ?case

using tranclp-cdcl_W-stgy-S0-decreasing

by (auto intro!: cdcl_W-stgy-step-decreasing[of]

simp del: cdcl_W-restart-measure.simps)

qed

thm do-all-cdcl_W-stgy.induct

lemma do-all-cdcl_W-stgy-induct:

$(\bigwedge S. (do_cdcl_W\text{-stgy}\text{-step}' S \neq S \implies P (do_cdcl_W\text{-stgy}\text{-step}' S)) \implies P S) \implies P a0$
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 ≠ S*)

proof –

fix *Sa :: 'a cdcl_W-restart-state-inv-from-init-state*

assume *a1: ¬ do-cdcl_W-stgy-step' Sa ≠ Sa*

{ fix *pp*

have (*if True then Sa else do-all-cdcl_W-stgy Sa*) = *do-all-cdcl_W-stgy Sa*

using *a1* **by** *auto*

then have $\neg cdcl_W\text{-stgy (toS (rough-state-from-init-state-of (do-all-cdcl}_W\text{-stgy Sa)))}$ *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 ≠ Sa*

$\implies no\text{-step } cdcl_W\text{-stgy (toS (rough-state-from-init-state-of$

$(do\text{-all-cdcl}_W\text{-stgy (do-cdcl}_W\text{-stgy-step' Sa))))$

assume *a2: do-cdcl_W-stgy-step' Sa ≠ 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-rtrancpl-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-cdclW-stgy-step by fastforce
qed
qed

```

Final theorem:

lemma *DPLL-tot-correct*:

```

assumes
  r: rough-state-from-init-state-of (do-all-cdclW-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: cdclW-all-struct-inv (toS ([], map remdups N, [], None))
    unfolding cdclW-all-struct-inv-def distinct-cdclW-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 cdclW-stgy (toS ([], ?N, [], None)) (toS S)
    unfolding full-def apply rule
    using do-all-cdclW-stgy-is-rtrancpl-cdclW-stgy[of
      state-from-init-state-of ([], map remdups N, [], None)] inv
      no-step-cdclW-stgy-cdclW-restart-all
    apply (auto simp del: do-all-cdclW-stgy.simps simp: state-from-init-state-of-inverse
      r[symmetric] comp-def)[]
    using do-all-cdclW-stgy-is-rtrancpl-cdclW-stgy[of
      state-from-init-state-of ([], map remdups N, [], None)] inv
      no-step-cdclW-stgy-cdclW-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 cdclW-all-struct-inv (toS S)
      by (metis (no-types) cdclW-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 cdclW-all-struct-inv-def cdclW-M-level-inv-def S[symmetric] by auto
  moreover
    have raw-init-clss (toS ([], ?N, [], None)) = raw-init-clss (toS S)
      apply (rule rtrancpl-cdclW-stgy-no-more-init-clss)
      using 1 unfolding full-def by (auto simp add: rtrancpl-cdclW-stgy-rtrancpl-cdclW-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-cdclW-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  $\implies e \in \#$  convert-to-mset Cs and
    in-mset-cls-exists-preimage:
      b  $\in \#$  convert-to-mset Cs  $\implies \exists b'. \text{get-lit } Cs \text{ } b' = \text{Some } b$ 

```

```

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  $\implies e \in \#$  convert-to-mset Cs and
    in-mset-clss-exists-preimage:
      b  $\in \#$  convert-to-mset Cs  $\implies \exists b'. \text{get-lit } Cs \text{ } b' = \text{Some } b$ 

```

```

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-cls
  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 cdclW-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 cdclW-restart-mset ⇒ ('v, 'v clause) ann-lit list where
trail (M, -) = M

```

```

fun init-cls :: 'v cdclW-restart-mset ⇒ 'v clauses where
init-cls (-, N, -) = N

```

```

fun learned-cls :: 'v cdclW-restart-mset ⇒ 'v clauses where
learned-cls (-, -, U, -) = U

```

```

fun conflicting :: 'v cdclW-restart-mset ⇒ 'v clause option where
conflicting (-, -, -, C) = C

```

```

fun cons-trail :: ('v, 'v clause) ann-lit ⇒ 'v cdclW-restart-mset ⇒ 'v cdclW-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**

$\langle \text{state } S = (\text{trail } S, \text{init-clss } S, \text{learned-clss } S, \text{conflicting } S, ()) \rangle$

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* \Rightarrow '*v cdcl_W-restart-mset* \Rightarrow bool (**infix** \sim_m 50) **where**
 $\langle S \sim_m T \iff \text{state } S = \text{state } T \rangle$

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* \Rightarrow nat **where**

backtrack-lvl \equiv *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 cdclW-restart-mset-state-eq-eq: state-eq = (=)
  apply (intro ext)
  unfolding state-eq-def
  by (auto simp: cdclW-restart-mset-state state-def)

```

```

lemma clauses-def: ⟨cdclW-restart-mset.clauses (M, N, U, C) = N + U⟩
  by (subst cdclW-restart-mset.clauses-def) (simp add: cdclW-restart-mset-state)

```

```

lemma cdclW-restart-mset-reduce-trail-to:

```

```

  cdclW-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: cdclW-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: cdclW-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 cdclW-restart-mset.reduce-trail-to-length-ne[of S F] IH by auto

```

```

      subgoal

```

```

        using IH cdclW-restart-mset.reduce-trail-to-length-ne[of S F]

```

```

        apply (cases S)

```

```

        by (simp add: cdclW-restart-mset.trail-reduce-trail-to-drop cdclW-restart-mset-state)

```

```

      done

```

```

    qed

```

```

  qed

```

```

lemma full-cdclW-init-state:

```

```

  ⟨full cdclW-restart-mset.cdclW-stgy (init-state {#}) S ⟷ S = init-state {#}⟩

```

```

  unfolding full-def rtranclp-unfold

```

```

  by (subst tranclp-unfold-begin)

```

```

    (auto simp: cdclW-restart-mset.cdclW-stgy.simps

```

```

      cdclW-restart-mset.conflict.simps cdclW-restart-mset.cdclW-o.simps

```

```

      cdclW-restart-mset.propagate.simps cdclW-restart-mset.decide.simps

```

```

      cdclW-restart-mset.cdclW-bj.simps cdclW-restart-mset.backtrack.simps

```

```

      cdclW-restart-mset.skip.simps cdclW-restart-mset.resolve.simps

```

```

      cdclW-restart-mset-state clauses-def)

```



```

locale twl-restart-ops =
  fixes
     $f :: \langle \text{nat} \Rightarrow \text{nat} \rangle$ 
begin

interpretation cdclW-restart-mset: cdclW-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  $f :: \langle \text{nat} \Rightarrow \text{nat} \rangle +$ 
  assumes
     $f: \langle \text{unbounded } f \rangle$ 
begin

interpretation cdclW-restart-mset: cdclW-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-learningW
begin

lemma distinct-cdclW-state-alt-def:
   $\langle \text{distinct-cdcl}_W\text{-state } S =$ 
     $((\forall T. \text{conflicting } S = \text{Some } T \longrightarrow \text{distinct-mset } T) \wedge$ 

```

```

    distinct-mset-mset (clauses S) ∧
    (∀ L mark. Propagated L mark ∈ set (trail S) ⟶ distinct-mset mark))
unfolding distinct-cdclW-state-def clauses-def
by auto
end

```

```

lemma cdclW-stgy-cdclW-init-state-empty-no-step:
  ⟨cdclW-restart-mset.cdclW-stgy (init-state {#}) S ⟷ False⟩
unfolding rtrancpl-unfold
by (auto simp: cdclW-restart-mset.cdclW-stgy.simps
    cdclW-restart-mset.conflict.simps cdclW-restart-mset.cdclW-o.simps
    cdclW-restart-mset.propagate.simps cdclW-restart-mset.decide.simps
    cdclW-restart-mset.cdclW-bj.simps cdclW-restart-mset.backtrack.simps
    cdclW-restart-mset.skip.simps cdclW-restart-mset.resolve.simps
    cdclW-restart-mset-state clauses-def)

```

```

lemma cdclW-stgy-cdclW-init-state:
  ⟨cdclW-restart-mset.cdclW-stgy** (init-state {#}) S ⟷ S = init-state {#}⟩
unfolding rtrancpl-unfold
by (subst trancpl-unfold-begin)
  (auto simp: cdclW-stgy-cdclW-init-state-empty-no-step simp del: init-state.simps)

end

```