Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

March 30, 2016

Contents

1 Partial Clausal Logic

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

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

begin

1.1 Decided Literals

1.1.1 Definition

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

lemma *ann-literal-list-induct*[*case-names nil decided proped*]:
  assumes P [] **and**
  $\bigwedge L\ l\ xs.\ P\ xs \Longrightarrow P\ (Decided\ L\ l\ \#\ xs)$ **and**
  $\bigwedge L\ m\ xs.\ P\ xs \Longrightarrow P\ (Propagated\ L\ m\ \#\ xs)$
  shows $P\ xs$
  ⟨*proof*⟩

lemma *is-decided-ex-Decided*:
  $is\text{-}decided\ L \Longrightarrow \exists K\ lvl.\ L = Decided\ K\ lvl$
  ⟨*proof*⟩

type-synonym $('v, 'l, 'm)\ ann\text{-}literals = ('v, 'l, 'm)\ ann\text{-}literal\ list$

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

lemma *lits-of-empty*[*simp*]:
  $lits\text{-}of\ [] = \{\}$ ⟨*proof*⟩

lemma *lits-of-cons*[*simp*]:
  $lits\text{-}of\ (L\ \#\ Ls) = insert\ (lit\text{-}of\ L)\ (lits\text{-}of\ Ls)$
  ⟨*proof*⟩

lemma *lits-of-append*[*simp*]:
  $lits\text{-}of\ (l\ @\ l') = lits\text{-}of\ l \cup lits\text{-}of\ l'$
  ⟨*proof*⟩

lemma *finite-lits-of-def*[*simp*]: $finite\ (lits\text{-}of\ L)$
  ⟨*proof*⟩

lemma *lits-of-rev*[*simp*]: $lits\text{-}of\ (rev\ M) = lits\text{-}of\ M$
  ⟨*proof*⟩

lemma *set-map-lit-of-lits-of*[*simp*]:
  $set\ (map\ lit\text{-}of\ T) = lits\text{-}of\ T$
  ⟨*proof*⟩

Remove annotation and transform to a set of single literals.

abbreviation *unmark* :: $('a, 'b, 'c)\ ann\text{-}literal\ list \Rightarrow 'a\ literal\ multiset\ set$ **where**
$unmark\ M \equiv (\lambda a.\ \{\#lit\text{-}of\ a\#\})\ `\ set\ M$

lemma *atms-of-ms-lambda-lit-of-is-atm-of-lit-of*[*simp*]:
  $atms\text{-}of\text{-}ms\ (unmark\ M') = atm\text{-}of\ `\ lits\text{-}of\ M'$
  ⟨*proof*⟩

lemma *lits-of-empty-is-empty*[*iff*]:
  $lits\text{-}of\ M = \{\} \longleftrightarrow M = []$
  ⟨*proof*⟩

1.1.2 Entailment

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

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

lemma *true-annot-empty-model*[*simp*]:
  $\neg[] \models a\ \psi$
  ⟨*proof*⟩

lemma *true-annot-empty*[*simp*]:

$\neg I \models a \; \{\#\}$
⟨*proof*⟩

lemma *empty-true-annots-def*[*iff*]:
$[] \models as \; \psi \longleftrightarrow \psi = \{\}$
⟨*proof*⟩

lemma *true-annots-empty*[*simp*]:
$I \models as \; \{\}$
⟨*proof*⟩

lemma *true-annots-single-true-annot*[*iff*]:
$I \models as \; \{C\} \longleftrightarrow I \models a \; C$
⟨*proof*⟩

lemma *true-annot-insert-l*[*simp*]:
$M \models a \; A \Longrightarrow L \# M \models a \; A$
⟨*proof*⟩

lemma *true-annots-insert-l* [*simp*]:
$M \models as \; A \Longrightarrow L \# M \models as \; A$
⟨*proof*⟩

lemma *true-annots-union*[*iff*]:
$M \models as \; A \cup B \longleftrightarrow (M \models as \; A \wedge M \models as \; B)$
⟨*proof*⟩

lemma *true-annots-insert*[*iff*]:
$M \models as \; insert \; a \; A \longleftrightarrow (M \models a \; a \wedge M \models as \; A)$
⟨*proof*⟩

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

lemma *true-annots-true-cls*:
$I \models as \; CC \longleftrightarrow (lits\text{-}of \; I) \models s \; CC$
⟨*proof*⟩

lemma *in-lit-of-true-annot*:
$a \in lits\text{-}of \; M \longleftrightarrow M \models a \; \{\#a\#\}$
⟨*proof*⟩

lemma *true-annot-lit-of-notin-skip*:
$L \# M \models a \; A \Longrightarrow lit\text{-}of \; L \notin\# \; A \Longrightarrow M \models a \; A$
⟨*proof*⟩

lemma *true-clss-singleton-lit-of-implies-incl*:
$I \models s \; unmark \; MLs \Longrightarrow lits\text{-}of \; MLs \subseteq I$
⟨*proof*⟩

lemma *true-annot-true-clss-cls*:
$MLs \models a \; \psi \Longrightarrow set \; (map \; (\lambda a. \; \{\#lit\text{-}of \; a\#\}) \; MLs) \models p \; \psi$
⟨*proof*⟩

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

$\langle proof \rangle$

lemma *true-annots-decided-true-cls*[*iff*]:
$map\ (\lambda M.\ Decided\ M\ a)\ M \models as\ N \longleftrightarrow set\ M \models s\ N$
$\langle proof \rangle$

lemma *true-annot-singleton*[*iff*]: $M \models a\ \{\#L\#\} \longleftrightarrow L \in lits\text{-}of\ M$
$\langle proof \rangle$

lemma *true-annots-true-clss-clss*:
$A \models as\ \Psi \Longrightarrow unmark\ A \models ps\ \Psi$
$\langle proof \rangle$

lemma *true-annot-commute*:
$M\ @\ M' \models a\ D \longleftrightarrow M'\ @\ M \models a\ D$
$\langle proof \rangle$

lemma *true-annots-commute*:
$M\ @\ M' \models as\ D \longleftrightarrow M'\ @\ M \models as\ D$
$\langle proof \rangle$

lemma *true-annot-mono*[*dest*]:
$set\ I \subseteq set\ I' \Longrightarrow I \models a\ N \Longrightarrow I' \models a\ N$
$\langle proof \rangle$

lemma *true-annots-mono*:
$set\ I \subseteq set\ I' \Longrightarrow I \models as\ N \Longrightarrow I' \models as\ N$
$\langle proof \rangle$

1.1.3 Defined and undefined literals

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

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

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

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

lemma *defined-lit-rev*[*simp*]:
$defined\text{-}lit\ (rev\ M)\ L \longleftrightarrow defined\text{-}lit\ M\ L$
$\langle proof \rangle$

lemma *atm-imp-decided-or-proped*:
assumes $x \in set\ I$
shows
$(\exists l.\ Decided\ (-\ lit\text{-}of\ x)\ l \in set\ I)$
$\lor (\exists l.\ Decided\ (lit\text{-}of\ x)\ l \in set\ I)$
$\lor (\exists l.\ Propagated\ (-\ lit\text{-}of\ x)\ l \in set\ I)$
$\lor (\exists l.\ Propagated\ (lit\text{-}of\ x)\ l \in set\ I)$
$\langle proof \rangle$

lemma *literal-is-lit-of-decided*:
assumes $L = \mathit{lit\text{-}of}\ x$
shows $(\exists l.\ x = \mathit{Decided}\ L\ l) \lor (\exists l'.\ x = \mathit{Propagated}\ L\ l')$
⟨*proof*⟩

lemma *true-annot-iff-decided-or-true-lit*:
$\mathit{defined\text{-}lit}\ I\ L \longleftrightarrow ((\mathit{lits\text{-}of}\ I) \models l\ L \lor (\mathit{lits\text{-}of}\ I) \models l\ -L)$
⟨*proof*⟩

lemma *consistent-inter-true-annots-satisfiable*:
$\mathit{consistent\text{-}interp}\ (\mathit{lits\text{-}of}\ I) \Longrightarrow I \models as\ N \Longrightarrow \mathit{satisfiable}\ N$
⟨*proof*⟩

lemma *defined-lit-map*:
$\mathit{defined\text{-}lit}\ Ls\ L \longleftrightarrow \mathit{atm\text{-}of}\ L \in (\lambda l.\ \mathit{atm\text{-}of}\ (\mathit{lit\text{-}of}\ l))\ `\ \mathit{set}\ Ls$
⟨*proof*⟩

lemma *defined-lit-uminus*[*iff*]:
$\mathit{defined\text{-}lit}\ I\ (-L) \longleftrightarrow \mathit{defined\text{-}lit}\ I\ L$
⟨*proof*⟩

lemma *Decided-Propagated-in-iff-in-lits-of*:
$\mathit{defined\text{-}lit}\ I\ L \longleftrightarrow (L \in \mathit{lits\text{-}of}\ I \lor -L \in \mathit{lits\text{-}of}\ I)$
⟨*proof*⟩

lemma *consistent-add-undefined-lit-consistent*[*simp*]:
assumes
$\mathit{consistent\text{-}interp}\ (\mathit{lits\text{-}of}\ Ls)$ **and**
$\mathit{undefined\text{-}lit}\ Ls\ L$
shows $\mathit{consistent\text{-}interp}\ (\mathit{insert}\ L\ (\mathit{lits\text{-}of}\ Ls))$
⟨*proof*⟩

lemma *decided-empty*[*simp*]:
$\neg \mathit{defined\text{-}lit}\ []\ L$
⟨*proof*⟩

1.2 Backtracking

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

lemma *backtrack-split-fst-not-decided*: $a \in \mathit{set}\ (\mathit{fst}\ (\mathit{backtrack\text{-}split}\ l)) \Longrightarrow \neg \mathit{is\text{-}decided}\ a$
⟨*proof*⟩

lemma *backtrack-split-snd-hd-decided*:
$\mathit{snd}\ (\mathit{backtrack\text{-}split}\ l) \neq [] \Longrightarrow \mathit{is\text{-}decided}\ (\mathit{hd}\ (\mathit{snd}\ (\mathit{backtrack\text{-}split}\ l)))$
⟨*proof*⟩

lemma *backtrack-split-list-eq*[*simp*]:
$\mathit{fst}\ (\mathit{backtrack\text{-}split}\ l)\ @\ (\mathit{snd}\ (\mathit{backtrack\text{-}split}\ l)) = l$
⟨*proof*⟩

lemma *backtrack-snd-empty-not-decided*:

$backtrack\text{-}split\ M = (M'',\ []) \Longrightarrow \forall\, l{\in}set\ M.\ \neg\ is\text{-}decided\ l$
⟨proof⟩

lemma *backtrack-split-some-is-decided-then-snd-has-hd*:
$\exists\, l{\in}set\ M.\ is\text{-}decided\ l \Longrightarrow \exists\, M'\ L'\ M''.\ backtrack\text{-}split\ M = (M'',\ L'\ \#\ M')$
⟨proof⟩

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

lemma *backtrack-split-takeWhile-dropWhile*:
$backtrack\text{-}split\ M = (takeWhile\ (Not\ o\ is\text{-}decided)\ M,\ dropWhile\ (Not\ o\ is\text{-}decided)\ M)$
⟨proof⟩

1.3 Decomposition with respect to the First Decided Literals

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

1.3.1 Definition

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

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

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

Now we can prove several simple properties about the function.

lemma *get-all-decided-decomposition-never-empty*[*iff*]:
get-all-decided-decomposition M = [] ⟷ *False*
⟨proof⟩

lemma *get-all-decided-decomposition-never-empty-sym*[*iff*]:
[] = *get-all-decided-decomposition M* ⟷ *False*
⟨proof⟩

lemma *get-all-decided-decomposition-decomp*:
hd (*get-all-decided-decomposition S*) = (*a*, *c*) ⟹ *S* = *c* @ *a*
⟨proof⟩

lemma *get-all-decided-decomposition-backtrack-split*:
backtrack-split S = (*M*, *M'*) ⟷ *hd* (*get-all-decided-decomposition S*) = (*M'*, *M*)
⟨proof⟩

lemma *get-all-decided-decomposition-nil-backtrack-split-snd-nil*:
get-all-decided-decomposition S = [([], *A*)] ⟹ *snd* (*backtrack-split S*) = []

$\langle proof \rangle$

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

lemma *get-all-decided-decomposition-length-1-fst-empty-or-length-1*:
assumes $\textit{get-all-decided-decomposition } M = (a, b) \mathbin{\#} []$
shows $a = [] \lor (\textit{length } a = 1 \land \textit{is-decided } (\textit{hd } a) \land \textit{hd } a \in \textit{set } M)$
$\langle proof \rangle$

lemma *get-all-decided-decomposition-fst-empty-or-hd-in-M*:
assumes $\textit{get-all-decided-decomposition } M = (a, b) \mathbin{\#} l$
shows $a = [] \lor (\textit{is-decided } (\textit{hd } a) \land \textit{hd } a \in \textit{set } M)$
$\langle proof \rangle$

lemma *get-all-decided-decomposition-snd-not-decided*:
assumes $(a, b) \in \textit{set } (\textit{get-all-decided-decomposition } M)$
and $L \in \textit{set } b$
shows $\neg\textit{is-decided } L$
$\langle proof \rangle$

lemma *tl-get-all-decided-decomposition-skip-some*:
assumes $x \in \textit{set } (\textit{tl } (\textit{get-all-decided-decomposition } M1))$
shows $x \in \textit{set } (\textit{tl } (\textit{get-all-decided-decomposition } (M0 \mathbin{@} M1)))$
$\langle proof \rangle$

lemma *hd-get-all-decided-decomposition-skip-some*:
assumes $(x, y) = \textit{hd } (\textit{get-all-decided-decomposition } M1)$
shows $(x, y) \in \textit{set } (\textit{get-all-decided-decomposition } (M0 \mathbin{@} \textit{Decided } K\ i \mathbin{\#} M1))$
$\langle proof \rangle$

lemma *in-get-all-decided-decomposition-in-get-all-decided-decomposition-prepend*:
$(a, b) \in \textit{set } (\textit{get-all-decided-decomposition } M') \Longrightarrow$
$\exists b'.\ (a, b' \mathbin{@} b) \in \textit{set } (\textit{get-all-decided-decomposition } (M \mathbin{@} M'))$
$\langle proof \rangle$

lemma *get-all-decided-decomposition-remove-undecided-length*:
assumes $\forall l \in \textit{set } M'.\ \neg\textit{is-decided } l$
shows $\textit{length } (\textit{get-all-decided-decomposition } (M' \mathbin{@} M''))$
$= \textit{length } (\textit{get-all-decided-decomposition } M'')$
$\langle proof \rangle$

lemma *get-all-decided-decomposition-not-is-decided-length*:
assumes $\forall l \in \textit{set } M'.\ \neg\textit{is-decided } l$
shows $1 + \textit{length } (\textit{get-all-decided-decomposition } (\textit{Propagated } (-L)\ P \mathbin{\#} M))$
$= \textit{length } (\textit{get-all-decided-decomposition } (M' \mathbin{@} \textit{Decided } L\ l \mathbin{\#} M))$
$\langle proof \rangle$

lemma *get-all-decided-decomposition-last-choice*:
assumes $\textit{tl } (\textit{get-all-decided-decomposition } (M' \mathbin{@} \textit{Decided } L\ l \mathbin{\#} M)) \neq []$
and $\forall l \in \textit{set } M'.\ \neg\textit{is-decided } l$
and $\textit{hd } (\textit{tl } (\textit{get-all-decided-decomposition } (M' \mathbin{@} \textit{Decided } L\ l \mathbin{\#} M))) = (M0', M0)$
shows $\textit{hd } (\textit{get-all-decided-decomposition } (\textit{Propagated } (-L)\ P \mathbin{\#} M)) = (M0', \textit{Propagated } (-L)\ P \mathbin{\#} M0)$
$\langle proof \rangle$

lemma *get-all-decided-decomposition-except-last-choice-equal*:
  assumes $\forall\, l \in$ *set* M'. $\neg$*is-decided* l
  shows *tl* (*get-all-decided-decomposition* (*Propagated* $(-L)$ P # M))
    = *tl* (*tl* (*get-all-decided-decomposition* (M' @ *Decided* L l # M)))
  ⟨*proof*⟩

lemma *get-all-decided-decomposition-hd-hd*:
  assumes *get-all-decided-decomposition* Ls = (M, C) # ($M0$, $M0'$) # l
  shows *tl* M = $M0'$ @ $M0$ $\wedge$ *is-decided* (*hd* M)
  ⟨*proof*⟩

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

lemma *get-all-decided-decomposition-incl*:
  assumes (a, b) $\in$ *set* (*get-all-decided-decomposition* M)
  shows *set* $b \subseteq$ *set* M **and** *set* $a \subseteq$ *set* M
  ⟨*proof*⟩

lemma *get-all-decided-decomposition-exists-prepend'*:
  assumes (a, b) $\in$ *set* (*get-all-decided-decomposition* M)
  obtains c **where** M = c @ b @ a
  ⟨*proof*⟩

lemma *union-in-get-all-decided-decomposition-is-subset*:
  assumes (a, b) $\in$ *set* (*get-all-decided-decomposition* M)
  shows *set* $a \cup$ *set* $b \subseteq$ *set* M
  ⟨*proof*⟩

1.3.2 Entailment of the Propagated by the Decided Literal

lemma *get-all-decided-decomposition-snd-union*:
  set M = $\bigcup$(*set* ‘ *snd* ‘ *set* (*get-all-decided-decomposition* M)) $\cup$ {L |L. *is-decided* L $\wedge$ $L \in$ *set* M}
  (**is** *?M* M = *?U* M $\cup$ *?Ls* M)
⟨*proof*⟩

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

lemma *all-decomposition-implies-empty*[*iff*]:
  all-decomposition-implies N [] ⟨*proof*⟩

lemma *all-decomposition-implies-single*[*iff*]:
  all-decomposition-implies N [(*Ls*, *seen*)] $\longleftrightarrow$ *unmark* $Ls \cup N$ $\models$*ps* *unmark seen*
  ⟨*proof*⟩

lemma *all-decomposition-implies-append*[*iff*]:
  all-decomposition-implies N (S @ S')
    $\longleftrightarrow$ (*all-decomposition-implies* N S $\wedge$ *all-decomposition-implies* N S')
  ⟨*proof*⟩

lemma *all-decomposition-implies-cons-pair*[*iff*]:

all-decomposition-implies N $((Ls,\ seen)\ \#\ S')$
$\longleftrightarrow$ (*all-decomposition-implies* N $[(Ls,\ seen)]$ $\wedge$ *all-decomposition-implies* N S')
⟨*proof*⟩

lemma *all-decomposition-implies-cons-single*[*iff*]:
all-decomposition-implies N $(l\ \#\ S')$ $\longleftrightarrow$
(*unmark* (*fst* l) $\cup$ N $\models$*ps* *unmark* (*snd* l) $\wedge$
all-decomposition-implies N S')
⟨*proof*⟩

lemma *all-decomposition-implies-trail-is-implied*:
assumes *all-decomposition-implies* N (*get-all-decided-decomposition* M)
shows $N \cup \{\{\#$*lit-of* $L\#\}$ $|L.$ *is-decided* $L \wedge L \in$ *set* $M\}$
$\models$*ps* $(\lambda a.\ \{\#$*lit-of* $a\#\})$ ‘ $\bigcup$(*set* ‘ *snd* ‘ *set* (*get-all-decided-decomposition* M))
⟨*proof*⟩

lemma *all-decomposition-implies-propagated-lits-are-implied*:
assumes *all-decomposition-implies* N (*get-all-decided-decomposition* M)
shows $N \cup \{\{\#$*lit-of* $L\#\}$ $|L.$ *is-decided* $L \wedge L \in$ *set* $M\}$ $\models$*ps* *unmark* M
(**is** *?I* $\models$*ps* *?A*)
⟨*proof*⟩

lemma *all-decomposition-implies-insert-single*:
all-decomposition-implies N M $\Longrightarrow$ *all-decomposition-implies* (*insert* C N) M
⟨*proof*⟩

1.4 Negation of Clauses

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

definition *CNot* :: $'v$ *clause* $\Rightarrow$ $'v$ *clauses* **where**
CNot $\psi = \{\ \{\#-L\#\}\ |\ L.\ \ L \in\#\ \psi\ \}$

lemma *in-CNot-uminus*[*iff*]:
shows $\{\#L\#\} \in$ *CNot* $\psi \longleftrightarrow -L \in\#\ \psi$
⟨*proof*⟩

lemma *CNot-singleton*[*simp*]: *CNot* $\{\#L\#\} = \{\{\#-L\#\}\}$ ⟨*proof*⟩
lemma *CNot-empty*[*simp*]: *CNot* $\{\#\} = \{\}$ ⟨*proof*⟩
lemma *CNot-plus*[*simp*]: *CNot* $(A + B) =$ *CNot* $A \cup$ *CNot* B ⟨*proof*⟩

lemma *CNot-eq-empty*[*iff*]:
CNot $D = \{\} \longleftrightarrow D = \{\#\}$
⟨*proof*⟩

lemma *in-CNot-implies-uminus*:
assumes $L \in\#\ D$
and $M \models$*as* *CNot* D
shows $M \models$*a* $\{\#-L\#\}$ **and** $-L \in$ *lits-of* M
⟨*proof*⟩

lemma *CNot-remdups-mset*[*simp*]:
CNot (*remdups-mset* A) $=$ *CNot* A
⟨*proof*⟩

lemma *Ball-CNot-Ball-mset*[*simp*]:
$(\forall x{\in}CNot\ D.\ P\ x) \longleftrightarrow (\forall L{\in}\#\ D.\ P\ \{\#{-}L\#\})$
⟨*proof*⟩

lemma *consistent-CNot-not*:
assumes *consistent-interp I*
shows $I \models s\ CNot\ \varphi \Longrightarrow \neg I \models \varphi$
⟨*proof*⟩

lemma *total-not-true-cls-true-clss-CNot*:
assumes *total-over-m I* $\{\varphi\}$ **and** $\neg I \models \varphi$
shows $I \models s\ CNot\ \varphi$
⟨*proof*⟩

lemma *total-not-CNot*:
assumes *total-over-m I* $\{\varphi\}$ **and** $\neg I \models s\ CNot\ \varphi$
shows $I \models \varphi$
⟨*proof*⟩

lemma *atms-of-ms-CNot-atms-of*[*simp*]:
atms-of-ms (*CNot C*) = *atms-of C*
⟨*proof*⟩

lemma *true-clss-clss-contradiction-true-clss-cls-false*:
$C \in D \Longrightarrow D \models ps\ CNot\ C \Longrightarrow D \models p\ \{\#\}$
⟨*proof*⟩

lemma *true-annots-CNot-all-atms-defined*:
assumes $M \models as\ CNot\ T$ **and** *a1*: $L \in\#\ T$
shows *atm-of* $L \in$ *atm-of* ‘ *lits-of M*
⟨*proof*⟩

lemma *true-clss-clss-false-left-right*:
assumes $\{\{\#L\#\}\} \cup B \models p\ \{\#\}$
shows $B \models ps\ CNot\ \{\#L\#\}$
⟨*proof*⟩

lemma *true-annots-true-cls-def-iff-negation-in-model*:
$M \models as\ CNot\ C \longleftrightarrow (\forall L \in\#\ C.\ {-L} \in$ *lits-of M*$)$
⟨*proof*⟩

lemma *consistent-CNot-not-tautology*:
consistent-interp M $\Longrightarrow M \models s\ CNot\ D \Longrightarrow \neg$*tautology D*
⟨*proof*⟩

lemma *atms-of-ms-CNot-atms-of-ms*: *atms-of-ms* (*CNot CC*) = *atms-of-ms* {*CC*}
⟨*proof*⟩

lemma *total-over-m-CNot-toal-over-m*[*simp*]:
total-over-m I (*CNot C*) = *total-over-set I* (*atms-of C*)
⟨*proof*⟩

lemma *uminus-lit-swap*: $-(a{::}'a\ literal) = i \longleftrightarrow a = -i$
⟨*proof*⟩

lemma *true-clss-cls-plus-CNot*:
assumes
CC-L: $A \models p\ CC + \{\#L\#\}$ **and**
CNot-CC: $A \models ps\ CNot\ CC$
shows $A \models p\ \{\#L\#\}$
⟨*proof*⟩

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

lemma *true-clss-clss-union-false-true-clss-clss-cnot*:
$A \cup \{B\} \models ps\ \{\{\#\}\} \longleftrightarrow A \models ps\ CNot\ B$
⟨*proof*⟩

lemma *true-annot-remove-hd-if-notin-vars*:
assumes $a \# M' \models a\ D$
and $atm\text{-}of\ (lit\text{-}of\ a) \notin atms\text{-}of\ D$
shows $M' \models a\ D$
⟨*proof*⟩

lemma *true-annot-remove-if-notin-vars*:
assumes $M\ @\ M' \models a\ D$
and $\forall x \in atms\text{-}of\ D.\ x \notin atm\text{-}of\ `\ lits\text{-}of\ M$
shows $M' \models a\ D$
⟨*proof*⟩

lemma *true-annots-remove-if-notin-vars*:
assumes $M\ @\ M' \models as\ D$
and $\forall x \in atms\text{-}of\text{-}ms\ D.\ x \notin atm\text{-}of\ `\ lits\text{-}of\ M$
shows $M' \models as\ D$ ⟨*proof*⟩

lemma *all-variables-defined-not-imply-cnot*:
assumes $\forall s \in atms\text{-}of\text{-}ms\ \{B\}.\ s \in atm\text{-}of\ `\ lits\text{-}of\ A$
and $\neg\ A \models a\ B$
shows $A \models as\ CNot\ B$
⟨*proof*⟩

lemma *CNot-union-mset*[*simp*]:
$CNot\ (A\ \#\cup\ B) = CNot\ A \cup CNot\ B$
⟨*proof*⟩

1.5 Other

abbreviation $no\text{-}dup\ L \equiv distinct\ (map\ (\lambda l.\ atm\text{-}of\ (lit\text{-}of\ l))\ L)$

lemma *no-dup-rev*[*simp*]:
$no\text{-}dup\ (rev\ M) \longleftrightarrow no\text{-}dup\ M$
⟨*proof*⟩

lemma *no-dup-length-eq-card-atm-of-lits-of*:
assumes $no\text{-}dup\ M$
shows $length\ M = card\ (atm\text{-}of\ `\ lits\text{-}of\ M)$
⟨*proof*⟩

lemma *distinctconsistent-interp*:
$\quad$ *no-dup M* $\Longrightarrow$ *consistent-interp* (*lits-of M*)
⟨*proof*⟩

lemma *distinct-get-all-decided-decomposition-no-dup*:
$\quad$ **assumes** $(a, b) \in$ *set* (*get-all-decided-decomposition M*)
$\quad$ **and** *no-dup M*
$\quad$ **shows** *no-dup* (a @ b)
$\quad$ ⟨*proof*⟩

lemma *true-annots-lit-of-notin-skip*:
$\quad$ **assumes** $L \# M \models as\ CNot\ A$
$\quad$ **and** $-$*lit-of* $L \notin\# A$
$\quad$ **and** *no-dup* ($L \# M$)
$\quad$ **shows** $M \models as\ CNot\ A$
⟨*proof*⟩

1.6 Extending Entailments to multisets

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

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

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

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

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

lemma *true-clss-clssm-subsetE*: $N \models psm\ B \Longrightarrow A \subseteq\# B \Longrightarrow N \models psm\ A$
$\quad$ ⟨*proof*⟩

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

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

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

abbreviation *atms-of-msu* **where**
atms-of-msu U $\equiv$ *atms-of-ms* (*set-mset U*)

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

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

2 NOT's CDCL

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

2.1 Auxiliary Lemmas and Measure

lemma *no-dup-cannot-not-lit-and-uminus*:
$no\text{-}dup\ M \Longrightarrow -\ lit\text{-}of\ xa = lit\text{-}of\ x \Longrightarrow x \in set\ M \Longrightarrow xa \notin set\ M$
⟨*proof*⟩

lemma *true-clss-single-iff-incl*:
$I \models s\ single\ `\ B \longleftrightarrow B \subseteq I$
⟨*proof*⟩

lemma *atms-of-ms-single-atm-of*[*simp*]:
$atms\text{-}of\text{-}ms\ \{\{\#lit\text{-}of\ L\#\}\ |L.\ P\ L\} = atm\text{-}of\ `\ \{lit\text{-}of\ L\ |L.\ P\ L\}$
⟨*proof*⟩

lemma *atms-of-uminus-lit-atm-of-lit-of*:
$atms\text{-}of\ \{\#-\ lit\text{-}of\ x.\ x \in\#\ A\#\} = atm\text{-}of\ `\ (lit\text{-}of\ `\ (set\text{-}mset\ A))$
⟨*proof*⟩

lemma *atms-of-ms-single-image-atm-of-lit-of*:
$atms\text{-}of\text{-}ms\ ((\lambda x.\ \{\#lit\text{-}of\ x\#\})\ `\ A) = atm\text{-}of\ `\ (lit\text{-}of\ `\ A)$
⟨*proof*⟩

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

definition μ_C :: *nat* ⇒ *nat* ⇒ *nat list* ⇒ *nat* **where**
$\mu_C\ s\ b\ M \equiv (\sum i{=}0..{<}length\ M.\ M!i * b\widehat{\ }\ (s + i - length\ M))$

lemma μ_C-*nil*[*simp*]:
$\mu_C\ s\ b\ [] = 0$
⟨*proof*⟩

lemma μ_C-*single*[*simp*]:
$\mu_C\ s\ b\ [L] = L * b\ \widehat{}\ (s - Suc\ 0)$
⟨*proof*⟩

lemma *set-sum-atLeastLessThan-add*:
$(\sum i{=}k..{<}k{+}(b{::}nat).\ f\ i) = (\sum i{=}0..{<}b.\ f\ (k{+}\ i))$
⟨*proof*⟩

lemma *set-sum-atLeastLessThan-Suc*:
$(\sum i{=}1..{<}Suc\ j.\ f\ i) = (\sum i{=}0..{<}j.\ f\ (Suc\ i))$
⟨*proof*⟩

lemma μ_C-*cons*:
$\mu_C\ s\ b\ (L\ \#\ M) = L * b\ \widehat{}\ (s - 1 - length\ M) + \mu_C\ s\ b\ M$
⟨*proof*⟩

lemma μ_C-*append*:
assumes $s \geq length\ (M@M')$
shows $\mu_C\ s\ b\ (M@M') = \mu_C\ (s - length\ M')\ b\ M + \mu_C\ s\ b\ M'$

⟨*proof*⟩

lemma μ_C*-cons-non-empty-inf*:
  assumes *M-ge-1*: $\forall i{\in}set\ M.\ i \geq 1$ **and** *M*: $M \neq []$
  shows $\mu_C\ s\ b\ M \geq b\ \hat{}\ (s - length\ M)$
  ⟨*proof*⟩

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

lemma *sum-of-powers*: $0 \leq k \Longrightarrow (k - 1) * (\sum i{=}0..{<}n.\ k\hat{}i) = k\hat{}n - (1::nat)$
  ⟨*proof*⟩

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

lemma μ_C*-bounded-non-degenerated*:
  fixes b ::*nat*
  assumes
    $b > 0$ **and**
    $M \neq []$ **and**
    M-le: $\forall\, i < length\ M.\ M!i < b$ **and**
    $s \geq length\ M$
  shows $\mu_C\ s\ b\ M < b\hat{}s$
⟨*proof*⟩

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

lemma μ_C*-bounded*:
  fixes b ::*nat*
  assumes
    M-le: $\forall\, i < length\ M.\ M!i < b$ **and**
    $s \geq length\ M$
    $b > 0$
  shows $\mu_C\ s\ b\ M < b\ \hat{}\ s$
⟨*proof*⟩

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

lemma μ_C*-base-0*:
  assumes $length\ M \leq s$
  shows $\mu_C\ s\ 0\ M \leq M!0$
⟨*proof*⟩

2.2 Initial definitions

2.2.1 The state

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

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

trail-prepend-trail[*simp*]:
$\bigwedge st\ L.$ *undefined-lit* (*trail st*) (*lit-of L*) $\Longrightarrow$ *trail* (*prepend-trail L st*) = *L* # *trail st*
and
tl-trail[*simp*]: *trail* (*tl-trail S*) = *tl* (*trail S*) **and**
trail-add-cls$_{NOT}$[*simp*]: $\bigwedge st\ C.$ *no-dup* (*trail st*) $\Longrightarrow$ *trail* (*add-cls*$_{NOT}$ *C st*) = *trail st* **and**
trail-remove-cls$_{NOT}$[*simp*]: $\bigwedge st\ C.$ *trail* (*remove-cls*$_{NOT}$ *C st*) = *trail st* **and**

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

function *reduce-trail-to*$_{NOT}$ :: *'a list* ⇒ *'st* ⇒ *'st* **where**
reduce-trail-to$_{NOT}$ *F S* =
(*if length* (*trail S*) = *length F* ∨ *trail S* = [] *then S else reduce-trail-to*$_{NOT}$ *F* (*tl-trail S*))
⟨*proof*⟩
termination ⟨*proof*⟩
declare *reduce-trail-to*$_{NOT}$*.simps*[*simp del*]

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

lemma *reduce-trail-to*$_{NOT}$*-length-ne*[*simp*]:
length (*trail S*) ≠ *length F* $\Longrightarrow$ *trail S* ≠ [] $\Longrightarrow$
reduce-trail-to$_{NOT}$ *F S* = *reduce-trail-to*$_{NOT}$ *F* (*tl-trail S*)
⟨*proof*⟩

lemma *trail-reduce-trail-to*$_{NOT}$*-length-le*:
assumes *length F* > *length* (*trail S*)
shows *trail* (*reduce-trail-to*$_{NOT}$ *F S*) = []
⟨*proof*⟩

lemma *trail-reduce-trail-to*$_{NOT}$*-nil*[*simp*]:
trail (*reduce-trail-to*$_{NOT}$ [] *S*) = []
⟨*proof*⟩

lemma *clauses-reduce-trail-to*$_{NOT}$*-nil*:
clauses (*reduce-trail-to*$_{NOT}$ [] *S*) = *clauses S*
⟨*proof*⟩

lemma *trail-reduce-trail-to*$_{NOT}$*-drop*:
trail (*reduce-trail-to*$_{NOT}$ *F S*) =
(*if length* (*trail S*) ≥ *length F*
then drop (*length* (*trail S*) − *length F*) (*trail S*)
else [])
⟨*proof*⟩

lemma *reduce-trail-to*$_{NOT}$*-skip-beginning*:
  assumes *trail* $S = F' @ F$
  shows *trail* (*reduce-trail-to*$_{NOT}$ F S) $= F$
  ⟨*proof*⟩

lemma *reduce-trail-to*$_{NOT}$*-clauses*[*simp*]:
  clauses (*reduce-trail-to*$_{NOT}$ F S) = *clauses* S
  ⟨*proof*⟩

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

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

lemma *state-eq*$_{NOT}$*-ref*[*simp*]:
  $S \sim S$
  ⟨*proof*⟩

lemma *state-eq*$_{NOT}$*-sym*:
  $S \sim T \longleftrightarrow T \sim S$
  ⟨*proof*⟩

lemma *state-eq*$_{NOT}$*-trans*:
  $S \sim T \Longrightarrow T \sim U \Longrightarrow S \sim U$
  ⟨*proof*⟩

lemma
  shows
    state-eq$_{NOT}$*-trail*: $S \sim T \Longrightarrow$ *trail* S = *trail* T **and**
    state-eq$_{NOT}$*-clauses*: $S \sim T \Longrightarrow$ *clauses* S = *clauses* T
  ⟨*proof*⟩

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

lemma *trail-eq-reduce-trail-to*$_{NOT}$*-eq*:
  trail S = *trail* $T \Longrightarrow$ *trail* (*reduce-trail-to*$_{NOT}$ F S) = *trail* (*reduce-trail-to*$_{NOT}$ F T)
  ⟨*proof*⟩

lemma *reduce-trail-to*$_{NOT}$*-state-eq*$_{NOT}$*-compatible*:
  assumes *ST*: $S \sim T$
  shows *reduce-trail-to*$_{NOT}$ F S ∼ *reduce-trail-to*$_{NOT}$ F T
⟨*proof*⟩

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

lemma *reduce-trail-to*$_{NOT}$*-trail-tl-trail-decomp*[*simp*]:
  trail $S = F' @$ *Decided* K () # $F \Longrightarrow$
    trail (*reduce-trail-to*$_{NOT}$ F (*tl-trail* S)) = F
  ⟨*proof*⟩

end

2.2.2 Definition of the operation

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

end

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

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

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

$\Longrightarrow F \models as\ CNot\ C'$
$\Longrightarrow backjump\text{-}conds\ C\ C'\ L\ S\ T$
$\Longrightarrow backjump\ S\ T$
inductive-cases *backjumpE*: *backjump S T*
end

2.3 DPLL with backjumping

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

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

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

2.3.1 Definition

We define dpll with backjumping:

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

lemmas *dpll-bj-induct* = *dpll-bj.induct*[*split-format*(*complete*)]
thm *dpll-bj-induct*[*OF dpll-with-backjumping-ops-axioms*]
lemma *dpll-bj-all-induct*[*consumes 2*, *case-names decide*$_{NOT}$ *propagate*$_{NOT}$ *backjump*]:
fixes $S\ T$ :: $'st$

assumes
  dpll-bj S T **and**
  inv S
  $\bigwedge L\ T.\ \textit{undefined-lit}\ (\textit{trail}\ S)\ L \Longrightarrow \textit{atm-of}\ L \in \textit{atms-of-msu}\ (\textit{clauses}\ S)$
    $\Longrightarrow T \sim \textit{prepend-trail}\ (\textit{Decided}\ L\ ())\ S$
    $\Longrightarrow P\ S\ T$ **and**
  $\bigwedge C\ L\ T.\ C + \{\#L\#\} \in\#\ \textit{clauses}\ S \Longrightarrow \textit{trail}\ S \models \textit{as CNot}\ C \Longrightarrow \textit{undefined-lit}\ (\textit{trail}\ S)\ L$
    $\Longrightarrow T \sim \textit{prepend-trail}\ (\textit{Propagated}\ L\ ())\ S$
    $\Longrightarrow P\ S\ T$ **and**
  $\bigwedge C\ F'\ K\ F\ L\ C'\ T.\ C \in\#\ \textit{clauses}\ S \Longrightarrow F'\ @\ \textit{Decided}\ K\ ()\ \#\ F \models \textit{as CNot}\ C$
    $\Longrightarrow \textit{trail}\ S = F'\ @\ \textit{Decided}\ K\ ()\ \#\ F$
    $\Longrightarrow \textit{undefined-lit}\ F\ L$
    $\Longrightarrow \textit{atm-of}\ L \in \textit{atms-of-msu}\ (\textit{clauses}\ S) \cup \textit{atm-of}\ `\ (\textit{lits-of}\ (F'\ @\ \textit{Decided}\ K\ ()\ \#\ F))$
    $\Longrightarrow \textit{clauses}\ S \models \textit{pm}\ C' + \{\#L\#\}$
    $\Longrightarrow F \models \textit{as CNot}\ C'$
    $\Longrightarrow T \sim \textit{prepend-trail}\ (\textit{Propagated}\ L\ ())\ (\textit{reduce-trail-to}_{NOT}\ F\ S)$
    $\Longrightarrow P\ S\ T$
shows $P\ S\ T$
⟨*proof*⟩

2.3.2 Basic properties

First, some better suited induction principle **lemma** *dpll-bj-clauses*:
  assumes *dpll-bj* S T **and** *inv* S
  shows $\textit{clauses}\ S = \textit{clauses}\ T$
  ⟨*proof*⟩

No duplicates in the trail **lemma** *dpll-bj-no-dup*:
  assumes *dpll-bj* S T **and** *inv* S
  and *no-dup* (*trail* S)
  shows *no-dup* (*trail* T)
  ⟨*proof*⟩

Valuations **lemma** *dpll-bj-sat-iff*:
  assumes *dpll-bj* S T **and** *inv* S
  shows $I \models \textit{sm clauses}\ S \longleftrightarrow I \models \textit{sm clauses}\ T$
  ⟨*proof*⟩

Clauses **lemma** *dpll-bj-atms-of-ms-clauses-inv*:
  assumes
    dpll-bj S T **and**
    inv S
  shows $\textit{atms-of-msu}\ (\textit{clauses}\ S) = \textit{atms-of-msu}\ (\textit{clauses}\ T)$
  ⟨*proof*⟩

lemma *dpll-bj-atms-in-trail*:
  assumes
    dpll-bj S T **and**
    inv S **and**
    $\textit{atm-of}\ `\ (\textit{lits-of}\ (\textit{trail}\ S)) \subseteq \textit{atms-of-msu}\ (\textit{clauses}\ S)$
  shows $\textit{atm-of}\ `\ (\textit{lits-of}\ (\textit{trail}\ T)) \subseteq \textit{atms-of-msu}\ (\textit{clauses}\ S)$
  ⟨*proof*⟩

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

inv S **and**
atms-of-msu (*clauses S*) ⊆ *A* **and**
atm-of ‘ (*lits-of* (*trail S*)) ⊆ *A*
shows *atm-of* ‘ (*lits-of* (*trail T*)) ⊆ *A*
⟨*proof*⟩

lemma *dpll-bj-all-decomposition-implies-inv*:
assumes
dpll-bj S T **and**
inv: *inv S* **and**
decomp: *all-decomposition-implies-m* (*clauses S*) (*get-all-decided-decomposition* (*trail S*))
shows *all-decomposition-implies-m* (*clauses T*) (*get-all-decided-decomposition* (*trail T*))
⟨*proof*⟩

2.3.3 Termination

Using a proper measure **lemma** *length-get-all-decided-decomposition-append-Decided*:
length (*get-all-decided-decomposition* (F' @ *Decided K* () # *F*)) =
length (*get-all-decided-decomposition* F')
+ *length* (*get-all-decided-decomposition* (*Decided K* () # *F*))
− *1*
⟨*proof*⟩

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

⟨*proof*⟩

lemma *length-get-all-decided-decomposition-length*:
length (*get-all-decided-decomposition M*) ≤ *1* + *length M*
⟨*proof*⟩

lemma *length-in-get-all-decided-decomposition-bounded*:
assumes *i*:*i* ∈ *set* (*trail-weight S*)
shows *i* ≤ *Suc* (*length* (*trail S*))
⟨*proof*⟩

Well-foundedness The bounds are the following:

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

abbreviation *unassigned-lit* :: *'b literal multiset set* ⇒ *'a list* ⇒ *nat* **where**
unassigned-lit N M ≡ *card* (*atms-of-ms N*) − *length M*
lemma *dpll-bj-trail-mes-increasing-prop*:
fixes *M* :: (*'v*, *unit*, *unit*) *ann-literals* **and** *N* :: *'v clauses*

assumes
dpll-bj S T **and**
inv S **and**
NA: *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* **and**
MA: *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite A*
shows μ_C (*1+card* (*atms-of-ms A*)) (*2+card* (*atms-of-ms A*)) (*trail-weight T*)
> μ_C (*1+card* (*atms-of-ms A*)) (*2+card* (*atms-of-ms A*)) (*trail-weight S*)
⟨*proof*⟩

lemma *dpll-bj-trail-mes-decreasing-prop*:
assumes *dpll*: *dpll-bj S T* **and** *inv*: *inv S* **and**
N-A: *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* **and**
M-A: *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-ms A* **and**
nd: *no-dup* (*trail S*) **and**
fin-A: *finite A*
shows (*2+card* (*atms-of-ms A*)) ^ (*1+card* (*atms-of-ms A*))
− μ_C (*1+card* (*atms-of-ms A*)) (*2+card* (*atms-of-ms A*)) (*trail-weight T*)
< (*2+card* (*atms-of-ms A*)) ^ (*1+card* (*atms-of-ms A*))
− μ_C (*1+card* (*atms-of-ms A*)) (*2+card* (*atms-of-ms A*)) (*trail-weight S*)
⟨*proof*⟩

lemma *wf-dpll-bj*:
assumes *fin*: *finite A*
shows *wf* {(*T*, *S*). *dpll-bj S T*
∧ *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* ∧ *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-ms A*
∧ *no-dup* (*trail S*) ∧ *inv S*}
(**is** *wf ?A*)
⟨*proof*⟩

2.3.4 Normal Forms

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

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

1. The *decide* rules tells us that every variable in N has a value.
2. ¬ M ⊨*as* N tells us that there is conflict.
3. There is at least one decision in the trail (otherwise, M is a model of N).
4. Now if we build the clause with all the decision literals of the trail, we can apply the *backjump* rule.

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

theorem *dpll-backjump-final-state*:
fixes *A* :: *'v literal multiset set* **and** *S T* :: *'st*
assumes
atms-of-msu (*clauses S*) ⊆ *atms-of-ms A* **and**
atm-of ‘ *lits-of* (*trail S*) ⊆ *atms-of-ms A* **and**

no-dup (*trail S*) **and**
finite A **and**
inv: *inv S* **and**
n-s: *no-step dpll-bj S* **and**
decomp: *all-decomposition-implies-m* (*clauses S*) (*get-all-decided-decomposition* (*trail S*))
shows *unsatisfiable* (*set-mset* (*clauses S*))
$\vee$ (*trail S* $\models$*asm clauses S* $\wedge$ *satisfiable* (*set-mset* (*clauses S*)))
⟨*proof*⟩

end

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

lemma *rtranclp-dpll-bj-inv*:
assumes *dpll-bj*** *S T* **and** *inv S*
shows *inv T*
⟨*proof*⟩

lemma *rtranclp-dpll-bj-no-dup*:
assumes *dpll-bj*** *S T* **and** *inv S*
and *no-dup* (*trail S*)
shows *no-dup* (*trail T*)
⟨*proof*⟩

lemma *rtranclp-dpll-bj-atms-of-ms-clauses-inv*:
assumes
dpll-bj** *S T* **and** *inv S*
shows *atms-of-msu* (*clauses S*) = *atms-of-msu* (*clauses T*)
⟨*proof*⟩

lemma *rtranclp-dpll-bj-atms-in-trail*:
assumes
dpll-bj** *S T* **and**
inv S **and**
atm-of ' (*lits-of* (*trail S*)) $\subseteq$ *atms-of-msu* (*clauses S*)
shows *atm-of* ' (*lits-of* (*trail T*)) $\subseteq$ *atms-of-msu* (*clauses T*)
⟨*proof*⟩

lemma *rtranclp-dpll-bj-sat-iff*:
assumes *dpll-bj*** *S T* **and** *inv S*
shows *I* $\models$*sm clauses S* $\longleftrightarrow$ *I* $\models$*sm clauses T*
⟨*proof*⟩

lemma *rtranclp-dpll-bj-atms-in-trail-in-set*:
  assumes
    $dpll\text{-}bj^{**}\ S\ T$ **and**
    inv S
    atms-of-msu (*clauses S*) $\subseteq A$ **and**
    atm-of ‘ (*lits-of* (*trail S*)) $\subseteq A$
  shows *atm-of* ‘ (*lits-of* (*trail T*)) $\subseteq A$
  ⟨*proof*⟩

lemma *rtranclp-dpll-bj-all-decomposition-implies-inv*:
  assumes
    $dpll\text{-}bj^{**}\ S\ T$ **and**
    inv S
    all-decomposition-implies-m (*clauses S*) (*get-all-decided-decomposition* (*trail S*))
  shows *all-decomposition-implies-m* (*clauses T*) (*get-all-decided-decomposition* (*trail T*))
  ⟨*proof*⟩

lemma *rtranclp-dpll-bj-inv-incl-dpll-bj-inv-trancl*:
  {(*T*, *S*). $dpll\text{-}bj^{++}\ S\ T$
    ∧ *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* ∧ *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-ms A*
    ∧ *no-dup* (*trail S*) ∧ *inv S*}
     ⊆ {(*T*, *S*). *dpll-bj S T* ∧ *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A*
        ∧ *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-ms A* ∧ *no-dup* (*trail S*) ∧ *inv S*}$^+$
    (**is** *?A* ⊆ $?B^+$)
⟨*proof*⟩

lemma *wf-tranclp-dpll-bj*:
  assumes *fin*: *finite A*
  shows *wf* {(*T*, *S*). $dpll\text{-}bj^{++}\ S\ T$
    ∧ *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* ∧ *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-ms A*
    ∧ *no-dup* (*trail S*) ∧ *inv S*}
  ⟨*proof*⟩

lemma *dpll-bj-sat-ext-iff*:
  dpll-bj S T ⟹ *inv S* ⟹ *I*⊨*sextm clauses S* ⟷ *I*⊨*sextm clauses T*
  ⟨*proof*⟩

lemma *rtranclp-dpll-bj-sat-ext-iff*:
  $dpll\text{-}bj^{**}\ S\ T$ ⟹ *inv S* ⟹ *I*⊨*sextm clauses S* ⟷ *I*⊨*sextm clauses T*
  ⟨*proof*⟩

theorem *full-dpll-backjump-final-state*:
  fixes *A* :: *′v literal multiset set* **and** *S T* :: *′st*
  assumes
    full: *full dpll-bj S T* **and**
    atms-S: *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* **and**
    atms-trail: *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-ms A* **and**
    n-d: *no-dup* (*trail S*) **and**
    finite A **and**
    inv: *inv S* **and**
    decomp: *all-decomposition-implies-m* (*clauses S*) (*get-all-decided-decomposition* (*trail S*))
  shows *unsatisfiable* (*set-mset* (*clauses S*))
  ∨ (*trail T* ⊨*asm clauses S* ∧ *satisfiable* (*set-mset* (*clauses S*)))
⟨*proof*⟩

corollary *full-dpll-backjump-final-state-from-init-state*:
fixes A :: *'v literal multiset set* **and** S T :: *'st*
assumes
full: *full dpll-bj* S T **and**
trail S = [] **and**
clauses S = N **and**
inv S
shows *unsatisfiable* (*set-mset* N) $\vee$ (*trail* T $\models$*asm* N $\wedge$ *satisfiable* (*set-mset* N))
⟨*proof*⟩

lemma *tranclp-dpll-bj-trail-mes-decreasing-prop*:
assumes *dpll*: *dpll-bj*$^{++}$ S T **and** *inv*: *inv* S **and**
N-A: *atms-of-msu* (*clauses* S) $\subseteq$ *atms-of-ms* A **and**
M-A: *atm-of* ‘ *lits-of* (*trail* S) $\subseteq$ *atms-of-ms* A **and**
n-d: *no-dup* (*trail* S) **and**
fin-A: *finite* A
shows (*2+card* (*atms-of-ms* A)) ^ (*1+card* (*atms-of-ms* A))
$-$ μ_C (*1+card* (*atms-of-ms* A)) (*2+card* (*atms-of-ms* A)) (*trail-weight* T)
$<$ (*2+card* (*atms-of-ms* A)) ^ (*1+card* (*atms-of-ms* A))
$-$ μ_C (*1+card* (*atms-of-ms* A)) (*2+card* (*atms-of-ms* A)) (*trail-weight* S)
⟨*proof*⟩

end

2.4 CDCL

2.4.1 Learn and Forget

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

begin
inductive *learn* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
clauses S $\models$*pm* C $\Longrightarrow$ *atms-of* C $\subseteq$ *atms-of-msu* (*clauses* S) $\cup$ *atm-of* ‘ (*lits-of* (*trail* S))
$\Longrightarrow$ *learn-cond* C S
$\Longrightarrow$ T $\sim$ *add-cls*$_{NOT}$ C S
$\Longrightarrow$ *learn* S T
inductive-cases *learn*$_{NOT}$*E*: *learn* S T

lemma *learn-*μ_C*-stable*:
assumes *learn* S T **and** *no-dup* (*trail* S)
shows μ_C A B (*trail-weight* S) = μ_C A B (*trail-weight* T)
⟨*proof*⟩
end

locale *forget-ops* =
dpll-state trail clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$
for

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

lemma *forget-*μ_C*-stable*:
assumes *forget*$_{NOT}$ *S T*
shows μ_C *A B* (*trail-weight S*) = μ_C *A B* (*trail-weight T*)
⟨*proof*⟩
end

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

2.4.2 Definition of CDCL

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

inductive $cdcl_{NOT}$ :: $'st \Rightarrow 'st \Rightarrow bool$ **for** S :: $'st$ **where**
c-dpll-bj: $dpll\text{-}bj\ S\ S' \Longrightarrow cdcl_{NOT}\ S\ S'$ |
c-learn: $learn\ S\ S' \Longrightarrow cdcl_{NOT}\ S\ S'$ |
c-forget$_{NOT}$: $forget_{NOT}\ S\ S' \Longrightarrow cdcl_{NOT}\ S\ S'$

lemma $cdcl_{NOT}$*-all-induct*[*consumes 1*, *case-names dpll-bj learn forget*$_{NOT}$]:
fixes S T :: $'st$
assumes $cdcl_{NOT}\ S\ T$ **and**
dpll: $\bigwedge T.\ dpll\text{-}bj\ S\ T \Longrightarrow P\ S\ T$ **and**
learning:
$\bigwedge C\ T.\ clauses\ S \models pm\ C \Longrightarrow$
$atms\text{-}of\ C \subseteq atms\text{-}of\text{-}msu\ (clauses\ S) \cup atm\text{-}of\ `\ (lits\text{-}of\ (trail\ S)) \Longrightarrow$
$T \sim add\text{-}cls_{NOT}\ C\ S \Longrightarrow$
$P\ S\ T$ **and**
forgetting: $\bigwedge C\ T.\ clauses\ S - replicate\text{-}mset\ (count\ (clauses\ S)\ C)\ C \models pm\ C \Longrightarrow$
$C \in\#\ clauses\ S \Longrightarrow$
$T \sim remove\text{-}cls_{NOT}\ C\ S \Longrightarrow$
$P\ S\ T$
shows $P\ S\ T$
⟨*proof*⟩

lemma $cdcl_{NOT}$*-no-dup*:
assumes
$cdcl_{NOT}\ S\ T$ **and**
inv S **and**
no-dup (*trail* S)
shows *no-dup* (*trail* T)
⟨*proof*⟩

Consistency of the trail **lemma** $cdcl_{NOT}$*-consistent*:
assumes
$cdcl_{NOT}\ S\ T$ **and**
inv S **and**
no-dup (*trail* S)
shows *consistent-interp* (*lits-of* (*trail* T))
⟨*proof*⟩

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

lemma $cdcl_{NOT}$*-atms-of-ms-clauses-decreasing*:
assumes $cdcl_{NOT}\ S\ T$**and** *inv* S **and** *no-dup* (*trail* S)
shows $atms\text{-}of\text{-}msu\ (clauses\ T) \subseteq atms\text{-}of\text{-}msu\ (clauses\ S) \cup atm\text{-}of\ `\ (lits\text{-}of\ (trail\ S))$
⟨*proof*⟩

lemma $cdcl_{NOT}$*-atms-in-trail*:
assumes $cdcl_{NOT}\ S\ T$**and** *inv* S **and** *no-dup* (*trail* S)
and $atm\text{-}of\ `\ (lits\text{-}of\ (trail\ S)) \subseteq atms\text{-}of\text{-}msu\ (clauses\ S)$
shows $atm\text{-}of\ `\ (lits\text{-}of\ (trail\ T)) \subseteq atms\text{-}of\text{-}msu\ (clauses\ S)$
⟨*proof*⟩

lemma $cdcl_{NOT}$*-atms-in-trail-in-set*:
assumes
$cdcl_{NOT}\ S\ T$ **and** *inv* S **and** *no-dup* (*trail* S) **and**

atms-of-msu (*clauses* S) $\subseteq A$ **and**
atm-of ' (*lits-of* (*trail* S)) $\subseteq A$
shows *atm-of* ' (*lits-of* (*trail* T)) $\subseteq A$
⟨*proof*⟩

lemma $cdcl_{NOT}$*-all-decomposition-implies*:
assumes $cdcl_{NOT}$ S T **and** *inv* S **and** *n-d*[*simp*]: *no-dup* (*trail* S) **and**
all-decomposition-implies-m (*clauses* S) (*get-all-decided-decomposition* (*trail* S))
shows
all-decomposition-implies-m (*clauses* T) (*get-all-decided-decomposition* (*trail* T))
⟨*proof*⟩

Extension of models **lemma** $cdcl_{NOT}$*-bj-sat-ext-iff*:
assumes $cdcl_{NOT}$ S T**and** *inv* S **and** *n-d*: *no-dup* (*trail* S)
shows $I \models$*sextm clauses* $S \longleftrightarrow I \models$*sextm clauses* T
⟨*proof*⟩

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

2.5 CDCL with invariant

locale *conflict-driven-clause-learning* =
conflict-driven-clause-learning-ops +
assumes $cdcl_{NOT}$*-inv*: $\bigwedge S\ T.\ cdcl_{NOT}\ S\ T \Longrightarrow inv\ S \Longrightarrow inv\ T$
begin
sublocale *dpll-with-backjumping*
⟨*proof*⟩

lemma *rtranclp-*$cdcl_{NOT}$*-inv*:
$cdcl_{NOT}^{**}\ S\ T \Longrightarrow inv\ S \Longrightarrow inv\ T$
⟨*proof*⟩

lemma *rtranclp-*$cdcl_{NOT}$*-no-dup*:
assumes $cdcl_{NOT}^{**}$ S T **and** *inv* S
and *no-dup* (*trail* S)
shows *no-dup* (*trail* T)
⟨*proof*⟩

lemma *rtranclp-*$cdcl_{NOT}$*-trail-clauses-bound*:
assumes
cdcl: $cdcl_{NOT}^{**}$ S T **and**
inv: *inv* S **and**
n-d: *no-dup* (*trail* S) **and**
atms-clauses-S: *atms-of-msu* (*clauses* S) $\subseteq A$ **and**
atms-trail-S: *atm-of* '(*lits-of* (*trail* S)) $\subseteq A$
shows *atm-of* '(*lits-of* (*trail* T)) $\subseteq A \wedge$ *atms-of-msu* (*clauses* T) $\subseteq A$
⟨*proof*⟩

lemma *rtranclp-*$cdcl_{NOT}$*-all-decomposition-implies*:
assumes $cdcl_{NOT}^{**}$ S T **and** *inv* S **and** *no-dup* (*trail* S) **and**
all-decomposition-implies-m (*clauses* S) (*get-all-decided-decomposition* (*trail* S))
shows
all-decomposition-implies-m (*clauses* T) (*get-all-decided-decomposition* (*trail* T))
⟨*proof*⟩

lemma *rtranclp-*$cdcl_{NOT}$*-bj-sat-ext-iff*:

assumes $cdcl_{NOT}^{**}$ S T**and** *inv* S **and** *no-dup* (*trail* S)
shows $I\models$*sextm clauses* $S \longleftrightarrow I\models$*sextm clauses* T
⟨*proof*⟩

definition $cdcl_{NOT}$*-NOT-all-inv* **where**
$cdcl_{NOT}$*-NOT-all-inv* A $S \longleftrightarrow$ (*finite* $A \land$ *inv* $S \land$ *atms-of-msu* (*clauses* S) $\subseteq$ *atms-of-ms* A
$\land$ *atm-of* ‘ *lits-of* (*trail* S) $\subseteq$ *atms-of-ms* $A \land$ *no-dup* (*trail* S))

lemma $cdcl_{NOT}$*-NOT-all-inv*:
assumes $cdcl_{NOT}^{**}$ S T **and** $cdcl_{NOT}$*-NOT-all-inv* A S
shows $cdcl_{NOT}$*-NOT-all-inv* A T
⟨*proof*⟩

abbreviation *learn-or-forget* **where**
learn-or-forget S $T \equiv (\lambda S\ T.$ *learn* S $T \lor$ $forget_{NOT}$ S T) S T

lemma *rtranclp-learn-or-forget-*$cdcl_{NOT}$:
learn-or-forget** S $T \Longrightarrow cdcl_{NOT}^{**}$ S T
⟨*proof*⟩

lemma *learn-or-forget-dpll-*μ_C:
assumes
l-f: *learn-or-forget*** S T **and**
dpll: *dpll-bj* T U **and**
inv: $cdcl_{NOT}$*-NOT-all-inv* A S
shows (*2*+*card* (*atms-of-ms* A)) ^ (*1*+*card* (*atms-of-ms* A))
$-\ \mu_C$ (*1*+*card* (*atms-of-ms* A)) (*2*+*card* (*atms-of-ms* A)) (*trail-weight* U)
$<$ (*2*+*card* (*atms-of-ms* A)) ^ (*1*+*card* (*atms-of-ms* A))
$-\ \mu_C$ (*1*+*card* (*atms-of-ms* A)) (*2*+*card* (*atms-of-ms* A)) (*trail-weight* S)
(**is** $?\mu$ U $<$ $?\mu$ S)
⟨*proof*⟩

lemma *infinite-*$cdcl_{NOT}$*-exists-learn-and-forget-infinite-chain*:
assumes
$\bigwedge i.$ $cdcl_{NOT}$ (f i) (f(*Suc* i)) **and**
inv: $cdcl_{NOT}$*-NOT-all-inv* A (f *0*)
shows $\exists j.\ \forall i\geq j.$ *learn-or-forget* (f i) (f (*Suc* i))
⟨*proof*⟩

lemma *wf-*$cdcl_{NOT}$*-no-learn-and-forget-infinite-chain*:
assumes
no-infinite-lf: $\bigwedge f\ j.\ \neg\ (\forall i\geq j.$ *learn-or-forget* (f i) (f (*Suc* i)))
shows *wf* $\{(T, S).\ cdcl_{NOT}\ S\ T \land cdcl_{NOT}$*-NOT-all-inv* A $S\}$ (**is** *wf* $\{(T, S).\ cdcl_{NOT}\ S\ T$
$\land\ ?inv\ S\}$)
⟨*proof*⟩

lemma *inv-and-tranclp-cdcl-*$_{NOT}$*-tranclp-*$cdcl_{NOT}$*-and-inv*:
$cdcl_{NOT}^{++}$ S $T \land cdcl_{NOT}$*-NOT-all-inv* A $S \longleftrightarrow (\lambda S\ T.\ cdcl_{NOT}\ S\ T \land cdcl_{NOT}$*-NOT-all-inv* A
$S)^{++}$ S T
(**is** $?A \land ?I \longleftrightarrow ?B$)
⟨*proof*⟩

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

no-infinite-lf: $\bigwedge f\ j.\ \neg\ (\forall\ i{\geq}j.\ \textit{learn-or-forget}\ (f\ i)\ (f\ (\textit{Suc}\ i)))$
shows $\textit{wf}\ \{(T,\ S).\ \textit{cdcl}_{NOT}{}^{++}\ S\ T \land \textit{cdcl}_{NOT}\textit{-NOT-all-inv}\ A\ S\}$
⟨*proof*⟩

lemma $\textit{cdcl}_{NOT}$*-final-state*:
assumes
n-s: *no-step* $\textit{cdcl}_{NOT}$ S **and**
inv: $\textit{cdcl}_{NOT}\textit{-NOT-all-inv}\ A\ S$ **and**
decomp: *all-decomposition-implies-m* (*clauses* S) (*get-all-decided-decomposition* (*trail* S))
shows *unsatisfiable* (*set-mset* (*clauses* S))
$\lor$ (*trail* S $\models$*asm clauses* S $\land$ *satisfiable* (*set-mset* (*clauses* S)))
⟨*proof*⟩

lemma *full-*$\textit{cdcl}_{NOT}$*-final-state*:
assumes
full: *full* $\textit{cdcl}_{NOT}$ $S\ T$ **and**
inv: $\textit{cdcl}_{NOT}\textit{-NOT-all-inv}\ A\ S$ **and**
n-d: *no-dup* (*trail* S) **and**
decomp: *all-decomposition-implies-m* (*clauses* S) (*get-all-decided-decomposition* (*trail* S))
shows *unsatisfiable* (*set-mset* (*clauses* T))
$\lor$ (*trail* T $\models$*asm clauses* T $\land$ *satisfiable* (*set-mset* (*clauses* T)))
⟨*proof*⟩

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

2.6 Termination

2.6.1 Restricting learn and forget

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

lemma $\textit{cdcl}_{NOT}$*-learn-all-induct*[*consumes 1*, *case-names dpll-bj learn* $\textit{forget}_{NOT}$]:
fixes $S\ T$ :: *'st*
assumes $\textit{cdcl}_{NOT}$ $S\ T$ **and**
dpll: $\bigwedge T.\ \textit{dpll-bj}\ S\ T \Longrightarrow P\ S\ T$ **and**
learning:
$\bigwedge C\ F\ K\ F'\ C'\ L\ T.\ \textit{clauses}\ S \models\!\textit{pm}\ C$
$\Longrightarrow \textit{atms-of}\ C \subseteq \textit{atms-of-msu}\ (\textit{clauses}\ S) \cup \textit{atm-of}\ `\ (\textit{lits-of}\ (\textit{trail}\ S))$

$\Longrightarrow$ *distinct-mset* $C \Longrightarrow \neg$ *tautology* $C \Longrightarrow$ *learn-restrictions* C S
$\Longrightarrow$ *trail* $S = F'$ @ *Decided* K () # $F \Longrightarrow C = C' + \{\#L\#\} \Longrightarrow F \models as$ *CNot* C'
$\Longrightarrow C' + \{\#L\#\} \notin\#$ *clauses* $S \Longrightarrow T \sim$ *add-cls*$_{NOT}$ C S
$\Longrightarrow P$ S T **and**
forgetting: $\bigwedge C$ T. *clauses* S $-$ *replicate-mset* (*count* (*clauses* S) C) $C \models pm$ C
$\Longrightarrow C \in\#$ *clauses* S
$\Longrightarrow \neg(\exists F'$ F K L. *trail* $S = F'$ @ *Decided* K () # $F \wedge F \models as$ *CNot* $(C - \{\#L\#\}))$
$\Longrightarrow T \sim$ *remove-cls*$_{NOT}$ C S
$\Longrightarrow$ *forget-restrictions* C $S \Longrightarrow P$ S T
shows P S T
⟨*proof*⟩

lemma *rtranclp-cdcl*$_{NOT}$*-inv*:
cdcl$_{NOT}$** S $T \Longrightarrow$ *inv* $S \Longrightarrow$ *inv* T
⟨*proof*⟩

lemma *learn-always-simple-clauses*:
assumes
learn: *learn* S T **and**
n-d: *no-dup* (*trail* S)
shows *set-mset* (*clauses* T $-$ *clauses* S)
$\subseteq$ *simple-clss* (*atms-of-msu* (*clauses* S) $\cup$ *atm-of* ' *lits-of* (*trail* S))
⟨*proof*⟩

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

lemma *conflicting-bj-clss-remove-cls*$_{NOT}$[*simp*]:
conflicting-bj-clss (*remove-cls*$_{NOT}$ C S) = *conflicting-bj-clss* S $-$ $\{C\}$
⟨*proof*⟩

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

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

lemma *conflicting-bj-clss-incl-clauses*:
conflicting-bj-clss $S \subseteq$ *set-mset* (*clauses* S)
⟨*proof*⟩

lemma *finite-conflicting-bj-clss*[*simp*]:
finite (*conflicting-bj-clss* S)

$\langle proof \rangle$

lemma *learn-conflicting-increasing*:
no-dup (*trail S*) $\Longrightarrow$ *learn S T* $\Longrightarrow$ *conflicting-bj-clss S* $\subseteq$ *conflicting-bj-clss T*
$\langle proof \rangle$

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

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

lemma *do-not-forget-before-backtrack-rule-clause-learned-clause-untouched*:
assumes $forget_{NOT}$ *S T*
shows *conflicting-bj-clss S* = *conflicting-bj-clss T*
$\langle proof \rangle$

lemma *forget-*μ_L*-decrease*:
assumes $forget_{NOT}$: $forget_{NOT}$ *S T*
shows (μ_L *b T*, μ_L *b S*) $\in$ *less-than* <*lex*> *less-than*
$\langle proof \rangle$

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

lemma *set-insert-neq*:
$A \neq$ *insert a A* $\longleftrightarrow$ $a \notin A$
$\langle proof \rangle$

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

We have to assume the following:

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

definition μ_{CDCL} **where**
μ_{CDCL} *A T* $\equiv$ ((*2*+*card* (*atms-of-ms A*)) ^ (*1*+*card* (*atms-of-ms A*))
$-$ μ_C (*1*+*card* (*atms-of-ms A*)) (*2*+*card* (*atms-of-ms A*)) (*trail-weight T*),
conflicting-bj-clss-yet (*card* (*atms-of-ms A*)) *T*, *card* (*set-mset* (*clauses T*)))
lemma $cdcl_{NOT}$*-decreasing-measure*:
assumes
$cdcl_{NOT}$ *S T* **and**
inv: *inv S* **and**

atm-clss: *atms-of-msu* (*clauses* S) $\subseteq$ *atms-of-ms* A **and**
atm-lits: *atm-of* ‘ *lits-of* (*trail* S) $\subseteq$ *atms-of-ms* A **and**
n-d: *no-dup* (*trail* S) **and**
fin-A: *finite* A
shows (μ_{CDCL} A T, μ_{CDCL} A S)
$\in$ *less-than* <*lex*> (*less-than* <*lex*> *less-than*)
⟨*proof*⟩

lemma *wf-cdcl*$_{NOT}$*-restricted-learning*:
assumes *finite* A
shows *wf* {(T, S).
(*atms-of-msu* (*clauses* S) $\subseteq$ *atms-of-ms* A $\wedge$ *atm-of* ‘ *lits-of* (*trail* S) $\subseteq$ *atms-of-ms* A
$\wedge$ *no-dup* (*trail* S)
$\wedge$ *inv* S)
$\wedge$ *cdcl*$_{NOT}$ S T }
⟨*proof*⟩

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

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

lemma *cdcl*$_{NOT}$*-decreasing-measure'*:
assumes
cdcl$_{NOT}$ S T **and**
inv: *inv* S **and**
atms-clss: *atms-of-msu* (*clauses* S) $\subseteq$ *atms-of-ms* A **and**
atms-trail: *atm-of* ‘ *lits-of* (*trail* S) $\subseteq$ *atms-of-ms* A **and**
n-d: *no-dup* (*trail* S) **and**
fin-A: *finite* A
shows $\mu_{CDCL}{}'$ A T $<$ $\mu_{CDCL}{}'$ A S
⟨*proof*⟩

lemma *cdcl*$_{NOT}$*-clauses-bound*:
assumes
cdcl$_{NOT}$ S T **and**
inv S **and**
atms-of-msu (*clauses* S) $\subseteq$ A **and**
atm-of ‘(*lits-of* (*trail* S)) $\subseteq$ A **and**
n-d: *no-dup* (*trail* S) **and**
fin-A[*simp*]: *finite* A
shows *set-mset* (*clauses* T) $\subseteq$ *set-mset* (*clauses* S) $\cup$ *simple-clss* A
⟨*proof*⟩

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

n-d: *no-dup* (*trail* S) **and**
finite: *finite* A
shows *set-mset* (*clauses* T) $\subseteq$ *set-mset* (*clauses* S) $\cup$ *simple-clss* A
⟨*proof*⟩

lemma *rtranclp-cdcl*$_{NOT}$*-card-clauses-bound*:
assumes
cdcl$_{NOT}$** S T **and**
inv S **and**
atms-of-msu (*clauses* S) $\subseteq A$ **and**
atm-of '(*lits-of* (*trail* S)) $\subseteq A$ **and**
n-d: *no-dup* (*trail* S) **and**
finite: *finite* A
shows *card* (*set-mset* (*clauses* T)) $\leq$ *card* (*set-mset* (*clauses* S)) + *3* ^ (*card* A)
⟨*proof*⟩

lemma *rtranclp-cdcl*$_{NOT}$*-card-clauses-bound'*:
assumes
cdcl$_{NOT}$** S T **and**
inv S **and**
atms-of-msu (*clauses* S) $\subseteq A$ **and**
atm-of '(*lits-of* (*trail* S)) $\subseteq A$ **and**
n-d: *no-dup* (*trail* S) **and**
finite: *finite* A
shows *card* $\{C|C.\ C \in\#$ *clauses* $T \wedge$ (*tautology* $C \vee \neg$*distinct-mset* C)$\}$
$\leq$ *card* $\{C|C.\ C\in\#$ *clauses* $S \wedge$ (*tautology* $C \vee \neg$*distinct-mset* C)$\}$ + *3* ^ (*card* A)
(**is** *card* *?T* $\leq$ *card* *?S* + -)
⟨*proof*⟩

lemma *rtranclp-cdcl*$_{NOT}$*-card-simple-clauses-bound*:
assumes
cdcl$_{NOT}$** S T **and**
inv S **and**
atms-of-msu (*clauses* S) $\subseteq A$ **and**
atm-of '(*lits-of* (*trail* S)) $\subseteq A$ **and**
n-d: *no-dup* (*trail* S) **and**
finite: *finite* A
shows *card* (*set-mset* (*clauses* T))
$\leq$ *card* $\{C.\ C \in\#$ *clauses* $S \wedge$ (*tautology* $C \vee \neg$*distinct-mset* C)$\}$ + *3* ^ (*card* A)
(**is** *card* *?T* $\leq$ *card* *?S* + -)
⟨*proof*⟩

definition μ_{CDCL}*'-bound* :: *'v literal multiset set* $\Rightarrow$ *'st* $\Rightarrow$ *nat* **where**
μ_{CDCL}*'-bound* A S =
((*2* + *card* (*atms-of-ms* A)) ^ (*1* + *card* (*atms-of-ms* A))) $*$ (*1* + *3* ^ *card* (*atms-of-ms* A)) $*$ *2*
+ *2*$*$*3* ^ (*card* (*atms-of-ms* A))
+ *card* $\{C.\ C \in\#$ *clauses* $S \wedge$ (*tautology* $C \vee \neg$*distinct-mset* C)$\}$ + *3* ^ (*card* (*atms-of-ms* A))

lemma μ_{CDCL}*'-bound-reduce-trail-to*$_{NOT}$[*simp*]:
μ_{CDCL}*'-bound* A (*reduce-trail-to*$_{NOT}$ M S) = μ_{CDCL}*'-bound* A S
⟨*proof*⟩

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

$cdcl_{NOT}^{**}$ *S T* **and**
inv S **and**
atms-of-msu (*clauses S*) $\subseteq$ *atms-of-ms A* **and**
atm-of ‘(*lits-of* (*trail S*)) $\subseteq$ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite* (*atms-of-ms A*) **and**
U: *U* $\sim$ *reduce-trail-to*$_{NOT}$ *M T*
shows $\mu_{CDCL}{}'$ *A U* $\leq$ $\mu_{CDCL}{}'$*-bound A S*
⟨*proof*⟩

lemma *rtranclp-cdcl*$_{NOT}$*-*$\mu_{CDCL}{}'$*-bound*:
assumes
$cdcl_{NOT}^{**}$ *S T* **and**
inv S **and**
atms-of-msu (*clauses S*) $\subseteq$ *atms-of-ms A* **and**
atm-of ‘(*lits-of* (*trail S*)) $\subseteq$ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite* (*atms-of-ms A*)
shows $\mu_{CDCL}{}'$ *A T* $\leq$ $\mu_{CDCL}{}'$*-bound A S*
⟨*proof*⟩

lemma *rtranclp-*$\mu_{CDCL}{}'$*-bound-decreasing*:
assumes
$cdcl_{NOT}^{**}$ *S T* **and**
inv S **and**
atms-of-msu (*clauses S*) $\subseteq$ *atms-of-ms A* **and**
atm-of ‘(*lits-of* (*trail S*)) $\subseteq$ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite[*simp*]: *finite* (*atms-of-ms A*)
shows $\mu_{CDCL}{}'$*-bound A T* $\leq$ $\mu_{CDCL}{}'$*-bound A S*
⟨*proof*⟩

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

2.7 CDCL with restarts

2.7.1 Definition

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

end

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

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

lemma *cdcl*$_{NOT}$*-iff-cdcl*$_{NOT}$*-raw-restart-no-restarts*:
cdcl$_{NOT}$ *S T* ⟷ *restart-ops.cdcl*$_{NOT}$*-raw-restart cdcl*$_{NOT}$ (λ- -. *False*) *S T*
(**is** *?C S T* ⟷ *?R S T*)
⟨*proof*⟩

lemma *cdcl*$_{NOT}$*-cdcl*$_{NOT}$*-raw-restart*:
cdcl$_{NOT}$ *S T* ⟹ *restart-ops.cdcl*$_{NOT}$*-raw-restart cdcl*$_{NOT}$ *restart S T*
⟨*proof*⟩
end

2.7.2 Increasing restarts

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

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

locale *cdcl*$_{NOT}$*-increasing-restarts-ops* =
restart-ops cdcl$_{NOT}$ *restart* **for**
restart :: *'st* ⇒ *'st* ⇒ *bool* **and**
cdcl$_{NOT}$ :: *'st* ⇒ *'st* ⇒ *bool* +
fixes
f :: *nat* ⇒ *nat* **and**
bound-inv :: *'bound* ⇒ *'st* ⇒ *bool* **and**
μ :: *'bound* ⇒ *'st* ⇒ *nat* **and**
cdcl$_{NOT}$*-inv* :: *'st* ⇒ *bool* **and**
μ-bound :: *'bound* ⇒ *'st* ⇒ *nat*
assumes
f: *unbounded f* **and**
f-ge-1:⋀*n*. *n*≥*1* ⟹ *f n* ≠ *0* **and**
bound-inv: ⋀*A S T*. *cdcl*$_{NOT}$*-inv S* ⟹ *bound-inv A S* ⟹ *cdcl*$_{NOT}$ *S T* ⟹ *bound-inv A T* **and**

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

lemma $cdcl_{NOT}\text{-}cdcl_{NOT}\text{-}inv$:
assumes
$(cdcl_{NOT}\widehat{\ }\widehat{\ }n)\ S\ T$ **and**
$cdcl_{NOT}\text{-}inv\ S$
shows $cdcl_{NOT}\text{-}inv\ T$
⟨*proof*⟩

lemma $cdcl_{NOT}\text{-}bound\text{-}inv$:
assumes
$(cdcl_{NOT}\widehat{\ }\widehat{\ }n)\ S\ T$ **and**
$cdcl_{NOT}\text{-}inv\ S$
$bound\text{-}inv\ A\ S$
shows $bound\text{-}inv\ A\ T$
⟨*proof*⟩

lemma $rtranclp\text{-}cdcl_{NOT}\text{-}cdcl_{NOT}\text{-}inv$:
assumes
$cdcl_{NOT}^{**}\ S\ T$ **and**
$cdcl_{NOT}\text{-}inv\ S$
shows $cdcl_{NOT}\text{-}inv\ T$
⟨*proof*⟩

lemma $rtranclp\text{-}cdcl_{NOT}\text{-}bound\text{-}inv$:
assumes
$cdcl_{NOT}^{**}\ S\ T$ **and**
$bound\text{-}inv\ A\ S$ **and**
$cdcl_{NOT}\text{-}inv\ S$
shows $bound\text{-}inv\ A\ T$
⟨*proof*⟩

lemma $cdcl_{NOT}\text{-}comp\text{-}n\text{-}le$:
assumes
$(cdcl_{NOT}\widehat{\ }\widehat{\ }(Suc\ n))\ S\ T$ **and**
$bound\text{-}inv\ A\ S$
$cdcl_{NOT}\text{-}inv\ S$
shows $\mu\ A\ T < \mu\ A\ S - n$
⟨*proof*⟩

lemma $wf\text{-}cdcl_{NOT}$:
$wf\ \{(T,\ S).\ cdcl_{NOT}\ S\ T \wedge cdcl_{NOT}\text{-}inv\ S \wedge bound\text{-}inv\ A\ S\}$ (**is** *wf ?A*)
⟨*proof*⟩

lemma *rtranclp-cdcl*$_{NOT}$*-measure*:
  assumes
    cdcl$_{NOT}$** *S T* **and**
    bound-inv A S **and**
    cdcl$_{NOT}$*-inv S*
  shows μ *A T* $\leq$ μ *A S*
  ⟨*proof*⟩

lemma *cdcl*$_{NOT}$*-comp-bounded*:
  assumes
    bound-inv A S **and** *cdcl*$_{NOT}$*-inv S* **and** $m \geq 1+\mu$ *A S*
  shows ¬(*cdcl*$_{NOT}$ $\widehat{}\widehat{}$ *m*) *S T*
  ⟨*proof*⟩

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

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

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

lemma *cdcl*$_{NOT}$*-restart-cdcl*$_{NOT}$*-raw-restart*:
  cdcl$_{NOT}$*-restart S T* ⟹ *cdcl*$_{NOT}$*-raw-restart*** (*fst S*) (*fst T*)
⟨*proof*⟩

lemma *cdcl*$_{NOT}$*-with-restart-bound-inv*:
  assumes
    cdcl$_{NOT}$*-restart S T* **and**
    bound-inv A (*fst S*) **and**
    cdcl$_{NOT}$*-inv* (*fst S*)
  shows *bound-inv A* (*fst T*)
  ⟨*proof*⟩

lemma *cdcl*$_{NOT}$*-with-restart-cdcl*$_{NOT}$*-inv*:
  assumes
    cdcl$_{NOT}$*-restart S T* **and**
    cdcl$_{NOT}$*-inv* (*fst S*)
  shows *cdcl*$_{NOT}$*-inv* (*fst T*)
  ⟨*proof*⟩

lemma *rtranclp-cdcl*$_{NOT}$*-with-restart-cdcl*$_{NOT}$*-inv*:
  assumes
    cdcl$_{NOT}$*-restart*** *S T* **and**
    cdcl$_{NOT}$*-inv* (*fst S*)
  shows *cdcl*$_{NOT}$*-inv* (*fst T*)
  ⟨*proof*⟩

lemma *rtranclp-cdcl*$_{NOT}$*-with-restart-bound-inv*:
  assumes
    cdcl$_{NOT}$*-restart*** *S T* **and**
    cdcl$_{NOT}$*-inv* (*fst S*) **and**

bound-inv A (*fst S*)
shows *bound-inv A* (*fst T*)
⟨*proof*⟩

lemma $cdcl_{NOT}$*-with-restart-increasing-number*:
$cdcl_{NOT}$*-restart S T* $\Longrightarrow$ *snd T* = *1* + *snd S*
⟨*proof*⟩
end

locale $cdcl_{NOT}$*-increasing-restarts* =
$cdcl_{NOT}$*-increasing-restarts-ops restart* $cdcl_{NOT}$ *f bound-inv* μ $cdcl_{NOT}$*-inv* μ*-bound*
for
trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *ann-literals* **and**
clauses :: *'st* ⇒ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *ann-literal* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒ *'st* **and**
add-cls$_{NOT}$ *remove-cls*$_{NOT}$:: *'v clause* ⇒ *'st* ⇒ *'st* **and**
f :: *nat* ⇒ *nat* **and**
restart :: *'st* ⇒ *'st* ⇒ *bool* **and**
bound-inv :: *'bound* ⇒ *'st* ⇒ *bool* **and**
μ :: *'bound* ⇒ *'st* ⇒ *nat* **and**
$cdcl_{NOT}$ :: *'st* ⇒ *'st* ⇒ *bool* **and**
$cdcl_{NOT}$*-inv* :: *'st* ⇒ *bool* **and**
μ*-bound* :: *'bound* ⇒ *'st* ⇒ *nat* +
assumes
measure-bound: $\bigwedge$*A T V n.* $cdcl_{NOT}$*-inv T* $\Longrightarrow$ *bound-inv A T*
$\Longrightarrow$ $cdcl_{NOT}$*-restart* (*T*, *n*) (*V*, *Suc n*) $\Longrightarrow$ μ *A V* $\leq$ μ*-bound A T* **and**
$cdcl_{NOT}$*-raw-restart-*μ*-bound*:
$cdcl_{NOT}$*-restart* (*T*, *a*) (*V*, *b*) $\Longrightarrow$ $cdcl_{NOT}$*-inv T* $\Longrightarrow$ *bound-inv A T*
$\Longrightarrow$ μ*-bound A V* $\leq$ μ*-bound A T*
begin

lemma *rtranclp-*$cdcl_{NOT}$*-raw-restart-*μ*-bound*:
$cdcl_{NOT}$*-restart*** (*T*, *a*) (*V*, *b*) $\Longrightarrow$ $cdcl_{NOT}$*-inv T* $\Longrightarrow$ *bound-inv A T*
$\Longrightarrow$ μ*-bound A V* $\leq$ μ*-bound A T*
⟨*proof*⟩

lemma $cdcl_{NOT}$*-raw-restart-measure-bound*:
$cdcl_{NOT}$*-restart* (*T*, *a*) (*V*, *b*) $\Longrightarrow$ $cdcl_{NOT}$*-inv T* $\Longrightarrow$ *bound-inv A T*
$\Longrightarrow$ μ *A V* $\leq$ μ*-bound A T*
⟨*proof*⟩

lemma *rtranclp-*$cdcl_{NOT}$*-raw-restart-measure-bound*:
$cdcl_{NOT}$*-restart*** (*T*, *a*) (*V*, *b*) $\Longrightarrow$ $cdcl_{NOT}$*-inv T* $\Longrightarrow$ *bound-inv A T*
$\Longrightarrow$ μ *A V* $\leq$ μ*-bound A T*
⟨*proof*⟩

lemma *wf-*$cdcl_{NOT}$*-restart*:
wf {(*T*, *S*). $cdcl_{NOT}$*-restart S T* ∧ $cdcl_{NOT}$*-inv* (*fst S*)} (**is** *wf ?A*)
⟨*proof*⟩

lemma $cdcl_{NOT}$*-restart-steps-bigger-than-bound*:
assumes
$cdcl_{NOT}$*-restart S T* **and**
bound-inv A (*fst S*) **and**

$cdcl_{NOT}$*-inv* (*fst S*) **and**
f (*snd S*) > μ*-bound A* (*fst S*)
shows *full1* $cdcl_{NOT}$ (*fst S*) (*fst T*)
⟨*proof*⟩

lemma *rtranclp-*$cdcl_{NOT}$*-with-inv-inv-rtranclp-*$cdcl_{NOT}$:
assumes
inv: $cdcl_{NOT}$*-inv S* **and**
binv: *bound-inv A S*
shows ($\lambda S\ T.\ cdcl_{NOT}\ S\ T \wedge cdcl_{NOT}\text{-}inv\ S \wedge bound\text{-}inv\ A\ S)^{**}\ S\ T \longleftrightarrow cdcl_{NOT}{}^{**}\ S\ T$
(**is** $?A^{**}\ S\ T \longleftrightarrow ?B^{**}\ S\ T$)
⟨*proof*⟩

lemma *no-step-*$cdcl_{NOT}$*-restart-no-step-*$cdcl_{NOT}$:
assumes
n-s: *no-step* $cdcl_{NOT}$*-restart S* **and**
inv: $cdcl_{NOT}$*-inv* (*fst S*) **and**
binv: *bound-inv A* (*fst S*)
shows *no-step* $cdcl_{NOT}$ (*fst S*)
⟨*proof*⟩

end

2.8 Merging backjump and learning

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

inductive $cdcl_{NOT}$*-merged-bj-learn* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
$cdcl_{NOT}$*-merged-bj-learn-decide*$_{NOT}$: *decide*$_{NOT}$ *S S'* ⟹ $cdcl_{NOT}$*-merged-bj-learn S S'* |

$cdcl_{NOT}$*-merged-bj-learn-propagate*$_{NOT}$: *propagate*$_{NOT}$ S S' $\Longrightarrow$ $cdcl_{NOT}$*-merged-bj-learn* S S' |
$cdcl_{NOT}$*-merged-bj-learn-backjump-l*: *backjump-l* S S' $\Longrightarrow$ $cdcl_{NOT}$*-merged-bj-learn* S S' |
$cdcl_{NOT}$*-merged-bj-learn-forget*$_{NOT}$: *forget*$_{NOT}$ S S' $\Longrightarrow$ $cdcl_{NOT}$*-merged-bj-learn* S S'

lemma $cdcl_{NOT}$*-merged-bj-learn-no-dup-inv*:
  $cdcl_{NOT}$*-merged-bj-learn* S T $\Longrightarrow$ *no-dup* (*trail* S) $\Longrightarrow$ *no-dup* (*trail* T)
  ⟨*proof*⟩
end

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

sublocale *dpll-with-backjumping-ops trail clauses prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$
  propagate-conds inv backjump-conds
⟨*proof*⟩

end

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

$add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$:: $'v$ *clause* $\Rightarrow$ $'st$ $\Rightarrow$ $'st$ **and**
propagate-conds :: ($'v$, *unit*, *unit*) *ann-literal* $\Rightarrow$ $'st$ $\Rightarrow$ *bool* **and**
inv :: $'st$ $\Rightarrow$ *bool* **and**
forget-conds :: $'v$ *clause* $\Rightarrow$ $'st$ $\Rightarrow$ *bool* **and**
backjump-l-cond :: $'v$ *clause* $\Rightarrow$ $'v$ *clause* $\Rightarrow$ $'v$ *literal* $\Rightarrow$ $'st$ $\Rightarrow$ *bool*
begin

sublocale *conflict-driven-clause-learning-ops trail clauses prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ *propagate-conds inv backjump-conds* λC -. *distinct-mset* $C \land \neg$*tautology* C *forget-conds*
⟨*proof*⟩
end

locale $cdcl_{NOT}$*-merge-bj-learn* =
$cdcl_{NOT}$*-merge-bj-learn-proxy2 trail clauses prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
propagate-conds inv forget-conds backjump-l-cond
for
trail :: $'st$ $\Rightarrow$ ($'v$, *unit*, *unit*) *ann-literals* **and**
clauses :: $'st$ $\Rightarrow$ $'v$ *clauses* **and**
prepend-trail :: ($'v$, *unit*, *unit*) *ann-literal* $\Rightarrow$ $'st$ $\Rightarrow$ $'st$ **and**
tl-trail :: $'st$ $\Rightarrow$ $'st$ **and**
$add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$:: $'v$ *clause* $\Rightarrow$ $'st$ $\Rightarrow$ $'st$ **and**
propagate-conds :: ($'v$, *unit*, *unit*) *ann-literal* $\Rightarrow$ $'st$ $\Rightarrow$ *bool* **and**
inv :: $'st$ $\Rightarrow$ *bool* **and**
forget-conds :: $'v$ *clause* $\Rightarrow$ $'st$ $\Rightarrow$ *bool* **and**
backjump-l-cond :: $'v$ *clause* $\Rightarrow$ $'v$ *clause* $\Rightarrow$ $'v$ *literal* $\Rightarrow$ $'st$ $\Rightarrow$ *bool* +
assumes
dpll-bj-inv: $\bigwedge S\ T$. *dpll-bj* S $T \Longrightarrow$ *inv* $S \Longrightarrow$ *inv* T **and**
learn-inv: $\bigwedge S\ T$. *learn* S $T \Longrightarrow$ *inv* $S \Longrightarrow$ *inv* T
begin

interpretation $cdcl_{NOT}$:
conflict-driven-clause-learning trail clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
propagate-conds inv backjump-conds λC -. *distinct-mset* $C \land \neg$*tautology* C *forget-conds*
⟨*proof*⟩

lemma *backjump-l-learn-backjump*:
assumes *bt*: *backjump-l* S T **and** *inv*: *inv* S **and** *n-d*: *no-dup* (*trail* S)
shows $\exists\, C'\ L$. *learn* S ($add\text{-}cls_{NOT}$ (C' + {#L#}) S)
$\land$ *backjump* ($add\text{-}cls_{NOT}$ (C' + {#L#}) S) T
$\land$ *atms-of* (C' + {#L#}) $\subseteq$ *atms-of-msu* (*clauses* S) $\cup$ *atm-of* ' (*lits-of* (*trail* S))
⟨*proof*⟩

lemma $cdcl_{NOT}$*-merged-bj-learn-is-tranclp-*$cdcl_{NOT}$:
$cdcl_{NOT}$*-merged-bj-learn* S $T \Longrightarrow$ *inv* $S \Longrightarrow$ *no-dup* (*trail* S) $\Longrightarrow$ $cdcl_{NOT}{}^{++}$ S T
⟨*proof*⟩

lemma *rtranclp-*$cdcl_{NOT}$*-merged-bj-learn-is-rtranclp-*$cdcl_{NOT}$*-and-inv*:
$cdcl_{NOT}$*-merged-bj-learn*** S $T \Longrightarrow$ *inv* $S \Longrightarrow$ *no-dup* (*trail* S) $\Longrightarrow$ $cdcl_{NOT}{}^{**}$ S $T \land$ *inv* T
⟨*proof*⟩

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

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

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

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

lemma *cdcl*$_{NOT}$*-decreasing-measure'*:
assumes
cdcl$_{NOT}$*-merged-bj-learn* S T **and**
inv: *inv* S **and**
atm-clss: *atms-of-msu* (*clauses* S) ⊆ *atms-of-ms* A **and**
atm-trail: *atm-of* ‘ *lits-of* (*trail* S) ⊆ *atms-of-ms* A **and**
n-d: *no-dup* (*trail* S) **and**
fin-A: *finite* A
shows $\mu_{CDCL}{}'$*-merged* A T < $\mu_{CDCL}{}'$*-merged* A S
⟨*proof*⟩

lemma *wf-cdcl*$_{NOT}$*-merged-bj-learn*:
assumes
fin-A: *finite* A
shows *wf* {(T, S).
(*inv* S ∧ *atms-of-msu* (*clauses* S) ⊆ *atms-of-ms* A ∧ *atm-of* ‘ *lits-of* (*trail* S) ⊆ *atms-of-ms* A
∧ *no-dup* (*trail* S))
∧ *cdcl*$_{NOT}$*-merged-bj-learn* S T}
⟨*proof*⟩

lemma *tranclp-cdcl*$_{NOT}$*-cdcl*$_{NOT}$*-tranclp*:
assumes
cdcl$_{NOT}$*-merged-bj-learn*$^{++}$ S T **and**
inv: *inv* S **and**
atm-clss: *atms-of-msu* (*clauses* S) ⊆ *atms-of-ms* A **and**
atm-trail: *atm-of* ‘ *lits-of* (*trail* S) ⊆ *atms-of-ms* A **and**
n-d: *no-dup* (*trail* S) **and**
fin-A[*simp*]: *finite* A
shows (T, S) ∈ {(T, S).
(*inv* S ∧ *atms-of-msu* (*clauses* S) ⊆ *atms-of-ms* A ∧ *atm-of* ‘ *lits-of* (*trail* S) ⊆ *atms-of-ms* A
∧ *no-dup* (*trail* S))
∧ *cdcl*$_{NOT}$*-merged-bj-learn* S T}$^+$ (**is** - ∈ *?P*$^+$)
⟨*proof*⟩

lemma *wf-tranclp-cdcl*$_{NOT}$*-merged-bj-learn*:
assumes *finite* A
shows *wf* {(T, S).
(*inv* S ∧ *atms-of-msu* (*clauses* S) ⊆ *atms-of-ms* A ∧ *atm-of* ‘ *lits-of* (*trail* S) ⊆ *atms-of-ms* A
∧ *no-dup* (*trail* S))
∧ *cdcl*$_{NOT}$*-merged-bj-learn*$^{++}$ S T}
⟨*proof*⟩

lemma *backjump-no-step-backjump-l*:
backjump S $T \Longrightarrow$ *inv* $S \Longrightarrow$ ¬*no-step backjump-l* S

⟨*proof*⟩

lemma *cdcl*$_{NOT}$*-merged-bj-learn-final-state*:
fixes A :: *'v literal multiset set* **and** S T :: *'st*
assumes
n-s: *no-step cdcl*$_{NOT}$*-merged-bj-learn* S **and**
atms-S: *atms-of-msu* (*clauses* S) $\subseteq$ *atms-of-ms* A **and**
atms-trail: *atm-of* ‘ *lits-of* (*trail* S) $\subseteq$ *atms-of-ms* A **and**
n-d: *no-dup* (*trail* S) **and**
finite A **and**
inv: *inv* S **and**
decomp: *all-decomposition-implies-m* (*clauses* S) (*get-all-decided-decomposition* (*trail* S))
shows *unsatisfiable* (*set-mset* (*clauses* S))
$\vee$ (*trail* S $\models$*asm clauses* S $\wedge$ *satisfiable* (*set-mset* (*clauses* S)))
⟨*proof*⟩

lemma *full-cdcl*$_{NOT}$*-merged-bj-learn-final-state*:
fixes A :: *'v literal multiset set* **and** S T :: *'st*
assumes
full: *full cdcl*$_{NOT}$*-merged-bj-learn* S T **and**
atms-S: *atms-of-msu* (*clauses* S) $\subseteq$ *atms-of-ms* A **and**
atms-trail: *atm-of* ‘ *lits-of* (*trail* S) $\subseteq$ *atms-of-ms* A **and**
n-d: *no-dup* (*trail* S) **and**
finite A **and**
inv: *inv* S **and**
decomp: *all-decomposition-implies-m* (*clauses* S) (*get-all-decided-decomposition* (*trail* S))
shows *unsatisfiable* (*set-mset* (*clauses* T))
$\vee$ (*trail* T $\models$*asm clauses* T $\wedge$ *satisfiable* (*set-mset* (*clauses* T)))
⟨*proof*⟩

end

2.8.1 Instantiations

locale *cdcl*$_{NOT}$*-with-backtrack-and-restarts* =
conflict-driven-clause-learning-learning-before-backjump-only-distinct-learnt trail clauses
prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$ *propagate-conds inv backjump-conds*
learn-restrictions forget-restrictions
for
trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *ann-literals* **and**
clauses :: *'st* ⇒ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *ann-literal* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒ *'st* **and**
add-cls$_{NOT}$ *remove-cls*$_{NOT}$:: *'v clause* ⇒ *'st* ⇒ *'st* **and**
propagate-conds :: (*'v*, *unit*, *unit*) *ann-literal* ⇒ *'st* ⇒ *bool* **and**
inv :: *'st* ⇒ *bool* **and**
backjump-conds :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool* **and**
learn-restrictions forget-restrictions :: *'v clause* ⇒ *'st* ⇒ *bool*
+
fixes f :: *nat* ⇒ *nat*
assumes
unbounded: *unbounded* f **and** *f-ge-1*: $\bigwedge n.\ n \geq 1 \Longrightarrow f\ n \geq 1$ **and**
inv-restart:$\bigwedge S\ T.$ *inv* $S \Longrightarrow T \sim$ *reduce-trail-to*$_{NOT}$ ([]::*'a list*) $S \Longrightarrow$ *inv* T
begin

lemma *bound-inv-inv*:

assumes
inv S **and**
n-d: *no-dup* (*trail* S) **and**
atms-clss-S-A: *atms-of-msu* (*clauses* S) $\subseteq$ *atms-of-ms* A **and**
atms-trail-S-A:*atm-of* ' *lits-of* (*trail* S) $\subseteq$ *atms-of-ms* A **and**
finite A **and**
$cdcl_{NOT}$: $cdcl_{NOT}$ S T
shows
atms-of-msu (*clauses* T) $\subseteq$ *atms-of-ms* A **and**
atm-of ' *lits-of* (*trail* T) $\subseteq$ *atms-of-ms* A **and**
finite A
⟨*proof*⟩

sublocale $cdcl_{NOT}$*-increasing-restarts-ops* $\lambda S\ T.\ T \sim$ *reduce-trail-to*$_{NOT}$ ([]::'*a list*) S $cdcl_{NOT}$ f
$\lambda A\ S.$ *atms-of-msu* (*clauses* S) $\subseteq$ *atms-of-ms* A $\wedge$ *atm-of* ' *lits-of* (*trail* S) $\subseteq$ *atms-of-ms* A $\wedge$
finite A
$\mu_{CDCL}{}'$ $\lambda S.$ *inv* S $\wedge$ *no-dup* (*trail* S)
$\mu_{CDCL}{}'$*-bound*
⟨*proof*⟩

abbreviation $cdcl_{NOT}$*-l* **where**
$cdcl_{NOT}$*-l* $\equiv$
conflict-driven-clause-learning-ops.$cdcl_{NOT}$ *trail clauses prepend-trail tl-trail add-cls*$_{NOT}$
remove-cls$_{NOT}$ *propagate-conds* (λ- - - S T. *backjump* S T)
($\lambda C\ S.$ *distinct-mset* C $\wedge$ $\neg$ *tautology* C $\wedge$ *learn-restrictions* C S
$\wedge$ ($\exists F\ K\ F'\ C'\ L.$ *trail* S = F' @ *Decided* K () # F $\wedge$ $C = C' + \{\#L\#\}$
$\wedge$ F $\models$*as CNot* C' $\wedge$ $C' + \{\#L\#\} \notin\#$ *clauses* S))
($\lambda C\ S.$ $\neg$ ($\exists F'\ F\ K\ L.$ *trail* S = F' @ *Decided* K () # F $\wedge$ F $\models$*as CNot* ($C - \{\#L\#\}$))
$\wedge$ *forget-restrictions* C S)

lemma $cdcl_{NOT}$*-with-restart-*$\mu_{CDCL}{}'$*-le-*$\mu_{CDCL}{}'$*-bound*:
assumes
$cdcl_{NOT}$: $cdcl_{NOT}$*-restart* (T, a) (V, b) **and**
$cdcl_{NOT}$*-inv*:
inv T
no-dup (*trail* T) **and**
bound-inv:
atms-of-msu (*clauses* T) $\subseteq$ *atms-of-ms* A
atm-of ' *lits-of* (*trail* T) $\subseteq$ *atms-of-ms* A
finite A
shows $\mu_{CDCL}{}'$ A $V \leq \mu_{CDCL}{}'$*-bound* A T
⟨*proof*⟩

lemma $cdcl_{NOT}$*-with-restart-*$\mu_{CDCL}{}'$*-bound-le-*$\mu_{CDCL}{}'$*-bound*:
assumes
$cdcl_{NOT}$: $cdcl_{NOT}$*-restart* (T, a) (V, b) **and**
$cdcl_{NOT}$*-inv*:
inv T
no-dup (*trail* T) **and**
bound-inv:
atms-of-msu (*clauses* T) $\subseteq$ *atms-of-ms* A
atm-of ' *lits-of* (*trail* T) $\subseteq$ *atms-of-ms* A
finite A
shows $\mu_{CDCL}{}'$*-bound* A $V \leq \mu_{CDCL}{}'$*-bound* A T
⟨*proof*⟩

sublocale $cdcl_{NOT}$*-increasing-restarts* - - - - - - *f*
  $\lambda S\ T.\ T \sim$ *reduce-trail-to*$_{NOT}$ ([]::'*a list*) *S*
 $\lambda A\ S.$ *atms-of-msu* (*clauses S*) $\subseteq$ *atms-of-ms A*
  $\wedge$ *atm-of* ‘ *lits-of* (*trail S*) $\subseteq$ *atms-of-ms A* $\wedge$ *finite A*
 $\mu_{CDCL}{}'$ $cdcl_{NOT}$
  $\lambda S.$ *inv S* $\wedge$ *no-dup* (*trail S*)
 $\mu_{CDCL}{}'$*-bound*
 ⟨*proof*⟩

lemma $cdcl_{NOT}$*-restart-all-decomposition-implies*:
 assumes $cdcl_{NOT}$*-restart S T* **and**
  inv (*fst S*) **and**
  no-dup (*trail* (*fst S*))
  all-decomposition-implies-m (*clauses* (*fst S*)) (*get-all-decided-decomposition* (*trail* (*fst S*)))
 shows
  all-decomposition-implies-m (*clauses* (*fst T*)) (*get-all-decided-decomposition* (*trail* (*fst T*)))
 ⟨*proof*⟩

lemma *rtranclp-*$cdcl_{NOT}$*-restart-all-decomposition-implies*:
 assumes $cdcl_{NOT}$*-restart*** *S T* **and**
  inv: *inv* (*fst S*) **and**
  n-d: *no-dup* (*trail* (*fst S*)) **and**
  decomp:
   all-decomposition-implies-m (*clauses* (*fst S*)) (*get-all-decided-decomposition* (*trail* (*fst S*)))
 shows
  all-decomposition-implies-m (*clauses* (*fst T*)) (*get-all-decided-decomposition* (*trail* (*fst T*)))
 ⟨*proof*⟩

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

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

theorem *full-*$cdcl_{NOT}$*-restart-backjump-final-state*:
 fixes *A* :: '*v literal multiset set* **and** *S T* :: '*st*
 assumes
  full: *full* $cdcl_{NOT}$*-restart* (*S*, *n*) (*T*, *m*) **and**
  atms-S: *atms-of-msu* (*clauses S*) $\subseteq$ *atms-of-ms A* **and**
  atms-trail: *atm-of* ‘ *lits-of* (*trail S*) $\subseteq$ *atms-of-ms A* **and**
  n-d: *no-dup* (*trail S*) **and**
  fin-A[*simp*]: *finite A* **and**
  inv: *inv S* **and**
  decomp: *all-decomposition-implies-m* (*clauses S*) (*get-all-decided-decomposition* (*trail S*))

shows *unsatisfiable* (*set-mset* (*clauses S*))
∨ (*lits-of* (*trail T*) ⊨*sextm clauses S* ∧ *satisfiable* (*set-mset* (*clauses S*)))
⟨*proof*⟩
end — end of $cdcl_{NOT}$*-with-backtrack-and-restarts* locale

locale *most-general-*$cdcl_{NOT}$ =
dpll-state trail clauses prepend-trail tl-trail add-cls_{NOT} *remove-*cls_{NOT} +
propagate-ops trail clauses prepend-trail tl-trail add-cls_{NOT} *remove-*cls_{NOT} *propagate-conds* +
backjumping-ops trail clauses prepend-trail tl-trail add-cls_{NOT} *remove-*cls_{NOT} λ- - - - -. *True*
for
trail :: ′*st* ⇒ (′*v*, *unit*, *unit*) *ann-literals* **and**
clauses :: ′*st* ⇒ ′*v clauses* **and**
prepend-trail :: (′*v*, *unit*, *unit*) *ann-literal* ⇒ ′*st* ⇒ ′*st* **and**
tl-trail :: ′*st* ⇒ ′*st* **and**
add-cls_{NOT} *remove-*cls_{NOT}:: ′*v clause* ⇒ ′*st* ⇒ ′*st* **and**
propagate-conds :: (′*v*, *unit*, *unit*) *ann-literal* ⇒ ′*st* ⇒ *bool* **and**
inv :: ′*st* ⇒ *bool*
begin
lemma *backjump-bj-can-jump*:
assumes
tr-S: *trail S* = *F*′ @ *Decided K* () # *F* **and**
C: *C* ∈# *clauses S* **and**
tr-S-C: *trail S* ⊨*as CNot C* **and**
undef: *undefined-lit F L* **and**
atm-L: *atm-of L* ∈ *atms-of-msu* (*clauses S*) ∪ *atm-of* ‘ (*lits-of* (*F*′ @ *Decided K* () # *F*)) **and**
cls-S-C′: *clauses S* ⊨*pm C*′ + {#*L*#} **and**
F-C′: *F* ⊨*as CNot C*′
shows ¬*no-step backjump S*
⟨*proof*⟩

sublocale *dpll-with-backjumping-ops* - - - - - - - *inv* λ- - - - -. *True*
⟨*proof*⟩
end

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

locale $cdcl_{NOT}$*-merge-bj-learn-with-backtrack-restarts* =
$cdcl_{NOT}$*-merge-bj-learn trail clauses prepend-trail tl-trail add-*cls_{NOT} *remove-*cls_{NOT}
propagate-conds inv forget-conds
λ*C C*′ *L*′ *S*. *distinct-mset* (*C*′ + {#*L*′#}) ∧ *backjump-l-cond C C*′ *L*′ *S*
for
trail :: ′*st* ⇒ (′*v*, *unit*, *unit*) *ann-literals* **and**
clauses :: ′*st* ⇒ ′*v clauses* **and**
prepend-trail :: (′*v*, *unit*, *unit*) *ann-literal* ⇒ ′*st* ⇒ ′*st* **and**
tl-trail :: ′*st* ⇒ ′*st* **and**
add-cls_{NOT} *remove-*cls_{NOT}:: ′*v clause* ⇒ ′*st* ⇒ ′*st* **and**
propagate-conds :: (′*v*, *unit*, *unit*) *ann-literal* ⇒ ′*st* ⇒ *bool* **and**
inv :: ′*st* ⇒ *bool* **and**
forget-conds :: ′*v clause* ⇒ ′*st* ⇒ *bool* **and**
backjump-l-cond :: ′*v clause* ⇒ ′*v clause* ⇒ ′*v literal* ⇒ ′*st* ⇒ *bool*
\+
fixes *f* :: *nat* ⇒ *nat*
assumes
unbounded: *unbounded f* **and** *f-ge-1*: ⋀*n*. *n* ≥ *1* ⟹ *f n* ≥ *1* **and**
inv-restart:⋀*S T*. *inv S* ⟹ *T* ∼ *reduce-trail-to*$_{NOT}$ [] *S* ⟹ *inv T*

begin

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

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

definition *not-simplified-cls* A = {#C ∈# A. *tautology* C ∨ ¬*distinct-mset* C#}

lemma *simple-clss-or-not-simplified-cls*:
assumes *atms-of-msu* (*clauses* S) ⊆ *atms-of-ms* A **and**
x ∈# *clauses* S **and** *finite* A
shows x ∈ *simple-clss* (*atms-of-ms* A) ∨ x ∈# *not-simplified-cls* (*clauses* S)
⟨*proof*⟩

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

lemma *cdcl*$_{NOT}$*-merged-bj-learn-not-simplified-decreasing*:
assumes *cdcl*$_{NOT}$*-merged-bj-learn* S T
shows (*not-simplified-cls* (*clauses* T)) ⊆# (*not-simplified-cls* (*clauses* S))
⟨*proof*⟩

lemma *rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-not-simplified-decreasing*:
assumes *cdcl*$_{NOT}$*-merged-bj-learn*** S T
shows (*not-simplified-cls* (*clauses* T)) ⊆# (*not-simplified-cls* (*clauses* S))
⟨*proof*⟩

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

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

lemma *rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-clauses-bound-card*:
assumes
cdcl$_{NOT}$*-merged-bj-learn*** *S T* **and**
inv S **and**
atms-of-msu (*clauses S*) ⊆ *atms-of-ms A* **and**
atm-of ‘(*lits-of* (*trail S*)) ⊆ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite A*
shows $\mu_{CDCL}{}'$*-merged A T* ≤ $\mu_{CDCL}{}'$*-bound A S*
⟨*proof*⟩

sublocale *cdcl*$_{NOT}$*-increasing-restarts-ops* λ*S T*. *T* ∼ *reduce-trail-to*$_{NOT}$ ([]::′*a list*) *S*
cdcl$_{NOT}$*-merged-bj-learn f*
λ*A S*. *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A*
∧ *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-ms A* ∧ *finite A*
$\mu_{CDCL}{}'$*-merged*
λ*S*. *inv S* ∧ *no-dup* (*trail S*)
$\mu_{CDCL}{}'$*-bound*
⟨*proof*⟩

lemma *cdcl*$_{NOT}$*-restart-*$\mu_{CDCL}{}'$*-merged-le-*$\mu_{CDCL}{}'$*-bound*:
assumes
cdcl$_{NOT}$*-restart T V*
inv (*fst T*) **and**
no-dup (*trail* (*fst T*)) **and**
atms-of-msu (*clauses* (*fst T*)) ⊆ *atms-of-ms A* **and**
atm-of ‘ *lits-of* (*trail* (*fst T*)) ⊆ *atms-of-ms A* **and**
finite A
shows $\mu_{CDCL}{}'$*-merged A* (*fst V*) ≤ $\mu_{CDCL}{}'$*-bound A* (*fst T*)
⟨*proof*⟩

lemma *cdcl*$_{NOT}$*-restart-*$\mu_{CDCL}{}'$*-bound-le-*$\mu_{CDCL}{}'$*-bound*:
assumes
cdcl$_{NOT}$*-restart T V* **and**
no-dup (*trail* (*fst T*)) **and**
inv (*fst T*) **and**
fin: *finite A*
shows $\mu_{CDCL}{}'$*-bound A* (*fst V*) ≤ $\mu_{CDCL}{}'$*-bound A* (*fst T*)
⟨*proof*⟩

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

⟨*proof*⟩

lemma $cdcl_{NOT}$*-restart-eq-sat-iff*:
assumes
$cdcl_{NOT}$*-restart S T* **and**
no-dup (*trail* (*fst S*))
inv (*fst S*)
shows $I \models sextm\ clauses\ (fst\ S) \longleftrightarrow I \models sextm\ clauses\ (fst\ T)$
⟨*proof*⟩

lemma *rtranclp-*$cdcl_{NOT}$*-restart-eq-sat-iff*:
assumes
$cdcl_{NOT}$*-restart*** *S T* **and**
inv: *inv* (*fst S*) **and** *n-d*: *no-dup*(*trail* (*fst S*))
shows $I \models sextm\ clauses\ (fst\ S) \longleftrightarrow I \models sextm\ clauses\ (fst\ T)$
⟨*proof*⟩

lemma $cdcl_{NOT}$*-restart-all-decomposition-implies-m*:
assumes
$cdcl_{NOT}$*-restart S T* **and**
inv: *inv* (*fst S*) **and** *n-d*: *no-dup*(*trail* (*fst S*)) **and**
all-decomposition-implies-m (*clauses* (*fst S*))
(*get-all-decided-decomposition* (*trail* (*fst S*)))
shows *all-decomposition-implies-m* (*clauses* (*fst T*))
(*get-all-decided-decomposition* (*trail* (*fst T*)))
⟨*proof*⟩

lemma *rtranclp-*$cdcl_{NOT}$*-restart-all-decomposition-implies-m*:
assumes
$cdcl_{NOT}$*-restart*** *S T* **and**
inv: *inv* (*fst S*) **and** *n-d*: *no-dup*(*trail* (*fst S*)) **and**
decomp: *all-decomposition-implies-m* (*clauses* (*fst S*))
(*get-all-decided-decomposition* (*trail* (*fst S*)))
shows *all-decomposition-implies-m* (*clauses* (*fst T*))
(*get-all-decided-decomposition* (*trail* (*fst T*)))
⟨*proof*⟩

lemma *full-*$cdcl_{NOT}$*-restart-normal-form*:
assumes
full: *full* $cdcl_{NOT}$*-restart S T* **and**
inv: *inv* (*fst S*) **and** *n-d*: *no-dup*(*trail* (*fst S*)) **and**
decomp: *all-decomposition-implies-m* (*clauses* (*fst S*))
(*get-all-decided-decomposition* (*trail* (*fst S*))) **and**
atms-cls: *atms-of-msu* (*clauses* (*fst S*)) $\subseteq$ *atms-of-ms A* **and**
atms-trail: *atm-of* ‘ *lits-of* (*trail* (*fst S*)) $\subseteq$ *atms-of-ms A* **and**
fin: *finite A*
shows *unsatisfiable* (*set-mset* (*clauses* (*fst S*)))
$\vee$ *lits-of* (*trail* (*fst T*)) $\models$*sextm clauses* (*fst S*) $\wedge$ *satisfiable* (*set-mset* (*clauses* (*fst S*)))
⟨*proof*⟩

corollary *full-*$cdcl_{NOT}$*-restart-normal-form-init-state*:
assumes
init-state: *trail S* = [] *clauses S* = *N* **and**
full: *full* $cdcl_{NOT}$*-restart* (*S*, *0*) *T* **and**
inv: *inv S*

shows *unsatisfiable* (*set-mset* N)
  ∨ *lits-of* (*trail* (*fst* T)) ⊨*sextm* N ∧ *satisfiable* (*set-mset* N)
  ⟨*proof*⟩

end

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

3 DPLL as an instance of NOT

3.1 DPLL with simple backtrack

locale *dpll-with-backtrack*
begin
inductive *backtrack* :: (′*v*, *unit*, *unit*) *ann-literal list* × ′*v clauses*
  ⇒ (′*v*, *unit*, *unit*) *ann-literal list* × ′*v clauses* ⇒ *bool* **where**
backtrack-split (*fst* S) = (M', L # M) ⟹ *is-decided* L ⟹ D ∈# *snd* S
  ⟹ *fst* S ⊨*as CNot* D ⟹ *backtrack* S (*Propagated* (− (*lit-of* L)) () # M, *snd* S)

inductive-cases *backtrackE*[*elim*]: *backtrack* (M, N) (M', N')
lemma *backtrack-is-backjump*:
  fixes M M' :: (′*v*, *unit*, *unit*) *ann-literal list*
  assumes
    backtrack: *backtrack* (M, N) (M', N') **and**
    no-dup: (*no-dup* ∘ *fst*) (M, N) **and**
    decomp: *all-decomposition-implies-m* N (*get-all-decided-decomposition* M)
    shows
      ∃ C F' K F L l C'.
        M = F' @ *Decided* K () # F ∧
        M' = *Propagated* L l # F ∧ N = N' ∧ C ∈# N ∧ F' @ *Decided* K d # F ⊨*as CNot* C ∧
        undefined-lit F L ∧ *atm-of* L ∈ *atms-of-msu* N ∪ *atm-of* ‘ *lits-of* (F' @ *Decided* K d # F) ∧
        N ⊨*pm* C' + {#L#} ∧ F ⊨*as CNot* C'
⟨*proof*⟩

lemma *backtrack-is-backjump′*:
  fixes M M' :: (′*v*, *unit*, *unit*) *ann-literal list*
  assumes
    backtrack: *backtrack* S T **and**
    no-dup: (*no-dup* ∘ *fst*) S **and**
    decomp: *all-decomposition-implies-m* (*snd* S) (*get-all-decided-decomposition* (*fst* S))
    shows
      ∃ C F' K F L l C'.
        fst S = F' @ *Decided* K () # F ∧
        T = (*Propagated* L l # F, *snd* S) ∧ C ∈# *snd* S ∧ *fst* S ⊨*as CNot* C
        ∧ *undefined-lit* F L ∧ *atm-of* L ∈ *atms-of-msu* (*snd* S) ∪ *atm-of* ‘ *lits-of* (*fst* S) ∧
        snd S ⊨*pm* C' + {#L#} ∧ F ⊨*as CNot* C'
  ⟨*proof*⟩

sublocale *dpll-state fst snd* λL (M, N). (L # M, N) λ(M, N). (*tl* M, N)
  λC (M, N). (M, {#C#} + N) λC (M, N). (M, *remove-mset* C N)
  ⟨*proof*⟩

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

lemma *backtrack-is-backjump''*:
fixes M M' :: ('*v*, *unit*, *unit*) *ann-literal list*
assumes
backtrack: *backtrack* S T **and**
no-dup: (*no-dup* ∘ *fst*) S **and**
decomp: *all-decomposition-implies-m* (*snd* S) (*get-all-decided-decomposition* (*fst* S))
shows *backjump* S T
⟨*proof*⟩

lemma *can-do-bt-step*:
assumes
M: *fst* $S = F'$ @ *Decided* K d # F **and**
C ∈# *snd* S **and**
C: *fst* S ⊨*as CNot* C
shows ¬ *no-step backtrack* S
⟨*proof*⟩

end

sublocale *dpll-with-backtrack* ⊆ *dpll-with-backjumping-ops fst snd* λL (M, N). $(L \# M, N)$
$\lambda(M, N)$. $(tl\ M, N)$ λC (M, N). $(M, \{\#C\#\} + N)$ λC (M, N). $(M, \textit{remove-mset}\ C\ N)$ λ- -. *True*
$\lambda(M, N)$. *no-dup* M ∧ *all-decomposition-implies-m* N (*get-all-decided-decomposition* M)
λ- - - S T. *backtrack* S T
⟨*proof*⟩

sublocale *dpll-with-backtrack* ⊆ *dpll-with-backjumping fst snd* λL (M, N). $(L \# M, N)$
$\lambda(M, N)$. $(tl\ M, N)$ λC (M, N). $(M, \{\#C\#\} + N)$ λC (M, N). $(M, \textit{remove-mset}\ C\ N)$ λ- -. *True*
$\lambda(M, N)$. *no-dup* M ∧ *all-decomposition-implies-m* N (*get-all-decided-decomposition* M)
λ- - - S T. *backtrack* S T
⟨*proof*⟩

sublocale *dpll-with-backtrack* ⊆ *conflict-driven-clause-learning-ops*
fst snd λL (M, N). $(L \# M, N)$
$\lambda(M, N)$. $(tl\ M, N)$ λC (M, N). $(M, \{\#C\#\} + N)$ λC (M, N). $(M, \textit{remove-mset}\ C\ N)$ λ- -. *True*
$\lambda(M, N)$. *no-dup* M ∧ *all-decomposition-implies-m* N (*get-all-decided-decomposition* M)
λ- - - S T. *backtrack* S T λ- -. *False* λ- -. *False*
⟨*proof*⟩

sublocale *dpll-with-backtrack* ⊆ *conflict-driven-clause-learning*
fst snd λL (M, N). $(L \# M, N)$
$\lambda(M, N)$. $(tl\ M, N)$ λC (M, N). $(M, \{\#C\#\} + N)$ λC (M, N). $(M, \textit{remove-mset}\ C\ N)$ λ- -. *True*
$\lambda(M, N)$. *no-dup* M ∧ *all-decomposition-implies-m* N (*get-all-decided-decomposition* M)
λ- - - S T. *backtrack* S T λ- -. *False* λ- -. *False*
⟨*proof*⟩

context *dpll-with-backtrack*
begin
lemma *wf-tranclp-dpll-inital-state*:
assumes *fin*: *finite* A
shows *wf* $\{((M'::('v, unit, unit)\ \textit{ann-literals}, N'::'v\ clauses), ([], N))|M'\ N'\ N.$
dpll-bj^{++} ([], N) (M', N') ∧ *atms-of-msu* N ⊆ *atms-of-ms* $A\}$

⟨*proof*⟩

corollary *full-dpll-final-state-conclusive*:
fixes $M\ M'$ :: $('v,\ unit,\ unit)$ *ann-literal list*
assumes
full: *full dpll-bj* $([],\ N)\ (M',\ N')$
shows *unsatisfiable* $(set\text{-}mset\ N) \lor (M' \models asm\ N \land satisfiable\ (set\text{-}mset\ N))$
⟨*proof*⟩

corollary *full-dpll-normal-form-from-init-state*:
fixes $M\ M'$ :: $('v,\ unit,\ unit)$ *ann-literal list*
assumes
full: *full dpll-bj* $([],\ N)\ (M',\ N')$
shows $M' \models asm\ N \longleftrightarrow satisfiable\ (set\text{-}mset\ N)$
⟨*proof*⟩

lemma $cdcl_{NOT}$*-is-dpll*:
$cdcl_{NOT}\ S\ T \longleftrightarrow dpll\text{-}bj\ S\ T$
⟨*proof*⟩

Another proof of termination:

lemma $wf\ \{(T,\ S).\ dpll\text{-}bj\ S\ T \land cdcl_{NOT}\text{-}NOT\text{-}all\text{-}inv\ A\ S\}$
⟨*proof*⟩
end

3.2 Adding restarts

locale *dpll-withbacktrack-and-restarts* =
dpll-with-backtrack +
fixes f :: $nat \Rightarrow nat$
assumes *unbounded*: *unbounded f* **and** *f-ge-1*: $\bigwedge n.\ n \geq 1 \Longrightarrow f\ n \geq 1$
begin
sublocale $cdcl_{NOT}$*-increasing-restarts* $fst\ snd\ \lambda L\ (M,\ N).\ (L\ \#\ M,\ N)\ \lambda(M,\ N).\ (tl\ M,\ N)$
$\lambda C\ (M,\ N).\ (M,\ \{\#C\#\} + N)\ \lambda C\ (M,\ N).\ (M,\ remove\text{-}mset\ C\ N)\ f\ \lambda(\text{-},\ N)\ S.\ S = ([],\ N)$
$\lambda A\ (M,\ N).\ atms\text{-}of\text{-}msu\ N \subseteq atms\text{-}of\text{-}ms\ A \land atm\text{-}of\ `\ lits\text{-}of\ M \subseteq atms\text{-}of\text{-}ms\ A \land finite\ A$
$\land\ all\text{-}decomposition\text{-}implies\text{-}m\ N\ (get\text{-}all\text{-}decided\text{-}decomposition\ M)$
$\lambda A\ T.\ (2{+}card\ (atms\text{-}of\text{-}ms\ A))\ \hat{}\ (1{+}card\ (atms\text{-}of\text{-}ms\ A))$
$-\ \mu_C\ (1{+}card\ (atms\text{-}of\text{-}ms\ A))\ (2{+}card\ (atms\text{-}of\text{-}ms\ A))\ (trail\text{-}weight\ T)\ dpll\text{-}bj$
$\lambda(M,\ N).\ no\text{-}dup\ M \land all\text{-}decomposition\text{-}implies\text{-}m\ N\ (get\text{-}all\text{-}decided\text{-}decomposition\ M)$
$\lambda A\ \text{-}.\ (2{+}card\ (atms\text{-}of\text{-}ms\ A))\ \hat{}\ (1{+}card\ (atms\text{-}of\text{-}ms\ A))$
⟨*proof*⟩
end

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

4 DPLL

4.1 Rules

type-synonym $'a\ dpll_W$*-ann-literal* = $('a,\ unit,\ unit)$ *ann-literal*
type-synonym $'a\ dpll_W$*-ann-literals* = $('a,\ unit,\ unit)$ *ann-literals*

type-synonym $'v$ $dpll_W$*-state* = $'v$ $dpll_W$*-ann-literals* × $'v$ *clauses*

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

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

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

4.2 Invariants

lemma $dpll_W$*-distinct-inv*:
assumes $dpll_W$ *S S'*
and *no-dup* (*trail S*)
shows *no-dup* (*trail S'*)
⟨*proof*⟩

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

lemma $dpll_W$*-vars-in-snd-inv*:
assumes $dpll_W$ *S S'*
and *atm-of* ‘ (*lits-of* (*trail S*)) ⊆ *atms-of-msu* (*clauses S*)
shows *atm-of* ‘ (*lits-of* (*trail S'*)) ⊆ *atms-of-msu* (*clauses S'*)
⟨*proof*⟩

lemma *atms-of-ms-lit-of-atms-of*: *atms-of-ms* ((λ*a*. {#*lit-of a*#}) ‘ *c*) = *atm-of* ‘ *lit-of* ‘ *c*
⟨*proof*⟩

Lemma theorem 2.8.2 page 71 of CW

lemma $dpll_W$*-propagate-is-conclusion*:
assumes $dpll_W$ *S S'*
and *all-decomposition-implies-m* (*clauses S*) (*get-all-decided-decomposition* (*trail S*))
and *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-msu* (*clauses S*)
shows *all-decomposition-implies-m* (*clauses S'*) (*get-all-decided-decomposition* (*trail S'*))
⟨*proof*⟩

Lemma theorem 2.8.3 page 72 of CW

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

⟨*proof*⟩

Lemma theorem 2.8.4 page 72 of CW

lemma *only-propagated-vars-unsat*:
assumes *decided*: $\forall x \in$ *set M*. ¬ *is-decided x*
and *DN*: $D \in N$ **and** *D*: $M \models as$ *CNot D*
and *inv*: *all-decomposition-implies N* (*get-all-decided-decomposition M*)
and *atm-incl*: *atm-of* ‘ *lits-of M* ⊆ *atms-of-ms N*
shows *unsatisfiable N*
⟨*proof*⟩

lemma *dpll$_W$-same-clauses*:
assumes *dpll$_W$ S S′*
shows *clauses S* = *clauses S′*
⟨*proof*⟩

lemma *rtranclp-dpll$_W$-inv*:
assumes *rtranclp dpll$_W$ S S′*
and *inv*: *all-decomposition-implies-m* (*clauses S*) (*get-all-decided-decomposition* (*trail S*))
and *atm-incl*: *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-msu* (*clauses S*)
and *consistent-interp* (*lits-of* (*trail S*))
and *no-dup* (*trail S*)
shows *all-decomposition-implies-m* (*clauses S′*) (*get-all-decided-decomposition* (*trail S′*))
and *atm-of* ‘ *lits-of* (*trail S′*) ⊆ *atms-of-msu* (*clauses S′*)
and *clauses S* = *clauses S′*
and *consistent-interp* (*lits-of* (*trail S′*))
and *no-dup* (*trail S′*)
⟨*proof*⟩

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

lemma *dpll$_W$-all-inv-dest*[*dest*]:
assumes *dpll$_W$-all-inv S*
shows *all-decomposition-implies-m* (*clauses S*) (*get-all-decided-decomposition* (*trail S*))
and *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-msu* (*clauses S*)
and *consistent-interp* (*lits-of* (*trail S*)) ∧ *no-dup* (*trail S*)
⟨*proof*⟩

lemma *rtranclp-dpll$_W$-all-inv*:
assumes *rtranclp dpll$_W$ S S′*
and *dpll$_W$-all-inv S*
shows *dpll$_W$-all-inv S′*
⟨*proof*⟩

lemma *dpll$_W$-all-inv*:
assumes *dpll$_W$ S S′*
and *dpll$_W$-all-inv S*
shows *dpll$_W$-all-inv S′*
⟨*proof*⟩

lemma *rtranclp-dpll$_W$-inv-starting-from-0*:

assumes *rtranclp* $dpll_W$ S S'
and *inv*: *trail* S = []
shows $dpll_W$*-all-inv* S'
⟨*proof*⟩

lemma $dpll_W$*-can-do-step*:
assumes *consistent-interp* (*set* M)
and *distinct* M
and *atm-of* ‘ (*set* M) ⊆ *atms-of-msu* N
shows *rtranclp* $dpll_W$ ([], N) (*map* (λM. *Decided* M ()) M, N)
⟨*proof*⟩

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

lemma $dpll_W$*-strong-completeness*:
assumes *set* M ⊨*sm* N
and *consistent-interp* (*set* M)
and *distinct* M
and *atm-of* ‘ (*set* M) ⊆ *atms-of-msu* N
shows $dpll_W{}^{**}$ ([], N) (*map* (λM. *Decided* M ()) M, N)
and *conclusive-*$dpll_W$*-state* (*map* (λM. *Decided* M ()) M, N)
⟨*proof*⟩

lemma $dpll_W$*-sound*:
assumes
rtranclp $dpll_W$ ([], N) (M, N) **and**
∀ S. ¬$dpll_W$ (M, N) S
shows M ⊨*asm* N ⟷ *satisfiable* (*set-mset* N) (**is** *?A* ⟷ *?B*)
⟨*proof*⟩

4.3 Termination

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

lemma *length-*$dpll_W$*-mes*:
assumes *length* $M \le n$
shows *length* ($dpll_W$*-mes* M n) = n
⟨*proof*⟩

lemma *distinctcard-atm-of-lit-of-eq-length*:
assumes *no-dup* S
shows *card* (*atm-of* ‘ *lits-of* S) = *length* S
⟨*proof*⟩

lemma $dpll_W$*-card-decrease*:
assumes *dpll*: $dpll_W$ S S' **and** *length* (*trail* S') ≤ *card vars*
and *length* (*trail* S) ≤ *card vars*
shows ($dpll_W$*-mes* (*trail* S') (*card vars*), $dpll_W$*-mes* (*trail* S) (*card vars*))
∈ *lexn* {(a, b). $a < b$} (*card vars*)
⟨*proof*⟩

Proposition theorem 2.8.7 page 73 of CW

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

lemma *wf-lexn*: *wf* (*lexn* {(*a*, *b*). (*a*::*nat*) < *b*} (*card* (*atms-of-msu* (*clauses S*))))
⟨*proof*⟩

lemma $dpll_W$*-wf*:
  wf {(*S'*, *S*). $dpll_W$*-all-inv S* ∧ $dpll_W$ *S S'*}
  ⟨*proof*⟩

lemma $dpll_W$*-tranclp-star-commute*:
  {(*S'*, *S*). $dpll_W$*-all-inv S* ∧ $dpll_W$ *S S'*}$^+$ = {(*S'*, *S*). $dpll_W$*-all-inv S* ∧ *tranclp* $dpll_W$ *S S'*}
    (**is** *?A* = *?B*)
⟨*proof*⟩

lemma $dpll_W$*-wf-tranclp*: *wf* {(*S'*, *S*). $dpll_W$*-all-inv S* ∧ ${dpll_W}^{++}$ *S S'*}
  ⟨*proof*⟩

lemma $dpll_W$*-wf-plus*:
  shows *wf* {(*S'*, ([], *N*))| *S'*. ${dpll_W}^{++}$ ([], *N*) *S'*} (**is** *wf ?P*)
  ⟨*proof*⟩

4.4 Final States

lemma $dpll_W$*-no-more-step-is-a-conclusive-state*:
  assumes ∀ *S'*. ¬$dpll_W$ *S S'*
  shows *conclusive-*$dpll_W$*-state S*
⟨*proof*⟩

lemma $dpll_W$*-conclusive-state-correct*:
  assumes ${dpll_W}^{**}$ ([], *N*) (*M*, *N*) **and** *conclusive-*$dpll_W$*-state* (*M*, *N*)
  shows *M* ⊨*asm N* ⟷ *satisfiable* (*set-mset N*) (**is** *?A* ⟷ *?B*)
⟨*proof*⟩

4.5 Link with NOT's DPLL

interpretation $dpll_W$*-*$_{NOT}$: *dpll-with-backtrack* ⟨*proof*⟩

lemma *state-eq*$_{NOT}$*-iff-eq*[*iff*, *simp*]: $dpll_W$*-*$_{NOT}$*.state-eq*$_{NOT}$ *S T* ⟷ *S* = *T*
  ⟨*proof*⟩

declare $dpll_W$*-*$_{NOT}$*.state-simp*$_{NOT}$[*simp del*]

lemma $dpll_W$*-*$dpll_W$*-bj*:
  assumes *inv*: $dpll_W$*-all-inv S* **and** *dpll*: $dpll_W$ *S T*
  shows $dpll_W$*-*$_{NOT}$*.dpll-bj S T*
  ⟨*proof*⟩

lemma *$dpll_W$-bj-dpll*:
  assumes *inv*: *$dpll_W$-all-inv S* **and** *dpll*: *$dpll_W$-$_{NOT}$.dpll-bj S T*
  shows *$dpll_W$ S T*
  ⟨*proof*⟩

lemma *rtranclp-$dpll_W$-rtranclp-$dpll_W$-$_{NOT}$*:
  assumes *$dpll_W^{**}$ S T* **and** *$dpll_W$-all-inv S*
  shows *$dpll_W$-$_{NOT}$.dpll-bj** S T*
  ⟨*proof*⟩

lemma *rtranclp-dpll-rtranclp-$dpll_W$*:
  assumes *$dpll_W$-$_{NOT}$.dpll-bj** S T* **and** *$dpll_W$-all-inv S*
  shows *$dpll_W^{**}$ S T*
  ⟨*proof*⟩

lemma *dpll-conclusive-state-correctness*:
  assumes *$dpll_W$-$_{NOT}$.dpll-bj*** ([], *N*) (*M*, *N*) **and** *conclusive-$dpll_W$-state* (*M*, *N*)
  shows *M* ⊨*asm N* ⟷ *satisfiable* (*set-mset N*)
⟨*proof*⟩

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

4.5.1 Level of literals and clauses

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

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

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

lemma *get-rev-level-uminus*[*simp*]: *get-rev-level M n*(−*L*) = *get-rev-level M n L*
  ⟨*proof*⟩

lemma *atm-of-notin-get-rev-level-eq-0*[*simp*]:
  assumes *atm-of L* ∉ *atm-of ‘ lits-of M*
  shows *get-rev-level M n L* = *0*
  ⟨*proof*⟩

lemma *get-rev-level-ge-0-atm-of-in*:
  assumes *get-rev-level M n L* > *n*
  shows *atm-of L* ∈ *atm-of ‘ lits-of M*
  ⟨*proof*⟩

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

lemma *get-rev-level-skip*[*simp*]:
  assumes *atm-of L* ∉ *atm-of ‘ lits-of M*

shows *get-rev-level* $(M\ @\ Decided\ K\ i\ \#\ M')\ n\ L = get\text{-}rev\text{-}level\ (Decided\ K\ i\ \#\ M')\ i\ L$
⟨*proof*⟩

lemma *get-rev-level-notin-end*[*simp*]:
assumes $atm\text{-}of\ L \notin atm\text{-}of\ `\ lits\text{-}of\ M'$
shows $get\text{-}rev\text{-}level\ (M\ @\ M')\ n\ L = get\text{-}rev\text{-}level\ M\ n\ L$
⟨*proof*⟩

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

lemma *get-rev-level-skip-end*[*simp*]:
assumes $atm\text{-}of\ L \in atm\text{-}of\ `\ lits\text{-}of\ M$
shows $get\text{-}rev\text{-}level\ (M\ @\ M')\ n\ L = get\text{-}rev\text{-}level\ M\ n\ L$
⟨*proof*⟩

lemma *get-level-skip-beginning*:
assumes $atm\text{-}of\ L' \neq atm\text{-}of\ (lit\text{-}of\ K)$
shows $get\text{-}level\ (K\ \#\ M)\ L' = get\text{-}level\ M\ L'$
⟨*proof*⟩

lemma *get-level-skip-beginning-not-decided-rev*:
assumes $atm\text{-}of\ L \notin atm\text{-}of\ `\ lit\text{-}of\ `(set\ S)$
and $\forall\, s{\in}set\ S.\ \neg is\text{-}decided\ s$
shows $get\text{-}level\ (M\ @\ rev\ S)\ L = get\text{-}level\ M\ L$
⟨*proof*⟩

lemma *get-level-skip-beginning-not-decided*[*simp*]:
assumes $atm\text{-}of\ L \notin atm\text{-}of\ `\ lit\text{-}of\ `(set\ S)$
and $\forall\, s{\in}set\ S.\ \neg is\text{-}decided\ s$
shows $get\text{-}level\ (M\ @\ S)\ L = get\text{-}level\ M\ L$
⟨*proof*⟩

lemma *get-rev-level-skip-beginning-not-decided*[*simp*]:
assumes $atm\text{-}of\ L \notin atm\text{-}of\ `\ lit\text{-}of\ `(set\ S)$
and $\forall\, s{\in}set\ S.\ \neg is\text{-}decided\ s$
shows $get\text{-}rev\text{-}level\ (rev\ S\ @\ rev\ M)\ 0\ L = get\text{-}level\ M\ L$
⟨*proof*⟩

lemma *get-level-skip-in-all-not-decided*:
fixes $M :: ('a,\ nat,\ 'b)\ ann\text{-}literal\ list$ **and** $L :: 'a\ literal$
assumes $\forall\, m{\in}set\ M.\ \neg\ is\text{-}decided\ m$
and $atm\text{-}of\ L \in atm\text{-}of\ `\ lit\text{-}of\ `\ (set\ M)$
shows $get\text{-}rev\text{-}level\ M\ n\ L = n$
⟨*proof*⟩

lemma *get-level-skip-all-not-decided*[*simp*]:
fixes M
defines $M' \equiv rev\ M$
assumes $\forall\, m{\in}set\ M.\ \neg\ is\text{-}decided\ m$
shows $get\text{-}level\ M\ L = 0$
⟨*proof*⟩

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

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

definition $get\text{-}maximum\text{-}level :: ('a,\ nat,\ 'b)\ ann\text{-}literal\ list \Rightarrow 'a\ literal\ multiset \Rightarrow nat$

where
get-maximum-level M D = *MMax* ({#*0*#} + *image-mset* (*get-level M*) *D*)

lemma *get-maximum-level-ge-get-level*:
$L \in\# D \Longrightarrow$ *get-maximum-level M D* $\geq$ *get-level M L*
⟨*proof*⟩

lemma *get-maximum-level-empty*[*simp*]:
get-maximum-level M {#} = *0*
⟨*proof*⟩

lemma *get-maximum-level-exists-lit-of-max-level*:
$D \neq \{\#\} \Longrightarrow \exists L \in\# D.$ *get-level M L* = *get-maximum-level M D*
⟨*proof*⟩

lemma *get-maximum-level-empty-list*[*simp*]:
get-maximum-level [] *D* = *0*
⟨*proof*⟩

lemma *get-maximum-level-single*[*simp*]:
get-maximum-level M {#*L*#} = *get-level M L*
⟨*proof*⟩

lemma *get-maximum-level-plus*:
get-maximum-level M ($D + D'$) = *max* (*get-maximum-level M D*) (*get-maximum-level M D'*)
⟨*proof*⟩

lemma *get-maximum-level-exists-lit*:
assumes *n*: $n > 0$
and *max*: *get-maximum-level M D* = *n*
shows $\exists L \in\# D.$ *get-level M L* = *n*
⟨*proof*⟩

lemma *get-maximum-level-skip-first*[*simp*]:
assumes *atm-of L* $\notin$ *atms-of D*
shows *get-maximum-level* (*Propagated L C* # *M*) *D* = *get-maximum-level M D*
⟨*proof*⟩

lemma *get-maximum-level-skip-beginning*:
assumes *DH*: *atms-of D* $\subseteq$ *atm-of* '*lits-of H*
shows *get-maximum-level* (*c* @ *Decided Kh i* # *H*) *D* = *get-maximum-level H D*
⟨*proof*⟩

lemma *get-maximum-level-D-single-propagated*:
get-maximum-level [*Propagated x21 x22*] *D* = *0*
⟨*proof*⟩

lemma *get-maximum-level-skip-notin*:
assumes *D*: $\forall L \in\# D.$ *atm-of L* $\in$ *atm-of* '*lits-of M*
shows *get-maximum-level M D* = *get-maximum-level* (*Propagated x21 x22* # *M*) *D*
⟨*proof*⟩

lemma *get-maximum-level-skip-un-decided-not-present*:
assumes $\forall L \in\# D.$ *atm-of L* $\in$ *atm-of* ' *lits-of aa* **and**

$\forall m \in set\ M.\ \neg\ is\text{-}decided\ m$
shows *get-maximum-level aa D = get-maximum-level (M @ aa) D*
⟨*proof*⟩

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

lemma *get-maximum-possible-level-append*[*simp*]:
get-maximum-possible-level (M@M')
= max (get-maximum-possible-level M) (get-maximum-possible-level M')
⟨*proof*⟩

lemma *get-maximum-possible-level-rev*[*simp*]:
get-maximum-possible-level (rev M) = get-maximum-possible-level M
⟨*proof*⟩

lemma *get-maximum-possible-level-ge-get-rev-level*:
max (get-maximum-possible-level M) i ≥ get-rev-level M i L
⟨*proof*⟩

lemma *get-maximum-possible-level-ge-get-level*[*simp*]:
get-maximum-possible-level M ≥ get-level M L
⟨*proof*⟩

lemma *get-maximum-possible-level-ge-get-maximum-level*[*simp*]:
get-maximum-possible-level M ≥ get-maximum-level M D
⟨*proof*⟩

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
⟨*proof*⟩

4.5.2 Properties about the levels

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

lemma *get-all-levels-of-decided-nil-iff-not-is-decided*:
get-all-levels-of-decided xs = [] ⟷ (∀ x ∈ set xs. ¬is-decided x)
⟨*proof*⟩

lemma *get-all-levels-of-decided-cons*:
get-all-levels-of-decided (a # b) =
(if is-decided a then [level-of a] else []) @ get-all-levels-of-decided b
⟨*proof*⟩

lemma *get-all-levels-of-decided-append*[*simp*]:

get-all-levels-of-decided $(a\ @\ b) =$ *get-all-levels-of-decided* $a\ @$ *get-all-levels-of-decided* b
⟨*proof*⟩

lemma *in-get-all-levels-of-decided-iff-decomp*:
$i \in$ *set* (*get-all-levels-of-decided* M) $\longleftrightarrow$ $(\exists\, c\ K\ c'.\ M = c\ @$ *Decided* $K\ i\ \#\ c')$ (**is** $?A \longleftrightarrow ?B$)
⟨*proof*⟩

lemma *get-rev-level-less-max-get-all-levels-of-decided*:
get-rev-level $M\ n\ L \leq$ *Max* (*set* ($n\ \#$ *get-all-levels-of-decided* M))
⟨*proof*⟩

lemma *get-rev-level-ge-min-get-all-levels-of-decided*:
assumes *atm-of* $L \in$ *atm-of* ‘ *lits-of* M
shows *get-rev-level* $M\ n\ L \geq$ *Min* (*set* ($n\ \#$ *get-all-levels-of-decided* M))
⟨*proof*⟩

lemma *get-all-levels-of-decided-rev-eq-rev-get-all-levels-of-decided*[*simp*]:
get-all-levels-of-decided (*rev* M) = *rev* (*get-all-levels-of-decided* M)
⟨*proof*⟩

lemma *get-maximum-possible-level-max-get-all-levels-of-decided*:
get-maximum-possible-level M = *Max* (*insert* 0 (*set* (*get-all-levels-of-decided* M)))
⟨*proof*⟩

lemma *get-rev-level-in-levels-of-decided*:
get-rev-level $M\ n\ L \in \{0,\ n\} \cup$ *set* (*get-all-levels-of-decided* M)
⟨*proof*⟩

lemma *get-rev-level-in-atms-in-levels-of-decided*:
atm-of $L \in$ *atm-of* ‘ (*lits-of* M) $\Longrightarrow$ *get-rev-level* $M\ n\ L \in \{n\} \cup$ *set* (*get-all-levels-of-decided* M)
⟨*proof*⟩

lemma *get-all-levels-of-decided-no-decided*:
$(\forall\, l{\in}$*set* $Ls.\ \neg$ *is-decided* $l) \longleftrightarrow$ *get-all-levels-of-decided* $Ls = []$
⟨*proof*⟩

lemma *get-level-in-levels-of-decided*:
get-level $M\ L \in \{0\} \cup$ *set* (*get-all-levels-of-decided* M)
⟨*proof*⟩

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

lemma *get-level-get-rev-level-get-all-levels-of-decided*:
assumes *atm-of* $L \notin$ *atm-of* ‘ (*lits-of* M)
shows *get-level* $(K\ @\ M)\ L$ = *get-rev-level* (*rev* K) (*last* ($0\ \#$ *get-all-levels-of-decided* (*rev* M)))
L
⟨*proof*⟩

lemma *get-rev-level-can-skip-correctly-ordered*:
assumes
no-dup M **and**
atm-of $L \notin$ *atm-of* ‘ (*lits-of* M) **and**
get-all-levels-of-decided M = *rev* [*Suc* $0..<$*Suc* (*length* (*get-all-levels-of-decided* M))]
shows *get-rev-level* (*rev* $M\ @\ K$) $0\ L$ = *get-rev-level* K (*length* (*get-all-levels-of-decided* M)) L
⟨*proof*⟩

lemma *get-level-skip-beginning-hd-get-all-levels-of-decided*:
assumes *atm-of* $L \notin$ *atm-of* ' *lits-of* S
and *get-all-levels-of-decided* $S \neq []$
shows *get-level* (M@ S) L = *get-rev-level* (*rev* M) (*hd* (*get-all-levels-of-decided* S)) L
⟨*proof*⟩

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

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

lemma *Bex-set-set-Bex-set*[*iff*]: ($\exists x \in$*set-mset* C. P) $\longleftrightarrow$ ($\exists x \in \# C$. P)
⟨*proof*⟩

5 Weidenbach's CDCL

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

5.1 The State

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

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

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

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

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

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

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

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

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

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

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

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

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

clss-tl-trail[*simp*]: *clauses* (*tl-trail S*) = *clauses S* **and**
clauses-add-learned-cls-unfolded:
no-dup (*trail S*) $\Longrightarrow$ *clauses* (*add-learned-cls U S*) = {#*U*#} + *learned-clss S* + *init-clss S*
and
clauses-add-init-cls[*simp*]:
no-dup (*trail S*) $\Longrightarrow$ *clauses* (*add-init-cls N S*) = {#*N*#} + *init-clss S* + *learned-clss S* **and**
clauses-update-backtrack-lvl[*simp*]: *clauses* (*update-backtrack-lvl k S*) = *clauses S* **and**
clauses-update-conflicting[*simp*]: *clauses* (*update-conflicting D S*) = *clauses S* **and**
clauses-remove-cls[*simp*]:
clauses (*remove-cls C S*) = *clauses S* − *replicate-mset* (*count* (*clauses S*) *C*) *C* **and**
clauses-add-learned-cls[*simp*]:
no-dup (*trail S*) $\Longrightarrow$ *clauses* (*add-learned-cls C S*) = {#*C*#} + *clauses S* **and**
clauses-restart[*simp*]: *clauses* (*restart-state S*) $\subseteq$# *clauses S* **and**
clauses-init-state[*simp*]: $\bigwedge$*N. clauses* (*init-state N*) = *N*
$\langle$*proof*$\rangle$

abbreviation *state* :: *'st* $\Rightarrow$ (*'v*, *nat*, *'v clause*) *ann-literal list* $\times$ *'v clauses* $\times$ *'v clauses*
$\times$ *nat* $\times$ *'v clause option* **where**
state S $\equiv$ (*trail S*, *init-clss S*, *learned-clss S*, *backtrack-lvl S*, *conflicting S*)

abbreviation *incr-lvl* :: *'st* $\Rightarrow$ *'st* **where**
incr-lvl S $\equiv$ *update-backtrack-lvl* (*backtrack-lvl S* + *1*) *S*

definition *state-eq* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* (**infix** $\sim$ *50*) **where**
S $\sim$ *T* $\longleftrightarrow$ *state S* = *state T*

lemma *state-eq-ref* [*simp*, *intro*]:
  $S \sim S$
  ⟨*proof*⟩

lemma *state-eq-sym*:
  $S \sim T \longleftrightarrow T \sim S$
  ⟨*proof*⟩

lemma *state-eq-trans*:
  $S \sim T \Longrightarrow T \sim U \Longrightarrow S \sim U$
  ⟨*proof*⟩

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

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

lemma *atms-of-ms-learned-clss-restart-state-in-atms-of-ms-learned-clssI* [*intro*]:
  $x \in$ *atms-of-msu* (*learned-clss* (*restart-state* S)) $\Longrightarrow x \in$ *atms-of-msu* (*learned-clss* S)
  ⟨*proof*⟩

function *reduce-trail-to* :: $'a$ *list* $\Rightarrow$ $'st \Rightarrow$ $'st$ **where**
reduce-trail-to F S =
  (*if length* (*trail* S) = *length* $F \vee$ *trail* $S = []$ *then* S *else reduce-trail-to* F (*tl-trail* S))
⟨*proof*⟩
termination
  ⟨*proof*⟩

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

lemma
  shows
  reduce-trail-to-nil[*simp*]: *trail* $S = [] \Longrightarrow$ *reduce-trail-to* F $S = S$ **and**
  reduce-trail-to-eq-length[*simp*]: *length* (*trail* S) = *length* $F \Longrightarrow$ *reduce-trail-to* F $S = S$
  ⟨*proof*⟩

lemma *reduce-trail-to-length-ne*:
  length (*trail* S) $\neq$ *length* $F \Longrightarrow$ *trail* $S \neq [] \Longrightarrow$
    reduce-trail-to F S = *reduce-trail-to* F (*tl-trail* S)
  ⟨*proof*⟩

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

lemma *trail-reduce-trail-to-nil*[*simp*]:
trail (*reduce-trail-to* [] S) = []
⟨*proof*⟩

lemma *clauses-reduce-trail-to-nil*:
clauses (*reduce-trail-to* [] S) = *clauses* S
⟨*proof*⟩

lemma *reduce-trail-to-skip-beginning*:
assumes *trail* $S = F'$ @ F
shows *trail* (*reduce-trail-to* F S) = F
⟨*proof*⟩

lemma *clauses-reduce-trail-to*[*simp*]:
clauses (*reduce-trail-to* F S) = *clauses* S
⟨*proof*⟩

lemma *conflicting-update-trial*[*simp*]:
conflicting (*reduce-trail-to* F S) = *conflicting* S
⟨*proof*⟩

lemma *backtrack-lvl-update-trial*[*simp*]:
backtrack-lvl (*reduce-trail-to* F S) = *backtrack-lvl* S
⟨*proof*⟩

lemma *init-clss-update-trial*[*simp*]:
init-clss (*reduce-trail-to* F S) = *init-clss* S
⟨*proof*⟩

lemma *learned-clss-update-trial*[*simp*]:
learned-clss (*reduce-trail-to* F S) = *learned-clss* S
⟨*proof*⟩

lemma *trail-eq-reduce-trail-to-eq*:
trail S = *trail* $T \Longrightarrow$ *trail* (*reduce-trail-to* F S) = *trail* (*reduce-trail-to* F T)
⟨*proof*⟩

lemma *reduce-trail-to-state-eq*$_{NOT}$*-compatible*:
assumes ST: $S \sim T$
shows *reduce-trail-to* F S $\sim$ *reduce-trail-to* F T
⟨*proof*⟩

lemma *reduce-trail-to-trail-tl-trail-decomp*[*simp*]:
trail $S = F'$ @ *Decided* K d # $F \Longrightarrow$ (*trail* (*reduce-trail-to* F S)) = F
⟨*proof*⟩

lemma *reduce-trail-to-add-learned-cls*[*simp*]:
no-dup (*trail* S) $\Longrightarrow$
trail (*reduce-trail-to* F (*add-learned-cls* C S)) = *trail* (*reduce-trail-to* F S)
⟨*proof*⟩

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

⟨*proof*⟩

lemma *reduce-trail-to-remove-learned-cls*[*simp*]:
trail (*reduce-trail-to F* (*remove-cls C S*)) = *trail* (*reduce-trail-to F S*)
⟨*proof*⟩

lemma *reduce-trail-to-update-conflicting*[*simp*]:
trail (*reduce-trail-to F* (*update-conflicting C S*)) = *trail* (*reduce-trail-to F S*)
⟨*proof*⟩

lemma *reduce-trail-to-update-backtrack-lvl*[*simp*]:
trail (*reduce-trail-to F* (*update-backtrack-lvl C S*)) = *trail* (*reduce-trail-to F S*)
⟨*proof*⟩

lemma *in-get-all-decided-decomposition-decided-or-empty*:
assumes (*a*, *b*) ∈ *set* (*get-all-decided-decomposition M*)
shows *a* = [] ∨ (*is-decided* (*hd a*))
⟨*proof*⟩

lemma *in-get-all-decided-decomposition-trail-update-trail*[*simp*]:
assumes *H*: (*L* # *M1*, *M2*) ∈ *set* (*get-all-decided-decomposition* (*trail S*))
shows *trail* (*reduce-trail-to M1 S*) = *M1*
⟨*proof*⟩

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

lemma *trail-append-trail*:
no-dup (*M* @ *trail S*) ⟹ *trail* (*append-trail M S*) = *rev M* @ *trail S*
⟨*proof*⟩

lemma *init-clss-append-trail*:
no-dup (*M* @ *trail S*) ⟹ *init-clss* (*append-trail M S*) = *init-clss S*
⟨*proof*⟩

lemma *learned-clss-append-trail*:
no-dup (*M* @ *trail S*) ⟹ *learned-clss* (*append-trail M S*) = *learned-clss S*
⟨*proof*⟩

lemma *conflicting-append-trail*:
no-dup (*M* @ *trail S*) ⟹ *conflicting* (*append-trail M S*) = *conflicting S*
⟨*proof*⟩

lemma *backtrack-lvl-append-trail*:
no-dup (*M* @ *trail S*) ⟹ *backtrack-lvl* (*append-trail M S*) = *backtrack-lvl S*
⟨*proof*⟩

lemma *clauses-append-trail*:
no-dup (*M* @ *trail S*) ⟹ *clauses* (*append-trail M S*) = *clauses S*
⟨*proof*⟩

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

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

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

end

5.2 Special Instantiation: using Triples as State

5.3 CDCL Rules

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

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

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

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

inductive *propagate* :: *'st* ⇒ *'st* ⇒ *bool* **where**
propagate-rule[*intro*]:
 state S = (*M*, *N*, *U*, *k*, *None*) ⟹ *C* + {#*L*#} ∈# *clauses S* ⟹ *M* ⊨*as CNot C*
 ⟹ *undefined-lit* (*trail S*) *L*
 ⟹ *T* ∼ *cons-trail* (*Propagated L* (*C* + {#*L*#})) *S*
 ⟹ *propagate S T*
inductive-cases *propagateE*[*elim*]: *propagate S T*
thm *propagateE*

inductive *conflict* :: *'st* ⇒ *'st* ⇒ *bool* **where**
conflict-rule[*intro*]: *state S* = (*M*, *N*, *U*, *k*, *None*) ⟹ *D* ∈# *clauses S* ⟹ *M* ⊨*as CNot D*
 ⟹ *T* ∼ *update-conflicting* (*Some D*) *S*
 ⟹ *conflict S T*

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

inductive *backtrack* :: *'st* ⇒ *'st* ⇒ *bool* **where**
backtrack-rule[*intro*]: *state S* = (*M*, *N*, *U*, *k*, *Some* (*D* + {#*L*#}))
 ⟹ (*Decided K* (*i+1*) # *M1*, *M2*) ∈ *set* (*get-all-decided-decomposition M*)

$\Longrightarrow$ *get-level M L = k*
$\Longrightarrow$ *get-level M L = get-maximum-level M (D+{#L#})*
$\Longrightarrow$ *get-maximum-level M D = i*
$\Longrightarrow$ *T ∼ cons-trail (Propagated L (D+{#L#}))*
(reduce-trail-to M1
(add-learned-cls (D + {#L#})
(update-backtrack-lvl i
(update-conflicting None S))))
$\Longrightarrow$ *backtrack S T*
inductive-cases *backtrackE[elim]: backtrack S S′*
thm *backtrackE*

inductive *decide* :: *′st ⇒ ′st ⇒ bool* **where**
decide-rule[intro]: state S = (M, N, U, k, None)
$\Longrightarrow$ *undefined-lit M L* $\Longrightarrow$ *atm-of L ∈ atms-of-msu (init-clss S)*
$\Longrightarrow$ *T ∼ cons-trail (Decided L (k+1)) (incr-lvl S)*
$\Longrightarrow$ *decide S T*
inductive-cases *decideE[elim]: decide S S′*
thm *decideE*

inductive *skip* :: *′st ⇒ ′st ⇒ bool* **where**
skip-rule[intro]: state S = (Propagated L C′ # M, N, U, k, Some D) $\Longrightarrow$ *−L ∉# D* $\Longrightarrow$ *D ≠ {#}*
$\Longrightarrow$ *T ∼ tl-trail S*
$\Longrightarrow$ *skip S T*
inductive-cases *skipE[elim]: skip S S′*
thm *skipE*

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

inductive *resolve* :: *′st ⇒ ′st ⇒ bool* **where**
resolve-rule[intro]:
state S = (Propagated L (C + {#L#}) # M, N, U, k, Some (D + {#−L#}))
$\Longrightarrow$ *get-maximum-level (Propagated L (C + {#L#}) # M) D = k*
$\Longrightarrow$ *T ∼ update-conflicting (Some (D #∪ C)) (tl-trail S)*
$\Longrightarrow$ *resolve S T*
inductive-cases *resolveE[elim]: resolve S S′*
thm *resolveE*

inductive *restart* :: *′st ⇒ ′st ⇒ bool* **where**
restart: state S = (M, N, U, k, None) $\Longrightarrow$ *¬M ⊨asm clauses S*
$\Longrightarrow$ *T ∼ restart-state S*
$\Longrightarrow$ *restart S T*
inductive-cases *restartE[elim]: restart S T*
thm *restartE*

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

inductive *forget* :: *′st ⇒ ′st ⇒ bool* **where**
forget-rule: state S = (M, N, {#C#} + U, k, None)
$\Longrightarrow$ *¬M ⊨asm clauses S*
$\Longrightarrow$ *C ∉ set (get-all-mark-of-propagated (trail S))*
$\Longrightarrow$ *C ∉# init-clss S*
$\Longrightarrow$ *C ∈# learned-clss S*
$\Longrightarrow$ *T ∼ remove-cls C S*
$\Longrightarrow$ *forget S T*
inductive-cases *forgetE[elim]: forget S T*

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

inductive $cdcl_W$-*bj* :: *'st* ⇒ *'st* ⇒ *bool* **where**
skip[*intro*]: *skip* S S' ⟹ $cdcl_W$-*bj* S S' |
resolve[*intro*]: *resolve* S S' ⟹ $cdcl_W$-*bj* S S' |
backtrack[*intro*]: *backtrack* S S' ⟹ $cdcl_W$-*bj* S S'

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

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

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

lemma *rtranclp-propagate-is-rtranclp-*$cdcl_W$:
  $propagate^{**}$ S S' ⟹ $cdcl_W{}^{**}$ S S'
  ⟨*proof*⟩

lemma $cdcl_W$-*all-rules-induct*[*consumes 1*, *case-names propagate conflict forget restart decide skip resolve backtrack*]:
  fixes S :: *'st*
  assumes
    $cdcl_W$: $cdcl_W$ S S' **and**
    propagate: ⋀T. *propagate* S T ⟹ P S T **and**
    conflict: ⋀T. *conflict* S T ⟹ P S T **and**
    forget: ⋀T. *forget* S T ⟹ P S T **and**
    restart: ⋀T. *restart* S T ⟹ P S T **and**
    decide: ⋀T. *decide* S T ⟹ P S T **and**
    skip: ⋀T. *skip* S T ⟹ P S T **and**
    resolve: ⋀T. *resolve* S T ⟹ P S T **and**
    backtrack: ⋀T. *backtrack* S T ⟹ P S T
  shows P S S'
  ⟨*proof*⟩

lemma $cdcl_W$-*all-induct*[*consumes 1*, *case-names propagate conflict forget restart decide skip resolve backtrack*]:
  fixes S :: *'st*
  assumes
    $cdcl_W$: $cdcl_W$ S S' **and**
    propagateH: ⋀C L T. C + {#L#} ∈# *clauses* S ⟹ *trail* S ⊨*as CNot* C
      ⟹ *undefined-lit* (*trail* S) L ⟹ *conflicting* S = *None*
      ⟹ T ∼ *cons-trail* (*Propagated* L (C + {#L#})) S
      ⟹ P S T **and**
    conflictH: ⋀D T. D ∈# *clauses* S ⟹ *conflicting* S = *None* ⟹ *trail* S ⊨*as CNot* D
      ⟹ T ∼ *update-conflicting* (*Some* D) S
      ⟹ P S T **and**
    forgetH: ⋀C T. ¬*trail* S ⊨*asm clauses* S

$\Longrightarrow C \notin set\ (get\text{-}all\text{-}mark\text{-}of\text{-}propagated\ (trail\ S))$
$\Longrightarrow C \notin\#\ init\text{-}clss\ S$
$\Longrightarrow C \in\#\ learned\text{-}clss\ S$
$\Longrightarrow conflicting\ S = None$
$\Longrightarrow T \sim remove\text{-}cls\ C\ S$
$\Longrightarrow P\ S\ T$ **and**
$restartH$: $\bigwedge T.\ \neg trail\ S \models asm\ clauses\ S$
$\Longrightarrow conflicting\ S = None$
$\Longrightarrow T \sim restart\text{-}state\ S$
$\Longrightarrow P\ S\ T$ **and**
$decideH$: $\bigwedge L\ T.\ conflicting\ S = None \Longrightarrow\ undefined\text{-}lit\ (trail\ S)\ L$
$\Longrightarrow atm\text{-}of\ L \in atms\text{-}of\text{-}msu\ (init\text{-}clss\ S)$
$\Longrightarrow T \sim cons\text{-}trail\ (Decided\ L\ (backtrack\text{-}lvl\ S\ +1))\ (incr\text{-}lvl\ S)$
$\Longrightarrow P\ S\ T$ **and**
$skipH$: $\bigwedge L\ C'\ M\ D\ T.\ trail\ S = Propagated\ L\ C'\ \#\ M$
$\Longrightarrow conflicting\ S = Some\ D \Longrightarrow -L \notin\#\ D \Longrightarrow D \neq \{\#\}$
$\Longrightarrow T \sim tl\text{-}trail\ S$
$\Longrightarrow P\ S\ T$ **and**
$resolveH$: $\bigwedge L\ C\ M\ D\ T.$
$trail\ S = Propagated\ L\ (\ (C + \{\#L\#\}))\ \#\ M$
$\Longrightarrow conflicting\ S = Some\ (D + \{\#-L\#\})$
$\Longrightarrow get\text{-}maximum\text{-}level\ (Propagated\ L\ (C + \{\#L\#\})\ \#\ M)\ D = backtrack\text{-}lvl\ S$
$\Longrightarrow T \sim (update\text{-}conflicting\ (Some\ (D\ \#\cup\ C))\ (tl\text{-}trail\ S))$
$\Longrightarrow P\ S\ T$ **and**
$backtrackH$: $\bigwedge K\ i\ M1\ M2\ L\ D\ T.$
$(Decided\ K\ (Suc\ i)\ \#\ M1,\ M2) \in set\ (get\text{-}all\text{-}decided\text{-}decomposition\ (trail\ S))$
$\Longrightarrow get\text{-}level\ (trail\ S)\ L = backtrack\text{-}lvl\ S$
$\Longrightarrow conflicting\ S = Some\ (D + \{\#L\#\})$
$\Longrightarrow get\text{-}maximum\text{-}level\ (trail\ S)\ (D{+}\{\#L\#\}) = get\text{-}level\ (trail\ S)\ L$
$\Longrightarrow get\text{-}maximum\text{-}level\ (trail\ S)\ D \equiv i$
$\Longrightarrow T \sim cons\text{-}trail\ (Propagated\ L\ (D{+}\{\#L\#\}))$
$(reduce\text{-}trail\text{-}to\ M1$
$(add\text{-}learned\text{-}cls\ (D + \{\#L\#\})$
$(update\text{-}backtrack\text{-}lvl\ i$
$(update\text{-}conflicting\ None\ S))))$
$\Longrightarrow P\ S\ T$
shows $P\ S\ S'$
$\langle proof\rangle$

lemma $cdcl_W\text{-}o\text{-}induct[consumes\ 1,\ case\text{-}names\ decide\ skip\ resolve\ backtrack]$:
fixes $S\ ::\ 'st$
assumes $cdcl_W$: $cdcl_W\text{-}o\ S\ T$ **and**
$decideH$: $\bigwedge L\ T.\ conflicting\ S = None \Longrightarrow undefined\text{-}lit\ (trail\ S)\ L$
$\Longrightarrow atm\text{-}of\ L \in atms\text{-}of\text{-}msu\ (init\text{-}clss\ S)$
$\Longrightarrow T \sim cons\text{-}trail\ (Decided\ L\ (backtrack\text{-}lvl\ S\ +1))\ (incr\text{-}lvl\ S)$
$\Longrightarrow P\ S\ T$ **and**
$skipH$: $\bigwedge L\ C'\ M\ D\ T.\ trail\ S = Propagated\ L\ C'\ \#\ M$
$\Longrightarrow conflicting\ S = Some\ D \Longrightarrow -L \notin\#\ D \Longrightarrow D \neq \{\#\}$
$\Longrightarrow T \sim tl\text{-}trail\ S$
$\Longrightarrow P\ S\ T$ **and**
$resolveH$: $\bigwedge L\ C\ M\ D\ T.$
$trail\ S = Propagated\ L\ (\ (C + \{\#L\#\}))\ \#\ M$
$\Longrightarrow conflicting\ S = Some\ (D + \{\#-L\#\})$
$\Longrightarrow get\text{-}maximum\text{-}level\ (Propagated\ L\ (C + \{\#L\#\})\ \#\ M)\ D = backtrack\text{-}lvl\ S$
$\Longrightarrow T \sim update\text{-}conflicting\ (Some\ (D\ \#\cup\ C))\ (tl\text{-}trail\ S)$

$\Longrightarrow$ *P S T* **and**
backtrackH: $\bigwedge$*K i M1 M2 L D T*.
(*Decided K* (*Suc i*) # *M1*, *M2*) ∈ *set* (*get-all-decided-decomposition* (*trail S*))
$\Longrightarrow$ *get-level* (*trail S*) *L* = *backtrack-lvl S*
$\Longrightarrow$ *conflicting S* = *Some* (*D* + {#*L*#})
$\Longrightarrow$ *get-level* (*trail S*) *L* = *get-maximum-level* (*trail S*) (*D*+{#*L*#})
$\Longrightarrow$ *get-maximum-level* (*trail S*) *D* ≡ *i*
$\Longrightarrow$ *T* ∼ *cons-trail* (*Propagated L* (*D*+{#*L*#}))
(*reduce-trail-to M1*
(*add-learned-cls* (*D* + {#*L*#})
(*update-backtrack-lvl i*
(*update-conflicting None S*))))
$\Longrightarrow$ *P S T*
shows *P S T*
⟨*proof*⟩

thm $cdcl_W$*-o.induct*
lemma $cdcl_W$*-o-all-rules-induct*[*consumes 1*, *case-names decide backtrack skip resolve*]:
fixes *S T* :: *'st*
assumes
$cdcl_W$*-o S T* **and**
$\bigwedge$*T*. *decide S T* $\Longrightarrow$ *P S T* **and**
$\bigwedge$*T*. *backtrack S T* $\Longrightarrow$ *P S T* **and**
$\bigwedge$*T*. *skip S T* $\Longrightarrow$ *P S T* **and**
$\bigwedge$*T*. *resolve S T* $\Longrightarrow$ *P S T*
shows *P S T*
⟨*proof*⟩

lemma $cdcl_W$*-o-rule-cases*[*consumes 1*, *case-names decide backtrack skip resolve*]:
fixes *S T* :: *'st*
assumes
$cdcl_W$*-o S T* **and**
decide S T $\Longrightarrow$ *P* **and**
backtrack S T $\Longrightarrow$ *P* **and**
skip S T $\Longrightarrow$ *P* **and**
resolve S T $\Longrightarrow$ *P*
shows *P*
⟨*proof*⟩

5.4 Invariants

5.4.1 Properties of the trail

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

lemma *backtrack-lit-skiped*:
assumes *L*: *get-level* (*trail S*) *L* = *backtrack-lvl S*
and *M1*: (*Decided K* (*i* + *1*) # *M1*, *M2*) ∈ *set* (*get-all-decided-decomposition* (*trail S*))
and *no-dup*: *no-dup* (*trail S*)
and *bt-l*: *backtrack-lvl S* = *length* (*get-all-levels-of-decided* (*trail S*))
and *order*: *get-all-levels-of-decided* (*trail S*)
= *rev* ([*1*..<(*1*+*length* (*get-all-levels-of-decided* (*trail S*)))])
shows *atm-of L* ∉ *atm-of* ' *lits-of M1*
⟨*proof*⟩

lemma $cdcl_W$*-distinctinv-1*:
assumes
$cdcl_W$ S S' **and**
no-dup (*trail* S) **and**
backtrack-lvl S = *length* (*get-all-levels-of-decided* (*trail* S)) **and**
get-all-levels-of-decided (*trail* S) = *rev* [*1*..<*1*+*length* (*get-all-levels-of-decided* (*trail* S))]
shows *no-dup* (*trail* S')
⟨*proof*⟩

lemma $cdcl_W$*-consistent-inv-2*:
assumes
$cdcl_W$ S S' **and**
no-dup (*trail* S) **and**
backtrack-lvl S = *length* (*get-all-levels-of-decided* (*trail* S)) **and**
get-all-levels-of-decided (*trail* S) = *rev* [*1*..<*1*+*length* (*get-all-levels-of-decided* (*trail* S))]
shows *consistent-interp* (*lits-of* (*trail* S'))
⟨*proof*⟩

lemma $cdcl_W$*-o-bt*:
assumes
$cdcl_W$*-o* S S' **and**
backtrack-lvl S = *length* (*get-all-levels-of-decided* (*trail* S)) **and**
get-all-levels-of-decided (*trail* S) =
rev ([*1*..<(*1*+*length* (*get-all-levels-of-decided* (*trail* S)))]) **and**
n-d[*simp*]: *no-dup* (*trail* S)
shows *backtrack-lvl* S' = *length* (*get-all-levels-of-decided* (*trail* S'))
⟨*proof*⟩

lemma $cdcl_W$*-rf-bt*:
assumes
$cdcl_W$*-rf* S S' **and**
backtrack-lvl S = *length* (*get-all-levels-of-decided* (*trail* S)) **and**
get-all-levels-of-decided (*trail* S) = *rev* [*1*..<(*1*+*length* (*get-all-levels-of-decided* (*trail* S)))]
shows *backtrack-lvl* S' = *length* (*get-all-levels-of-decided* (*trail* S'))
⟨*proof*⟩

lemma $cdcl_W$*-bt*:
assumes
$cdcl_W$ S S' **and**
backtrack-lvl S = *length* (*get-all-levels-of-decided* (*trail* S)) **and**
get-all-levels-of-decided (*trail* S)
= *rev* ([*1*..<(*1*+*length* (*get-all-levels-of-decided* (*trail* S)))]) **and**
no-dup (*trail* S)
shows *backtrack-lvl* S' = *length* (*get-all-levels-of-decided* (*trail* S'))
⟨*proof*⟩

lemma $cdcl_W$*-bt-level'*:
assumes
$cdcl_W$ S S' **and**
backtrack-lvl S = *length* (*get-all-levels-of-decided* (*trail* S)) **and**
get-all-levels-of-decided (*trail* S)
= *rev* ([*1*..<(*1*+*length* (*get-all-levels-of-decided* (*trail* S)))]) **and**
n-d: *no-dup* (*trail* S)
shows *get-all-levels-of-decided* (*trail* S')
= *rev* ([*1*..<(*1*+*length* (*get-all-levels-of-decided* (*trail* S')))])

⟨*proof*⟩

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

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

lemma $cdcl_W$*-M-level-inv-decomp*:
assumes $cdcl_W$*-M-level-inv S*
shows *consistent-interp* (*lits-of* (*trail S*))
and *no-dup* (*trail S*)
⟨*proof*⟩

lemma $cdcl_W$*-consistent-inv*:
fixes *S S'* :: *'st*
assumes
$cdcl_W$ *S S'* **and**
$cdcl_W$*-M-level-inv S*
shows $cdcl_W$*-M-level-inv S'*
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-consistent-inv*:
assumes $cdcl_W^{**}$ *S S'*
and $cdcl_W$*-M-level-inv S*
shows $cdcl_W$*-M-level-inv S'*
⟨*proof*⟩

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

lemma $cdcl_W$*-M-level-inv-S0-*$cdcl_W$[*simp*]:
$cdcl_W$*-M-level-inv* (*init-state N*)
⟨*proof*⟩

lemma $cdcl_W$*-M-level-inv-get-level-le-backtrack-lvl*:
assumes *inv*: $cdcl_W$*-M-level-inv S*
shows *get-level* (*trail S*) *L* ≤ *backtrack-lvl S*
⟨*proof*⟩

lemma *backtrack-ex-decomp*:
assumes *M-l*: $cdcl_W$*-M-level-inv S*
and *i-S*: *i* < *backtrack-lvl S*
shows ∃*K M1 M2*. (*Decided K* (*i* + *1*) # *M1*, *M2*) ∈ *set* (*get-all-decided-decomposition* (*trail S*))
⟨*proof*⟩

5.4.2 Better-Suited Induction Principle

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

lemma *backtrack-induction-lev*[*consumes 1*, *case-names M-devel-inv backtrack*]:
  assumes
    bt: *backtrack S T* **and**
    inv: $cdcl_W$*-M-level-inv S* **and**
    backtrackH: $\bigwedge$*K i M1 M2 L D T*.
      (*Decided K* (*Suc i*) # *M1*, *M2*) ∈ *set* (*get-all-decided-decomposition* (*trail S*))
      ⟹ *get-level* (*trail S*) *L* = *backtrack-lvl S*
      ⟹ *conflicting S* = *Some* (*D* + {#*L*#})
      ⟹ *get-level* (*trail S*) *L* = *get-maximum-level* (*trail S*) (*D*+{#*L*#})
      ⟹ *get-maximum-level* (*trail S*) *D* ≡ *i*
      ⟹ *undefined-lit M1 L*
      ⟹ *T* ∼ *cons-trail* (*Propagated L* (*D*+{#*L*#}))
            (*reduce-trail-to M1*
              (*add-learned-cls* (*D* + {#*L*#})
                (*update-backtrack-lvl i*
                  (*update-conflicting None S*))))
      ⟹ *P S T*
  shows *P S T*
⟨*proof*⟩

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

lemma $cdcl_W$*-all-induct-lev-full*:
  fixes *S* :: *'st*
  assumes
    $cdcl_W$: $cdcl_W$ *S S'* **and**
    inv[*simp*]: $cdcl_W$*-M-level-inv S* **and**
    propagateH: $\bigwedge$*C L T*. *C* + {#*L*#} ∈# *clauses S* ⟹ *trail S* ⊨*as CNot C*
      ⟹ *undefined-lit* (*trail S*) *L* ⟹ *conflicting S* = *None*
      ⟹ *T* ∼ *cons-trail* (*Propagated L* (*C* + {#*L*#})) *S*
      ⟹ $cdcl_W$*-M-level-inv S*
      ⟹ *P S T* **and**
    conflictH: $\bigwedge$*D T*. *D* ∈# *clauses S* ⟹ *conflicting S* = *None* ⟹ *trail S* ⊨*as CNot D*
      ⟹ *T* ∼ *update-conflicting* (*Some D*) *S*
      ⟹ *P S T* **and**
    forgetH: $\bigwedge$*C T*. ¬*trail S* ⊨*asm clauses S*
      ⟹ *C* ∉ *set* (*get-all-mark-of-propagated* (*trail S*))
      ⟹ *C* ∉# *init-clss S*
      ⟹ *C* ∈# *learned-clss S*
      ⟹ *conflicting S* = *None*
      ⟹ *T* ∼ *remove-cls C S*
      ⟹ $cdcl_W$*-M-level-inv S*
      ⟹ *P S T* **and**
    restartH: $\bigwedge$*T*. ¬*trail S* ⊨*asm clauses S*
      ⟹ *conflicting S* = *None*
      ⟹ *T* ∼ *restart-state S*
      ⟹ $cdcl_W$*-M-level-inv S*
      ⟹ *P S T* **and**
    decideH: $\bigwedge$*L T*. *conflicting S* = *None* ⟹ *undefined-lit* (*trail S*) *L*
      ⟹ *atm-of L* ∈ *atms-of-msu* (*init-clss S*)

$\Longrightarrow T \sim$ *cons-trail* (*Decided* L (*backtrack-lvl* S +*1*)) (*incr-lvl* S)
$\Longrightarrow$ *cdcl*$_W$*-M-level-inv* S
$\Longrightarrow P\ S\ T$ **and**
skipH: $\bigwedge L\ C'\ M\ D\ T$. *trail* S = *Propagated* $L\ C'$ # M
$\Longrightarrow$ *conflicting* S = *Some* $D \Longrightarrow -L \notin\#\ D \Longrightarrow D \neq \{\#\}$
$\Longrightarrow T \sim$ *tl-trail* S
$\Longrightarrow$ *cdcl*$_W$*-M-level-inv* S
$\Longrightarrow P\ S\ T$ **and**
resolveH: $\bigwedge L\ C\ M\ D\ T$.
trail S = *Propagated* L ((C + {#L#})) # M
$\Longrightarrow$ *conflicting* S = *Some* (D + {#$-L$#})
$\Longrightarrow$ *get-maximum-level* (*Propagated* L (C + {#L#}) # M) D = *backtrack-lvl* S
$\Longrightarrow T \sim$ (*update-conflicting* (*Some* (D #∪ C)) (*tl-trail* S))
$\Longrightarrow$ *cdcl*$_W$*-M-level-inv* S
$\Longrightarrow P\ S\ T$ **and**
backtrackH: $\bigwedge K\ i\ M1\ M2\ L\ D\ T$.
(*Decided* K (*Suc* i) # *M1*, *M2*) $\in$ *set* (*get-all-decided-decomposition* (*trail* S))
$\Longrightarrow$ *get-level* (*trail* S) L = *backtrack-lvl* S
$\Longrightarrow$ *conflicting* S = *Some* (D + {#L#})
$\Longrightarrow$ *get-maximum-level* (*trail* S) (D+{#L#}) = *get-level* (*trail* S) L
$\Longrightarrow$ *get-maximum-level* (*trail* S) $D \equiv i$
$\Longrightarrow$ *undefined-lit M1* L
$\Longrightarrow T \sim$ *cons-trail* (*Propagated* L (D+{#L#}))
(*reduce-trail-to M1*
(*add-learned-cls* (D + {#L#})
(*update-backtrack-lvl* i
(*update-conflicting None* S))))
$\Longrightarrow$ *cdcl*$_W$*-M-level-inv* S
$\Longrightarrow P\ S\ T$
shows $P\ S\ S'$
⟨*proof*⟩

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

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

thm *cdcl*$_W$*-o-induct*
lemma *cdcl*$_W$*-o-induct-lev*[*consumes 1*, *case-names M-lev decide skip resolve backtrack*]:
fixes S :: *'st*
assumes
cdcl$_W$: *cdcl*$_W$*-o* $S\ T$ **and**
inv[*simp*]: *cdcl*$_W$*-M-level-inv* S **and**
decideH: $\bigwedge L\ T$. *conflicting* S = *None* $\Longrightarrow$ *undefined-lit* (*trail* S) L
$\Longrightarrow$ *atm-of* $L \in$ *atms-of-msu* (*init-clss* S)
$\Longrightarrow T \sim$ *cons-trail* (*Decided* L (*backtrack-lvl* S +*1*)) (*incr-lvl* S)
$\Longrightarrow$ *cdcl*$_W$*-M-level-inv* S
$\Longrightarrow P\ S\ T$ **and**
skipH: $\bigwedge L\ C'\ M\ D\ T$. *trail* S = *Propagated* $L\ C'$ # M
$\Longrightarrow$ *conflicting* S = *Some* $D \Longrightarrow -L \notin\#\ D \Longrightarrow D \neq \{\#\}$
$\Longrightarrow T \sim$ *tl-trail* S
$\Longrightarrow$ *cdcl*$_W$*-M-level-inv* S
$\Longrightarrow P\ S\ T$ **and**
resolveH: $\bigwedge L\ C\ M\ D\ T$.

trail S = *Propagated L* ($(C + \{\#L\#\})$) # *M*
$\Longrightarrow$ *conflicting S* = *Some* $(D + \{\#-L\#\})$
$\Longrightarrow$ *get-maximum-level* (*Propagated L* $(C + \{\#L\#\})$ # *M*) *D* = *backtrack-lvl S*
$\Longrightarrow$ *T* $\sim$ *update-conflicting* (*Some* (*D* #∪ *C*)) (*tl-trail S*)
$\Longrightarrow$ $cdcl_W$*-M-level-inv S*
$\Longrightarrow$ *P S T* **and**
backtrackH: $\bigwedge$*K i M1 M2 L D T*.
(*Decided K* (*Suc i*) # *M1*, *M2*) ∈ *set* (*get-all-decided-decomposition* (*trail S*))
$\Longrightarrow$ *get-level* (*trail S*) *L* = *backtrack-lvl S*
$\Longrightarrow$ *conflicting S* = *Some* $(D + \{\#L\#\})$
$\Longrightarrow$ *get-level* (*trail S*) *L* = *get-maximum-level* (*trail S*) $(D+\{\#L\#\})$
$\Longrightarrow$ *get-maximum-level* (*trail S*) *D* ≡ *i*
$\Longrightarrow$ *undefined-lit M1 L*
$\Longrightarrow$ *T* $\sim$ *cons-trail* (*Propagated L* $(D+\{\#L\#\})$)
(*reduce-trail-to M1*
(*add-learned-cls* $(D + \{\#L\#\})$
(*update-backtrack-lvl i*
(*update-conflicting None S*))))
$\Longrightarrow$ $cdcl_W$*-M-level-inv S*
$\Longrightarrow$ *P S T*
shows *P S T*
⟨*proof*⟩

lemmas $cdcl_W$*-o-induct-lev2* = $cdcl_W$*-o-induct-lev*[*consumes 2*, *case-names decide skip resolve backtrack*]

5.4.3 Compatibility with *op* $\sim$

lemma *propagate-state-eq-compatible*:
assumes
propagate S T **and**
$S \sim S'$ **and**
$T \sim T'$
shows *propagate* S' T'
⟨*proof*⟩

lemma *conflict-state-eq-compatible*:
assumes
conflict S T **and**
$S \sim S'$ **and**
$T \sim T'$
shows *conflict* S' T'
⟨*proof*⟩

lemma *backtrack-state-eq-compatible*:
assumes
backtrack S T **and**
$S \sim S'$ **and**
$T \sim T'$ **and**
inv: $cdcl_W$*-M-level-inv S*
shows *backtrack* S' T'
⟨*proof*⟩

lemma *decide-state-eq-compatible*:
assumes
decide S T **and**

$S \sim S'$ **and**
$T \sim T'$
shows *decide* S' T'
⟨*proof*⟩

lemma *skip-state-eq-compatible*:
assumes
skip S T **and**
$S \sim S'$ **and**
$T \sim T'$
shows *skip* S' T'
⟨*proof*⟩

lemma *resolve-state-eq-compatible*:
assumes
resolve S T **and**
$S \sim S'$ **and**
$T \sim T'$
shows *resolve* S' T'
⟨*proof*⟩

lemma *forget-state-eq-compatible*:
assumes
forget S T **and**
$S \sim S'$ **and**
$T \sim T'$
shows *forget* S' T'
⟨*proof*⟩

lemma $cdcl_W$*-state-eq-compatible*:
assumes
$cdcl_W$ S T **and** $\neg$*restart* S T **and**
$S \sim S'$ **and**
$T \sim T'$ **and**
inv: $cdcl_W$*-M-level-inv* S
shows $cdcl_W$ S' T'
⟨*proof*⟩

lemma $cdcl_W$*-bj-state-eq-compatible*:
assumes
$cdcl_W$*-bj* S T **and** $cdcl_W$*-M-level-inv* S
$S \sim S'$ **and**
$T \sim T'$
shows $cdcl_W$*-bj* S' T'
⟨*proof*⟩

lemma *tranclp-*$cdcl_W$*-bj-state-eq-compatible*:
assumes
$cdcl_W$*-bj*$^{++}$ S T **and** *inv*: $cdcl_W$*-M-level-inv* S **and**
$S \sim S'$ **and**
$T \sim T'$
shows $cdcl_W$*-bj*$^{++}$ S' T'
⟨*proof*⟩

5.4.4 Conservation of some Properties

lemma *level-of-decided-ge-1*:
  assumes
    $cdcl_W\ S\ S'$ **and**
    inv: $cdcl_W$*-M-level-inv* S **and**
    $\forall L\ l.\ Decided\ L\ l \in set\ (trail\ S) \longrightarrow l > 0$
  shows $\forall L\ l.\ Decided\ L\ l \in set\ (trail\ S') \longrightarrow l > 0$
  ⟨*proof*⟩

lemma $cdcl_W$*-o-no-more-init-clss*:
  assumes
    $cdcl_W$*-o* $S\ S'$ **and**
    inv: $cdcl_W$*-M-level-inv* S
  shows *init-clss* S = *init-clss* S'
  ⟨*proof*⟩

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-clss* S = *init-clss* S'
  ⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-o-no-more-init-clss*:
  assumes
    $cdcl_W\text{-}o^{**}\ S\ S'$ **and**
    inv: $cdcl_W$*-M-level-inv* S
  shows *init-clss* S = *init-clss* S'
  ⟨*proof*⟩

lemma $cdcl_W$*-init-clss*:
  $cdcl_W\ S\ T \Longrightarrow cdcl_W\text{-}M\text{-}level\text{-}inv\ S \Longrightarrow init\text{-}clss\ S = init\text{-}clss\ T$
  ⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-init-clss*:
  ${cdcl_W}^{**}\ S\ T \Longrightarrow cdcl_W\text{-}M\text{-}level\text{-}inv\ S \Longrightarrow init\text{-}clss\ S = init\text{-}clss\ T$
  ⟨*proof*⟩

lemma *tranclp-*$cdcl_W$*-init-clss*:
  ${cdcl_W}^{++}\ S\ T \Longrightarrow cdcl_W\text{-}M\text{-}level\text{-}inv\ S \Longrightarrow init\text{-}clss\ S = init\text{-}clss\ T$
  ⟨*proof*⟩

5.4.5 Learned Clause

This invariant shows that:

- the learned clauses are entailed by the initial set of clauses.
- the conflicting clause is entailed by the initial set of clauses.
- the marks are entailed by the clauses. A more precise version would be to show that either these decided are learned or are in the set of clauses

definition $cdcl_W$*-learned-clause* $(S{::}\ 'st) \longleftrightarrow$

(*init-clss* S $\models$*psm* *learned-clss* S
$\wedge$ ($\forall$ T. *conflicting* S = *Some* T $\longrightarrow$ *init-clss* S $\models$*pm* T)
$\wedge$ *set* (*get-all-mark-of-propagated* (*trail* S)) $\subseteq$ *set-mset* (*clauses* S))

lemma $cdcl_W$*-learned-clause-S0-*$cdcl_W$[*simp*]:
$cdcl_W$*-learned-clause* (*init-state* N)
⟨*proof*⟩

lemma $cdcl_W$*-learned-clss*:
assumes
$cdcl_W$ S S' **and**
learned: $cdcl_W$*-learned-clause* S **and**
lev-inv: $cdcl_W$*-M-level-inv* S
shows $cdcl_W$*-learned-clause* S'
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-learned-clss*:
assumes
${cdcl_W}^{**}$ S S' **and**
$cdcl_W$*-M-level-inv* S
$cdcl_W$*-learned-clause* S
shows $cdcl_W$*-learned-clause* S'
⟨*proof*⟩

5.4.6 No alien atom in the state

This invariant means that all the literals are in the set of clauses.

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

lemma *no-strange-atm-decomp*:
assumes *no-strange-atm* S
shows *conflicting* S = *Some* T $\Longrightarrow$ *atms-of* T $\subseteq$ *atms-of-msu* (*init-clss* S)
and ($\forall$ L *mark*. *Propagated* L *mark* $\in$ *set* (*trail* S)
$\longrightarrow$ *atms-of* (*mark*) $\subseteq$ *atms-of-msu* (*init-clss* S))
and *atms-of-msu* (*learned-clss* S) $\subseteq$ *atms-of-msu* (*init-clss* S)
and *atm-of* ‘ (*lits-of* (*trail* S)) $\subseteq$ *atms-of-msu* (*init-clss* S)
⟨*proof*⟩

lemma *no-strange-atm-S0* [*simp*]: *no-strange-atm* (*init-state* N)
⟨*proof*⟩

lemma $cdcl_W$*-no-strange-atm-explicit*:
assumes
$cdcl_W$ S S' **and**
lev: $cdcl_W$*-M-level-inv* S **and**
conf: $\forall$ T. *conflicting* S = *Some* T $\longrightarrow$ *atms-of* T $\subseteq$ *atms-of-msu* (*init-clss* S) **and**
decided: $\forall$ L *mark*. *Propagated* L *mark* $\in$ *set* (*trail* S)
$\longrightarrow$ *atms-of mark* $\subseteq$ *atms-of-msu* (*init-clss* S) **and**
learned: *atms-of-msu* (*learned-clss* S) $\subseteq$ *atms-of-msu* (*init-clss* S) **and**

trail: *atm-of* ' (*lits-of* (*trail* S)) $\subseteq$ *atms-of-msu* (*init-clss* S)
shows ($\forall$ T. *conflicting* S' = *Some* T $\longrightarrow$ *atms-of* T $\subseteq$ *atms-of-msu* (*init-clss* S')) $\wedge$
($\forall$ L *mark*. *Propagated* L *mark* $\in$ *set* (*trail* S')
$\longrightarrow$ *atms-of* (*mark*) $\subseteq$ *atms-of-msu* (*init-clss* S')) $\wedge$
atms-of-msu (*learned-clss* S') $\subseteq$ *atms-of-msu* (*init-clss* S') $\wedge$
atm-of ' (*lits-of* (*trail* S')) $\subseteq$ *atms-of-msu* (*init-clss* S') (**is** *?C* S' $\wedge$ *?M* S' $\wedge$ *?U* S' $\wedge$ *?V* S')
⟨*proof*⟩

lemma *cdcl*$_W$*-no-strange-atm-inv*:
assumes *cdcl*$_W$ S S' **and** *no-strange-atm* S **and** *cdcl*$_W$*-M-level-inv* S
shows *no-strange-atm* S'
⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-no-strange-atm-inv*:
assumes *cdcl*$_W$** S S' **and** *no-strange-atm* S **and** *cdcl*$_W$*-M-level-inv* S
shows *no-strange-atm* S'
⟨*proof*⟩

5.4.7 No duplicates all around

This invariant shows that there is no duplicate (no literal appearing twice in the formula). The last part could be proven using the previous invariant moreover.

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

lemma *distinct-cdcl*$_W$*-state-decomp*:
assumes *distinct-cdcl*$_W$*-state* (S::'*st*)
shows $\forall$ T. *conflicting* S = *Some* T $\longrightarrow$ *distinct-mset* T
and *distinct-mset-mset* (*learned-clss* S)
and *distinct-mset-mset* (*init-clss* S)
and $\forall$ L *mark*. (*Propagated* L *mark* $\in$ *set* (*trail* S) $\longrightarrow$ *distinct-mset* (*mark*))
⟨*proof*⟩

lemma *distinct-cdcl*$_W$*-state-decomp-2*:
assumes *distinct-cdcl*$_W$*-state* (S::'*st*)
shows *conflicting* S = *Some* T $\Longrightarrow$ *distinct-mset* T
⟨*proof*⟩

lemma *distinct-cdcl*$_W$*-state-S0-cdcl*$_W$[*simp*]:
distinct-mset-mset N $\Longrightarrow$ *distinct-cdcl*$_W$*-state* (*init-state* N)
⟨*proof*⟩

lemma *distinct-cdcl*$_W$*-state-inv*:
assumes
cdcl$_W$ S S' **and**
cdcl$_W$*-M-level-inv* S **and**
distinct-cdcl$_W$*-state* S
shows *distinct-cdcl*$_W$*-state* S'
⟨*proof*⟩

lemma *rtanclp-distinct-cdcl*$_W$*-state-inv*:
assumes

$cdcl_W{}^{**}$ S S' **and**
$cdcl_W$*-M-level-inv* S **and**
distinct-$cdcl_W$*-state* S
shows *distinct-*$cdcl_W$*-state* S'
⟨*proof*⟩

5.4.8 Conflicts and co

This invariant shows that each mark contains a contradiction only related to the previously defined variable.

abbreviation *every-mark-is-a-conflict* :: $'st \Rightarrow$ *bool* **where**
every-mark-is-a-conflict $S \equiv$
$\forall L$ *mark a b. a* @ *Propagated L mark* # $b =$ (*trail S*)
$\longrightarrow$ ($b \models$*as CNot* (*mark* $-$ {#L#}) $\wedge$ $L \in\#$ *mark*)

definition $cdcl_W$*-conflicting* $S \equiv$
($\forall T$. *conflicting* $S =$ *Some* $T \longrightarrow$ *trail* $S \models$*as CNot* T)
$\wedge$ *every-mark-is-a-conflict* S

lemma *backtrack-atms-of-D-in-M1*:
fixes *M1* :: ('v, *nat*, 'v *clause*) *ann-literals*
assumes
inv: $cdcl_W$*-M-level-inv* S **and**
undef: *undefined-lit M1 L* **and**
i: *get-maximum-level* (*trail S*) $D = i$ **and**
decomp: (*Decided K* (*Suc i*) # *M1*, *M2*)
$\in$ *set* (*get-all-decided-decomposition* (*trail S*)) **and**
S-lvl: *backtrack-lvl* $S =$ *get-maximum-level* (*trail S*) (D + {#L#}) **and**
S-confl: *conflicting* $S =$ *Some* (D + {#L#}) **and**
undef: *undefined-lit M1 L* **and**
T: $T \sim$ (*cons-trail* (*Propagated L* (D+{#L#}))
(*reduce-trail-to M1*
(*add-learned-cls* (D + {#L#})
(*update-backtrack-lvl i*
(*update-conflicting None S*))))) **and**
confl: $\forall T$. *conflicting* $S =$ *Some* $T \longrightarrow$ *trail* $S \models$*as CNot* T
shows *atms-of* $D \subseteq$ *atm-of* ‘ *lits-of* (*tl* (*trail T*))
⟨*proof*⟩

lemma *distinct-atms-of-incl-not-in-other*:
assumes
a1: *no-dup* (M @ M') **and** *a2*:
atms-of $D \subseteq$ *atm-of* ‘ *lits-of* M'
shows $\forall x \in$*atms-of D*. $x \notin$ *atm-of* ‘ *lits-of* M
⟨*proof*⟩

lemma $cdcl_W$*-propagate-is-conclusion*:
assumes
$cdcl_W$ S S' **and**
inv: $cdcl_W$*-M-level-inv* S **and**
decomp: *all-decomposition-implies-m* (*init-clss S*) (*get-all-decided-decomposition* (*trail S*)) **and**
learned: $cdcl_W$*-learned-clause* S **and**
confl: $\forall T$. *conflicting* $S =$ *Some* $T \longrightarrow$ *trail* $S \models$*as CNot* T **and**
alien: *no-strange-atm* S
shows *all-decomposition-implies-m* (*init-clss* S') (*get-all-decided-decomposition* (*trail* S'))

$\langle proof \rangle$

lemma *cdcl$_W$-propagate-is-false*:
assumes
cdcl$_W$ S S' **and**
lev: *cdcl$_W$-M-level-inv S* **and**
learned: *cdcl$_W$-learned-clause S* **and**
decomp: *all-decomposition-implies-m* (*init-clss S*) (*get-all-decided-decomposition* (*trail S*)) **and**
confl: $\forall$ *T*. *conflicting S = Some T* $\longrightarrow$ *trail S* $\models$*as CNot T* **and**
alien: *no-strange-atm S* **and**
mark-confl: *every-mark-is-a-conflict S*
shows *every-mark-is-a-conflict S'*
$\langle proof \rangle$

lemma *cdcl$_W$-conflicting-is-false*:
assumes
cdcl$_W$ S S' **and**
M-lev: *cdcl$_W$-M-level-inv S* **and**
confl-inv: $\forall$ *T*. *conflicting S = Some T* $\longrightarrow$ *trail S* $\models$*as CNot T* **and**
decided-confl: $\forall$ *L mark a b*. *a* @ *Propagated L mark* # *b* = (*trail S*)
$\longrightarrow$ (*b* $\models$*as CNot* (*mark* − {#*L*#}) $\wedge$ *L* $\in$# *mark*) **and**
dist: *distinct-cdcl$_W$-state S*
shows $\forall$ *T*. *conflicting S' = Some T* $\longrightarrow$ *trail S'* $\models$*as CNot T*
$\langle proof \rangle$

lemma *cdcl$_W$-conflicting-decomp*:
assumes *cdcl$_W$-conflicting S*
shows $\forall$ *T*. *conflicting S = Some T* $\longrightarrow$ *trail S* $\models$*as CNot T*
and $\forall$ *L mark a b*. *a* @ *Propagated L mark* # *b* = (*trail S*)
$\longrightarrow$ (*b* $\models$*as CNot* (*mark* − {#*L*#}) $\wedge$ *L* $\in$# *mark*)
$\langle proof \rangle$

lemma *cdcl$_W$-conflicting-decomp2*:
assumes *cdcl$_W$-conflicting S* **and** *conflicting S = Some T*
shows *trail S* $\models$*as CNot T*
$\langle proof \rangle$

lemma *cdcl$_W$-conflicting-decomp2'*:
assumes
cdcl$_W$-conflicting S **and**
conflicting S = Some D
shows *trail S* $\models$*as CNot D*
$\langle proof \rangle$

lemma *cdcl$_W$-conflicting-S0-cdcl$_W$*[*simp*]:
cdcl$_W$-conflicting (*init-state N*)
$\langle proof \rangle$

5.4.9 Putting all the invariants together

lemma *cdcl$_W$-all-inv*:
assumes *cdcl$_W$*: *cdcl$_W$ S S'* **and**
1: *all-decomposition-implies-m* (*init-clss S*) (*get-all-decided-decomposition* (*trail S*)) **and**
2: *cdcl$_W$-learned-clause S* **and**
4: *cdcl$_W$-M-level-inv S* **and**
5: *no-strange-atm S* **and**

7: *distinct-cdcl$_W$-state S* **and**
8: *cdcl$_W$-conflicting S*
shows *all-decomposition-implies-m* (*init-clss S'*) (*get-all-decided-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*⟩

lemma *rtranclp-cdcl$_W$-all-inv*:
assumes
cdcl$_W$: *rtranclp cdcl$_W$ S S'* **and**
1: *all-decomposition-implies-m* (*init-clss S*) (*get-all-decided-decomposition* (*trail S*)) **and**
2: *cdcl$_W$-learned-clause S* **and**
4: *cdcl$_W$-M-level-inv S* **and**
5: *no-strange-atm S* **and**
7: *distinct-cdcl$_W$-state S* **and**
8: *cdcl$_W$-conflicting S*
shows
all-decomposition-implies-m (*init-clss S'*) (*get-all-decided-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*⟩

lemma *all-invariant-S0-cdcl$_W$*:
assumes *distinct-mset-mset N*
shows *all-decomposition-implies-m* (*init-clss* (*init-state N*))
(*get-all-decided-decomposition* (*trail* (*init-state N*)))
and *cdcl$_W$-learned-clause* (*init-state N*)
and $\forall$ *T*. *conflicting* (*init-state N*) = *Some T* $\longrightarrow$ (*trail* (*init-state N*))$\models$*as CNot T*
and *no-strange-atm* (*init-state N*)
and *consistent-interp* (*lits-of* (*trail* (*init-state N*)))
and $\forall$ *L mark a b*. *a* @ *Propagated L mark* # *b* = *trail* (*init-state N*) $\longrightarrow$
(*b* $\models$*as CNot* (*mark* − {#*L*#}) $\wedge$ *L* $\in$# *mark*)
and *distinct-cdcl$_W$-state* (*init-state N*)
⟨*proof*⟩

lemma *cdcl$_W$-only-propagated-vars-unsat*:
assumes
decided: $\forall$ *x* $\in$ *set M*. $\neg$ *is-decided x* **and**
DN: *D* $\in$# *clauses S* **and**
D: *M* $\models$*as CNot D* **and**
inv: *all-decomposition-implies-m N* (*get-all-decided-decomposition M*) **and**
state: *state S* = (*M*, *N*, *U*, *k*, *C*) **and**
learned-cl: *cdcl$_W$-learned-clause S* **and**
atm-incl: *no-strange-atm S*
shows *unsatisfiable* (*set-mset N*)
⟨*proof*⟩

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

lemma
  assumes *all-decomposition-implies-m* N (*get-all-decided-decomposition* M)
  and $\forall\, m \in$ *set* M. $\neg$*is-decided* m
  shows *set-mset* N $\models$*ps unmark* M
$\langle$*proof*$\rangle$

lemma *conflict-with-false-implies-unsat*:
  assumes
    cdcl$_W$: *cdcl*$_W$ S S' **and**
    lev: *cdcl*$_W$*-M-level-inv* S **and**
    [*simp*]: *conflicting* S' = *Some* {#} **and**
    learned: *cdcl*$_W$*-learned-clause* S
  shows *unsatisfiable* (*set-mset* (*init-clss* S))
  $\langle$*proof*$\rangle$

lemma *conflict-with-false-implies-terminated*:
  assumes *cdcl*$_W$ S S'
  and *conflicting* S = *Some* {#}
  shows *False*
  $\langle$*proof*$\rangle$

5.4.10 No tautology is learned

This is a simple consequence of all we have shown previously. It is not strictly necessary, but helps finding a better bound on the number of learned clauses.

lemma *learned-clss-are-not-tautologies*:
  assumes
    cdcl$_W$ S S' **and**
    lev: *cdcl*$_W$*-M-level-inv* S **and**
    conflicting: *cdcl*$_W$*-conflicting* S **and**
    no-tauto: $\forall\, s \in\#$ *learned-clss* S. $\neg$*tautology* s
  shows $\forall\, s \in\#$ *learned-clss* S'. $\neg$*tautology* s
  $\langle$*proof*$\rangle$

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

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

5.5 CDCL Strong Completeness

fun *mapi* :: ($'a \Rightarrow$ *nat* $\Rightarrow$ $'b$) $\Rightarrow$ *nat* $\Rightarrow$ $'a$ *list* $\Rightarrow$ $'b$ *list* **where**
mapi - - [] = [] |
mapi f n (x # *xs*) = *f x n* # *mapi f* ($n - 1$) *xs*

lemma *mark-not-in-set-mapi*[*simp*]: $L \notin$ *set* M $\Longrightarrow$ *Decided* L $k \notin$ *set* (*mapi Decided* i M)
  $\langle$*proof*$\rangle$

lemma *propagated-not-in-set-mapi*[*simp*]: $L \notin set\ M \Longrightarrow Propagated\ L\ k \notin set\ (mapi\ Decided\ i\ M)$
⟨*proof*⟩

lemma *image-set-mapi*:
$f\ `\ set\ (mapi\ g\ i\ M) = set\ (mapi\ (\lambda x\ i.\ f\ (g\ x\ i))\ i\ M)$
⟨*proof*⟩

lemma *mapi-map-convert*:
$\forall x\ i\ j.\ f\ x\ i = f\ x\ j \Longrightarrow mapi\ f\ i\ M = map\ (\lambda x.\ f\ x\ 0)\ M$
⟨*proof*⟩

lemma *defined-lit-mapi*: $defined\text{-}lit\ (mapi\ Decided\ i\ M)\ L \longleftrightarrow atm\text{-}of\ L \in atm\text{-}of\ `\ set\ M$
⟨*proof*⟩

lemma $cdcl_W$*-can-do-step*:
assumes
consistent-interp (*set M*) **and**
distinct M **and**
atm-of ‘ (*set M*) $\subseteq$ *atms-of-msu N*
shows $\exists S.\ rtranclp\ cdcl_W\ (init\text{-}state\ N)\ S$
$\wedge\ state\ S = (mapi\ Decided\ (length\ M)\ M,\ N,\ \{\#\},\ length\ M,\ None)$
⟨*proof*⟩

lemma $cdcl_W$*-strong-completeness*:
assumes
set M $\models s$ *set-mset N* **and**
consistent-interp (*set M*) **and**
distinct M **and**
atm-of ‘ (*set M*) $\subseteq$ *atms-of-msu N*
obtains *S* **where**
$state\ S = (mapi\ Decided\ (length\ M)\ M,\ N,\ \{\#\},\ length\ M,\ None)$ **and**
$rtranclp\ cdcl_W\ (init\text{-}state\ N)\ S$ **and**
$final\text{-}cdcl_W\text{-}state\ S$
⟨*proof*⟩

5.6 Higher level strategy

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

5.6.1 Definition

lemma *tranclp-conflict-iff*[*iff*]:
$full1\ conflict\ S\ S' \longleftrightarrow conflict\ S\ S'$
⟨*proof*⟩

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

lemma *rtranclp-*$cdcl_W$*-cp-rtranclp-*$cdcl_W$:
$cdcl_W\text{-}cp^{**}\ S\ T \Longrightarrow cdcl_W^{**}\ S\ T$
⟨*proof*⟩

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

$cdcl_W$-cp S T **and**
$S \sim S'$ **and**
$T \sim T'$
shows $cdcl_W$-cp S' T'
⟨*proof*⟩

lemma *tranclp-cdcl$_W$-cp-state-eq-compatible*:
assumes
$cdcl_W$-cp^{++} S T **and**
$S \sim S'$ **and**
$T \sim T'$
shows $cdcl_W$-cp^{++} S' T'
⟨*proof*⟩

lemma *option-full-cdcl$_W$-cp*:
conflicting $S \neq$ *None* $\Longrightarrow$ *full* $cdcl_W$-cp S S
⟨*proof*⟩

lemma *skip-unique*:
skip S $T \Longrightarrow$ *skip* S $T' \Longrightarrow T \sim T'$
⟨*proof*⟩

lemma *resolve-unique*:
resolve S $T \Longrightarrow$ *resolve* S $T' \Longrightarrow T \sim T'$
⟨*proof*⟩

lemma *cdcl$_W$-cp-no-more-clauses*:
assumes $cdcl_W$-cp S S'
shows *clauses* S = *clauses* S'
⟨*proof*⟩

lemma *tranclp-cdcl$_W$-cp-no-more-clauses*:
assumes $cdcl_W$-cp^{++} S S'
shows *clauses* S = *clauses* S'
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-cp-no-more-clauses*:
assumes $cdcl_W$-cp^{**} S S'
shows *clauses* S = *clauses* S'
⟨*proof*⟩

lemma *no-conflict-after-conflict*:
conflict S $T \Longrightarrow \neg$*conflict* T U
⟨*proof*⟩

lemma *no-propagate-after-conflict*:
conflict S $T \Longrightarrow \neg$*propagate* T U
⟨*proof*⟩

lemma *tranclp-cdcl$_W$-cp-propagate-with-conflict-or-not*:
assumes $cdcl_W$-cp^{++} S U
shows (*propagate*$^{++}$ S U $\wedge$ *conflicting* U = *None*)
$\vee$ ($\exists$ T D. *propagate*** S T $\wedge$ *conflict* T U $\wedge$ *conflicting* U = *Some* D)
⟨*proof*⟩

lemma $cdcl_W$*-cp-conflicting-not-empty*[*simp*]: *conflicting* S = *Some* $D \implies \neg cdcl_W$*-cp* S S'
⟨*proof*⟩

lemma *no-step-*$cdcl_W$*-cp-no-conflict-no-propagate*:
assumes *no-step* $cdcl_W$*-cp* S
shows *no-step conflict* S **and** *no-step propagate* S
⟨*proof*⟩

CDCL with the reasonable strategy: we fully propagate the conflict and propagate, then we apply any other possible rule $cdcl_W$*-o* S S' and re-apply conflict and propagate $cdcl_W$*-cp*$^{\downarrow}$ S' S''

inductive $cdcl_W$*-stgy* :: $'st \Rightarrow 'st \Rightarrow$ *bool* **for** S :: $'st$ **where**
conflict': *full1* $cdcl_W$*-cp* S $S' \Longrightarrow cdcl_W$*-stgy* S S' |
other': $cdcl_W$*-o* S $S' \implies$ *no-step* $cdcl_W$*-cp* $S \Longrightarrow$ *full* $cdcl_W$*-cp* S' $S'' \Longrightarrow cdcl_W$*-stgy* S S''

5.6.2 Invariants

These are the same invariants as before, but lifted

lemma $cdcl_W$*-cp-learned-clause-inv*:
assumes $cdcl_W$*-cp* S S'
shows *learned-clss* S = *learned-clss* S'
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-cp-learned-clause-inv*:
assumes $cdcl_W$*-cp*** S S'
shows *learned-clss* S = *learned-clss* S'
⟨*proof*⟩

lemma *tranclp-*$cdcl_W$*-cp-learned-clause-inv*:
assumes $cdcl_W$*-cp*$^{++}$ S S'
shows *learned-clss* S = *learned-clss* S'
⟨*proof*⟩

lemma $cdcl_W$*-cp-backtrack-lvl*:
assumes $cdcl_W$*-cp* S S'
shows *backtrack-lvl* S = *backtrack-lvl* S'
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-cp-backtrack-lvl*:
assumes $cdcl_W$*-cp*** S S'
shows *backtrack-lvl* S = *backtrack-lvl* S'
⟨*proof*⟩

lemma $cdcl_W$*-cp-consistent-inv*:
assumes $cdcl_W$*-cp* S S'
and $cdcl_W$*-M-level-inv* S
shows $cdcl_W$*-M-level-inv* S'
⟨*proof*⟩

lemma *full1-*$cdcl_W$*-cp-consistent-inv*:
assumes *full1* $cdcl_W$*-cp* S S'
and $cdcl_W$*-M-level-inv* S
shows $cdcl_W$*-M-level-inv* S'
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-cp-consistent-inv*:
assumes *rtranclp cdcl$_W$-cp S S'*
and *cdcl$_W$-M-level-inv S*
shows *cdcl$_W$-M-level-inv S'*
⟨*proof*⟩

lemma *cdcl$_W$-stgy-consistent-inv*:
assumes *cdcl$_W$-stgy S S'*
and *cdcl$_W$-M-level-inv S*
shows *cdcl$_W$-M-level-inv S'*
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-stgy-consistent-inv*:
assumes *cdcl$_W$-stgy** S S'*
and *cdcl$_W$-M-level-inv S*
shows *cdcl$_W$-M-level-inv S'*
⟨*proof*⟩

lemma *cdcl$_W$-cp-no-more-init-clss*:
assumes *cdcl$_W$-cp S S'*
shows *init-clss S = init-clss S'*
⟨*proof*⟩

lemma *tranclp-cdcl$_W$-cp-no-more-init-clss*:
assumes *cdcl$_W$-cp^{++} S S'*
shows *init-clss S = init-clss S'*
⟨*proof*⟩

lemma *cdcl$_W$-stgy-no-more-init-clss*:
assumes *cdcl$_W$-stgy S S'* **and** *cdcl$_W$-M-level-inv S*
shows *init-clss S = init-clss S'*
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-stgy-no-more-init-clss*:
assumes *cdcl$_W$-stgy** S S'* **and** *cdcl$_W$-M-level-inv S*
shows *init-clss S = init-clss S'*
⟨*proof*⟩

lemma *cdcl$_W$-cp-dropWhile-trail'*:
assumes *cdcl$_W$-cp S S'*
obtains *M* **where** *trail S' = M @ trail S* **and** (∀ *l* ∈ *set M*. ¬*is-decided l*)
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-cp-dropWhile-trail'*:
assumes *cdcl$_W$-cp** S S'*
obtains *M* :: ('*v*, *nat*, '*v clause*) *ann-literal list* **where**
trail S' = M @ trail S **and** ∀ *l* ∈ *set M*. ¬*is-decided l*
⟨*proof*⟩

lemma *cdcl$_W$-cp-dropWhile-trail*:
assumes *cdcl$_W$-cp S S'*
shows ∃ *M*. *trail S' = M @ trail S* ∧ (∀ *l* ∈ *set M*. ¬*is-decided l*)
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-cp-dropWhile-trail*:

assumes $cdcl_W\text{-}cp^{**}\ S\ S'$
shows $\exists M.\ trail\ S' = M\ @\ trail\ S \wedge (\forall l \in set\ M.\ \neg is\text{-}decided\ l)$
⟨proof⟩

This theorem can be seen a a termination theorem for $cdcl_W\text{-}cp$.

lemma *length-model-le-vars*:
assumes
no-strange-atm S **and**
no-d: *no-dup (trail S)* **and**
finite (atms-of-msu (init-clss S))
shows $length\ (trail\ S) \leq card\ (atms\text{-}of\text{-}msu\ (init\text{-}clss\ S))$
⟨proof⟩

lemma $cdcl_W$*-cp-decreasing-measure*:
assumes
$cdcl_W$: $cdcl_W\text{-}cp\ S\ T$ **and**
M-lev: $cdcl_W$*-M-level-inv S* **and**
alien: *no-strange-atm S*
shows $(\lambda S.\ card\ (atms\text{-}of\text{-}msu\ (init\text{-}clss\ S)) - length\ (trail\ S)$
$+ (if\ conflicting\ S = None\ then\ 1\ else\ 0))\ S$
$> (\lambda S.\ card\ (atms\text{-}of\text{-}msu\ (init\text{-}clss\ S)) - length\ (trail\ S)$
$+ (if\ conflicting\ S = None\ then\ 1\ else\ 0))\ T$
⟨proof⟩

lemma $cdcl_W$*-cp-wf*: $wf\ \{(b,a).\ (cdcl_W\text{-}M\text{-}level\text{-}inv\ a \wedge no\text{-}strange\text{-}atm\ a)$
$\wedge\ cdcl_W\text{-}cp\ a\ b\}$
⟨proof⟩

lemma *rtranclp-*$cdcl_W$*-all-struct-inv-*$cdcl_W$*-cp-iff-rtranclp-*$cdcl_W$*-cp*:
assumes
lev: $cdcl_W$*-M-level-inv S* **and**
alien: *no-strange-atm S*
shows $(\lambda a\ b.\ (cdcl_W\text{-}M\text{-}level\text{-}inv\ a \wedge no\text{-}strange\text{-}atm\ a) \wedge cdcl_W\text{-}cp\ a\ b)^{**}\ S\ T$
$\longleftrightarrow cdcl_W\text{-}cp^{**}\ S\ T$
(**is** $?I\ S\ T \longleftrightarrow\ ?C\ S\ T$)
⟨proof⟩

lemma $cdcl_W$*-cp-normalized-element*:
assumes
lev: $cdcl_W$*-M-level-inv S* **and**
no-strange-atm S
obtains *T* **where** *full* $cdcl_W$*-cp S T*
⟨proof⟩

lemma *in-atms-of-implies-atm-of-on-atms-of-ms*:
$C + \{\#L\#\} \in\#\ A \Longrightarrow x \in atms\text{-}of\ C \Longrightarrow x \in atms\text{-}of\text{-}msu\ A$
⟨proof⟩

lemma *propagate-no-stange-atm*:
assumes
propagate S S' **and**
no-strange-atm S
shows *no-strange-atm S'*
⟨proof⟩

lemma *always-exists-full-cdcl$_W$-cp-step*:
  assumes *no-strange-atm S*
  shows $\exists S''$. *full cdcl$_W$-cp S S''*
  ⟨*proof*⟩

5.6.3 Literal of highest level in conflicting clauses

One important property of the *local.cdcl$_W$* with strategy is that, whenever a conflict takes place, there is at least a literal of level k involved (except if we have derived the false clause). The reason is that we apply conflicts before a decision is taken.

abbreviation *no-clause-is-false* :: *'st* ⇒ *bool* **where**
no-clause-is-false ≡
  λS. (*conflicting S = None* ⟶ ($\forall D \in\#$ *clauses S*. ¬*trail S* ⊨*as CNot D*))

abbreviation *conflict-is-false-with-level* :: *'st* ⇒ *bool* **where**
conflict-is-false-with-level S ≡ $\forall D$. *conflicting S = Some D* ⟶ $D \neq \{\#\}$
  ⟶ ($\exists L \in\# D$. *get-level* (*trail S*) *L = backtrack-lvl S*)

lemma *not-conflict-not-any-negated-init-clss*:
  assumes ∀ *S'*. ¬*conflict S S'*
  shows *no-clause-is-false S*
  ⟨*proof*⟩

lemma *full-cdcl$_W$-cp-not-any-negated-init-clss*:
  assumes *full cdcl$_W$-cp S S'*
  shows *no-clause-is-false S'*
  ⟨*proof*⟩

lemma *full1-cdcl$_W$-cp-not-any-negated-init-clss*:
  assumes *full1 cdcl$_W$-cp S S'*
  shows *no-clause-is-false S'*
  ⟨*proof*⟩

lemma *cdcl$_W$-stgy-not-non-negated-init-clss*:
  assumes *cdcl$_W$-stgy S S'*
  shows *no-clause-is-false S'*
  ⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-stgy-not-non-negated-init-clss*:
  assumes *cdcl$_W$-stgy** S S'* **and** *no-clause-is-false S*
  shows *no-clause-is-false S'*
  ⟨*proof*⟩

lemma *cdcl$_W$-stgy-conflict-ex-lit-of-max-level*:
  assumes *cdcl$_W$-cp S S'*
  and *no-clause-is-false S*
  and *cdcl$_W$-M-level-inv S*
  shows *conflict-is-false-with-level S'*
  ⟨*proof*⟩

lemma *no-chained-conflict*:
  assumes *conflict S S'*
  and *conflict S' S''*
  shows *False*
  ⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-cp-propa-or-propa-confl*:
  assumes $cdcl_W\text{-}cp^{**}\ S\ U$
  shows $propagate^{**}\ S\ U \lor (\exists T.\ propagate^{**}\ S\ T\ \land conflict\ T\ U)$
  ⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-co-conflict-ex-lit-of-max-level*:
  assumes *full*: *full* $cdcl_W$*-cp* $S\ U$
  and *cls-f*: *no-clause-is-false* S
  and *conflict-is-false-with-level* S
  and *lev*: $cdcl_W$*-M-level-inv* S
  shows *conflict-is-false-with-level* U
⟨*proof*⟩

5.6.4 Literal of highest level in decided literals

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

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

lemma *propagate-no-more-propagation-to-do*:
  assumes *propagate*: *propagate* $S\ S'$
  and H: *no-more-propagation-to-do* S
  and M: $cdcl_W$*-M-level-inv* S
  shows *no-more-propagation-to-do* S'
  ⟨*proof*⟩

lemma *conflict-no-more-propagation-to-do*:
  assumes *conflict*: *conflict* $S\ S'$
  and H: *no-more-propagation-to-do* S
  and M: $cdcl_W$*-M-level-inv* S
  shows *no-more-propagation-to-do* S'
  ⟨*proof*⟩

lemma *cdcl$_W$-cp-no-more-propagation-to-do*:
  assumes *conflict*: $cdcl_W$*-cp* $S\ S'$
  and H: *no-more-propagation-to-do* S
  and M: $cdcl_W$*-M-level-inv* S
  shows *no-more-propagation-to-do* S'
  ⟨*proof*⟩

lemma *cdcl$_W$-then-exists-cdcl$_W$-stgy-step*:
  assumes
    o: $cdcl_W$*-o* $S\ S'$ **and**
    alien: *no-strange-atm* S **and**
    lev: $cdcl_W$*-M-level-inv* S
  shows $\exists S'.\ cdcl_W\text{-}stgy\ S\ S'$
⟨*proof*⟩

lemma *backtrack-no-decomp*:
assumes *S*: *state* $S = (M, N, U, k, Some\ (D + \{\#L\#\}))$
and *L*: $get\text{-}level\ M\ L = k$
and *D*: $get\text{-}maximum\text{-}level\ M\ D < k$
and *M-L*: $cdcl_W\text{-}M\text{-}level\text{-}inv\ S$
shows $\exists S'.\ cdcl_W\text{-}o\ S\ S'$
⟨*proof*⟩

lemma $cdcl_W$*-stgy-final-state-conclusive*:
assumes *termi*: $\forall S'.\ \neg cdcl_W\text{-}stgy\ S\ S'$
and *decomp*: *all-decomposition-implies-m* (*init-clss S*) (*get-all-decided-decomposition* (*trail S*))
and *learned*: $cdcl_W\text{-}learned\text{-}clause\ S$
and *level-inv*: $cdcl_W\text{-}M\text{-}level\text{-}inv\ S$
and *alien*: *no-strange-atm S*
and *no-dup*: $distinct\text{-}cdcl_W\text{-}state\ S$
and *confl*: $cdcl_W\text{-}conflicting\ S$
and *confl-k*: *conflict-is-false-with-level S*
shows $(conflicting\ S = Some\ \{\#\} \wedge unsatisfiable\ (set\text{-}mset\ (init\text{-}clss\ S)))$
$\vee\ (conflicting\ S = None \wedge trail\ S \models as\ set\text{-}mset\ (init\text{-}clss\ S))$
⟨*proof*⟩

lemma $cdcl_W\text{-}cp\text{-}tranclp\text{-}cdcl_W$:
$cdcl_W\text{-}cp\ S\ S' \Longrightarrow cdcl_W^{++}\ S\ S'$
⟨*proof*⟩

lemma $tranclp\text{-}cdcl_W\text{-}cp\text{-}tranclp\text{-}cdcl_W$:
$cdcl_W\text{-}cp^{++}\ S\ S' \Longrightarrow cdcl_W^{++}\ S\ S'$
⟨*proof*⟩

lemma $cdcl_W\text{-}stgy\text{-}tranclp\text{-}cdcl_W$:
$cdcl_W\text{-}stgy\ S\ S' \Longrightarrow cdcl_W^{++}\ S\ S'$
⟨*proof*⟩

lemma $tranclp\text{-}cdcl_W\text{-}stgy\text{-}tranclp\text{-}cdcl_W$:
$cdcl_W\text{-}stgy^{++}\ S\ S' \Longrightarrow cdcl_W^{++}\ S\ S'$
⟨*proof*⟩

lemma $rtranclp\text{-}cdcl_W\text{-}stgy\text{-}rtranclp\text{-}cdcl_W$:
$cdcl_W\text{-}stgy^{**}\ S\ S' \Longrightarrow cdcl_W^{**}\ S\ S'$
⟨*proof*⟩

lemma $cdcl_W$*-o-conflict-is-false-with-level-inv*:
assumes
$cdcl_W\text{-}o\ S\ S'$ **and**
lev: $cdcl_W\text{-}M\text{-}level\text{-}inv\ S$ **and**
confl-inv: *conflict-is-false-with-level S* **and**
n-d: $distinct\text{-}cdcl_W\text{-}state\ S$ **and**
conflicting: $cdcl_W\text{-}conflicting\ S$
shows *conflict-is-false-with-level* S'
⟨*proof*⟩

5.6.5 Strong completeness

lemma $cdcl_W$*-cp-propagate-confl*:
assumes $cdcl_W\text{-}cp\ S\ T$
shows $propagate^{**}\ S\ T \vee (\exists S'.\ propagate^{**}\ S\ S' \wedge conflict\ S'\ T)$

$\langle proof \rangle$

lemma *rtranclp-cdcl$_W$-cp-propagate-confl*:
assumes $cdcl_W\text{-}cp^{**}\ S\ T$
shows $propagate^{**}\ S\ T \lor (\exists S'.\ propagate^{**}\ S\ S' \land conflict\ S'\ T)$
$\langle proof \rangle$

lemma *cdcl$_W$-cp-propagate-completeness*:
assumes *MN*: $set\ M \models s\ set\text{-}mset\ N$ **and**
cons: $consistent\text{-}interp\ (set\ M)$ **and**
tot: $total\text{-}over\text{-}m\ (set\ M)\ (set\text{-}mset\ N)$ **and**
$lits\text{-}of\ (trail\ S) \subseteq set\ M$ **and**
$init\text{-}clss\ S = N$ **and**
$propagate^{**}\ S\ S'$ **and**
$learned\text{-}clss\ S = \{\#\}$
shows $length\ (trail\ S) \le length\ (trail\ S') \land lits\text{-}of\ (trail\ S') \subseteq set\ M$
$\langle proof \rangle$

lemma *completeness-is-a-full1-propagation*:
fixes $S :: 'st$ **and** $M :: 'v\ literal\ list$
assumes *MN*: $set\ M \models s\ set\text{-}mset\ N$
and *cons*: $consistent\text{-}interp\ (set\ M)$
and *tot*: $total\text{-}over\text{-}m\ (set\ M)\ (set\text{-}mset\ N)$
and *alien*: $no\text{-}strange\text{-}atm\ S$
and *learned*: $learned\text{-}clss\ S = \{\#\}$
and *clsS*[*simp*]: $init\text{-}clss\ S = N$
and *lits*: $lits\text{-}of\ (trail\ S) \subseteq set\ M$
shows $\exists S'.\ propagate^{**}\ S\ S' \land full\ cdcl_W\text{-}cp\ S\ S'$
$\langle proof \rangle$

See also $cdcl_W\text{-}cp^{**}\ ?S\ ?S' \Longrightarrow \exists M.\ trail\ ?S' = M\ @\ trail\ ?S \land (\forall l \in set\ M.\ \neg\ is\text{-}decided\ l)$

lemma *rtranclp-propagate-is-trail-append*:
$propagate^{**}\ S\ T \Longrightarrow \exists c.\ trail\ T = c\ @\ trail\ S$
$\langle proof \rangle$

lemma *rtranclp-propagate-is-update-trail*:
$propagate^{**}\ S\ T \Longrightarrow cdcl_W\text{-}M\text{-}level\text{-}inv\ S \Longrightarrow T \sim delete\text{-}trail\text{-}and\text{-}rebuild\ (trail\ T)\ S$
$\langle proof \rangle$

lemma *cdcl$_W$-stgy-strong-completeness-n*:
assumes
MN: $set\ M \models s\ set\text{-}mset\ N$ **and**
cons: $consistent\text{-}interp\ (set\ M)$ **and**
tot: $total\text{-}over\text{-}m\ (set\ M)\ (set\text{-}mset\ N)$ **and**
atm-incl: $atm\text{-}of\ `\ (set\ M) \subseteq atms\text{-}of\text{-}msu\ N$ **and**
distM: $distinct\ M$ **and**
length: $n \le length\ M$
shows
$\exists M'\ k\ S.\ length\ M' \ge n \land$
$lits\text{-}of\ M' \subseteq set\ M \land$
$no\text{-}dup\ M' \land$
$S \sim update\text{-}backtrack\text{-}lvl\ k\ (append\text{-}trail\ (rev\ M')\ (init\text{-}state\ N)) \land$
$cdcl_W\text{-}stgy^{**}\ (init\text{-}state\ N)\ S$
$\langle proof \rangle$

lemma $cdcl_W$*-stgy-strong-completeness*:
assumes *MN*: *set M* $\models$*s set-mset N*
and *cons*: *consistent-interp* (*set M*)
and *tot*: *total-over-m* (*set M*) (*set-mset N*)
and *atm-incl*: *atm-of* ` (*set M*) $\subseteq$ *atms-of-msu N*
and *distM*: *distinct M*
shows
$\exists M'$ *k S*.
lits-of M' = *set M* $\wedge$
S $\sim$ *update-backtrack-lvl k* (*append-trail* (*rev* M') (*init-state N*)) $\wedge$
$cdcl_W$*-stgy*** (*init-state N*) *S* $\wedge$
final-$cdcl_W$*-state S*
⟨*proof*⟩

5.6.6 No conflict with only variables of level less than backtrack level

This invariant is stronger than the previous argument in the sense that it is a property about all possible conflicts.

definition *no-smaller-confl* (*S*::'*st*) $\equiv$
($\forall$ *M K i* M' *D*. M' @ *Decided K i* # *M* = *trail S* $\longrightarrow$ *D* $\in$# *clauses S*
$\longrightarrow$ $\neg$*M* $\models$*as CNot D*)

lemma *no-smaller-confl-init-sate*[*simp*]:
no-smaller-confl (*init-state N*) ⟨*proof*⟩

lemma $cdcl_W$*-o-no-smaller-confl-inv*:
fixes *S* S' :: '*st*
assumes
$cdcl_W$*-o S* S' **and**
lev: $cdcl_W$*-M-level-inv S* **and**
max-lev: *conflict-is-false-with-level S* **and**
smaller: *no-smaller-confl S* **and**
no-f: *no-clause-is-false S*
shows *no-smaller-confl* S'
⟨*proof*⟩

lemma *conflict-no-smaller-confl-inv*:
assumes *conflict S* S'
and *no-smaller-confl S*
shows *no-smaller-confl* S'
⟨*proof*⟩

lemma *propagate-no-smaller-confl-inv*:
assumes *propagate*: *propagate S* S'
and *n-l*: *no-smaller-confl S*
shows *no-smaller-confl* S'
⟨*proof*⟩

lemma $cdcl_W$*-cp-no-smaller-confl-inv*:
assumes *propagate*: $cdcl_W$*-cp S* S'
and *n-l*: *no-smaller-confl S*
shows *no-smaller-confl* S'
⟨*proof*⟩

lemma *rtrancp-*$cdcl_W$*-cp-no-smaller-confl-inv*:

assumes *propagate*: $cdcl_W$*-cp*** S S'
and *n-l*: *no-smaller-confl* S
shows *no-smaller-confl* S'
⟨*proof*⟩

lemma *trancp-*$cdcl_W$*-cp-no-smaller-confl-inv*:
assumes *propagate*: $cdcl_W$*-cp*$^{++}$ S S'
and *n-l*: *no-smaller-confl* S
shows *no-smaller-confl* S'
⟨*proof*⟩

lemma *full-*$cdcl_W$*-cp-no-smaller-confl-inv*:
assumes *full* $cdcl_W$*-cp* S S'
and *n-l*: *no-smaller-confl* S
shows *no-smaller-confl* S'
⟨*proof*⟩

lemma *full1-*$cdcl_W$*-cp-no-smaller-confl-inv*:
assumes *full1* $cdcl_W$*-cp* S S'
and *n-l*: *no-smaller-confl* S
shows *no-smaller-confl* S'
⟨*proof*⟩

lemma $cdcl_W$*-stgy-no-smaller-confl-inv*:
assumes $cdcl_W$*-stgy* S S'
and *n-l*: *no-smaller-confl* S
and *conflict-is-false-with-level* S
and $cdcl_W$*-M-level-inv* S
shows *no-smaller-confl* S'
⟨*proof*⟩

lemma *conflict-conflict-is-no-clause-is-false-test*:
assumes *conflict* S S'
and $(\forall D \in\#$ *init-clss* S + *learned-clss* S. *trail* $S \models as$ *CNot* D
$\longrightarrow (\exists L.\ L \in\#\ D \wedge$ *get-level* (*trail* S) L = *backtrack-lvl* S))
shows $\forall D \in\#$ *init-clss* S' + *learned-clss* S'. *trail* $S' \models as$ *CNot* D
$\longrightarrow (\exists L.\ L \in\#\ D \wedge$ *get-level* (*trail* S') L = *backtrack-lvl* S')
⟨*proof*⟩

lemma *is-conflicting-exists-conflict*:
assumes $\neg(\forall D\in\#$*init-clss* S' + *learned-clss* S'. $\neg$ *trail* $S' \models as$ *CNot* D)
and *conflicting* S' = *None*
shows $\exists S''$. *conflict* S' S''
⟨*proof*⟩

lemma $cdcl_W$*-o-conflict-is-no-clause-is-false*:
fixes S S' :: *'st*
assumes
$cdcl_W$*-o* S S' **and**
lev: $cdcl_W$*-M-level-inv* S **and**
max-lev: *conflict-is-false-with-level* S **and**
no-f: *no-clause-is-false* S **and**
no-l: *no-smaller-confl* S
shows *no-clause-is-false* S'

$\vee$ (*conflicting* S' = *None*
$\longrightarrow$ ($\forall D \in\#$ *clauses* S'. *trail* $S' \models as$ *CNot* D
$\longrightarrow$ ($\exists L$. $L \in\#$ D $\wedge$ *get-level* (*trail* S') L = *backtrack-lvl* S')))
⟨*proof*⟩

lemma *full1-cdcl*$_W$*-cp-exists-conflict-decompose*:
assumes *confl*: $\exists D \in\#$*clauses* S. *trail* $S \models as$ *CNot* D
and *full*: *full cdcl*$_W$*-cp* S U
and *no-confl*: *conflicting* S = *None*
shows $\exists T$. *propagate*** S T $\wedge$ *conflict* T U
⟨*proof*⟩

lemma *full1-cdcl*$_W$*-cp-exists-conflict-full1-decompose*:
assumes *confl*: $\exists D \in\#$*clauses* S. *trail* $S \models as$ *CNot* D
and *full*: *full cdcl*$_W$*-cp* S U
and *no-confl*: *conflicting* S = *None*
shows $\exists T$ D. *propagate*** S T $\wedge$ *conflict* T U
$\wedge$ *trail* $T \models as$ *CNot* D $\wedge$ *conflicting* U = *Some* D $\wedge$ $D \in\#$ *clauses* S
⟨*proof*⟩

lemma *cdcl*$_W$*-stgy-no-smaller-confl*:
assumes *cdcl*$_W$*-stgy* S S'
and *n-l*: *no-smaller-confl* S
and *conflict-is-false-with-level* S
and *cdcl*$_W$*-M-level-inv* S
and *no-clause-is-false* S
and *distinct-cdcl*$_W$*-state* S
and *cdcl*$_W$*-conflicting* S
shows *no-smaller-confl* S'
⟨*proof*⟩

lemma *cdcl*$_W$*-stgy-ex-lit-of-max-level*:
assumes *cdcl*$_W$*-stgy* S S'
and *n-l*: *no-smaller-confl* S
and *conflict-is-false-with-level* S
and *cdcl*$_W$*-M-level-inv* S
and *no-clause-is-false* S
and *distinct-cdcl*$_W$*-state* S
and *cdcl*$_W$*-conflicting* S
shows *conflict-is-false-with-level* S'
⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-stgy-no-smaller-confl-inv*:
assumes
cdcl$_W$*-stgy*** S S' **and**
n-l: *no-smaller-confl* S **and**
cls-false: *conflict-is-false-with-level* S **and**
lev: *cdcl*$_W$*-M-level-inv* S **and**
no-f: *no-clause-is-false* S **and**
dist: *distinct-cdcl*$_W$*-state* S **and**
conflicting: *cdcl*$_W$*-conflicting* S **and**
decomp: *all-decomposition-implies-m* (*init-clss* S) (*get-all-decided-decomposition* (*trail* S)) **and**
learned: *cdcl*$_W$*-learned-clause* S **and**
alien: *no-strange-atm* S
shows *no-smaller-confl* S' $\wedge$ *conflict-is-false-with-level* S'

$\langle proof \rangle$

5.6.7 Final States are Conclusive

lemma *full-cdcl$_W$-stgy-final-state-conclusive-non-false*:
fixes S' :: *'st*
assumes *full*: *full cdcl$_W$-stgy* (*init-state N*) S'
and *no-d*: *distinct-mset-mset N*
and *no-empty*: $\forall D \in \# N.\ D \neq \{\#\}$
shows (*conflicting* S' = *Some* $\{\#\}$ $\wedge$ *unsatisfiable* (*set-mset* (*init-clss* S')))
$\vee$ (*conflicting* S' = *None* $\wedge$ *trail* S' $\models$*asm init-clss* S')
$\langle proof \rangle$

lemma *conflict-is-full1-cdcl$_W$-cp*:
assumes *cp*: *conflict S S'*
shows *full1 cdcl$_W$-cp S S'*
$\langle proof \rangle$

lemma *cdcl$_W$-cp-fst-empty-conflicting-false*:
assumes *cdcl$_W$-cp S S'*
and *trail S* = []
and *conflicting S* $\neq$ *None*
shows *False*
$\langle proof \rangle$

lemma *cdcl$_W$-o-fst-empty-conflicting-false*:
assumes *cdcl$_W$-o S S'*
and *trail S* = []
and *conflicting S* $\neq$ *None*
shows *False*
$\langle proof \rangle$

lemma *cdcl$_W$-stgy-fst-empty-conflicting-false*:
assumes *cdcl$_W$-stgy S S'*
and *trail S* = []
and *conflicting S* $\neq$ *None*
shows *False*
$\langle proof \rangle$
thm *cdcl$_W$-cp.induct*[*split-format*(*complete*)]

lemma *cdcl$_W$-cp-conflicting-is-false*:
cdcl$_W$-cp S S' $\Longrightarrow$ *conflicting S* = *Some* $\{\#\}$ $\Longrightarrow$ *False*
$\langle proof \rangle$

lemma *rtranclp-cdcl$_W$-cp-conflicting-is-false*:
cdcl$_W$-cp^{++} S S' $\Longrightarrow$ *conflicting S* = *Some* $\{\#\}$ $\Longrightarrow$ *False*
$\langle proof \rangle$

lemma *cdcl$_W$-o-conflicting-is-false*:
cdcl$_W$-o S S' $\Longrightarrow$ *conflicting S* = *Some* $\{\#\}$ $\Longrightarrow$ *False*
$\langle proof \rangle$

lemma *cdcl$_W$-stgy-conflicting-is-false*:
cdcl$_W$-stgy S S' $\Longrightarrow$ *conflicting S* = *Some* $\{\#\}$ $\Longrightarrow$ *False*

⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-stgy-conflicting-is-false*:
cdcl$_W$*-stgy*** *S S'* ⟹ *conflicting S* = *Some* {#} ⟹ *S'* = *S*
⟨*proof*⟩

lemma *full-cdcl*$_W$*-init-clss-with-false-normal-form*:
assumes
∀ *m*∈ *set M*. ¬*is-decided m* **and**
E = *Some D* **and**
state S = (*M*, *N*, *U*, *0*, *E*)
full cdcl$_W$*-stgy S S'* **and**
all-decomposition-implies-m (*init-clss S*) (*get-all-decided-decomposition* (*trail S*))
cdcl$_W$*-learned-clause S*
cdcl$_W$*-M-level-inv S*
no-strange-atm S
distinct-cdcl$_W$*-state S*
cdcl$_W$*-conflicting S*
shows ∃ *M''*. *state S'* = (*M''*, *N*, *U*, *0*, *Some* {#})
⟨*proof*⟩

lemma *full-cdcl*$_W$*-stgy-final-state-conclusive-is-one-false*:
fixes *S'* :: *'st*
assumes *full*: *full cdcl*$_W$*-stgy* (*init-state N*) *S'*
and *no-d*: *distinct-mset-mset N*
and *empty*: {#} ∈# *N*
shows *conflicting S'* = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss S'*))
⟨*proof*⟩

lemma *full-cdcl*$_W$*-stgy-final-state-conclusive*:
fixes *S'* :: *'st*
assumes *full*: *full cdcl*$_W$*-stgy* (*init-state N*) *S'* **and** *no-d*: *distinct-mset-mset N*
shows (*conflicting S'* = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss S'*)))
∨ (*conflicting S'* = *None* ∧ *trail S'* ⊨*asm init-clss S'*)
⟨*proof*⟩

lemma *full-cdcl*$_W$*-stgy-final-state-conclusive-from-init-state*:
fixes *S'* :: *'st*
assumes *full*: *full cdcl*$_W$*-stgy* (*init-state N*) *S'*
and *no-d*: *distinct-mset-mset N*
shows (*conflicting S'* = *Some* {#} ∧ *unsatisfiable* (*set-mset N*))
∨ (*conflicting S'* = *None* ∧ *trail S'* ⊨*asm N* ∧ *satisfiable* (*set-mset N*))
⟨*proof*⟩
end
end
theory *CDCL-W-Termination*
imports *CDCL-W*
begin

context *cdcl*$_W$
begin

5.7 Termination

The condition that no learned clause is a tautology is overkill (in the sense that the no-duplicate condition is enough), but we can reuse *simple-clss*.

The invariant contains all the structural invariants that holds,

definition cdcl_W*-all-struct-inv* **where**
cdcl_W*-all-struct-inv* S =
($\mathit{no\text{-}strange\text{-}atm}\ S \wedge \mathit{cdcl}_W\text{-}\mathit{M\text{-}level\text{-}inv}\ S$
$\wedge\ (\forall s \in\#\ \mathit{learned\text{-}clss}\ S.\ \neg \mathit{tautology}\ s)$
$\wedge\ \mathit{distinct\text{-}cdcl}_W\text{-}\mathit{state}\ S \wedge \mathit{cdcl}_W\text{-}\mathit{conflicting}\ S$
$\wedge\ \mathit{all\text{-}decomposition\text{-}implies\text{-}m}\ (\mathit{init\text{-}clss}\ S)\ (\mathit{get\text{-}all\text{-}decided\text{-}decomposition}\ (\mathit{trail}\ S))$
$\wedge\ \mathit{cdcl}_W\text{-}\mathit{learned\text{-}clause}\ S$)

lemma cdcl_W*-all-struct-inv-inv*:
assumes $\mathit{cdcl}_W\ S\ S'$ **and** $\mathit{cdcl}_W\text{-}\mathit{all\text{-}struct\text{-}inv}\ S$
shows $\mathit{cdcl}_W\text{-}\mathit{all\text{-}struct\text{-}inv}\ S'$
⟨*proof*⟩

lemma *rtranclp-*cdcl_W*-all-struct-inv-inv*:
assumes $\mathit{cdcl}_W^{**}\ S\ S'$ **and** $\mathit{cdcl}_W\text{-}\mathit{all\text{-}struct\text{-}inv}\ S$
shows $\mathit{cdcl}_W\text{-}\mathit{all\text{-}struct\text{-}inv}\ S'$
⟨*proof*⟩

lemma cdcl_W*-stgy-*cdcl_W*-all-struct-inv*:
$\mathit{cdcl}_W\text{-}\mathit{stgy}\ S\ T \Longrightarrow \mathit{cdcl}_W\text{-}\mathit{all\text{-}struct\text{-}inv}\ S \Longrightarrow \mathit{cdcl}_W\text{-}\mathit{all\text{-}struct\text{-}inv}\ T$
⟨*proof*⟩

lemma *rtranclp-*cdcl_W*-stgy-*cdcl_W*-all-struct-inv*:
$\mathit{cdcl}_W\text{-}\mathit{stgy}^{**}\ S\ T \Longrightarrow \mathit{cdcl}_W\text{-}\mathit{all\text{-}struct\text{-}inv}\ S \Longrightarrow \mathit{cdcl}_W\text{-}\mathit{all\text{-}struct\text{-}inv}\ T$
⟨*proof*⟩

5.8 No Relearning of a clause

lemma cdcl_W*-o-new-clause-learned-is-backtrack-step*:
assumes *learned*: $D \in\#\ \mathit{learned\text{-}clss}\ T$ **and**
new: $D \notin\#\ \mathit{learned\text{-}clss}\ S$ **and**
cdcl_W: $\mathit{cdcl}_W\text{-}o\ S\ T$ **and**
lev: $\mathit{cdcl}_W\text{-}\mathit{M\text{-}level\text{-}inv}\ S$
shows $\mathit{backtrack}\ S\ T \wedge \mathit{conflicting}\ S = \mathit{Some}\ D$
⟨*proof*⟩

lemma cdcl_W*-cp-new-clause-learned-has-backtrack-step*:
assumes *learned*: $D \in\#\ \mathit{learned\text{-}clss}\ T$ **and**
new: $D \notin\#\ \mathit{learned\text{-}clss}\ S$ **and**
cdcl_W: $\mathit{cdcl}_W\text{-}\mathit{stgy}\ S\ T$ **and**
lev: $\mathit{cdcl}_W\text{-}\mathit{M\text{-}level\text{-}inv}\ S$
shows $\exists S'.\ \mathit{backtrack}\ S\ S' \wedge \mathit{cdcl}_W\text{-}\mathit{stgy}^{**}\ S'\ T \wedge \mathit{conflicting}\ S = \mathit{Some}\ D$
⟨*proof*⟩

lemma *rtranclp-*cdcl_W*-cp-new-clause-learned-has-backtrack-step*:
assumes *learned*: $D \in\#\ \mathit{learned\text{-}clss}\ T$ **and**
new: $D \notin\#\ \mathit{learned\text{-}clss}\ S$ **and**
cdcl_W: $\mathit{cdcl}_W\text{-}\mathit{stgy}^{**}\ S\ T$ **and**
lev: $\mathit{cdcl}_W\text{-}\mathit{M\text{-}level\text{-}inv}\ S$
shows $\exists S'\ S''.\ \mathit{cdcl}_W\text{-}\mathit{stgy}^{**}\ S\ S' \wedge \mathit{backtrack}\ S'\ S'' \wedge \mathit{conflicting}\ S' = \mathit{Some}\ D\ \wedge$

$cdcl_W$-$stgy^{**}$ S'' T
⟨proof⟩

lemma *propagate-no-more-Decided-lit*:
assumes *propagate* S S'
shows *Decided* K i $\in$ *set* (*trail* S) $\longleftrightarrow$ *Decided* K i $\in$ *set* (*trail* S')
⟨proof⟩

lemma *conflict-no-more-Decided-lit*:
assumes *conflict* S S'
shows *Decided* K i $\in$ *set* (*trail* S) $\longleftrightarrow$ *Decided* K i $\in$ *set* (*trail* S')
⟨proof⟩

lemma $cdcl_W$*-cp-no-more-Decided-lit*:
assumes $cdcl_W$-*cp* S S'
shows *Decided* K i $\in$ *set* (*trail* S) $\longleftrightarrow$ *Decided* K i $\in$ *set* (*trail* S')
⟨proof⟩

lemma *rtranclp-*$cdcl_W$*-cp-no-more-Decided-lit*:
assumes $cdcl_W$-cp^{**} S S'
shows *Decided* K i $\in$ *set* (*trail* S) $\longleftrightarrow$ *Decided* K i $\in$ *set* (*trail* S')
⟨proof⟩

lemma $cdcl_W$*-o-no-more-Decided-lit*:
assumes $cdcl_W$-*o* S S' **and** $cdcl_W$-*M-level-inv* S **and** $\neg$*decide* S S'
shows *Decided* K i $\in$ *set* (*trail* S') $\longrightarrow$ *Decided* K i $\in$ *set* (*trail* S)
⟨proof⟩

lemma $cdcl_W$*-new-decided-at-beginning-is-decide*:
assumes $cdcl_W$-*stgy* S S' **and**
lev: $cdcl_W$-*M-level-inv* S **and**
trail S' = M' @ *Decided* L i # M **and**
trail S = M
shows $\exists$ T. *decide* S T $\wedge$ *no-step* $cdcl_W$-*cp* S
⟨proof⟩

lemma $cdcl_W$*-o-is-decide*:
assumes $cdcl_W$-*o* S' T **and** $cdcl_W$-*M-level-inv* S'
trail T = *drop* (*length* M_0) M' @ *Decided* L i # H @ M**and**
$\neg$ ($\exists$ M'. *trail* S' = M' @ *Decided* L i # H @ M)
shows *decide* S' T
⟨proof⟩

lemma *rtranclp-*$cdcl_W$*-new-decided-at-beginning-is-decide*:
assumes $cdcl_W$-$stgy^{**}$ R U **and**
trail U = M' @ *Decided* L i # H @ M **and**
trail R = M **and**
$cdcl_W$-*M-level-inv* R
shows
$\exists$ S T T'. $cdcl_W$-$stgy^{**}$ R S $\wedge$ *decide* S T $\wedge$ $cdcl_W$-$stgy^{**}$ T U $\wedge$ $cdcl_W$-$stgy^{**}$ S U $\wedge$
no-step $cdcl_W$-*cp* S $\wedge$ *trail* T = *Decided* L i # H @ M $\wedge$ *trail* S = H @ M $\wedge$ $cdcl_W$-*stgy* S T' $\wedge$
$cdcl_W$-$stgy^{**}$ T' U
⟨proof⟩

lemma *rtranclp-*$cdcl_W$*-new-decided-at-beginning-is-decide*$'$:

assumes $cdcl_W$-*stgy*** *R U* **and**
trail U = *M'* @ *Decided L i* # *H* @ *M* **and**
trail R = *M* **and**
$cdcl_W$-*M-level-inv R*
shows ∃ *y y'*. $cdcl_W$-*stgy*** *R y* ∧ $cdcl_W$-*stgy y y'* ∧ ¬ (∃ *c*. *trail y* = *c* @ *Decided L i* # *H* @ *M*)
∧ (λ*a b*. $cdcl_W$-*stgy a b* ∧ (∃ *c*. *trail a* = *c* @ *Decided L i* # *H* @ *M*))** *y' U*
⟨*proof*⟩

lemma *beginning-not-decided-invert*:
assumes *A*: *M* @ *A* = *M'* @ *Decided K i* # *H* **and**
nm: ∀ *m*∈*set M*. ¬*is-decided m*
shows ∃ *M*. *A* = *M* @ *Decided K i* # *H*
⟨*proof*⟩

lemma $cdcl_W$-*stgy-trail-has-new-decided-is-decide-step*:
assumes $cdcl_W$-*stgy S T*
¬ (∃ *c*. *trail S* = *c* @ *Decided L i* # *H* @ *M*) **and**
(λ*a b*. $cdcl_W$-*stgy a b* ∧ (∃ *c*. *trail a* = *c* @ *Decided L i* # *H* @ *M*))** *T U* **and**
∃ *M'*. *trail U* = *M'* @ *Decided L i* # *H* @ *M* **and**
lev: $cdcl_W$-*M-level-inv S*
shows ∃ *S'*. *decide S S'* ∧ *full* $cdcl_W$-*cp S' T* ∧ *no-step* $cdcl_W$-*cp S*
⟨*proof*⟩

lemma *rtranclp*-$cdcl_W$-*stgy-with-trail-end-has-trail-end*:
assumes (λ*a b*. $cdcl_W$-*stgy a b* ∧ (∃ *c*. *trail a* = *c* @ *Decided L i* # *H* @ *M*))** *T U* **and**
∃ *M'*. *trail U* = *M'* @ *Decided L i* # *H* @ *M*
shows ∃ *M'*. *trail T* = *M'* @ *Decided L i* # *H* @ *M*
⟨*proof*⟩

lemma $cdcl_W$-*o-cannot-learn*:
assumes
$cdcl_W$-*o y z* **and**
lev: $cdcl_W$-*M-level-inv y* **and**
trM: *trail y* = *c* @ *Decided Kh i* # *H* **and**
DL: *D* + {#*L*#} ∉# *learned-clss y* **and**
DH: *atms-of D* ⊆ *atm-of* '*lits-of H* **and**
LH: *atm-of L* ∉ *atm-of* '*lits-of H* **and**
learned: ∀ *T*. *conflicting y* = *Some T* ⟶ *trail y* ⊨*as CNot T* **and**
z: *trail z* = *c'* @ *Decided Kh i* # *H*
shows *D* + {#*L*#} ∉# *learned-clss z*
⟨*proof*⟩

lemma $cdcl_W$-*stgy-with-trail-end-has-not-been-learned*:
assumes $cdcl_W$-*stgy y z* **and**
$cdcl_W$-*M-level-inv y* **and**
trail y = *c* @ *Decided Kh i* # *H* **and**
D + {#*L*#} ∉# *learned-clss y* **and**
DH: *atms-of D* ⊆ *atm-of* '*lits-of H* **and**
LH: *atm-of L* ∉ *atm-of* '*lits-of H* **and**
∀ *T*. *conflicting y* = *Some T* ⟶ *trail y* ⊨*as CNot T* **and**
trail z = *c'* @ *Decided Kh i* # *H*
shows *D* + {#*L*#} ∉# *learned-clss z*
⟨*proof*⟩

lemma *rtranclp*-$cdcl_W$-*stgy-with-trail-end-has-not-been-learned*:

assumes $(\lambda a\ b.\ cdcl_W\text{-}stgy\ a\ b \wedge (\exists c.\ trail\ a = c\ @\ Decided\ K\ i\ \#\ H\ @\ []))^{**}\ S\ z$ **and**
$cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$ **and**
$trail\ S = c\ @\ Decided\ K\ i\ \#\ H$ **and**
$D + \{\#L\#\} \notin\#\ learned\text{-}clss\ S$ **and**
DH: $atms\text{-}of\ D \subseteq atm\text{-}of\ \text{`}lits\text{-}of\ H$ **and**
LH: $atm\text{-}of\ L \notin atm\text{-}of\ \text{`}lits\text{-}of\ H$ **and**
$\exists c'.\ trail\ z = c'\ @\ Decided\ K\ i\ \#\ H$
shows $D + \{\#L\#\} \notin\#\ learned\text{-}clss\ z$
⟨*proof*⟩

lemma $cdcl_W\text{-}stgy\text{-}new\text{-}learned\text{-}clause$:
assumes $cdcl_W\text{-}stgy\ S\ T$ **and**
lev: $cdcl_W\text{-}M\text{-}level\text{-}inv\ S$ **and**
$E \notin\#\ learned\text{-}clss\ S$ **and**
$E \in\#\ learned\text{-}clss\ T$
shows $\exists S'.\ backtrack\ S\ S' \wedge conflicting\ S = Some\ E \wedge full\ cdcl_W\text{-}cp\ S'\ T$
⟨*proof*⟩

lemma $cdcl_W\text{-}stgy\text{-}no\text{-}relearned\text{-}clause$:
assumes
$invR$: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ R$ **and**
st': $cdcl_W\text{-}stgy^{**}\ R\ S$ **and**
bt: $backtrack\ S\ T$ **and**
$confl$: $conflicting\ S = Some\ E$ **and**
$already\text{-}learned$: $E \in\#\ clauses\ S$ **and**
R: $trail\ R = []$
shows $False$
⟨*proof*⟩

lemma $rtranclp\text{-}cdcl_W\text{-}stgy\text{-}distinct\text{-}mset\text{-}clauses$:
assumes
$invR$: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ R$ **and**
st: $cdcl_W\text{-}stgy^{**}\ R\ S$ **and**
$dist$: $distinct\text{-}mset\ (clauses\ R)$ **and**
R: $trail\ R = []$
shows $distinct\text{-}mset\ (clauses\ S)$
⟨*proof*⟩

lemma $cdcl_W\text{-}stgy\text{-}distinct\text{-}mset\text{-}clauses$:
assumes
st: $cdcl_W\text{-}stgy^{**}\ (init\text{-}state\ N)\ S$ **and**
$no\text{-}duplicate\text{-}clause$: $distinct\text{-}mset\ N$ **and**
$no\text{-}duplicate\text{-}in\text{-}clause$: $distinct\text{-}mset\text{-}mset\ N$
shows $distinct\text{-}mset\ (clauses\ S)$
⟨*proof*⟩

5.9 Decrease of a measure

fun $cdcl_W\text{-}measure$ **where**
$cdcl_W\text{-}measure\ S =$
$[(3::nat)\ \hat{}\ (card\ (atms\text{-}of\text{-}msu\ (init\text{-}clss\ S))) - card\ (set\text{-}mset\ (learned\text{-}clss\ S)),$
$if\ conflicting\ S = None\ then\ 1\ else\ 0,$
$if\ conflicting\ S = None\ then\ card\ (atms\text{-}of\text{-}msu\ (init\text{-}clss\ S)) - length\ (trail\ S)$
$else\ length\ (trail\ S)$
$]$

lemma *length-model-le-vars-all-inv*:
assumes $cdcl_W$*-all-struct-inv S*
shows *length* (*trail S*) $\leq$ *card* (*atms-of-msu* (*init-clss S*))
⟨*proof*⟩
end

context $cdcl_W$
begin

lemma *learned-clss-less-upper-bound*:
fixes *S* :: *'st*
assumes
distinct-$cdcl_W$*-state S* **and**
$\forall$ *s* ∈# *learned-clss S*. ¬*tautology s*
shows *card*(*set-mset* (*learned-clss S*)) $\leq$ *3* ^ *card* (*atms-of-msu* (*learned-clss S*))
⟨*proof*⟩

lemma *lexn3*[*intro*!, *simp*]:
$a < a' \lor (a = a' \land b < b') \lor (a = a' \land b = b' \land c < c')$
$\Longrightarrow$ ([*a*::*nat*, *b*, *c*], [*a'*, *b'*, *c'*]) ∈ *lexn* {(*x*, *y*). *x* < *y*} *3*
⟨*proof*⟩

lemma $cdcl_W$*-measure-decreasing*:
fixes *S* :: *'st*
assumes
$cdcl_W$ *S S'* **and**
no-restart:
¬(*learned-clss S* ⊆# *learned-clss S'* ∧ [] = *trail S'* ∧ *conflicting S'* = *None*)
and
learned-clss S ⊆# *learned-clss S'* **and**
no-relearn: $\bigwedge$*S'*. *backtrack S S'* $\Longrightarrow$ ∀ *T*. *conflicting S* = *Some T* ⟶ *T* ∉# *learned-clss S*
and
alien: *no-strange-atm S* **and**
M-level: $cdcl_W$*-M-level-inv S* **and**
no-taut: ∀ *s* ∈# *learned-clss S*. ¬*tautology s* **and**
no-dup: *distinct-*$cdcl_W$*-state S* **and**
confl: $cdcl_W$*-conflicting S*
shows ($cdcl_W$*-measure S'*, $cdcl_W$*-measure S*) ∈ *lexn* {(*a*, *b*). *a* < *b*} *3*
⟨*proof*⟩

lemma *propagate-measure-decreasing*:
fixes *S* :: *'st*
assumes *propagate S S'* **and** $cdcl_W$*-all-struct-inv S*
shows ($cdcl_W$*-measure S'*, $cdcl_W$*-measure S*) ∈ *lexn* {(*a*, *b*). *a* < *b*} *3*
⟨*proof*⟩

lemma *conflict-measure-decreasing*:
fixes *S* :: *'st*
assumes *conflict S S'* **and** $cdcl_W$*-all-struct-inv S*
shows ($cdcl_W$*-measure S'*, $cdcl_W$*-measure S*) ∈ *lexn* {(*a*, *b*). *a* < *b*} *3*
⟨*proof*⟩

lemma *decide-measure-decreasing*:
fixes *S* :: *'st*
assumes *decide S S'* **and** $cdcl_W$*-all-struct-inv S*

shows ($cdcl_W$*-measure* S', $cdcl_W$*-measure* S) $\in$ *lexn* $\{(a,\ b).\ a < b\}$ *3*
⟨*proof*⟩

lemma *trans-le*:
trans $\{(a,\ (b::nat)).\ a < b\}$
⟨*proof*⟩

lemma $cdcl_W$*-cp-measure-decreasing*:
fixes S :: *'st*
assumes $cdcl_W$*-cp* S S' **and** $cdcl_W$*-all-struct-inv* S
shows ($cdcl_W$*-measure* S', $cdcl_W$*-measure* S) $\in$ *lexn* $\{(a,\ b).\ a < b\}$ *3*
⟨*proof*⟩

lemma *tranclp-*$cdcl_W$*-cp-measure-decreasing*:
fixes S :: *'st*
assumes $cdcl_W$*-cp*$^{++}$ S S' **and** $cdcl_W$*-all-struct-inv* S
shows ($cdcl_W$*-measure* S', $cdcl_W$*-measure* S) $\in$ *lexn* $\{(a,\ b).\ a < b\}$ *3*
⟨*proof*⟩

lemma $cdcl_W$*-stgy-step-decreasing*:
fixes R S T :: *'st*
assumes $cdcl_W$*-stgy* S T **and**
$cdcl_W$*-stgy*** R S
trail R = [] **and**
$cdcl_W$*-all-struct-inv* R
shows ($cdcl_W$*-measure* T, $cdcl_W$*-measure* S) $\in$ *lexn* $\{(a,\ b).\ a < b\}$ *3*
⟨*proof*⟩

lemma *tranclp-*$cdcl_W$*-stgy-decreasing*:
fixes R S T :: *'st*
assumes $cdcl_W$*-stgy*$^{++}$ R S
trail R = [] **and**
$cdcl_W$*-all-struct-inv* R
shows ($cdcl_W$*-measure* S, $cdcl_W$*-measure* R) $\in$ *lexn* $\{(a,\ b).\ a < b\}$ *3*
⟨*proof*⟩

lemma *tranclp-*$cdcl_W$*-stgy-S0-decreasing*:
fixes R S T :: *'st*
assumes *pl*: $cdcl_W$*-stgy*$^{++}$ (*init-state* N) S **and**
no-dup: *distinct-mset-mset* N
shows ($cdcl_W$*-measure* S, $cdcl_W$*-measure* (*init-state* N)) $\in$ *lexn* $\{(a,\ b).\ a < b\}$ *3*
⟨*proof*⟩

lemma *wf-tranclp-*$cdcl_W$*-stgy*:
wf {(S::*'st*, *init-state* N)| S N. *distinct-mset-mset* N $\wedge$ $cdcl_W$*-stgy*$^{++}$ (*init-state* N) S}
⟨*proof*⟩
end

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

6 Simple Implementation of the DPLL and CDCL

6.1 Common Rules

6.1.1 Propagation

The following theorem holds:

lemma *lits-of-unfold*[*iff*]:
$(\forall\, c \in \mathit{set}\ C.\ -c \in \mathit{lits\text{-}of}\ Ms) \longleftrightarrow Ms \models\!as\ \mathit{CNot}\ (\mathit{mset}\ C)$
⟨*proof*⟩

The right-hand version is written at a high-level, but only the left-hand side is executable.

definition *is-unit-clause* :: *'a literal list* ⇒ (*'a*, *'b*, *'c*) *ann-literal list* ⇒ *'a literal option*
where
is-unit-clause l M =
(*case List.filter* (λ*a*. *atm-of a* ∉ *atm-of* ‘ *lits-of M*) *l of*
a # [] ⇒ *if M* ⊨*as CNot* (*mset l* − {#*a*#}) *then Some a else None*
| - ⇒ *None*)

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

lemma *is-unit-clause-is-unit-clause-code*[*code*]:
is-unit-clause l M = *is-unit-clause-code l M*
⟨*proof*⟩

lemma *is-unit-clause-some-undef*:
assumes *is-unit-clause l M* = *Some a*
shows *undefined-lit M a*
⟨*proof*⟩

lemma *is-unit-clause-some-CNot*: *is-unit-clause l M* = *Some a* ⟹ *M* ⊨*as CNot* (*mset l* − {#*a*#})
⟨*proof*⟩

lemma *is-unit-clause-some-in*: *is-unit-clause l M* = *Some a* ⟹ *a* ∈ *set l*
⟨*proof*⟩

lemma *is-unit-clause-nil*[*simp*]: *is-unit-clause* [] *M* = *None*
⟨*proof*⟩

6.1.2 Unit propagation for all clauses

Finding the first clause to propagate

fun *find-first-unit-clause* :: *'a literal list list* ⇒ (*'a*, *'b*, *'c*) *ann-literal list*
⇒ (*'a literal* × *'a literal list*) *option* **where**
find-first-unit-clause (*a* # *l*) *M* =
(*case is-unit-clause a M of*
None ⇒ *find-first-unit-clause l M*
| *Some L* ⇒ *Some* (*L*, *a*)) |
find-first-unit-clause [] - = *None*

lemma *find-first-unit-clause-some*:
  find-first-unit-clause l M = Some (a, c)
  $\Longrightarrow$ *c* $\in$ *set l* $\wedge$ *M* $\models$*as CNot (mset c* $-$ $\{\#a\#\}$) $\wedge$ *undefined-lit M a* $\wedge$ *a* $\in$ *set c*
  ⟨*proof*⟩

lemma *propagate-is-unit-clause-not-None*:
  assumes *dist*: *distinct c* **and**
  M: *M* $\models$*as CNot (mset c* $-$ $\{\#a\#\}$) **and**
  undef: *undefined-lit M a* **and**
  ac: *a* $\in$ *set c*
  shows *is-unit-clause c M* $\neq$ *None*
⟨*proof*⟩

lemma *find-first-unit-clause-none*:
  distinct c $\Longrightarrow$ *c* $\in$ *set l* $\Longrightarrow$ *M* $\models$*as CNot (mset c* $-$ $\{\#a\#\}$) $\Longrightarrow$ *undefined-lit M a* $\Longrightarrow$ *a* $\in$ *set c*
  $\Longrightarrow$ *find-first-unit-clause l M* $\neq$ *None*
  ⟨*proof*⟩

6.1.3 Decide

fun *find-first-unused-var* :: *'a literal list list* $\Rightarrow$ *'a literal set* $\Rightarrow$ *'a literal option* **where**
find-first-unused-var (a # l) M =
  (case List.find (λ*lit. lit* $\notin$ *M* $\wedge$ $-$*lit* $\notin$ *M) a of*
    None $\Rightarrow$ *find-first-unused-var l M*
  | *Some a* $\Rightarrow$ *Some a)* |
find-first-unused-var [] - = None

lemma *find-none[iff]*:
  List.find (λ*lit. lit* $\notin$ *M* $\wedge$ $-$*lit* $\notin$ *M) a = None* $\longleftrightarrow$ *atm-of ‘ set a* $\subseteq$ *atm-of ‘ M*
  ⟨*proof*⟩

lemma *find-some*: *List.find (*λ*lit. lit* $\notin$ *M* $\wedge$ $-$*lit* $\notin$ *M) a = Some b* $\Longrightarrow$ *b* $\in$ *set a* $\wedge$ *b* $\notin$ *M* $\wedge$ $-$*b* $\notin$ *M*
  ⟨*proof*⟩

lemma *find-first-unused-var-None[iff]*:
  find-first-unused-var l M = None $\longleftrightarrow$ ($\forall$ *a* $\in$ *set l. atm-of ‘ set a* $\subseteq$ *atm-of ‘ M*)
  ⟨*proof*⟩

lemma *find-first-unused-var-Some-not-all-incl*:
  assumes *find-first-unused-var l M = Some c*
  shows $\neg$($\forall$ *a* $\in$ *set l. atm-of ‘ set a* $\subseteq$ *atm-of ‘ M*)
⟨*proof*⟩

lemma *find-first-unused-var-Some*:
  find-first-unused-var l M = Some a $\Longrightarrow$ ($\exists$ *m* $\in$ *set l. a* $\in$ *set m* $\wedge$ *a* $\notin$ *M* $\wedge$ $-$*a* $\notin$ *M*)
  ⟨*proof*⟩

lemma *find-first-unused-var-undefined*:
  find-first-unused-var l (lits-of Ms) = Some a $\Longrightarrow$ *undefined-lit Ms a*
  ⟨*proof*⟩

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

6.2 Simple Implementation of DPLL

6.2.1 Combining the propagate and decide: a DPLL step

definition *DPLL-step* :: *int* $dpll_W$*-ann-literals* × *int literal list list*
 ⇒ *int* $dpll_W$*-ann-literals* × *int literal list list* **where**
DPLL-step = (λ(*Ms*, *N*).
 (*case find-first-unit-clause N Ms of*
  Some (*L*, -) ⇒ (*Propagated L* () # *Ms*, *N*)
 | - ⇒
  if ∃ *C* ∈ *set N*. (∀ *c* ∈ *set C*. −*c* ∈ *lits-of Ms*)
  then
   (*case backtrack-split Ms of*
    (-, *L* # *M*) ⇒ (*Propagated* (− (*lit-of L*)) () # *M*, *N*)
   | (-, -) ⇒ (*Ms*, *N*)
)
  else
  (*case find-first-unused-var N* (*lits-of Ms*) *of*
    Some a ⇒ (*Decided a* () # *Ms*, *N*)
   | *None* ⇒ (*Ms*, *N*))))

Example of propagation:

value *DPLL-step* ([*Decided* (*Neg 1*) ()], [[*Pos* (*1*::*int*), *Neg 2*]])

We define the conversion function between the states as defined in *Prop-DPLL* (with multisets) and here (with lists).

abbreviation *toS* ≡ λ(*Ms*::(*int*, *unit*, *unit*) *ann-literal list*)
      (*N*:: *int literal list list*). (*Ms*, *mset* (*map mset N*))
abbreviation *toS′* ≡ λ(*Ms*::(*int*, *unit*, *unit*) *ann-literal list*,
      N:: *int literal list list*). (*Ms*, *mset* (*map mset N*))

Proof of correctness of *DPLL-step*

lemma *DPLL-step-is-a-*$dpll_W$*-step*:
 assumes *step*: (*Ms′*, *N′*) = *DPLL-step* (*Ms*, *N*)
 and *neq*: (*Ms*, *N*) ≠ (*Ms′*, *N′*)
 shows $dpll_W$ (*toS Ms N*) (*toS Ms′ N′*)
⟨*proof*⟩

lemma *DPLL-step-stuck-final-state*:
 assumes *step*: (*Ms*, *N*) = *DPLL-step* (*Ms*, *N*)
 shows *conclusive-*$dpll_W$*-state* (*toS Ms N*)
⟨*proof*⟩

6.2.2 Adding invariants

Invariant tested in the function **function** *DPLL-ci* :: *int* $dpll_W$*-ann-literals* ⇒ *int literal list list*
 ⇒ *int* $dpll_W$*-ann-literals* × *int literal list list* **where**
DPLL-ci Ms N =
 (*if* ¬$dpll_W$*-all-inv* (*Ms*, *mset* (*map mset N*))
 then (*Ms*, *N*)
 else
  let (*Ms′*, *N′*) = *DPLL-step* (*Ms*, *N*) *in*
  if (*Ms′*, *N′*) = (*Ms*, *N*) *then* (*Ms*, *N*) *else DPLL-ci Ms′ N*)
 ⟨*proof*⟩
termination

⟨*proof*⟩

No invariant tested **function** (*domintros*) *DPLL-part*:: *int* $dpll_W$*-ann-literals* ⇒ *int literal list list* ⇒
int $dpll_W$*-ann-literals* × *int literal list list* **where**
DPLL-part Ms N =
(*let* (*Ms'*, *N'*) = *DPLL-step* (*Ms*, *N*) *in*
if (*Ms'*, *N'*) = (*Ms*, *N*) *then* (*Ms*, *N*) *else DPLL-part Ms' N*)
⟨*proof*⟩

lemma *snd-DPLL-step*[*simp*]:
snd (*DPLL-step* (*Ms*, *N*)) = *N*
⟨*proof*⟩

lemma $dpll_W$*-all-inv-implieS-2-eq3-and-dom*:
assumes $dpll_W$*-all-inv* (*Ms*, *mset* (*map mset N*))
shows *DPLL-ci Ms N* = *DPLL-part Ms N* ∧ *DPLL-part-dom* (*Ms*, *N*)
⟨*proof*⟩

lemma *DPLL-ci-*$dpll_W$*-rtranclp*:
assumes *DPLL-ci Ms N* = (*Ms'*, *N'*)
shows $dpll_W{}^{**}$ (*toS Ms N*) (*toS Ms' N*)
⟨*proof*⟩

lemma $dpll_W$*-all-inv-*$dpll_W$*-tranclp-irrefl*:
assumes $dpll_W$*-all-inv* (*Ms*, *N*)
and $dpll_W{}^{++}$ (*Ms*, *N*) (*Ms*, *N*)
shows *False*
⟨*proof*⟩

lemma *DPLL-ci-final-state*:
assumes *step*: *DPLL-ci Ms N* = (*Ms*, *N*)
and *inv*: $dpll_W$*-all-inv* (*toS Ms N*)
shows *conclusive-*$dpll_W$*-state* (*toS Ms N*)
⟨*proof*⟩

lemma *DPLL-step-obtains*:
obtains *Ms'* **where** (*Ms'*, *N*) = *DPLL-step* (*Ms*, *N*)
⟨*proof*⟩

lemma *DPLL-ci-obtains*:
obtains *Ms'* **where** (*Ms'*, *N*) = *DPLL-ci Ms N*
⟨*proof*⟩

lemma *DPLL-ci-no-more-step*:
assumes *step*: *DPLL-ci Ms N* = (*Ms'*, *N'*)
shows *DPLL-ci Ms' N'* = (*Ms'*, *N'*)
⟨*proof*⟩

lemma *DPLL-part-*$dpll_W$*-all-inv-final*:
fixes *M Ms'*:: (*int*, *unit*, *unit*) *ann-literal list* **and**
N :: *int literal list list*
assumes *inv*: $dpll_W$*-all-inv* (*Ms*, *mset* (*map mset N*))

and *MsN*: *DPLL-part Ms N* = (*Ms'*, *N*)
shows *conclusive-dpll*$_W$*-state* (*toS Ms' N*) ∧ *dpll*$_W$** (*toS Ms N*) (*toS Ms' N*)
⟨*proof*⟩

Embedding the invariant into the type

Defining the type **typedef** *dpll*$_W$*-state* =
{(*M*::(*int*, *unit*, *unit*) *ann-literal list*, *N*::*int literal list list*).
dpll$_W$*-all-inv* (*toS M N*)}
morphisms *rough-state-of state-of*
⟨*proof*⟩

lemma
DPLL-part-dom ([], *N*)
⟨*proof*⟩

Some type classes **instantiation** *dpll*$_W$*-state* :: *equal*
begin
definition *equal-dpll*$_W$*-state* :: *dpll*$_W$*-state* ⇒ *dpll*$_W$*-state* ⇒ *bool* **where**
equal-dpll$_W$*-state S S'* = (*rough-state-of S* = *rough-state-of S'*)
instance
⟨*proof*⟩
end

DPLL **definition** *DPLL-step'* :: *dpll*$_W$*-state* ⇒ *dpll*$_W$*-state* **where**
DPLL-step' S = *state-of* (*DPLL-step* (*rough-state-of S*))

declare *rough-state-of-inverse*[*simp*]

lemma *DPLL-step-dpll*$_W$*-conc-inv*:
DPLL-step (*rough-state-of S*) ∈ {(*M*, *N*). *dpll*$_W$*-all-inv* (*toS M N*)}
⟨*proof*⟩

lemma *rough-state-of-DPLL-step'-DPLL-step*[*simp*]:
rough-state-of (*DPLL-step' S*) = *DPLL-step* (*rough-state-of S*)
⟨*proof*⟩

function *DPLL-tot*:: *dpll*$_W$*-state* ⇒ *dpll*$_W$*-state* **where**
DPLL-tot S =
(*let S'* = *DPLL-step' S in*
if S' = *S then S else DPLL-tot S'*)
⟨*proof*⟩
termination
⟨*proof*⟩

lemma [*code*]:
DPLL-tot S =
(*let S'* = *DPLL-step' S in*
if S' = *S then S else DPLL-tot S'*) ⟨*proof*⟩

lemma *DPLL-tot-DPLL-step-DPLL-tot*[*simp*]: *DPLL-tot* (*DPLL-step' S*) = *DPLL-tot S*
⟨*proof*⟩

lemma *DOPLL-step'-DPLL-tot*[*simp*]:
DPLL-step' (*DPLL-tot S*) = *DPLL-tot S*

⟨*proof*⟩

lemma *DPLL-tot-final-state*:
assumes *DPLL-tot* $S = S$
shows *conclusive-dpll*$_W$*-state* (*toS'* (*rough-state-of* S))
⟨*proof*⟩

lemma *DPLL-tot-star*:
assumes *rough-state-of* (*DPLL-tot* S) $= S'$
shows *dpll*$_W$** (*toS'* (*rough-state-of* S)) (*toS'* S')
⟨*proof*⟩

lemma *rough-state-of-rough-state-of-nil*[*simp*]:
rough-state-of (*state-of* ([], N)) = ([], N)
⟨*proof*⟩

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*⟩

6.2.3 Code export

A conversion to *DPLL-W-Implementation.dpll*$_W$*-state* **definition** *Con* :: (*int*, *unit*, *unit*) *ann-literal list* × *int literal list list*
$\Rightarrow$ *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
⟨*proof*⟩

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*))
⟨*proof*⟩

A slightly different version of *DPLL-tot* where the returned boolean indicates the result.

definition *DPLL-tot-rep* **where**
DPLL-tot-rep S =
(*let* (M, N) = (*rough-state-of* (*DPLL-tot* S)) *in* (∀ A ∈ *set* N. (∃ a∈*set* A. a ∈ *lits-of* (M)), M))

One version of the generated SML code is here, but not included in the generated document. The only differences are:

- export *'a literal* from the SML Module *Clausal-Logic*;
- export the constructor *Con* from *DPLL-W-Implementation*;
- export the *int* constructor from *Arith*.

  All these allows to test on the code on some examples.

end
theory *CDCL-W-Implementation*
imports *DPLL-CDCL-W-Implementation CDCL-W-Termination*
begin

notation *image-mset* (**infixr** '# *90*)

type-synonym $'a$ $cdcl_W$*-mark* = $'a$ *clause*
type-synonym $cdcl_W$*-decided-level* = *nat*

type-synonym $'v$ $cdcl_W$*-ann-literal* = ($'v$, $cdcl_W$*-decided-level*, $'v$ $cdcl_W$*-mark*) *ann-literal*
type-synonym $'v$ $cdcl_W$*-ann-literals* = ($'v$, $cdcl_W$*-decided-level*, $'v$ $cdcl_W$*-mark*) *ann-literals*
type-synonym $'v$ $cdcl_W$*-state* =
$'v$ $cdcl_W$*-ann-literals* × $'v$ *clauses* × $'v$ *clauses* × *nat* × $'v$ *clause option*

abbreviation *trail* :: $'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'a$ **where**
trail ≡ (λ(M, -). M)

abbreviation *cons-trail* :: $'a \Rightarrow 'a$ *list* $\times 'b \times 'c \times 'd \times 'e \Rightarrow 'a$ *list* $\times 'b \times 'c \times 'd \times 'e$
where
cons-trail ≡ (λL (M, S). (L#M, S))

abbreviation *tl-trail* :: $'a$ *list* $\times 'b \times 'c \times 'd \times 'e \Rightarrow 'a$ *list* $\times 'b \times 'c \times 'd \times 'e$ **where**
tl-trail ≡ (λ(M, S). (*tl* M, S))

abbreviation *clss* :: $'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'b$ **where**
clss ≡ λ(M, N, -). N

abbreviation *learned-clss* :: $'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'c$ **where**
learned-clss ≡ λ(M, N, U, -). U

abbreviation *backtrack-lvl* :: $'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'd$ **where**
backtrack-lvl ≡ λ(M, N, U, k, -). k

abbreviation *update-backtrack-lvl* :: $'d \Rightarrow 'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'a \times 'b \times 'c \times 'd \times 'e$
where
update-backtrack-lvl ≡ λk (M, N, U, -, S). (M, N, U, k, S)

abbreviation *conflicting* :: $'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'e$ **where**
conflicting ≡ λ(M, N, U, k, D). D

abbreviation *update-conflicting* :: $'e \Rightarrow 'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'a \times 'b \times 'c \times 'd \times 'e$
where
update-conflicting ≡ λS (M, N, U, k, -). (M, N, U, k, S)

abbreviation *S0-*$cdcl_W$ N ≡ (([], N, {#}, *0*, *None*):: $'v$ $cdcl_W$*-state*)

abbreviation *add-learned-cls* **where**
add-learned-cls ≡ λC (M, N, U, S). (M, N, {#C#} + U, S)

abbreviation *remove-cls* **where**
remove-cls ≡ λC (M, N, U, S). (M, *remove-mset* C N, *remove-mset* C U, S)

lemma *trail-conv*: *trail* (M, N, U, k, D) = M **and**
clauses-conv: *clss* (M, N, U, k, D) = N **and**

learned-clss-conv: $learned\text{-}clss\ (M,\ N,\ U,\ k,\ D) = U$ **and**
conflicting-conv: $conflicting\ (M,\ N,\ U,\ k,\ D) = D$ **and**
backtrack-lvl-conv: $backtrack\text{-}lvl\ (M,\ N,\ U,\ k,\ D) = k$
$\langle proof \rangle$

lemma *state-conv*:
$S = (trail\ S,\ clss\ S,\ learned\text{-}clss\ S,\ backtrack\text{-}lvl\ S,\ conflicting\ S)$
$\langle proof \rangle$

interpretation $state_W$ *trail clss learned-clss backtrack-lvl conflicting*
$\lambda L\ (M,\ S).\ (L\ \#\ M,\ S)$
$\lambda(M,\ S).\ (tl\ M,\ S)$
$\lambda C\ (M,\ N,\ S).\ (M,\ \{\#C\#\} + N,\ S)$
$\lambda C\ (M,\ N,\ U,\ S).\ (M,\ N,\ \{\#C\#\} + U,\ S)$
$\lambda C\ (M,\ N,\ U,\ S).\ (M,\ remove\text{-}mset\ C\ N,\ remove\text{-}mset\ C\ U,\ S)$
$\lambda(k{::}nat)\ (M,\ N,\ U,\ \text{-},\ D).\ (M,\ N,\ U,\ k,\ D)$
$\lambda D\ (M,\ N,\ U,\ k,\ \text{-}).\ (M,\ N,\ U,\ k,\ D)$
$\lambda N.\ ([],\ N,\ \{\#\},\ 0,\ None)$
$\lambda(\text{-},\ N,\ U,\ \text{-}).\ ([],\ N,\ U,\ 0,\ None)$
$\langle proof \rangle$

interpretation $cdcl_W$ *trail clss learned-clss backtrack-lvl conflicting*
$\lambda L\ (M,\ S).\ (L\ \#\ M,\ S)$
$\lambda(M,\ S).\ (tl\ M,\ S)$
$\lambda C\ (M,\ N,\ S).\ (M,\ \{\#C\#\} + N,\ S)$
$\lambda C\ (M,\ N,\ U,\ S).\ (M,\ N,\ \{\#C\#\} + U,\ S)$
$\lambda C\ (M,\ N,\ U,\ S).\ (M,\ remove\text{-}mset\ C\ N,\ remove\text{-}mset\ C\ U,\ S)$
$\lambda(k{::}nat)\ (M,\ N,\ U,\ \text{-},\ D).\ (M,\ N,\ U,\ k,\ D)$
$\lambda D\ (M,\ N,\ U,\ k,\ \text{-}).\ (M,\ N,\ U,\ k,\ D)$
$\lambda N.\ ([],\ N,\ \{\#\},\ 0,\ None)$
$\lambda(\text{-},\ N,\ U,\ \text{-}).\ ([],\ N,\ U,\ 0,\ None)$
$\langle proof \rangle$

declare *clauses-def*[*simp*]

lemma $cdcl_W$*-state-eq-equality*[*iff*]: $state\text{-}eq\ S\ T \longleftrightarrow S = T$
$\langle proof \rangle$
declare *state-simp*[*simp del*]

6.3 CDCL Implementation

6.3.1 Definition of the rules

Types **lemma** *true-clss-remdups*[*simp*]:
$I \models s\ (mset \circ remdups)\ `\ N \longleftrightarrow\ I \models s\ mset\ `\ N$
$\langle proof \rangle$

lemma *satisfiable-mset-remdups*[*simp*]:
$satisfiable\ ((mset \circ remdups)\ `\ N) \longleftrightarrow satisfiable\ (mset\ `\ N)$
$\langle proof \rangle$

value $backtrack\text{-}split\ [Decided\ (Pos\ (Suc\ 0))\ ()]$
value $\exists\ C \in set\ [[Pos\ (Suc\ 0),\ Neg\ (Suc\ 0)]].\ (\forall\ c \in set\ C.\ -c \in lits\text{-}of\ [Decided\ (Pos\ (Suc\ 0))\ ()])$

type-synonym $cdcl_W$*-state-inv-st* $= (nat,\ nat,\ nat\ literal\ list)\ ann\text{-}literal\ list\ \times$

nat literal list list $\times$ *nat literal list list* $\times$ *nat* $\times$ *nat literal list option*

We need some functions to convert between our abstract state *nat* $cdcl_W$*-state* and the concrete state $cdcl_W$*-state-inv-st*.

fun *convert* :: $('a, 'b, 'c\ list)$ *ann-literal* $\Rightarrow$ $('a, 'b, 'c\ multiset)$ *ann-literal* **where**
convert (*Propagated L C*) = *Propagated L* (*mset C*) |
convert (*Decided K i*) = *Decided K i*

abbreviation *convertC* :: $'a$ *list option* $\Rightarrow$ $'a$ *multiset option* **where**
convertC $\equiv$ *map-option mset*

lemma *convert-Propagated*[*elim!*]:
convert z = *Propagated L C* $\Longrightarrow$ $(\exists\, C'.\ z = Propagated\ L\ C' \wedge C = mset\ C')$
⟨*proof*⟩

lemma *get-rev-level-map-convert*:
get-rev-level (*map convert M*) *n x* = *get-rev-level M n x*
⟨*proof*⟩

lemma *get-level-map-convert*[*simp*]:
get-level (*map convert M*) = *get-level M*
⟨*proof*⟩

lemma *get-maximum-level-map-convert*[*simp*]:
get-maximum-level (*map convert M*) *D* = *get-maximum-level M D*
⟨*proof*⟩

lemma *get-all-levels-of-decided-map-convert*[*simp*]:
get-all-levels-of-decided (*map convert M*) = (*get-all-levels-of-decided M*)
⟨*proof*⟩

Conversion function

fun *toS* :: $cdcl_W$*-state-inv-st* $\Rightarrow$ *nat* $cdcl_W$*-state* **where**
toS (*M*, *N*, *U*, *k*, *C*) = (*map convert M*, *mset* (*map mset N*), *mset* (*map mset U*), *k*, *convertC C*)

Definition an abstract type

typedef $cdcl_W$*-state-inv* = {*S*::$cdcl_W$*-state-inv-st*. $cdcl_W$*-all-struct-inv* (*toS S*)}
morphisms *rough-state-of state-of*
⟨*proof*⟩

instantiation $cdcl_W$*-state-inv* :: *equal*
begin
definition *equal-*$cdcl_W$*-state-inv* :: $cdcl_W$*-state-inv* $\Rightarrow$ $cdcl_W$*-state-inv* $\Rightarrow$ *bool* **where**
equal-$cdcl_W$*-state-inv S S'* = (*rough-state-of S* = *rough-state-of S'*)
instance
⟨*proof*⟩
end

lemma *lits-of-map-convert*[*simp*]: *lits-of* (*map convert M*) = *lits-of M*
⟨*proof*⟩

lemma *undefined-lit-map-convert*[*iff*]:
undefined-lit (*map convert M*) *L* $\longleftrightarrow$ *undefined-lit M L*
⟨*proof*⟩

lemma *true-annot-map-convert*[*simp*]: *map convert* $M \models a\ N \longleftrightarrow M \models a\ N$
⟨*proof*⟩

lemma *true-annots-map-convert*[*simp*]: *map convert* $M \models as\ N \longleftrightarrow M \models as\ N$
⟨*proof*⟩

lemmas *propagateE*
lemma *find-first-unit-clause-some-is-propagate*:
assumes H: *find-first-unit-clause* $(N\ @\ U)\ M = Some\ (L,\ C)$
shows *propagate* $(toS\ (M,\ N,\ U,\ k,\ None))\ (toS\ (Propagated\ L\ C\ \#\ M,\ N,\ U,\ k,\ None))$
⟨*proof*⟩

6.3.2 The Transitions

Propagate **definition** *do-propagate-step* **where**
do-propagate-step $S =$
$(case\ S\ of$
$(M,\ N,\ U,\ k,\ None) \Rightarrow$
$(case\ find\text{-}first\text{-}unit\text{-}clause\ (N\ @\ U)\ M\ of$
$Some\ (L,\ C) \Rightarrow (Propagated\ L\ C\ \#\ M,\ N,\ U,\ k,\ None)$
$|\ None \Rightarrow (M,\ N,\ U,\ k,\ None))$
$|\ S \Rightarrow S)$

lemma *do-propgate-step*:
$do\text{-}propagate\text{-}step\ S \neq S \Longrightarrow propagate\ (toS\ S)\ (toS\ (do\text{-}propagate\text{-}step\ S))$
⟨*proof*⟩

lemma *do-propagate-step-option*[*simp*]:
$conflicting\ S \neq None \Longrightarrow do\text{-}propagate\text{-}step\ S = S$
⟨*proof*⟩

lemma *do-propagate-step-no-step*:
assumes *dist*: $\forall c \in set\ (clss\ S\ @\ learned\text{-}clss\ S).\ distinct\ c$ **and**
prop-step: $do\text{-}propagate\text{-}step\ S = S$
shows *no-step propagate* $(toS\ S)$
⟨*proof*⟩

Conflict **fun** *find-conflict* **where**
$find\text{-}conflict\ M\ [] = None\ |$
$find\text{-}conflict\ M\ (N\ \#\ Ns) = (if\ (\forall c \in set\ N.\ -c \in lits\text{-}of\ M)\ then\ Some\ N\ else\ find\text{-}conflict\ M\ Ns)$

lemma *find-conflict-Some*:
$find\text{-}conflict\ M\ Ns = Some\ N \Longrightarrow N \in set\ Ns \land M \models as\ CNot\ (mset\ N)$
⟨*proof*⟩

lemma *find-conflict-None*:
$find\text{-}conflict\ M\ Ns = None \longleftrightarrow (\forall N \in set\ Ns.\ \neg M \models as\ CNot\ (mset\ N))$
⟨*proof*⟩

lemma *find-conflict-None-no-confl*:
$find\text{-}conflict\ M\ (N@U) = None \longleftrightarrow no\text{-}step\ conflict\ (toS\ (M,\ N,\ U,\ k,\ None))$
⟨*proof*⟩

definition *do-conflict-step* **where**
do-conflict-step $S =$
$(case\ S\ of$

$(M, N, U, k, None) \Rightarrow$
  $(case\ find\text{-}conflict\ M\ (N\ @\ U)\ of$
    $Some\ a \Rightarrow (M, N, U, k, Some\ a)$
  $|\ None \Rightarrow (M, N, U, k, None))$
$|\ S \Rightarrow S)$

lemma *do-conflict-step*:
$do\text{-}conflict\text{-}step\ S \neq S \Longrightarrow conflict\ (toS\ S)\ (toS\ (do\text{-}conflict\text{-}step\ S))$
⟨*proof*⟩

lemma *do-conflict-step-no-step*:
$do\text{-}conflict\text{-}step\ S = S \Longrightarrow no\text{-}step\ conflict\ (toS\ S)$
⟨*proof*⟩

lemma *do-conflict-step-option*[*simp*]:
$conflicting\ S \neq None \Longrightarrow do\text{-}conflict\text{-}step\ S = S$
⟨*proof*⟩

lemma *do-conflict-step-conflicting*[*dest*]:
$do\text{-}conflict\text{-}step\ S \neq S \Longrightarrow conflicting\ (do\text{-}conflict\text{-}step\ S) \neq None$
⟨*proof*⟩

definition *do-cp-step* **where**
$do\text{-}cp\text{-}step\ S =$
$(do\text{-}propagate\text{-}step\ o\ do\text{-}conflict\text{-}step)\ S$

lemma *cp-step-is-cdcl*$_W$*-cp*:
assumes H: $do\text{-}cp\text{-}step\ S \neq S$
shows $cdcl_W\text{-}cp\ (toS\ S)\ (toS\ (do\text{-}cp\text{-}step\ S))$
⟨*proof*⟩

lemma *do-cp-step-eq-no-prop-no-confl*:
$do\text{-}cp\text{-}step\ S = S \Longrightarrow do\text{-}conflict\text{-}step\ S = S \wedge do\text{-}propagate\text{-}step\ S = S$
⟨*proof*⟩

lemma *no-cdcl*$_W$*-cp-iff-no-propagate-no-conflict*:
$no\text{-}step\ cdcl_W\text{-}cp\ S \longleftrightarrow no\text{-}step\ propagate\ S \wedge no\text{-}step\ conflict\ S$
⟨*proof*⟩

lemma *do-cp-step-eq-no-step*:
assumes H: $do\text{-}cp\text{-}step\ S = S$ **and** $\forall\ c \in set\ (clss\ S\ @\ learned\text{-}clss\ S).\ distinct\ c$
shows $no\text{-}step\ cdcl_W\text{-}cp\ (toS\ S)$
⟨*proof*⟩

lemma *cdcl*$_W$*-cp-cdcl*$_W$*-st*: $cdcl_W\text{-}cp\ S\ S' \Longrightarrow cdcl_W{}^{**}\ S\ S'$
⟨*proof*⟩

lemma *cdcl*$_W$*-cp-wf-all-inv*:
$wf\ \{(S', S::'v::linorder\ cdcl_W\text{-}state).\ cdcl_W\text{-}all\text{-}struct\text{-}inv\ S \wedge cdcl_W\text{-}cp\ S\ S'\}$
(**is** $wf\ ?R$)
⟨*proof*⟩

lemma *cdcl*$_W$*-all-struct-inv-rough-state*[*simp*]: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ (toS\ (rough\text{-}state\text{-}of\ S))$
⟨*proof*⟩

lemma [*simp*]: $cdcl_W$*-all-struct-inv* (*toS S*) $\Longrightarrow$ *rough-state-of* (*state-of S*) = *S*
⟨*proof*⟩

lemma *rough-state-of-state-of-do-cp-step*[*simp*]:
rough-state-of (*state-of* (*do-cp-step* (*rough-state-of S*))) = *do-cp-step* (*rough-state-of S*)
⟨*proof*⟩

Skip **fun** *do-skip-step* :: $cdcl_W$*-state-inv-st* ⇒ $cdcl_W$*-state-inv-st* **where**
do-skip-step (*Propagated L C* # *Ls*,*N*,*U*,*k*, *Some D*) =
(*if* $-L \notin$ *set D* $\wedge$ *D* $\neq$ []
then (*Ls*, *N*, *U*, *k*, *Some D*)
else (*Propagated L C* #*Ls*, *N*, *U*, *k*, *Some D*)) |
do-skip-step S = *S*

lemma *do-skip-step*:
do-skip-step S $\neq$ *S* $\Longrightarrow$ *skip* (*toS S*) (*toS* (*do-skip-step S*))
⟨*proof*⟩

lemma *do-skip-step-no*:
do-skip-step S = *S* $\Longrightarrow$ *no-step skip* (*toS S*)
⟨*proof*⟩

lemma *do-skip-step-trail-is-None*[*iff*]:
do-skip-step S = (*a*, *b*, *c*, *d*, *None*) $\longleftrightarrow$ *S* = (*a*, *b*, *c*, *d*, *None*)
⟨*proof*⟩

Resolve **fun** *maximum-level-code*:: *'a literal list* ⇒ (*'a*, *nat*, *'a literal list*) *ann-literal list* ⇒ *nat*
where
maximum-level-code [] - = *0* |
maximum-level-code (*L* # *Ls*) *M* = *max* (*get-level M L*) (*maximum-level-code Ls M*)

lemma *maximum-level-code-eq-get-maximum-level*[*code*, *simp*]:
maximum-level-code D M = *get-maximum-level M* (*mset D*)
⟨*proof*⟩

fun *do-resolve-step* :: $cdcl_W$*-state-inv-st* ⇒ $cdcl_W$*-state-inv-st* **where**
do-resolve-step (*Propagated L C* # *Ls*, *N*, *U*, *k*, *Some D*) =
(*if* $-L \in$ *set D* $\wedge$ *maximum-level-code* (*remove1* ($-L$) *D*) (*Propagated L C* # *Ls*) = *k*
then (*Ls*, *N*, *U*, *k*, *Some* (*remdups* (*remove1 L C* @ *remove1* ($-L$) *D*)))
else (*Propagated L C* # *Ls*, *N*, *U*, *k*, *Some D*)) |
do-resolve-step S = *S*

lemma *do-resolve-step*:
$cdcl_W$*-all-struct-inv* (*toS S*) $\Longrightarrow$ *do-resolve-step S* $\neq$ *S*
$\Longrightarrow$ *resolve* (*toS S*) (*toS* (*do-resolve-step S*))
⟨*proof*⟩

lemma *do-resolve-step-no*:
do-resolve-step S = *S* $\Longrightarrow$ *no-step resolve* (*toS S*)
⟨*proof*⟩

lemma *rough-state-of-state-of-resolve*[*simp*]:
$cdcl_W$*-all-struct-inv* (*toS S*) $\Longrightarrow$ *rough-state-of* (*state-of* (*do-resolve-step S*)) = *do-resolve-step S*
⟨*proof*⟩

lemma *do-resolve-step-trail-is-None*[*iff*]:
do-resolve-step $S = (a,\ b,\ c,\ d,\ None) \longleftrightarrow S = (a,\ b,\ c,\ d,\ None)$
⟨*proof*⟩

Backjumping **fun** *find-level-decomp* **where**
find-level-decomp M [] $D\ k = None$ |
find-level-decomp $M\ (L\ \#\ Ls)\ D\ k =$
(*case* (*get-level* $M\ L$, *maximum-level-code* $(D\ @\ Ls)\ M$) *of*
$(i,\ j) \Rightarrow$ *if* $i = k \wedge j < i$ *then Some* $(L,\ j)$ *else find-level-decomp* $M\ Ls\ (L\#D)\ k$
)

lemma *find-level-decomp-some*:
assumes *find-level-decomp* $M\ Ls\ D\ k = Some\ (L,\ j)$
shows $L \in set\ Ls \wedge$ *get-maximum-level* M (*mset* (*remove1* $L\ (Ls\ @\ D))) = j \wedge$ *get-level* $M\ L = k$
⟨*proof*⟩

lemma *find-level-decomp-none*:
assumes *find-level-decomp* $M\ Ls\ E\ k = None$ **and** *mset* $(L\#D) =$ *mset* $(Ls\ @\ E)$
shows $\neg(L \in set\ Ls \wedge$ *get-maximum-level* M (*mset* $D) < k \wedge k =$ *get-level* $M\ L)$
⟨*proof*⟩

fun *bt-cut* **where**
bt-cut i (*Propagated* - - # Ls) = *bt-cut* $i\ Ls$ |
bt-cut i (*Decided* $K\ k$ # Ls) = (*if* $k = Suc\ i$ *then Some* (*Decided* $K\ k$ # Ls) *else bt-cut* $i\ Ls$) |
bt-cut i [] = *None*

lemma *bt-cut-some-decomp*:
bt-cut $i\ M = Some\ M' \Longrightarrow \exists K\ M2\ M1.\ M = M2\ @\ M' \wedge M' = Decided\ K\ (i+1)\ \#\ M1$
⟨*proof*⟩

lemma *bt-cut-not-none*: $M = M2\ @\ Decided\ K\ (Suc\ i)\ \#\ M' \Longrightarrow$ *bt-cut* $i\ M \neq None$
⟨*proof*⟩

lemma *get-all-decided-decomposition-ex*:
$\exists N.$ (*Decided* K (*Suc* i) # M', N) $\in$ *set* (*get-all-decided-decomposition* ($M2$@*Decided* K (*Suc* i) # M'))
⟨*proof*⟩

lemma *bt-cut-in-get-all-decided-decomposition*:
bt-cut $i\ M = Some\ M' \Longrightarrow \exists M2.\ (M',\ M2) \in$ *set* (*get-all-decided-decomposition* M)
⟨*proof*⟩

fun *do-backtrack-step* **where**
do-backtrack-step (M, N, U, k, *Some* D) =
(*case find-level-decomp* $M\ D$ [] k *of*
None ⇒ (M, N, U, k, *Some* D)
| *Some* (L, j) ⇒
(*case bt-cut* $j\ M$ *of*
Some (*Decided* - - # Ls) ⇒ (*Propagated* $L\ D$ # Ls, N, D # U, j, *None*)
| - ⇒ (M, N, U, k, *Some* D))
) |
do-backtrack-step $S = S$

lemma *get-all-decided-decomposition-map-convert*:

$(get\text{-}all\text{-}decided\text{-}decomposition\ (map\ convert\ M)) =$
$map\ (\lambda(a,\ b).\ (map\ convert\ a,\ map\ convert\ b))\ (get\text{-}all\text{-}decided\text{-}decomposition\ M)$
⟨*proof*⟩

lemma *do-backtrack-step*:
assumes
db: $do\text{-}backtrack\text{-}step\ S \neq S$ **and**
inv: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ (toS\ S)$
shows $backtrack\ (toS\ S)\ (toS\ (do\text{-}backtrack\text{-}step\ S))$
⟨*proof*⟩

lemma *do-backtrack-step-no*:
assumes *db*: $do\text{-}backtrack\text{-}step\ S = S$
and *inv*: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ (toS\ S)$
shows $no\text{-}step\ backtrack\ (toS\ S)$
⟨*proof*⟩

lemma *rough-state-of-state-of-backtrack*[*simp*]:
assumes *inv*: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ (toS\ S)$
shows $rough\text{-}state\text{-}of\ (state\text{-}of\ (do\text{-}backtrack\text{-}step\ S)) = do\text{-}backtrack\text{-}step\ S$
⟨*proof*⟩

Decide **fun** *do-decide-step* **where**
$do\text{-}decide\text{-}step\ (M,\ N,\ U,\ k,\ None) =$
$(case\ find\text{-}first\text{-}unused\text{-}var\ N\ (lits\text{-}of\ M)\ of$
$None \Rightarrow (M,\ N,\ U,\ k,\ None)$
$|\ Some\ L \Rightarrow (Decided\ L\ (Suc\ k)\ \#\ M,\ N,\ U,\ k{+}1,\ None))\ |$
$do\text{-}decide\text{-}step\ S = S$

lemma *do-decide-step*:
$do\text{-}decide\text{-}step\ S \neq S \Longrightarrow decide\ (toS\ S)\ (toS\ (do\text{-}decide\text{-}step\ S))$
⟨*proof*⟩

lemma *do-decide-step-no*:
$do\text{-}decide\text{-}step\ S = S \Longrightarrow no\text{-}step\ decide\ (toS\ S)$
⟨*proof*⟩

lemma *rough-state-of-state-of-do-decide-step*[*simp*]:
$cdcl_W\text{-}all\text{-}struct\text{-}inv\ (toS\ S) \Longrightarrow rough\text{-}state\text{-}of\ (state\text{-}of\ (do\text{-}decide\text{-}step\ S)) = do\text{-}decide\text{-}step\ S$
⟨*proof*⟩

lemma *rough-state-of-state-of-do-skip-step*[*simp*]:
$cdcl_W\text{-}all\text{-}struct\text{-}inv\ (toS\ S) \Longrightarrow rough\text{-}state\text{-}of\ (state\text{-}of\ (do\text{-}skip\text{-}step\ S)) = do\text{-}skip\text{-}step\ S$
⟨*proof*⟩

6.3.3 Code generation

Type definition There are two invariants: one while applying conflict and propagate and one for the other rules

declare *rough-state-of-inverse*[*simp add*]
definition *Con* **where**
$Con\ xs = state\text{-}of\ (if\ cdcl_W\text{-}all\text{-}struct\text{-}inv\ (toS\ (fst\ xs,\ snd\ xs))\ then\ xs$
$else\ ([],\ [],\ [],\ 0,\ None))$

lemma [*code abstype*]:

Con $(rough\text{-}state\text{-}of\ S) = S$
⟨*proof*⟩

definition $do\text{-}cp\text{-}step'$ **where**
$do\text{-}cp\text{-}step'\ S = state\text{-}of\ (do\text{-}cp\text{-}step\ (rough\text{-}state\text{-}of\ S))$

typedef $cdcl_W\text{-}state\text{-}inv\text{-}from\text{-}init\text{-}state = \{S::cdcl_W\text{-}state\text{-}inv\text{-}st.\ cdcl_W\text{-}all\text{-}struct\text{-}inv\ (toS\ S)$
$\wedge\ cdcl_W\text{-}stgy^{**}\ (S0\text{-}cdcl_W\ (clss\ (toS\ S)))\ (toS\ S)\}$
morphisms *rough-state-from-init-state-of state-from-init-state-of*
⟨*proof*⟩

instantiation $cdcl_W\text{-}state\text{-}inv\text{-}from\text{-}init\text{-}state :: equal$
begin
definition $equal\text{-}cdcl_W\text{-}state\text{-}inv\text{-}from\text{-}init\text{-}state :: cdcl_W\text{-}state\text{-}inv\text{-}from\text{-}init\text{-}state \Rightarrow$
$cdcl_W\text{-}state\text{-}inv\text{-}from\text{-}init\text{-}state \Rightarrow bool$ **where**
$equal\text{-}cdcl_W\text{-}state\text{-}inv\text{-}from\text{-}init\text{-}state\ S\ S' \longleftrightarrow$
$(rough\text{-}state\text{-}from\text{-}init\text{-}state\text{-}of\ S = rough\text{-}state\text{-}from\text{-}init\text{-}state\text{-}of\ S')$
instance
⟨*proof*⟩
end

definition *ConI* **where**
$ConI\ S = state\text{-}from\text{-}init\text{-}state\text{-}of\ (if\ cdcl_W\text{-}all\text{-}struct\text{-}inv\ (toS\ (fst\ S,\ snd\ S))$
$\wedge\ cdcl_W\text{-}stgy^{**}\ (S0\text{-}cdcl_W\ (clss\ (toS\ S)))\ (toS\ S)\ then\ S\ else\ ([], [], [], 0, None))$

lemma [*code abstype*]:
$ConI\ (rough\text{-}state\text{-}from\text{-}init\text{-}state\text{-}of\ S) = S$
⟨*proof*⟩

definition $id\text{-}of\text{-}I\text{-}to:: cdcl_W\text{-}state\text{-}inv\text{-}from\text{-}init\text{-}state \Rightarrow cdcl_W\text{-}state\text{-}inv$ **where**
$id\text{-}of\text{-}I\text{-}to\ S = state\text{-}of\ (rough\text{-}state\text{-}from\text{-}init\text{-}state\text{-}of\ S)$

lemma [*code abstract*]:
$rough\text{-}state\text{-}of\ (id\text{-}of\text{-}I\text{-}to\ S) = rough\text{-}state\text{-}from\text{-}init\text{-}state\text{-}of\ S$
⟨*proof*⟩

Conflict and Propagate **function** $do\text{-}full1\text{-}cp\text{-}step :: cdcl_W\text{-}state\text{-}inv \Rightarrow cdcl_W\text{-}state\text{-}inv$ **where**
$do\text{-}full1\text{-}cp\text{-}step\ S =$
$(let\ S' = do\text{-}cp\text{-}step'\ S\ in$
$if\ S = S'\ then\ S\ else\ do\text{-}full1\text{-}cp\text{-}step\ S')$
⟨*proof*⟩
termination
⟨*proof*⟩

lemma *do-full1-cp-step-fix-point-of-do-full1-cp-step*:
$do\text{-}cp\text{-}step(rough\text{-}state\text{-}of\ (do\text{-}full1\text{-}cp\text{-}step\ S)) = (rough\text{-}state\text{-}of\ (do\text{-}full1\text{-}cp\text{-}step\ S))$
⟨*proof*⟩

lemma *in-clauses-rough-state-of-is-distinct*:
$c \in set\ (clss\ (rough\text{-}state\text{-}of\ S)\ @\ learned\text{-}clss\ (rough\text{-}state\text{-}of\ S)) \Longrightarrow distinct\ c$
⟨*proof*⟩

lemma *do-full1-cp-step-full*:
$full\ cdcl_W\text{-}cp\ (toS\ (rough\text{-}state\text{-}of\ S))$
$(toS\ (rough\text{-}state\text{-}of\ (do\text{-}full1\text{-}cp\text{-}step\ S)))$

⟨*proof*⟩

lemma [*code abstract*]:
rough-state-of (*do-cp-step′ S*) = *do-cp-step* (*rough-state-of S*)
⟨*proof*⟩

The other rules **fun** *do-other-step* **where**
do-other-step S =
  (*let T* = *do-skip-step S in*
    if T ≠ *S*
    then T
    else
      (*let U* = *do-resolve-step T in*
      if U ≠ *T*
      then U else
      (*let V* = *do-backtrack-step U in*
      if V ≠ *U then V else do-decide-step V*)))

lemma *do-other-step*:
  assumes *inv*: $cdcl_W$*-all-struct-inv* (*toS S*) **and**
  st: *do-other-step S* ≠ *S*
  shows $cdcl_W$*-o* (*toS S*) (*toS* (*do-other-step S*))
  ⟨*proof*⟩

lemma *do-other-step-no*:
  assumes *inv*: $cdcl_W$*-all-struct-inv* (*toS S*) **and**
  st: *do-other-step S* = *S*
  shows *no-step* $cdcl_W$*-o* (*toS S*)
  ⟨*proof*⟩

lemma *rough-state-of-state-of-do-other-step*[*simp*]:
  rough-state-of (*state-of* (*do-other-step* (*rough-state-of S*))) = *do-other-step* (*rough-state-of S*)
⟨*proof*⟩

definition *do-other-step′* **where**
do-other-step′ S =
  state-of (*do-other-step* (*rough-state-of S*))

lemma *rough-state-of-do-other-step′*[*code abstract*]:
rough-state-of (*do-other-step′ S*) = *do-other-step* (*rough-state-of S*)
⟨*proof*⟩

definition *do-*$cdcl_W$*-stgy-step* **where**
do-$cdcl_W$*-stgy-step S* =
  (*let T* = *do-full1-cp-step S in*
    if T ≠ *S*
    then T
    else
      (*let U* = (*do-other-step′ T*) *in*
       (*do-full1-cp-step U*)))

definition *do-*$cdcl_W$*-stgy-step′* **where**
do-$cdcl_W$*-stgy-step′ S* = *state-from-init-state-of* (*rough-state-of* (*do-*$cdcl_W$*-stgy-step* (*id-of-I-to S*)))

lemma *toS-do-full1-cp-step-not-eq*: *do-full1-cp-step S* ≠ *S* ⟹

toS (*rough-state-of* S) $\neq$ *toS* (*rough-state-of* (*do-full1-cp-step* S))
⟨*proof*⟩

do-full1-cp-step should not be unfolded anymore:

declare *do-full1-cp-step.simps*[*simp del*]

Correction of the transformation **lemma** *do-cdcl$_W$-stgy-step*:
assumes *do-cdcl$_W$-stgy-step* $S \neq S$
shows *cdcl$_W$-stgy* (*toS* (*rough-state-of* S)) (*toS* (*rough-state-of* (*do-cdcl$_W$-stgy-step* S)))
⟨*proof*⟩

lemma *length-trail-toS*[*simp*]:
length (*trail* (*toS* S)) = *length* (*trail* S)
⟨*proof*⟩

lemma *conflicting-noTrue-iff-toS*[*simp*]:
conflicting (*toS* S) $\neq$ *None* $\longleftrightarrow$ *conflicting* $S \neq$ *None*
⟨*proof*⟩

lemma *trail-toS-neq-imp-trail-neq*:
trail (*toS* S) $\neq$ *trail* (*toS* S') $\Longrightarrow$ *trail* $S \neq$ *trail* S'
⟨*proof*⟩

lemma *do-skip-step-trail-changed-or-conflict*:
assumes *d*: *do-other-step* $S \neq S$
and *inv*: *cdcl$_W$-all-struct-inv* (*toS* S)
shows *trail* $S \neq$ *trail* (*do-other-step* S)
⟨*proof*⟩

lemma *do-full1-cp-step-induct*:
($\bigwedge S$. ($S \neq$ *do-cp-step'* $S \Longrightarrow P$ (*do-cp-step'* S)) $\Longrightarrow P\ S$) $\Longrightarrow P$ *a0*
⟨*proof*⟩

lemma *do-cp-step-neq-trail-increase*:
$\exists c$. *trail* (*do-cp-step* S) = c @ *trail* S $\wedge$($\forall m \in$ *set* c. $\neg$ *is-decided* m)
⟨*proof*⟩

lemma *do-full1-cp-step-neq-trail-increase*:
$\exists c$. *trail* (*rough-state-of* (*do-full1-cp-step* S)) = c @ *trail* (*rough-state-of* S)
$\wedge$ ($\forall m \in$ *set* c. $\neg$ *is-decided* m)
⟨*proof*⟩

lemma *do-cp-step-conflicting*:
conflicting (*rough-state-of* S) $\neq$ *None* $\Longrightarrow$ *do-cp-step'* $S = S$
⟨*proof*⟩

lemma *do-full1-cp-step-conflicting*:
conflicting (*rough-state-of* S) $\neq$ *None* $\Longrightarrow$ *do-full1-cp-step* $S = S$
⟨*proof*⟩

lemma *do-decide-step-not-conflicting-one-more-decide*:
assumes
conflicting S = *None* **and**
do-decide-step $S \neq S$
shows *Suc* (*length* (*filter is-decided* (*trail* S)))

$= length\ (filter\ is\text{-}decided\ (trail\ (do\text{-}decide\text{-}step\ S)))$
⟨*proof*⟩

lemma *do-decide-step-not-conflicting-one-more-decide-bt*:
assumes $conflicting\ S \neq None$ **and**
$do\text{-}decide\text{-}step\ S \neq S$
shows $length\ (filter\ is\text{-}decided\ (trail\ S)) < length\ (filter\ is\text{-}decided\ (trail\ (do\text{-}decide\text{-}step\ S)))$
⟨*proof*⟩

lemma *do-other-step-not-conflicting-one-more-decide-bt*:
assumes
$conflicting\ (rough\text{-}state\text{-}of\ S) \neq None$ **and**
$conflicting\ (rough\text{-}state\text{-}of\ (do\text{-}other\text{-}step'\ S)) = None$ **and**
$do\text{-}other\text{-}step'\ S \neq S$
shows $length\ (filter\ is\text{-}decided\ (trail\ (rough\text{-}state\text{-}of\ S)))$
$> length\ (filter\ is\text{-}decided\ (trail\ (rough\text{-}state\text{-}of\ (do\text{-}other\text{-}step'\ S))))$
⟨*proof*⟩

lemma *do-other-step-not-conflicting-one-more-decide*:
assumes $conflicting\ (rough\text{-}state\text{-}of\ S) = None$ **and**
$do\text{-}other\text{-}step'\ S \neq S$
shows $1 + length\ (filter\ is\text{-}decided\ (trail\ (rough\text{-}state\text{-}of\ S)))$
$= length\ (filter\ is\text{-}decided\ (trail\ (rough\text{-}state\text{-}of\ (do\text{-}other\text{-}step'\ S))))$
⟨*proof*⟩

lemma *rough-state-of-state-of-do-skip-step-rough-state-of*[*simp*]:
$rough\text{-}state\text{-}of\ (state\text{-}of\ (do\text{-}skip\text{-}step\ (rough\text{-}state\text{-}of\ S))) = do\text{-}skip\text{-}step\ (rough\text{-}state\text{-}of\ S)$
⟨*proof*⟩

lemma *conflicting-do-resolve-step-iff*[*iff*]:
$conflicting\ (do\text{-}resolve\text{-}step\ S) = None \longleftrightarrow conflicting\ S = None$
⟨*proof*⟩

lemma *conflicting-do-skip-step-iff*[*iff*]:
$conflicting\ (do\text{-}skip\text{-}step\ S) = None \longleftrightarrow conflicting\ S = None$
⟨*proof*⟩

lemma *conflicting-do-decide-step-iff*[*iff*]:
$conflicting\ (do\text{-}decide\text{-}step\ S) = None \longleftrightarrow conflicting\ S = None$
⟨*proof*⟩

lemma *conflicting-do-backtrack-step-imp*[*simp*]:
$do\text{-}backtrack\text{-}step\ S \neq S \Longrightarrow conflicting\ (do\text{-}backtrack\text{-}step\ S) = None$
⟨*proof*⟩

lemma *do-skip-step-eq-iff-trail-eq*:
$do\text{-}skip\text{-}step\ S = S \longleftrightarrow trail\ (do\text{-}skip\text{-}step\ S) = trail\ S$
⟨*proof*⟩

lemma *do-decide-step-eq-iff-trail-eq*:
$do\text{-}decide\text{-}step\ S = S \longleftrightarrow trail\ (do\text{-}decide\text{-}step\ S) = trail\ S$
⟨*proof*⟩

lemma *do-backtrack-step-eq-iff-trail-eq*:
$do\text{-}backtrack\text{-}step\ S = S \longleftrightarrow trail\ (do\text{-}backtrack\text{-}step\ S) = trail\ S$

$\langle proof \rangle$

lemma *do-resolve-step-eq-iff-trail-eq*:
do-resolve-step S = $S \longleftrightarrow$ *trail* (*do-resolve-step* S) = *trail* S
$\langle proof \rangle$

lemma *do-other-step-eq-iff-trail-eq*:
trail (*do-other-step* S) = *trail* $S \longleftrightarrow$ *do-other-step* S = S
$\langle proof \rangle$

lemma *do-full1-cp-step-do-other-step′-normal-form*[*dest*!]:
assumes H: *do-full1-cp-step* (*do-other-step′* S) = S
shows *do-other-step′* S = $S \wedge$ *do-full1-cp-step* S = S
$\langle proof \rangle$

lemma *do-cdcl*$_W$*-stgy-step-no*:
assumes S: *do-cdcl*$_W$*-stgy-step* S = S
shows *no-step cdcl*$_W$*-stgy* (*toS* (*rough-state-of* S))
$\langle proof \rangle$

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)
$\langle proof \rangle$

lemma *cdcl*$_W$*-cp-is-rtranclp-cdcl*$_W$: *cdcl*$_W$*-cp* S $T \Longrightarrow$ *cdcl*$_W$** S T
$\langle proof \rangle$

lemma *rtranclp-cdcl*$_W$*-cp-is-rtranclp-cdcl*$_W$: *cdcl*$_W$*-cp*** S $T \Longrightarrow$ *cdcl*$_W$** S T
$\langle proof \rangle$

lemma *cdcl*$_W$*-stgy-is-rtranclp-cdcl*$_W$:
cdcl$_W$*-stgy* S $T \Longrightarrow$ *cdcl*$_W$** S T
$\langle proof \rangle$

lemma *cdcl*$_W$*-stgy-init-clss*: *cdcl*$_W$*-stgy* S $T \Longrightarrow$ *cdcl*$_W$*-M-level-inv* $S \Longrightarrow$ *clss* S = *clss* T
$\langle proof \rangle$

lemma *clauses-toS-rough-state-of-do-cdcl*$_W$*-stgy-step*[*simp*]:
clss (*toS* (*rough-state-of* (*do-cdcl*$_W$*-stgy-step* (*state-of* (*rough-state-from-init-state-of* S)))))
= *clss* (*toS* (*rough-state-from-init-state-of* S)) (**is** - = *clss* (*toS* *?S*))
$\langle proof \rangle$

lemma *rough-state-from-init-state-of-do-cdcl*$_W$*-stgy-step′*[*code abstract*]:
rough-state-from-init-state-of (*do-cdcl*$_W$*-stgy-step′* S) =
rough-state-of (*do-cdcl*$_W$*-stgy-step* (*id-of-I-to* S))
$\langle proof \rangle$

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)
$\langle proof \rangle$
termination

⟨*proof*⟩

thm *do-all-cdcl$_W$-stgy.induct*
lemma *do-all-cdcl$_W$-stgy-induct*:
$(\bigwedge S.\ (\textit{do-cdcl}_W\textit{-stgy-step}'\ S \neq S \Longrightarrow P\ (\textit{do-cdcl}_W\textit{-stgy-step}'\ S)) \Longrightarrow P\ S) \Longrightarrow P\ a0$
⟨*proof*⟩

lemma *no-step-cdcl$_W$-stgy-cdcl$_W$-all*:
$\textit{no-step cdcl}_W\textit{-stgy}\ (\textit{toS}\ (\textit{rough-state-from-init-state-of}\ (\textit{do-all-cdcl}_W\textit{-stgy}\ S)))$
⟨*proof*⟩

lemma *do-all-cdcl$_W$-stgy-is-rtranclp-cdcl$_W$-stgy*:
$\textit{cdcl}_W\textit{-stgy}^{**}\ (\textit{toS}\ (\textit{rough-state-from-init-state-of}\ S))$
$(\textit{toS}\ (\textit{rough-state-from-init-state-of}\ (\textit{do-all-cdcl}_W\textit{-stgy}\ S)))$
⟨*proof*⟩

Final theorem:

lemma *DPLL-tot-correct*:
assumes
r: $\textit{rough-state-from-init-state-of}\ (\textit{do-all-cdcl}_W\textit{-stgy}\ (\textit{state-from-init-state-of}$
$(([], \textit{map remdups}\ N, [], 0, \textit{None})))) = S$ **and**
S: $(M', N', U', k, E) = \textit{toS}\ S$
shows $(E \neq \textit{Some}\ \{\#\} \wedge \textit{satisfiable}\ (\textit{set}\ (\textit{map mset}\ N)))$
$\vee (E = \textit{Some}\ \{\#\} \wedge \textit{unsatisfiable}\ (\textit{set}\ (\textit{map mset}\ N)))$
⟨*proof*⟩

The Code The SML code is skipped in the documentation, but stays to ensure that some version of the exported code is working. The only difference between the generated code and the one used here is the export of the constructor ConI.

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

7 Link between Weidenbach's and NOT's CDCL

7.1 Inclusion of the states

declare *upt.simps*(*2*)[*simp del*]
sledgehammer-params[*verbose*]

context *cdcl$_W$*
begin

lemma *backtrack-levE*:
$\textit{backtrack}\ S\ S' \Longrightarrow \textit{cdcl}_W\textit{-M-level-inv}\ S \Longrightarrow$
$(\bigwedge D\ L\ K\ M1\ M2.$
$(\textit{Decided}\ K\ (\textit{Suc}\ (\textit{get-maximum-level}\ (\textit{trail}\ S)\ D))\ \#\ M1, M2)$
$\in \textit{set}\ (\textit{get-all-decided-decomposition}\ (\textit{trail}\ S)) \Longrightarrow$
$\textit{get-level}\ (\textit{trail}\ S)\ L = \textit{get-maximum-level}\ (\textit{trail}\ S)\ (D + \{\#L\#\}) \Longrightarrow$
$\textit{undefined-lit}\ M1\ L \Longrightarrow$
$S' \sim \textit{cons-trail}\ (\textit{Propagated}\ L\ (D + \{\#L\#\}))$
$(\textit{reduce-trail-to}\ M1\ (\textit{add-learned-cls}\ (D + \{\#L\#\})$
$(\textit{update-backtrack-lvl}\ (\textit{get-maximum-level}\ (\textit{trail}\ S)\ D)\ (\textit{update-conflicting None}\ S)))) \Longrightarrow$

$backtrack\text{-}lvl\ S = get\text{-}maximum\text{-}level\ (trail\ S)\ (D + \{\#L\#\}) \Longrightarrow$
$conflicting\ S = Some\ (D + \{\#L\#\}) \Longrightarrow P) \Longrightarrow$
P
⟨proof⟩

lemma $backtrack\text{-}no\text{-}cdcl_W\text{-}bj$:
assumes $cdcl$: $cdcl_W\text{-}bj\ T\ U$ **and** inv: $cdcl_W\text{-}M\text{-}level\text{-}inv\ V$
shows $\neg backtrack\ V\ T$
⟨proof⟩

abbreviation $skip\text{-}or\text{-}resolve :: 'st \Rightarrow 'st \Rightarrow bool$ **where**
$skip\text{-}or\text{-}resolve \equiv (\lambda S\ T.\ skip\ S\ T \lor resolve\ S\ T)$

lemma $rtranclp\text{-}cdcl_W\text{-}bj\text{-}skip\text{-}or\text{-}resolve\text{-}backtrack$:
assumes $cdcl_W\text{-}bj^{**}\ S\ U$ **and** inv: $cdcl_W\text{-}M\text{-}level\text{-}inv\ S$
shows $skip\text{-}or\text{-}resolve^{**}\ S\ U \lor (\exists\ T.\ skip\text{-}or\text{-}resolve^{**}\ S\ T \land backtrack\ T\ U)$
⟨proof⟩

lemma $rtranclp\text{-}skip\text{-}or\text{-}resolve\text{-}rtranclp\text{-}cdcl_W$:
$skip\text{-}or\text{-}resolve^{**}\ S\ T \Longrightarrow cdcl_W^{**}\ S\ T$
⟨proof⟩

definition $backjump\text{-}l\text{-}cond :: 'v\ clause \Rightarrow 'v\ clause \Rightarrow 'v\ literal \Rightarrow 'st \Rightarrow bool$ **where**
$backjump\text{-}l\text{-}cond \equiv \lambda C\ C'\ L'\ S.\ True$

definition $inv_{NOT} :: 'st \Rightarrow bool$ **where**
$inv_{NOT} \equiv \lambda S.\ no\text{-}dup\ (trail\ S)$

declare $inv_{NOT}\text{-}def[simp]$
end

fun $convert\text{-}ann\text{-}literal\text{-}from\text{-}W$ **where**
$convert\text{-}ann\text{-}literal\text{-}from\text{-}W\ (Propagated\ L\ \text{-}) = Propagated\ L\ ()\ |$
$convert\text{-}ann\text{-}literal\text{-}from\text{-}W\ (Decided\ L\ \text{-}) = Decided\ L\ ()$

abbreviation $convert\text{-}trail\text{-}from\text{-}W ::$
$('v,\ 'lvl,\ 'a)\ ann\text{-}literal\ list$
$\Rightarrow ('v,\ unit,\ unit)\ ann\text{-}literal\ list$ **where**
$convert\text{-}trail\text{-}from\text{-}W \equiv map\ convert\text{-}ann\text{-}literal\text{-}from\text{-}W$

lemma $lits\text{-}of\text{-}convert\text{-}trail\text{-}from\text{-}W[simp]$:
$lits\text{-}of\ (convert\text{-}trail\text{-}from\text{-}W\ M) = lits\text{-}of\ M$
⟨proof⟩

lemma $lit\text{-}of\text{-}convert\text{-}trail\text{-}from\text{-}W[simp]$:
$lit\text{-}of\ (convert\text{-}ann\text{-}literal\text{-}from\text{-}W\ L) = lit\text{-}of\ L$
⟨proof⟩

lemma $no\text{-}dup\text{-}convert\text{-}from\text{-}W[simp]$:
$no\text{-}dup\ (convert\text{-}trail\text{-}from\text{-}W\ M) \longleftrightarrow no\text{-}dup\ M$
⟨proof⟩

lemma $convert\text{-}trail\text{-}from\text{-}W\text{-}true\text{-}annots[simp]$:
$convert\text{-}trail\text{-}from\text{-}W\ M \models as\ C \longleftrightarrow M \models as\ C$

⟨*proof*⟩

lemma *defined-lit-convert-trail-from-W*[*simp*]:
defined-lit (*convert-trail-from-W S*) *L* ⟷ *defined-lit S L*
⟨*proof*⟩

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

fun *convert-ann-literal-from-NOT*
:: ('*a*, '*e*, '*b*) *ann-literal* ⇒ ('*a*, *nat*, '*a literal multiset*) *ann-literal* **where**
convert-ann-literal-from-NOT (*Propagated L* -) = *Propagated L* {#} |
convert-ann-literal-from-NOT (*Decided L* -) = *Decided L 0*

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

lemma *undefined-lit-convert-trail-from-NOT*[*simp*]:
undefined-lit (*convert-trail-from-NOT F*) *L* ⟷ *undefined-lit F L*
⟨*proof*⟩

lemma *lits-of-convert-trail-from-NOT*:
lits-of (*convert-trail-from-NOT F*) = *lits-of F*
⟨*proof*⟩

lemma *convert-trail-from-W-from-NOT*[*simp*]:
convert-trail-from-W (*convert-trail-from-NOT M*) = *M*
⟨*proof*⟩

lemma *convert-trail-from-W-convert-lit-from-NOT*[*simp*]:
convert-ann-literal-from-W (*convert-ann-literal-from-NOT L*) = *L*
⟨*proof*⟩

abbreviation $trail_{NOT}$ **where**
$trail_{NOT}$ *S* ≡ *convert-trail-from-W* (*fst S*)

lemma *undefined-lit-convert-trail-from-W*[*iff*]:
undefined-lit (*convert-trail-from-W M*) *L* ⟷ *undefined-lit M L*
⟨*proof*⟩

lemma *lit-of-convert-ann-literal-from-NOT*[*iff*]:
lit-of (*convert-ann-literal-from-NOT L*) = *lit-of L*
⟨*proof*⟩

sublocale $state_W$ ⊆ *dpll-state*
λ*S*. *convert-trail-from-W* (*trail S*)
clauses
λ*L S*. *cons-trail* (*convert-ann-literal-from-NOT L*) *S*
λ*S*. *tl-trail S*
λ*C S*. *add-learned-cls C S*
λ*C S*. *remove-cls C S*
⟨*proof*⟩

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

sublocale $cdcl_W \subseteq cdcl_{NOT}$*-merge-bj-learn-ops*
$\lambda S.$ *convert-trail-from-W* (*trail* S)
clauses
$\lambda L\ S.$ *cons-trail* (*convert-ann-literal-from-NOT* L) S
$\lambda S.$ *tl-trail* S
$\lambda C\ S.$ *add-learned-cls* $C\ S$
$\lambda C\ S.$ *remove-cls* $C\ S$
λ- -. *True*
λ- $S.$ *conflicting* S = *None*
$\lambda C\ C'\ L'\ S.$ *backjump-l-cond* $C\ C'\ L'\ S \wedge$ *distinct-mset* $(C' + \{\#L'\#\}) \wedge \neg$*tautology* $(C' + \{\#L'\#\})$
⟨*proof*⟩

sublocale $cdcl_W \subseteq cdcl_{NOT}$*-merge-bj-learn-proxy*
$\lambda S.$ *convert-trail-from-W* (*trail* S)
clauses
$\lambda L\ S.$ *cons-trail* (*convert-ann-literal-from-NOT* L) S
$\lambda S.$ *tl-trail* S
$\lambda C\ S.$ *add-learned-cls* $C\ S$
$\lambda C\ S.$ *remove-cls* $C\ S$
λ- -. *True*
λ- $S.$ *conflicting* S = *None* *backjump-l-cond* inv_{NOT}
⟨*proof*⟩

sublocale $cdcl_W \subseteq cdcl_{NOT}$*-merge-bj-learn-proxy2*
$\lambda S.$ *convert-trail-from-W* (*trail* S)
clauses
$\lambda L\ S.$ *cons-trail* (*convert-ann-literal-from-NOT* L) S
$\lambda S.$ *tl-trail* S
$\lambda C\ S.$ *add-learned-cls* $C\ S$
$\lambda C\ S.$ *remove-cls* $C\ S$ λ- -. *True* inv_{NOT}
λ- $S.$ *conflicting* S = *None* *backjump-l-cond*
⟨*proof*⟩

sublocale $cdcl_W \subseteq cdcl_{NOT}$*-merge-bj-learn*
$\lambda S.$ *convert-trail-from-W* (*trail* S)
clauses
$\lambda L\ S.$ *cons-trail* (*convert-ann-literal-from-NOT* L) S
$\lambda S.$ *tl-trail* S
$\lambda C\ S.$ *add-learned-cls* $C\ S$
$\lambda C\ S.$ *remove-cls* $C\ S$ λ- -. *True* inv_{NOT}
λ- $S.$ *conflicting* S = *None* *backjump-l-cond*
⟨*proof*⟩

context $cdcl_W$
begin

Notations are lost while proving locale inclusion:

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

7.2 Additional Lemmas between NOT and W states

lemma $trail_W$*-eq-reduce-trail-to*$_{NOT}$*-eq*:
trail S = *trail* $T \Longrightarrow$ *trail* (*reduce-trail-to*$_{NOT}$ $F\ S$) = *trail* (*reduce-trail-to*$_{NOT}$ $F\ T$)
⟨*proof*⟩

lemma *trail-reduce-trail-to*$_{NOT}$*-add-learned-cls*:
no-dup (*trail* S) $\Longrightarrow$
trail (*reduce-trail-to*$_{NOT}$ M (*add-learned-cls* D S)) = *trail* (*reduce-trail-to*$_{NOT}$ M S)
⟨*proof*⟩

lemma *reduce-trail-to*$_{NOT}$*-reduce-trail-convert*:
reduce-trail-to$_{NOT}$ C S = *reduce-trail-to* (*convert-trail-from-NOT* C) S
⟨*proof*⟩

lemma *reduce-trail-to-length*:
length M = *length* M' $\Longrightarrow$ *reduce-trail-to* M S = *reduce-trail-to* M' S
⟨*proof*⟩

7.3 More lemmas conflict–propagate and backjumping

7.3.1 Termination

lemma *cdcl*$_W$*-cp-normalized-element-all-inv*:
assumes *inv*: *cdcl*$_W$*-all-struct-inv* S
obtains T **where** *full cdcl*$_W$*-cp* S T
⟨*proof*⟩
thm *backtrackE*

lemma *cdcl*$_W$*-bj-measure*:
assumes *cdcl*$_W$*-bj* S T **and** *cdcl*$_W$*-M-level-inv* S
shows *length* (*trail* S) + (*if conflicting* S = *None then 0 else 1*)
$>$ *length* (*trail* T) + (*if conflicting* T = *None then 0 else 1*)
⟨*proof*⟩

lemma *wf-cdcl*$_W$*-bj*:
wf {(b,a). *cdcl*$_W$*-bj* a b $\wedge$ *cdcl*$_W$*-M-level-inv* a}
⟨*proof*⟩

lemma *cdcl*$_W$*-bj-exists-normal-form*:
assumes *lev*: *cdcl*$_W$*-M-level-inv* S
shows $\exists$ T. *full cdcl*$_W$*-bj* S T
⟨*proof*⟩

lemma *rtranclp-skip-state-decomp*:
assumes *skip*** S T **and** *no-dup* (*trail* S)
shows
$\exists$ M. *trail* S = M @ *trail* T $\wedge$ ($\forall$ $m$$\in$*set* M. $\neg$*is-decided* m) **and**
T $\sim$ *delete-trail-and-rebuild* (*trail* T) S
⟨*proof*⟩

7.3.2 More backjumping

Backjumping after skipping or jump directly **lemma** *rtranclp-skip-backtrack-backtrack*:
assumes
skip** S T **and**
backtrack T W **and**
cdcl$_W$*-all-struct-inv* S
shows *backtrack* S W
⟨*proof*⟩

lemma *fst-get-all-decided-decomposition-prepend-not-decided*:

assumes $\forall\, m \in set\ MS.\ \neg\ is\text{-}decided\ m$
shows $set\ (map\ fst\ (get\text{-}all\text{-}decided\text{-}decomposition\ M))$
$= set\ (map\ fst\ (get\text{-}all\text{-}decided\text{-}decomposition\ (MS\ @\ M)))$
$\langle proof \rangle$

See also $[\![skip^{**}\ ?S\ ?T;\ backtrack\ ?T\ ?W;\ cdcl_W\text{-}all\text{-}struct\text{-}inv\ ?S]\!] \Longrightarrow backtrack\ ?S\ ?W$

lemma *rtranclp-skip-backtrack-backtrack-end*:
assumes
skip: $skip^{**}\ S\ T$ **and**
bt: *backtrack S W* **and**
inv: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$
shows *backtrack T W*
$\langle proof \rangle$

lemma $cdcl_W$*-bj-decomp-resolve-skip-and-bj*:
assumes $cdcl_W\text{-}bj^{**}\ S\ T$ **and** *inv*: $cdcl_W\text{-}M\text{-}level\text{-}inv\ S$
shows $(skip\text{-}or\text{-}resolve^{**}\ S\ T$
$\vee (\exists\, U.\ skip\text{-}or\text{-}resolve^{**}\ S\ U \wedge backtrack\ U\ T))$
$\langle proof \rangle$

lemma *resolve-skip-deterministic*:
$resolve\ S\ T \Longrightarrow skip\ S\ U \Longrightarrow False$
$\langle proof \rangle$

lemma *backtrack-unique*:
assumes
bt-T: *backtrack S T* **and**
bt-U: *backtrack S U* **and**
inv: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$
shows $T \sim U$
$\langle proof \rangle$

lemma *if-can-apply-backtrack-no-more-resolve*:
assumes
skip: $skip^{**}\ S\ U$ **and**
bt: *backtrack S T* **and**
inv: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$
shows $\neg resolve\ U\ V$
$\langle proof \rangle$

lemma *if-can-apply-resolve-no-more-backtrack*:
assumes
skip: $skip^{**}\ S\ U$ **and**
resolve: *resolve S T* **and**
inv: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$
shows $\neg backtrack\ U\ V$
$\langle proof \rangle$

lemma *if-can-apply-backtrack-skip-or-resolve-is-skip*:
assumes
bt: *backtrack S T* **and**
skip: $skip\text{-}or\text{-}resolve^{**}\ S\ U$ **and**
inv: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$
shows $skip^{**}\ S\ U$
$\langle proof \rangle$

lemma *cdcl$_W$-bj-bj-decomp*:
assumes *cdcl$_W$-bj*** S W **and** *cdcl$_W$-all-struct-inv* S
shows
($\exists$ T U V. (λS T. *skip-or-resolve* S T $\wedge$ *no-step backtrack* S)** S T
$\wedge$ (λT U. *resolve* T U $\wedge$ *no-step backtrack* T) T U
$\wedge$ *skip*** U V $\wedge$ *backtrack* V W)
$\vee$ ($\exists$ T U. (λS T. *skip-or-resolve* S T $\wedge$ *no-step backtrack* S)** S T
$\wedge$ (λT U. *resolve* T U $\wedge$ *no-step backtrack* T) T U $\wedge$ *skip*** U W)
$\vee$ ($\exists$ T. *skip*** S T $\wedge$ *backtrack* T W)
$\vee$ *skip*** S W (**is** *?RB* S W $\vee$ *?R* S W $\vee$ *?SB* S W $\vee$ *?S* S W)
$\langle proof \rangle$

The case distinction is needed, since $T \sim V$ does not imply that R^{**} T V.

lemma *cdcl$_W$-bj-strongly-confluent*:
assumes
cdcl$_W$-bj** S V **and**
cdcl$_W$-bj** S T **and**
n-s: *no-step cdcl$_W$-bj* V **and**
inv: *cdcl$_W$-all-struct-inv* S
shows $T \sim V$ $\vee$ *cdcl$_W$-bj*** T V
$\langle proof \rangle$

lemma *cdcl$_W$-bj-unique-normal-form*:
assumes
ST: *cdcl$_W$-bj*** S T **and** *SU*: *cdcl$_W$-bj*** S U **and**
n-s-U: *no-step cdcl$_W$-bj* U **and**
n-s-T: *no-step cdcl$_W$-bj* T **and**
inv: *cdcl$_W$-all-struct-inv* S
shows $T \sim U$
$\langle proof \rangle$

lemma *full-cdcl$_W$-bj-unique-normal-form*:
assumes *full cdcl$_W$-bj* S T **and** *full cdcl$_W$-bj* S U **and**
inv: *cdcl$_W$-all-struct-inv* S
shows $T \sim U$
$\langle proof \rangle$

7.4 CDCL FW

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

lemma *cdcl$_W$-merge-restart-cdcl$_W$*:
assumes *cdcl$_W$-merge-restart* S T
shows *cdcl$_W$*** S T
$\langle proof \rangle$

lemma *cdcl$_W$-merge-restart-conflicting-true-or-no-step*:
assumes *cdcl$_W$-merge-restart* S T
shows *conflicting* T = *None* $\vee$ *no-step cdcl$_W$* T
$\langle proof \rangle$

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

lemma $cdcl_W$*-merge-*$cdcl_W$*-merge-restart*:
$cdcl_W$*-merge S T* ⟹ $cdcl_W$*-merge-restart S T*
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-merge-tranclp-*$cdcl_W$*-merge-restart*:
$cdcl_W$*-merge*** *S T* ⟹ $cdcl_W$*-merge-restart*** *S T*
⟨*proof*⟩

lemma $cdcl_W$*-merge-rtranclp-*$cdcl_W$:
$cdcl_W$*-merge S T* ⟹ $cdcl_W{}^{**}$ *S T*
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-merge-rtranclp-*$cdcl_W$:
$cdcl_W$*-merge*** *S T* ⟹ $cdcl_W{}^{**}$ *S T*
⟨*proof*⟩

lemma $cdcl_W$*-merge-is-*$cdcl_{NOT}$*-merged-bj-learn*:
assumes
inv: $cdcl_W$*-all-struct-inv S* **and**
$cdcl_W$:$cdcl_W$*-merge S T*
shows $cdcl_{NOT}$*-merged-bj-learn S T*
∨ (*no-step* $cdcl_W$*-merge T* ∧ *conflicting T* ≠ *None*)
⟨*proof*⟩

abbreviation $cdcl_{NOT}$*-restart* **where**
$cdcl_{NOT}$*-restart* ≡ *restart-ops.*$cdcl_{NOT}$*-raw-restart* $cdcl_{NOT}$ *restart*

lemma $cdcl_W$*-merge-restart-is-*$cdcl_{NOT}$*-merged-bj-learn-restart-no-step*:
assumes
inv: $cdcl_W$*-all-struct-inv S* **and**
$cdcl_W$:$cdcl_W$*-merge-restart S T*
shows $cdcl_{NOT}$*-restart*** *S T* ∨ (*no-step* $cdcl_W$*-merge T* ∧ *conflicting T* ≠ *None*)
⟨*proof*⟩

abbreviation μ_{FW} :: *'st* ⇒ *nat* **where**
μ_{FW} *S* ≡ (*if no-step* $cdcl_W$*-merge S then 0 else 1*+$\mu_{CDCL}{}'$*-merged* (*set-mset* (*init-clss S*)) *S*)

lemma $cdcl_W$*-merge-*μ_{FW}*-decreasing*:
assumes
inv: $cdcl_W$*-all-struct-inv S* **and**
fw: $cdcl_W$*-merge S T*
shows μ_{FW} *T* < μ_{FW} *S*
⟨*proof*⟩

lemma *wf-*$cdcl_W$*-merge*: *wf* {(*T*, *S*). $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$*-merge S T*}
⟨*proof*⟩

lemma $cdcl_W$*-all-struct-inv-tranclp-*$cdcl_W$*-merge-tranclp-*$cdcl_W$*-merge-*$cdcl_W$*-all-struct-inv*:

assumes
inv: $cdcl_W$*-all-struct-inv b*
$cdcl_W$*-merge*$^{++}$ *b a*
shows $(\lambda S\ T.\ cdcl_W\text{-}all\text{-}struct\text{-}inv\ S \wedge cdcl_W\text{-}merge\ S\ T)^{++}\ b\ a$
⟨*proof*⟩

lemma *wf-tranclp-*$cdcl_W$*-merge*: $wf\ \{(T, S).\ cdcl_W\text{-}all\text{-}struct\text{-}inv\ S \wedge cdcl_W\text{-}merge^{++}\ S\ T\}$
⟨*proof*⟩

lemma *backtrack-is-full1-*$cdcl_W$*-bj*:
assumes *bt*: *backtrack S T* **and** *inv*: $cdcl_W$*-M-level-inv S*
shows *full1* $cdcl_W$*-bj S T*
⟨*proof*⟩

lemma *rtrancl-*$cdcl_W$*-conflicting-true-*$cdcl_W$*-merge-restart*:
assumes $cdcl_W^{**}\ S\ V$ **and** *inv*: $cdcl_W$*-M-level-inv S* **and** $conflicting\ S = None$
shows $(cdcl_W\text{-}merge\text{-}restart^{**}\ S\ V \wedge conflicting\ V = None)$
$\vee (\exists\ T\ U.\ cdcl_W\text{-}merge\text{-}restart^{**}\ S\ T \wedge conflicting\ V \neq None \wedge conflict\ T\ U \wedge cdcl_W\text{-}bj^{**}\ U\ V)$
⟨*proof*⟩

lemma *no-step-*$cdcl_W$*-no-step-*$cdcl_W$*-merge-restart*: *no-step* $cdcl_W$ $S \Longrightarrow$ *no-step* $cdcl_W$*-merge-restart* *S*
⟨*proof*⟩

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

lemma *rtranclp-*$cdcl_W$*-merge-restart-no-step-*$cdcl_W$*-bj*:
assumes
$cdcl_W\text{-}merge\text{-}restart^{**}\ S\ T$ **and**
$conflicting\ S = None$
shows *no-step* $cdcl_W$*-bj T*
⟨*proof*⟩

If $conflicting\ S \neq None$, we cannot say anything.

Remark that this theorem does not say anything about well-foundedness: even if you know that one relation is well-founded, it only states that the normal forms are shared.

lemma *conflicting-true-full-*$cdcl_W$*-iff-full-*$cdcl_W$*-merge*:
assumes *confl*: $conflicting\ S = None$ **and** *lev*: $cdcl_W$*-M-level-inv S*
shows *full* $cdcl_W$ $S\ V \longleftrightarrow$ *full* $cdcl_W$*-merge-restart S V*
⟨*proof*⟩

lemma *init-state-true-full-*$cdcl_W$*-iff-full-*$cdcl_W$*-merge*:
shows *full* $cdcl_W$ $(init\text{-}state\ N)\ V \longleftrightarrow$ *full* $cdcl_W$*-merge-restart* $(init\text{-}state\ N)\ V$
⟨*proof*⟩

7.5 FW with strategy

7.5.1 The intermediate step

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

$conflict'$: $full1\ cdcl_W\text{-}cp\ S\ S' \Longrightarrow cdcl_W\text{-}s'\ S\ S'$ |
$decide'$: $decide\ S\ S' \Longrightarrow no\text{-}step\ cdcl_W\text{-}cp\ S \Longrightarrow full\ cdcl_W\text{-}cp\ S'\ S'' \Longrightarrow cdcl_W\text{-}s'\ S\ S''$ |
bj': $full1\ cdcl_W\text{-}bj\ S\ S' \Longrightarrow no\text{-}step\ cdcl_W\text{-}cp\ S \Longrightarrow full\ cdcl_W\text{-}cp\ S'\ S'' \Longrightarrow cdcl_W\text{-}s'\ S\ S''$

inductive-cases $cdcl_W\text{-}s'E$: $cdcl_W\text{-}s'\ S\ T$

lemma $rtranclp\text{-}cdcl_W\text{-}bj\text{-}full1\text{-}cdclp\text{-}cdcl_W\text{-}stgy$:
$cdcl_W\text{-}bj^{**}\ S\ S' \Longrightarrow full\ cdcl_W\text{-}cp\ S'\ S'' \Longrightarrow cdcl_W\text{-}stgy^{**}\ S\ S''$
⟨*proof*⟩

lemma $cdcl_W\text{-}s'\text{-}is\text{-}rtranclp\text{-}cdcl_W\text{-}stgy$:
$cdcl_W\text{-}s'\ S\ T \Longrightarrow cdcl_W\text{-}stgy^{**}\ S\ T$
⟨*proof*⟩

lemma $cdcl_W\text{-}cp\text{-}cdcl_W\text{-}bj\text{-}bissimulation$:
assumes
$full\ cdcl_W\text{-}cp\ T\ U$ **and**
$cdcl_W\text{-}bj^{**}\ T\ T'$ **and**
$cdcl_W\text{-}all\text{-}struct\text{-}inv\ T$ **and**
$no\text{-}step\ cdcl_W\text{-}bj\ T'$
shows $full\ cdcl_W\text{-}cp\ T'\ U$
$\vee (\exists\, U'\ U''.\ full\ cdcl_W\text{-}cp\ T'\ U'' \wedge full1\ cdcl_W\text{-}bj\ U\ U' \wedge full\ cdcl_W\text{-}cp\ U'\ U'' \wedge cdcl_W\text{-}s'^{**}\ U\ U'')$
⟨*proof*⟩

lemma $cdcl_W\text{-}cp\text{-}cdcl_W\text{-}bj\text{-}bissimulation'$:
assumes
$full\ cdcl_W\text{-}cp\ T\ U$ **and**
$cdcl_W\text{-}bj^{**}\ T\ T'$ **and**
$cdcl_W\text{-}all\text{-}struct\text{-}inv\ T$ **and**
$no\text{-}step\ cdcl_W\text{-}bj\ T'$
shows $full\ cdcl_W\text{-}cp\ T'\ U$
$\vee (\exists\, U'.\ full1\ cdcl_W\text{-}bj\ U\ U' \wedge (\forall\, U''.\ full\ cdcl_W\text{-}cp\ U'\ U'' \longrightarrow full\ cdcl_W\text{-}cp\ T'\ U''$
$\wedge\ cdcl_W\text{-}s'^{**}\ U\ U''))$
⟨*proof*⟩

lemma $cdcl_W\text{-}stgy\text{-}cdcl_W\text{-}s'\text{-}connected$:
assumes $cdcl_W\text{-}stgy\ S\ U$ **and** $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$
shows $cdcl_W\text{-}s'\ S\ U$
$\vee (\exists\, U'.\ full1\ cdcl_W\text{-}bj\ U\ U' \wedge (\forall\, U''.\ full\ cdcl_W\text{-}cp\ U'\ U'' \longrightarrow cdcl_W\text{-}s'\ S\ U''))$
⟨*proof*⟩

lemma $cdcl_W\text{-}stgy\text{-}cdcl_W\text{-}s'\text{-}connected'$:
assumes $cdcl_W\text{-}stgy\ S\ U$ **and** $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$
shows $cdcl_W\text{-}s'\ S\ U$
$\vee (\exists\, U'\ U''.\ cdcl_W\text{-}s'\ S\ U'' \wedge full1\ cdcl_W\text{-}bj\ U\ U' \wedge full\ cdcl_W\text{-}cp\ U'\ U'')$
⟨*proof*⟩

lemma $cdcl_W\text{-}stgy\text{-}cdcl_W\text{-}s'\text{-}no\text{-}step$:
assumes $cdcl_W\text{-}stgy\ S\ U$ **and** $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$ **and** $no\text{-}step\ cdcl_W\text{-}bj\ U$
shows $cdcl_W\text{-}s'\ S\ U$
⟨*proof*⟩

lemma $rtranclp\text{-}cdcl_W\text{-}stgy\text{-}connected\text{-}to\text{-}rtranclp\text{-}cdcl_W\text{-}s'$:
assumes $cdcl_W\text{-}stgy^{**}\ S\ U$ **and** inv: $cdcl_W\text{-}M\text{-}level\text{-}inv\ S$
shows $cdcl_W\text{-}s'^{**}\ S\ U \vee (\exists\, T.\ cdcl_W\text{-}s'^{**}\ S\ T \wedge cdcl_W\text{-}bj^{++}\ T\ U \wedge conflicting\ U \neq None)$

$\langle proof \rangle$

lemma $n\text{-}step\text{-}cdcl_W\text{-}stgy\text{-}iff\text{-}no\text{-}step\text{-}cdcl_W\text{-}cl\text{-}cdcl_W\text{-}o$:
assumes inv: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$
shows $no\text{-}step\ cdcl_W\text{-}s'\ S \longleftrightarrow no\text{-}step\ cdcl_W\text{-}cp\ S \wedge no\text{-}step\ cdcl_W\text{-}o\ S$ (**is** $?S'\ S \longleftrightarrow ?C\ S \wedge ?O\ S$)
$\langle proof \rangle$

lemma $cdcl_W\text{-}s'\text{-}tranclp\text{-}cdcl_W$:
$cdcl_W\text{-}s'\ S\ S' \Longrightarrow cdcl_W^{++}\ S\ S'$
$\langle proof \rangle$

lemma $tranclp\text{-}cdcl_W\text{-}s'\text{-}tranclp\text{-}cdcl_W$:
$cdcl_W\text{-}s'^{++}\ S\ S' \Longrightarrow cdcl_W^{++}\ S\ S'$
$\langle proof \rangle$

lemma $rtranclp\text{-}cdcl_W\text{-}s'\text{-}rtranclp\text{-}cdcl_W$:
$cdcl_W\text{-}s'^{**}\ S\ S' \Longrightarrow cdcl_W^{**}\ S\ S'$
$\langle proof \rangle$

lemma $full\text{-}cdcl_W\text{-}stgy\text{-}iff\text{-}full\text{-}cdcl_W\text{-}s'$:
assumes inv: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$
shows $full\ cdcl_W\text{-}stgy\ S\ T \longleftrightarrow full\ cdcl_W\text{-}s'\ S\ T$ (**is** $?S \longleftrightarrow ?S'$)
$\langle proof \rangle$

lemma $conflict\text{-}step\text{-}cdcl_W\text{-}stgy\text{-}step$:
assumes
$conflict\ S\ T$
$cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$
shows $\exists T.\ cdcl_W\text{-}stgy\ S\ T$
$\langle proof \rangle$

lemma $decide\text{-}step\text{-}cdcl_W\text{-}stgy\text{-}step$:
assumes
$decide\ S\ T$
$cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$
shows $\exists T.\ cdcl_W\text{-}stgy\ S\ T$
$\langle proof \rangle$

lemma $rtranclp\text{-}cdcl_W\text{-}cp\text{-}conflicting\text{-}Some$:
$cdcl_W\text{-}cp^{**}\ S\ T \Longrightarrow conflicting\ S = Some\ D \Longrightarrow S = T$
$\langle proof \rangle$

inductive $cdcl_W\text{-}merge\text{-}cp :: 'st \Rightarrow 'st \Rightarrow bool$ **where**
$conflict'[intro]$: $conflict\ S\ T \Longrightarrow full\ cdcl_W\text{-}bj\ T\ U \Longrightarrow cdcl_W\text{-}merge\text{-}cp\ S\ U\ |$
$propagate'[intro]$: $propagate^{++}\ S\ S' \Longrightarrow cdcl_W\text{-}merge\text{-}cp\ S\ S'$

lemma $cdcl_W\text{-}merge\text{-}restart\text{-}cases[consumes\ 1,\ case\text{-}names\ conflict\ propagate]$:
assumes
$cdcl_W\text{-}merge\text{-}cp\ S\ U$ **and**
$\bigwedge T.\ conflict\ S\ T \Longrightarrow full\ cdcl_W\text{-}bj\ T\ U \Longrightarrow P$ **and**
$propagate^{++}\ S\ U \Longrightarrow P$
shows P
$\langle proof \rangle$

lemma $cdcl_W\text{-}merge\text{-}cp\text{-}tranclp\text{-}cdcl_W\text{-}merge$:

$cdcl_W$*-merge-cp* S T $\Longrightarrow$ $cdcl_W$*-merge*$^{++}$ S T
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-merge-cp-rtranclp-*$cdcl_W$:
$cdcl_W$*-merge-cp*** S T $\Longrightarrow$ $cdcl_W{}^{**}$ S T
⟨*proof*⟩

lemma *full1-*$cdcl_W$*-bj-no-step-*$cdcl_W$*-bj*:
full1 $cdcl_W$*-bj* S T $\Longrightarrow$ *no-step* $cdcl_W$*-cp* S
⟨*proof*⟩

inductive $cdcl_W$*-s′-without-decide* **where**
conflict′-without-decide[*intro*]: *full1* $cdcl_W$*-cp* S S' $\Longrightarrow$ $cdcl_W$*-s′-without-decide* S S' |
bj′-without-decide[*intro*]: *full1* $cdcl_W$*-bj* S S' $\Longrightarrow$ *no-step* $cdcl_W$*-cp* S $\Longrightarrow$ *full* $cdcl_W$*-cp* S' S''
$\Longrightarrow$ $cdcl_W$*-s′-without-decide* S S''

lemma *rtranclp-*$cdcl_W$*-s′-without-decide-rtranclp-*$cdcl_W$:
$cdcl_W$*-s′-without-decide*** S T $\Longrightarrow$ $cdcl_W{}^{**}$ S T
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-s′-without-decide-rtranclp-*$cdcl_W$*-s′*:
$cdcl_W$*-s′-without-decide*** S T $\Longrightarrow$ $cdcl_W$*-s′*** S T
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-merge-cp-is-rtranclp-*$cdcl_W$*-s′-without-decide*:
assumes
$cdcl_W$*-merge-cp*** S V
conflicting S = *None*
shows
($cdcl_W$*-s′-without-decide*** S V)
∨ (∃ T. $cdcl_W$*-s′-without-decide*** S T ∧ *propagate*$^{++}$ T V)
∨ (∃ T U. $cdcl_W$*-s′-without-decide*** S T ∧ *full1* $cdcl_W$*-bj* T U ∧ *propagate*** U V)
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-s′-without-decide-is-rtranclp-*$cdcl_W$*-merge-cp*:
assumes
$cdcl_W$*-s′-without-decide*** S V **and**
confl: *conflicting* S = *None*
shows
($cdcl_W$*-merge-cp*** S V ∧ *conflicting* V = *None*)
∨ ($cdcl_W$*-merge-cp*** S V ∧ *conflicting* V ≠ *None* ∧ *no-step* $cdcl_W$*-cp* V ∧ *no-step* $cdcl_W$*-bj* V)
∨ (∃ T. $cdcl_W$*-merge-cp*** S T ∧ *conflict* T V)
⟨*proof*⟩

lemma *no-step-*$cdcl_W$*-s′-no-ste-*$cdcl_W$*-merge-cp*:
assumes
$cdcl_W$*-all-struct-inv* S
conflicting S = *None*
no-step $cdcl_W$*-s′* S
shows *no-step* $cdcl_W$*-merge-cp* S
⟨*proof*⟩

The *no-step decide* S is needed, since $cdcl_W$*-merge-cp* is $cdcl_W$*-s′* without *decide*.

lemma *conflicting-true-no-step-*$cdcl_W$*-merge-cp-no-step-s′-without-decide*:

assumes
confl: *conflicting S = None* **and**
inv: $cdcl_W$*-M-level-inv S* **and**
n-s: *no-step* $cdcl_W$*-merge-cp S*
shows *no-step* $cdcl_W$*-s'-without-decide S*
⟨*proof*⟩

lemma *conflicting-true-no-step-s'-without-decide-no-step-*$cdcl_W$*-merge-cp*:
assumes
inv: $cdcl_W$*-all-struct-inv S* **and**
n-s: *no-step* $cdcl_W$*-s'-without-decide S*
shows *no-step* $cdcl_W$*-merge-cp S*
⟨*proof*⟩

lemma *no-step-*$cdcl_W$*-merge-cp-no-step-*$cdcl_W$*-cp*:
no-step $cdcl_W$*-merge-cp S* $\Longrightarrow$ $cdcl_W$*-M-level-inv S* $\Longrightarrow$ *no-step* $cdcl_W$*-cp S*
⟨*proof*⟩

lemma *conflicting-not-true-rtranclp-*$cdcl_W$*-merge-cp-no-step-*$cdcl_W$*-bj*:
assumes
conflicting S = None **and**
$cdcl_W$*-merge-cp*** *S T*
shows *no-step* $cdcl_W$*-bj T*
⟨*proof*⟩

lemma *conflicting-true-full-*$cdcl_W$*-merge-cp-iff-full-*$cdcl_W$*-s'-without-decode*:
assumes
confl: *conflicting S = None* **and**
inv: $cdcl_W$*-all-struct-inv S*
shows
full $cdcl_W$*-merge-cp S V* $\longleftrightarrow$ *full* $cdcl_W$*-s'-without-decide S V* (**is** *?fw* $\longleftrightarrow$ *?s'*)
⟨*proof*⟩

lemma *conflicting-true-full1-*$cdcl_W$*-merge-cp-iff-full1-*$cdcl_W$*-s'-without-decode*:
assumes
confl: *conflicting S = None* **and**
inv: $cdcl_W$*-all-struct-inv S*
shows
full1 $cdcl_W$*-merge-cp S V* $\longleftrightarrow$ *full1* $cdcl_W$*-s'-without-decide S V*
⟨*proof*⟩

lemma *conflicting-true-full1-*$cdcl_W$*-merge-cp-imp-full1-*$cdcl_W$*-s'-without-decode*:
assumes
fw: *full1* $cdcl_W$*-merge-cp S V* **and**
inv: $cdcl_W$*-all-struct-inv S*
shows
full1 $cdcl_W$*-s'-without-decide S V*
⟨*proof*⟩

inductive $cdcl_W$*-merge-stgy* **where**
fw-s-cp[*intro*]: *full1* $cdcl_W$*-merge-cp S T* $\Longrightarrow$ $cdcl_W$*-merge-stgy S T* |
fw-s-decide[*intro*]: *decide S T* $\Longrightarrow$ *no-step* $cdcl_W$*-merge-cp S* $\Longrightarrow$ *full* $cdcl_W$*-merge-cp T U*
$\Longrightarrow$ $cdcl_W$*-merge-stgy S U*

lemma $cdcl_W$*-merge-stgy-tranclp-*$cdcl_W$*-merge*:

assumes *fw*: $cdcl_W$*-merge-stgy* S T
shows $cdcl_W$*-merge*$^{++}$ S T
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-merge-stgy-rtranclp-*$cdcl_W$*-merge*:
assumes *fw*: $cdcl_W$*-merge-stgy*** S T
shows $cdcl_W$*-merge*** S T
⟨*proof*⟩

lemma $cdcl_W$*-merge-stgy-rtranclp-*$cdcl_W$:
$cdcl_W$*-merge-stgy* S $T \Longrightarrow cdcl_W{}^{**}$ S T
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-merge-stgy-rtranclp-*$cdcl_W$:
$cdcl_W$*-merge-stgy*** S $T \Longrightarrow cdcl_W{}^{**}$ S T
⟨*proof*⟩

lemma $cdcl_W$*-merge-stgy-cases*[*consumes 1*, *case-names fw-s-cp fw-s-decide*]:
assumes
$cdcl_W$*-merge-stgy* S U
full1 $cdcl_W$*-merge-cp* S $U \Longrightarrow P$
$\bigwedge T$. *decide* S $T \Longrightarrow$ *no-step* $cdcl_W$*-merge-cp* $S \Longrightarrow$ *full* $cdcl_W$*-merge-cp* T $U \Longrightarrow P$
shows P
⟨*proof*⟩

inductive $cdcl_W$*-s′-w* :: *′st* ⇒ *′st* ⇒ *bool* **where**
conflict′: *full1* $cdcl_W$*-s′-without-decide* S $S' \Longrightarrow cdcl_W$*-s′-w* S S' |
decide′: *decide* S $S' \Longrightarrow$ *no-step* $cdcl_W$*-s′-without-decide* $S \Longrightarrow$ *full* $cdcl_W$*-s′-without-decide* S' S''
$\Longrightarrow cdcl_W$*-s′-w* S S''

lemma $cdcl_W$*-s′-w-rtranclp-*$cdcl_W$:
$cdcl_W$*-s′-w* S $T \Longrightarrow cdcl_W{}^{**}$ S T
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-s′-w-rtranclp-*$cdcl_W$:
$cdcl_W$*-s′-w*** S $T \Longrightarrow cdcl_W{}^{**}$ S T
⟨*proof*⟩

lemma *no-step-*$cdcl_W$*-cp-no-step-*$cdcl_W$*-s′-without-decide*:
assumes *no-step* $cdcl_W$*-cp* S **and** *conflicting* S = *None* **and** *inv*: $cdcl_W$*-M-level-inv* S
shows *no-step* $cdcl_W$*-s′-without-decide* S
⟨*proof*⟩

lemma *no-step-*$cdcl_W$*-cp-no-step-*$cdcl_W$*-merge-restart*:
assumes *no-step* $cdcl_W$*-cp* S **and** *conflicting* S = *None*
shows *no-step* $cdcl_W$*-merge-cp* S
⟨*proof*⟩
lemma *after-*$cdcl_W$*-s′-without-decide-no-step-*$cdcl_W$*-cp*:
assumes $cdcl_W$*-s′-without-decide* S T
shows *no-step* $cdcl_W$*-cp* T
⟨*proof*⟩

lemma *no-step-*$cdcl_W$*-s′-without-decide-no-step-*$cdcl_W$*-cp*:
$cdcl_W$*-all-struct-inv* $S \Longrightarrow$ *no-step* $cdcl_W$*-s′-without-decide* $S \Longrightarrow$ *no-step* $cdcl_W$*-cp* S
⟨*proof*⟩

lemma *after-cdcl$_W$-s'-w-no-step-cdcl$_W$-cp*:
assumes *cdcl$_W$-s'-w* S T **and** *cdcl$_W$-all-struct-inv* S
shows *no-step cdcl$_W$-cp* T
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-s'-w-no-step-cdcl$_W$-cp-or-eq*:
assumes *cdcl$_W$-s'-w*** S T **and** *cdcl$_W$-all-struct-inv* S
shows $S = T \lor$ *no-step cdcl$_W$-cp* T
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-merge-stgy'-no-step-cdcl$_W$-cp-or-eq*:
assumes *cdcl$_W$-merge-stgy*** S T **and** *inv*: *cdcl$_W$-all-struct-inv* S
shows $S = T \lor$ *no-step cdcl$_W$-cp* T
⟨*proof*⟩

lemma *no-step-cdcl$_W$-s'-without-decide-no-step-cdcl$_W$-bj*:
assumes *no-step cdcl$_W$-s'-without-decide* S **and** *inv*: *cdcl$_W$-all-struct-inv* S
shows *no-step cdcl$_W$-bj* S
⟨*proof*⟩

lemma *cdcl$_W$-s'-w-no-step-cdcl$_W$-bj*:
assumes *cdcl$_W$-s'-w* S T **and** *cdcl$_W$-all-struct-inv* S
shows *no-step cdcl$_W$-bj* T
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-s'-w-no-step-cdcl$_W$-bj-or-eq*:
assumes *cdcl$_W$-s'-w*** S T **and** *cdcl$_W$-all-struct-inv* S
shows $S = T \lor$ *no-step cdcl$_W$-bj* T
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-s'-no-step-cdcl$_W$-s'-without-decide-decomp-into-cdcl$_W$-merge*:
assumes
cdcl$_W$-s'** R V **and**
conflicting $R =$ *None* **and**
inv: *cdcl$_W$-all-struct-inv* R
shows (*cdcl$_W$-merge-stgy*** R V $\land$ *conflicting* $V =$ *None*)
$\lor$ (*cdcl$_W$-merge-stgy*** R V $\land$ *conflicting* $V \neq$ *None* $\land$ *no-step cdcl$_W$-bj* V)
$\lor$ ($\exists S\ T\ U$. *cdcl$_W$-merge-stgy*** R S $\land$ *no-step cdcl$_W$-merge-cp* S $\land$ *decide* S T
$\land$ *cdcl$_W$-merge-cp*** T U $\land$ *conflict* U V)
$\lor$ ($\exists S\ T$. *cdcl$_W$-merge-stgy*** R S $\land$ *no-step cdcl$_W$-merge-cp* S $\land$ *decide* S T
$\land$ *cdcl$_W$-merge-cp*** T V
$\land$ *conflicting* $V =$ *None*)
$\lor$ (*cdcl$_W$-merge-cp*** R V $\land$ *conflicting* $V =$ *None*)
$\lor$ ($\exists U$. *cdcl$_W$-merge-cp*** R U $\land$ *conflict* U V)
⟨*proof*⟩

lemma *decide-rtranclp-cdcl$_W$-s'-rtranclp-cdcl$_W$-s'*:
assumes
dec: *decide* S T **and**
cdcl$_W$-s'** T U **and**
n-s-S: *no-step cdcl$_W$-cp* S **and**
no-step cdcl$_W$-cp U
shows *cdcl$_W$-s'*** S U
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-merge-stgy-rtranclp-cdcl$_W$-s'*:
  assumes
    cdcl$_W$-merge-stgy** *R V* **and**
    inv: *cdcl$_W$-all-struct-inv R*
  shows *cdcl$_W$-s'*** *R V*
  ⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-merge-stgy-distinct-mset-clauses*:
  assumes *invR*: *cdcl$_W$-all-struct-inv R* **and**
  st: *cdcl$_W$-merge-stgy*** *R S* **and**
  dist: *distinct-mset* (*clauses R*) **and**
  R: *trail R* = []
  shows *distinct-mset* (*clauses S*)
  ⟨*proof*⟩

lemma *no-step-cdcl$_W$-s'-no-step-cdcl$_W$-merge-stgy*:
  assumes
    inv: *cdcl$_W$-all-struct-inv R* **and** *s'*: *no-step cdcl$_W$-s' R*
  shows *no-step cdcl$_W$-merge-stgy R*
⟨*proof*⟩

lemma *wf-cdcl$_W$-merge-cp*:
  wf{(*T*, *S*). *cdcl$_W$-all-struct-inv S* ∧ *cdcl$_W$-merge-cp S T*}
  ⟨*proof*⟩

lemma *wf-cdcl$_W$-merge-stgy*:
  wf{(*T*, *S*). *cdcl$_W$-all-struct-inv S* ∧ *cdcl$_W$-merge-stgy S T*}
  ⟨*proof*⟩

lemma *cdcl$_W$-merge-cp-obtain-normal-form*:
  assumes *inv*: *cdcl$_W$-all-struct-inv R*
  obtains *S* **where** *full cdcl$_W$-merge-cp R S*
⟨*proof*⟩

lemma *no-step-cdcl$_W$-merge-stgy-no-step-cdcl$_W$-s'*:
  assumes
    inv: *cdcl$_W$-all-struct-inv R* **and**
    confl: *conflicting R* = *None* **and**
    n-s: *no-step cdcl$_W$-merge-stgy R*
  shows *no-step cdcl$_W$-s' R*
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-merge-cp-no-step-cdcl$_W$-bj*:
  assumes *conflicting R* = *None* **and** *cdcl$_W$-merge-cp*** *R S*
  shows *no-step cdcl$_W$-bj S*
  ⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-merge-stgy-no-step-cdcl$_W$-bj*:
  assumes *confl*: *conflicting R* = *None* **and** *cdcl$_W$-merge-stgy*** *R S*
  shows *no-step cdcl$_W$-bj S*
  ⟨*proof*⟩

lemma *full-cdcl$_W$-s'-full-cdcl$_W$-merge-restart*:
  assumes

conflicting R = None **and**
inv: *cdcl$_W$-all-struct-inv R*
shows *full cdcl$_W$-s′ R V* ⟷ *full cdcl$_W$-merge-stgy R V* (**is** *?s′* ⟷ *?fw*)
⟨*proof*⟩

lemma *full-cdcl$_W$-stgy-full-cdcl$_W$-merge*:
assumes
conflicting R = None **and**
inv: *cdcl$_W$-all-struct-inv R*
shows *full cdcl$_W$-stgy R V* ⟷ *full cdcl$_W$-merge-stgy R V*
⟨*proof*⟩

lemma *full-cdcl$_W$-merge-stgy-final-state-conclusive′*:
fixes *S′* :: *′st*
assumes *full*: *full cdcl$_W$-merge-stgy* (*init-state N*) *S′*
and *no-d*: *distinct-mset-mset N*
shows (*conflicting S′ = Some* {#} ∧ *unsatisfiable* (*set-mset N*))
∨ (*conflicting S′ = None* ∧ *trail S′* ⊨*asm N* ∧ *satisfiable* (*set-mset N*))
⟨*proof*⟩

end

7.6 Adding Restarts

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

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

init-state :: *′v clauses* ⇒ *′st* **and**
restart-state :: *′st* ⇒ *′st* +
fixes *f* :: *nat* ⇒ *nat*
assumes *f*: *unbounded f*
begin

The condition of the differences of cardinality has to be strict. Otherwise, you could be in a strange state, where nothing remains to do, but a restart is done. See the proof of well-foundedness.

inductive *cdcl$_W$-merge-with-restart* **where**
restart-step:
(*cdcl$_W$-merge-stgy*^^(*card* (*set-mset* (*learned-clss T*)) − *card* (*set-mset* (*learned-clss S*)))) *S T*

$\Longrightarrow$ *card* (*set-mset* (*learned-clss* T)) $-$ *card* (*set-mset* (*learned-clss* S)) $>$ *f n*
$\Longrightarrow$ *restart* T U $\Longrightarrow$ *cdcl*$_W$*-merge-with-restart* (S, n) (U, *Suc n*) |
restart-full: *full1 cdcl*$_W$*-merge-stgy* S T $\Longrightarrow$ *cdcl*$_W$*-merge-with-restart* (S, n) (T, *Suc n*)

lemma *cdcl*$_W$*-merge-with-restart* S T $\Longrightarrow$ *cdcl*$_W$*-merge-restart*** (*fst* S) (*fst* T)
⟨*proof*⟩

lemma *cdcl*$_W$*-merge-with-restart-rtranclp-cdcl*$_W$:
cdcl$_W$*-merge-with-restart* S T $\Longrightarrow$ *cdcl*$_W{}^{**}$ (*fst* S) (*fst* T)
⟨*proof*⟩

lemma *cdcl*$_W$*-merge-with-restart-increasing-number*:
cdcl$_W$*-merge-with-restart* S T $\Longrightarrow$ *snd* T = *1* + *snd* S
⟨*proof*⟩

lemma *full1 cdcl*$_W$*-merge-stgy* S T $\Longrightarrow$ *cdcl*$_W$*-merge-with-restart* (S, n) (T, *Suc n*)
⟨*proof*⟩

lemma *cdcl*$_W$*-all-struct-inv-learned-clss-bound*:
assumes *inv*: *cdcl*$_W$*-all-struct-inv* S
shows *set-mset* (*learned-clss* S) $\subseteq$ *simple-clss* (*atms-of-msu* (*init-clss* S))
⟨*proof*⟩

lemma *cdcl*$_W$*-merge-with-restart-init-clss*:
cdcl$_W$*-merge-with-restart* S T $\Longrightarrow$ *cdcl*$_W$*-M-level-inv* (*fst* S) $\Longrightarrow$
init-clss (*fst* S) = *init-clss* (*fst* T)
⟨*proof*⟩

lemma
wf {(T, S). *cdcl*$_W$*-all-struct-inv* (*fst* S) $\wedge$ *cdcl*$_W$*-merge-with-restart* S T}
⟨*proof*⟩

lemma *cdcl*$_W$*-merge-with-restart-distinct-mset-clauses*:
assumes *invR*: *cdcl*$_W$*-all-struct-inv* (*fst* R) **and**
st: *cdcl*$_W$*-merge-with-restart* R S **and**
dist: *distinct-mset* (*clauses* (*fst* R)) **and**
R: *trail* (*fst* R) = []
shows *distinct-mset* (*clauses* (*fst* S))
⟨*proof*⟩

inductive *cdcl*$_W$*-with-restart* **where**
restart-step:
(*cdcl*$_W$*-stgy*^^(*card* (*set-mset* (*learned-clss* T)) $-$ *card* (*set-mset* (*learned-clss* S)))) S T $\Longrightarrow$
card (*set-mset* (*learned-clss* T)) $-$ *card* (*set-mset* (*learned-clss* S)) $>$ *f n* $\Longrightarrow$
restart T U $\Longrightarrow$
cdcl$_W$*-with-restart* (S, n) (U, *Suc n*) |
restart-full: *full1 cdcl*$_W$*-stgy* S T $\Longrightarrow$ *cdcl*$_W$*-with-restart* (S, n) (T, *Suc n*)

lemma *cdcl*$_W$*-with-restart-rtranclp-cdcl*$_W$:
cdcl$_W$*-with-restart* S T $\Longrightarrow$ *cdcl*$_W{}^{**}$ (*fst* S) (*fst* T)
⟨*proof*⟩

lemma *cdcl*$_W$*-with-restart-increasing-number*:
cdcl$_W$*-with-restart* S T $\Longrightarrow$ *snd* T = *1* + *snd* S
⟨*proof*⟩

lemma *full1 cdcl*$_W$*-stgy S T* $\Longrightarrow$ *cdcl*$_W$*-with-restart* (*S*, *n*) (*T*, *Suc n*)
⟨*proof*⟩

lemma *cdcl*$_W$*-with-restart-init-clss*:
cdcl$_W$*-with-restart S T* $\Longrightarrow$ *cdcl*$_W$*-M-level-inv* (*fst S*) $\Longrightarrow$ *init-clss* (*fst S*) = *init-clss* (*fst T*)
⟨*proof*⟩

lemma
wf {(*T*, *S*). *cdcl*$_W$*-all-struct-inv* (*fst S*) ∧ *cdcl*$_W$*-with-restart S T*}
⟨*proof*⟩

lemma *cdcl*$_W$*-with-restart-distinct-mset-clauses*:
assumes *invR*: *cdcl*$_W$*-all-struct-inv* (*fst R*) **and**
st: *cdcl*$_W$*-with-restart R S* **and**
dist: *distinct-mset* (*clauses* (*fst R*)) **and**
R: *trail* (*fst R*) = []
shows *distinct-mset* (*clauses* (*fst S*))
⟨*proof*⟩
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*⟩

Luby sequences are defined by:

- $2^k - 1$, if $i = (2::'a)^k - (1::'a)$
- *luby-sequence-core* $(i - 2^{k-1} + 1)$, if $(2::'a)^{k-1} \leq i$ and $i \leq (2::'a)^k - (1::'a)$

Then the sequence is then scaled by a constant unit run (called *ur* here), strictly positive.

function *luby-sequence-core* :: *nat* ⇒ *nat* **where**
luby-sequence-core i =
(*if* ∃ *k*. *i* = *2*^*k* − *1*
then *2*^((*SOME k*. *i* = *2*^*k* − *1*) − *1*)
else luby-sequence-core (*i* − *2*^((*SOME k*. *2*^(*k*−*1*)≤ *i* ∧ *i* < *2*^*k* − *1*) − *1*) + *1*))
⟨*proof*⟩
termination
⟨*proof*⟩

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

lemma *two-pover-n-eq-two-power-n′-eq*:
assumes *H*: (*2*::*nat*) ^ (*k*::*nat*) − *1* = *2* ^ *k′* − *1*
shows *k′* = *k*
⟨*proof*⟩

lemma *luby-sequence-core-two-power-minus-one*:

luby-sequence-core $(2^k - 1) = 2^{(k-1)}$ (**is** $?L = ?K$)
⟨*proof*⟩

lemma *different-luby-decomposition-false*:
assumes
H: $2 \mathbin{\char`\^} (k - Suc\ 0) \leq i$ **and**
k': $i < 2 \mathbin{\char`\^} k' - Suc\ 0$ **and**
k-k': $k > k'$
shows *False*
⟨*proof*⟩

lemma *luby-sequence-core-not-two-power-minus-one*:
assumes
k-i: $2 \mathbin{\char`\^} (k - 1) \leq i$ **and**
i-k: $i < 2 \mathbin{\char`\^} k - 1$
shows *luby-sequence-core* i = *luby-sequence-core* $(i - 2 \mathbin{\char`\^} (k - 1) + 1)$
⟨*proof*⟩

lemma *unbounded-luby-sequence-core*: *unbounded luby-sequence-core*
⟨*proof*⟩

abbreviation *luby-sequence* :: *nat* ⇒ *nat* **where**
luby-sequence $n \equiv$ *ur* ∗ *luby-sequence-core* n

lemma *bounded-luby-sequence*: *unbounded luby-sequence*
⟨*proof*⟩

lemma *luby-sequence-core-0*: *luby-sequence-core* $0 = 1$
⟨*proof*⟩

lemma *luby-sequence-core* $n \geq 1$
⟨*proof*⟩
end

locale *luby-sequence-restart* =
luby-sequence ur +
$cdcl_W$ *trail init-clss learned-clss backtrack-lvl conflicting cons-trail tl-trail*
add-init-cls
add-learned-cls remove-cls update-backtrack-lvl update-conflicting init-state
restart-state
for
ur :: *nat* **and**
trail :: *'st* ⇒ (*'v*, *nat*, *'v clause*) *ann-literals* **and**
init-clss :: *'st* ⇒ *'v clauses* **and**
learned-clss :: *'st* ⇒ *'v clauses* **and**
backtrack-lvl :: *'st* ⇒ *nat* **and**
conflicting :: *'st* ⇒*'v clause option* **and**
cons-trail :: (*'v*, *nat*, *'v clause*) *ann-literal* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒ *'st* **and**
add-init-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
add-learned-cls remove-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
update-backtrack-lvl :: *nat* ⇒ *'st* ⇒ *'st* **and**
update-conflicting :: *'v clause option* ⇒ *'st* ⇒ *'st* **and**

init-state :: *'v clauses* ⇒ *'st* **and**

restart-state :: *'st* ⇒ *'st*
begin

sublocale $cdcl_W$*-restart* - - - - - - - - - - - - - - *luby-sequence*
⟨*proof*⟩

end

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

8 Incremental SAT solving

context $cdcl_W$
begin

This invariant holds all the invariant related to the strategy. See the structural invariant in $cdcl_W$*-all-struct-inv*

definition $cdcl_W$*-stgy-invariant* **where**
$cdcl_W$*-stgy-invariant S* ⟷
conflict-is-false-with-level S
∧ *no-clause-is-false S*
∧ *no-smaller-confl S*
∧ *no-clause-is-false S*

lemma $cdcl_W$*-stgy-*$cdcl_W$*-stgy-invariant*:
assumes
$cdcl_W$: $cdcl_W$*-stgy S T* **and**
inv-s: $cdcl_W$*-stgy-invariant S* **and**
inv: $cdcl_W$*-all-struct-inv S*
shows
$cdcl_W$*-stgy-invariant T*
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-stgy-*$cdcl_W$*-stgy-invariant*:
assumes
$cdcl_W$: $cdcl_W$*-stgy*** *S T* **and**
inv-s: $cdcl_W$*-stgy-invariant S* **and**
inv: $cdcl_W$*-all-struct-inv S*
shows
$cdcl_W$*-stgy-invariant T*
⟨*proof*⟩

abbreviation *decr-bt-lvl* **where**
decr-bt-lvl S ≡ *update-backtrack-lvl* (*backtrack-lvl S* − *1*) *S*

When we add a new clause, we reduce the trail until we get to tho first literal included in C. Then we can mark the conflict.

fun *cut-trail-wrt-clause* **where**
cut-trail-wrt-clause C [] *S* = *S* |
cut-trail-wrt-clause C (*Decided L* - # *M*) *S* =
(*if* −*L* ∈# *C then S*

else cut-trail-wrt-clause C M (decr-bt-lvl (tl-trail S))) |
cut-trail-wrt-clause C (Propagated L - # M) S =
(*if* $-L \in\#$ *C then S*
else cut-trail-wrt-clause C M (tl-trail S))

definition *add-new-clause-and-update* :: *'v literal multiset* ⇒ *'st* ⇒ *'st* **where**
add-new-clause-and-update C S =
(*if trail S* ⊨*as CNot C*
then update-conflicting (Some C) (add-init-cls C (cut-trail-wrt-clause C (trail S) S))
else add-init-cls C S)

thm *cut-trail-wrt-clause.induct*
lemma *init-clss-cut-trail-wrt-clause*[*simp*]:
init-clss (cut-trail-wrt-clause C M S) = init-clss S
⟨*proof*⟩

lemma *learned-clss-cut-trail-wrt-clause*[*simp*]:
learned-clss (cut-trail-wrt-clause C M S) = learned-clss S
⟨*proof*⟩

lemma *conflicting-clss-cut-trail-wrt-clause*[*simp*]:
conflicting (cut-trail-wrt-clause C M S) = conflicting S
⟨*proof*⟩

lemma *trail-cut-trail-wrt-clause*:
∃ *M. trail S = M @ trail (cut-trail-wrt-clause C (trail S) S)*
⟨*proof*⟩

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*⟩

lemma *cut-trail-wrt-clause-backtrack-lvl-length-decided*:
assumes
backtrack-lvl T = length (get-all-levels-of-decided (trail T))
shows
backtrack-lvl (cut-trail-wrt-clause C (trail T) T) =
length (get-all-levels-of-decided (trail (cut-trail-wrt-clause C (trail T) T)))
⟨*proof*⟩

lemma *cut-trail-wrt-clause-get-all-levels-of-decided*:
assumes *get-all-levels-of-decided (trail T) = rev [Suc 0..<*
Suc (length (get-all-levels-of-decided (trail T)))]
shows
get-all-levels-of-decided (trail ((cut-trail-wrt-clause C (trail T) T))) = rev [Suc 0..<
Suc (length (get-all-levels-of-decided (trail ((cut-trail-wrt-clause C (trail T) T)))))]
⟨*proof*⟩

lemma *cut-trail-wrt-clause-CNot-trail*:
assumes *trail T* ⊨*as CNot C*
shows
(*trail ((cut-trail-wrt-clause C (trail T) T)))* ⊨*as CNot C*
⟨*proof*⟩

lemma *cut-trail-wrt-clause-hd-trail-in-or-empty-trail*:
$((\forall L \in\# C.\ -L \notin \textit{lits-of}\ (\textit{trail}\ T)) \land \textit{trail}\ (\textit{cut-trail-wrt-clause}\ C\ (\textit{trail}\ T)\ T) = [])$
$\lor\ (-\textit{lit-of}\ (\textit{hd}\ (\textit{trail}\ (\textit{cut-trail-wrt-clause}\ C\ (\textit{trail}\ T)\ T))) \in\# C$
$\land\ \textit{length}\ (\textit{trail}\ (\textit{cut-trail-wrt-clause}\ C\ (\textit{trail}\ T)\ T)) \geq 1)$
⟨*proof*⟩

We can fully run $\textit{cdcl}_W\textit{-s}$ or add a clause. Remark that we use $\textit{cdcl}_W\textit{-s}$ to avoid an explicit *skip*, *resolve*, and *backtrack* normalisation to get rid of the conflict C if possible.

inductive $\textit{incremental-cdcl}_W$ :: $'st \Rightarrow 'st \Rightarrow \textit{bool}$ **for** S **where**
add-confl:
$\textit{trail}\ S \models\!\textit{asm}\ \textit{init-clss}\ S \Longrightarrow \textit{distinct-mset}\ C \Longrightarrow \textit{conflicting}\ S = \textit{None} \Longrightarrow$
$\textit{trail}\ S \models\!\textit{as}\ \textit{CNot}\ C \Longrightarrow$
$\textit{full}\ \textit{cdcl}_W\textit{-stgy}$
$(\textit{update-conflicting}\ (\textit{Some}\ C)\ (\textit{add-init-cls}\ C\ (\textit{cut-trail-wrt-clause}\ C\ (\textit{trail}\ S)\ S)))\ T \Longrightarrow$
$\textit{incremental-cdcl}_W\ S\ T$ |
add-no-confl:
$\textit{trail}\ S \models\!\textit{asm}\ \textit{init-clss}\ S \Longrightarrow \textit{distinct-mset}\ C \Longrightarrow \textit{conflicting}\ S = \textit{None} \Longrightarrow$
$\neg\textit{trail}\ S \models\!\textit{as}\ \textit{CNot}\ C \Longrightarrow$
$\textit{full}\ \textit{cdcl}_W\textit{-stgy}\ (\textit{add-init-cls}\ C\ S)\ T \Longrightarrow$
$\textit{incremental-cdcl}_W\ S\ T$

inductive *add-learned-clss* :: $'st \Rightarrow 'v\ \textit{clauses} \Rightarrow 'st \Rightarrow \textit{bool}$ **for** S :: $'st$ **where**
add-learned-clss-nil: *add-learned-clss* S {#} S |
add-learned-clss-plus:
add-learned-clss S A T $\Longrightarrow$ *add-learned-clss* S ({#x#} + A) (*add-learned-cls* x T)
declare *add-learned-clss.intros*[*intro*]

lemma *Ex-add-learned-clss*:
$\exists\, T.$ *add-learned-clss* S A T
⟨*proof*⟩

lemma *add-learned-clss-trail*:
assumes *add-learned-clss* S U T **and** *no-dup* (*trail* S)
shows *trail* T = *trail* S
⟨*proof*⟩

lemma *add-learned-clss-learned-clss*:
assumes *add-learned-clss* S U T **and** *no-dup* (*trail* S)
shows *learned-clss* T = U + *learned-clss* S
⟨*proof*⟩

lemma *add-learned-clss-init-clss*:
assumes *add-learned-clss* S U T **and** *no-dup* (*trail* S)
shows *init-clss* T = *init-clss* S
⟨*proof*⟩

lemma *add-learned-clss-conflicting*:
assumes *add-learned-clss* S U T **and** *no-dup* (*trail* S)
shows *conflicting* T = *conflicting* S
⟨*proof*⟩

lemma *add-learned-clss-backtrack-lvl*:
assumes *add-learned-clss* S U T **and** *no-dup* (*trail* S)
shows *backtrack-lvl* T = *backtrack-lvl* S
⟨*proof*⟩

lemma *add-learned-clss-init-state-mempty*[*dest*!]:
add-learned-clss (*init-state* N) {#} $T \Longrightarrow T$ = *init-state* N
⟨*proof*⟩

For multiset larger that 1 element, there is no way to know in which order the clauses are added. But contrary to a definition *fold-mset*, there is an element.

lemma *add-learned-clss-init-state-single*[*dest*!]:
add-learned-clss (*init-state* N) {#C#} $T \Longrightarrow T$ = *add-learned-cls* C (*init-state* N)
⟨*proof*⟩

thm *rtranclp-cdcl*$_W$*-stgy-no-smaller-confl-inv cdcl*$_W$*-stgy-final-state-conclusive*
lemma *cdcl*$_W$*-all-struct-inv-add-new-clause-and-update-cdcl*$_W$*-all-struct-inv*:
assumes
inv-T: *cdcl*$_W$*-all-struct-inv* T **and**
tr-T-N[*simp*]: *trail* T ⊨*asm* N **and**
tr-C[*simp*]: *trail* T ⊨*as CNot* C **and**
[*simp*]: *distinct-mset* C
shows *cdcl*$_W$*-all-struct-inv* (*add-new-clause-and-update* C T) (**is** *cdcl*$_W$*-all-struct-inv* ?T')
⟨*proof*⟩

lemma *cdcl*$_W$*-all-struct-inv-add-new-clause-and-update-cdcl*$_W$*-stgy-inv*:
assumes
inv-s: *cdcl*$_W$*-stgy-invariant* T **and**
inv: *cdcl*$_W$*-all-struct-inv* T **and**
tr-T-N[*simp*]: *trail* T ⊨*asm* N **and**
tr-C[*simp*]: *trail* T ⊨*as CNot* C **and**
[*simp*]: *distinct-mset* C
shows *cdcl*$_W$*-stgy-invariant* (*add-new-clause-and-update* C T) (**is** *cdcl*$_W$*-stgy-invariant* ?T')
⟨*proof*⟩

lemma *full-cdcl*$_W$*-stgy-inv-normal-form*:
assumes
full: *full cdcl*$_W$*-stgy* S T **and**
inv-s: *cdcl*$_W$*-stgy-invariant* S **and**
inv: *cdcl*$_W$*-all-struct-inv* S
shows *conflicting* T = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss* S))
∨ *conflicting* T = *None* ∧ *trail* T ⊨*asm init-clss* S ∧ *satisfiable* (*set-mset* (*init-clss* S))
⟨*proof*⟩

lemma *incremental-cdcl*$_W$*-inv*:
assumes
inc: *incremental-cdcl*$_W$ S T **and**
inv: *cdcl*$_W$*-all-struct-inv* S **and**
s-inv: *cdcl*$_W$*-stgy-invariant* S
shows
cdcl$_W$*-all-struct-inv* T **and**
cdcl$_W$*-stgy-invariant* T
⟨*proof*⟩

lemma *rtranclp-incremental-cdcl*$_W$*-inv*:
assumes
inc: *incremental-cdcl*$_W$** S T **and**
inv: *cdcl*$_W$*-all-struct-inv* S **and**
s-inv: *cdcl*$_W$*-stgy-invariant* S

shows
$cdcl_W$-all-struct-inv T **and**
$cdcl_W$-stgy-invariant T
⟨*proof*⟩

lemma *incremental-conclusive-state*:
assumes
inc: *incremental-$cdcl_W$ S T* **and**
inv: *$cdcl_W$-all-struct-inv S* **and**
s-inv: *$cdcl_W$-stgy-invariant S*
shows *conflicting T = Some {#} ∧ unsatisfiable (set-mset (init-clss T))*
∨ conflicting T = None ∧ trail T ⊨asm init-clss T ∧ satisfiable (set-mset (init-clss T))
⟨*proof*⟩

lemma *tranclp-incremental-correct*:
assumes
inc: *incremental-$cdcl_W{}^{++}$ S T* **and**
inv: *$cdcl_W$-all-struct-inv S* **and**
s-inv: *$cdcl_W$-stgy-invariant S*
shows *conflicting T = Some {#} ∧ unsatisfiable (set-mset (init-clss T))*
∨ conflicting T = None ∧ trail T ⊨asm init-clss T ∧ satisfiable (set-mset (init-clss T))
⟨*proof*⟩

lemma *blocked-induction-with-decided*:
assumes
n-d: *no-dup (L # M)* **and**
nil: *P []* **and**
append: $\bigwedge$*M L M′. P M ⟹ is-decided L ⟹ ∀m ∈ set M′. ¬is-decided m ⟹ no-dup (L # M′ @ M) ⟹*
P (L # M′ @ M) **and**
L: *is-decided L*
shows
P (L # M)
⟨*proof*⟩

lemma *trail-bloc-induction*:
assumes
n-d: *no-dup M* **and**
nil: *P []* **and**
append: $\bigwedge$*M L M′. P M ⟹ is-decided L ⟹ ∀m ∈ set M′. ¬is-decided m ⟹ no-dup (L # M′ @ M) ⟹*
P (L # M′ @ M) **and**
append-nm: $\bigwedge$*M′ M″. P M′ ⟹ M = M″ @ M′ ⟹ ∀m∈set M″. ¬is-decided m ⟹ P M*
shows
P M
⟨*proof*⟩

inductive *Tcons :: ('v, nat, 'v clause) ann-literals ⇒('v, nat, 'v clause) ann-literals ⇒ bool*
for *M :: ('v, nat, 'v clause) ann-literals* **where**
Tcons M [] |
Tcons M M′ ⟹ M = M″ @ M′ ⟹ (∀m ∈ set M″. ¬is-decided m) ⟹ Tcons M (M″ @ M′) |
Tcons M M′ ⟹ is-decided L ⟹ M = M‴ @ L # M″ @ M′ ⟹ (∀m ∈ set M″. ¬is-decided m) ⟹
Tcons M (L # M″ @ M′)

lemma *Tcons-same-end*: *Tcons M M′ ⟹ ∃M″. M = M″ @ M′*

⟨*proof*⟩

end

end

9 2-Watched-Literal

theory *CDCL-Two-Watched-Literals*
imports *CDCL-WNOT*
begin

9.1 Datastructure and Access Functions

Only the 2-watched literals have to be verified here: the backtrack level and the trail that appear in the state are not related to the 2-watched algoritm.

datatype *'v twl-clause* =
  TWL-Clause (*watched*: *'v*) (*unwatched*: *'v*)

abbreviation *raw-clause* :: *'v clause twl-clause* ⇒ *'v clause* **where**
  raw-clause C ≡ *watched C* + *unwatched C*

datatype (*'a*, *'b*, *'c*, *'d*) *twl-state* =
  TWL-State (*trail*: *'a list*) (*init-clss*: *'b*)
    (*learned-clss*: *'b*) (*backtrack-lvl*: *'c*)
    (*conflicting*: *'d option*)

type-synonym (*'v*, *'lvl*, *'mark*) *twl-state-abs* =
  ((*'v*, *'lvl*, *'mark*) *ann-literal*, *'v clause twl-clause multiset*, *'lvl*, *'v clause*) *twl-state*

abbreviation *raw-init-clss* **where**
  raw-init-clss S ≡ *image-mset raw-clause* (*init-clss S*)

abbreviation *raw-learned-clss* **where**
  raw-learned-clss S ≡ *image-mset raw-clause* (*learned-clss S*)

abbreviation *clauses* **where**
  clauses S ≡ *init-clss S* + *learned-clss S*

abbreviation *raw-clauses* **where**
  raw-clauses S ≡ *image-mset raw-clause* (*clauses S*)

definition
  candidates-propagate :: (*'v*, *'lvl*, *'mark*) *twl-state-abs* ⇒ (*'v literal* × *'v clause*) *set*
where
  candidates-propagate S =
   {(*L*, *raw-clause C*) | *L C*.
    C ∈# *clauses S* ∧ *watched C* − *mset-set* (*uminus* ‘ *lits-of* (*trail S*)) = {#*L*#} ∧
    undefined-lit (*trail S*) *L*}

definition *candidates-conflict* :: (*'v*, *'lvl*, *'mark*) *twl-state-abs* ⇒ *'v clause set* **where**
  candidates-conflict S =
   {*raw-clause C* | *C*. *C* ∈# *clauses S* ∧ *watched C* ⊆# *mset-set* (*uminus* ‘ *lits-of* (*trail S*))}

primrec (*nonexhaustive*) *index* :: $'a$ *list* $\Rightarrow 'a \Rightarrow$ *nat* **where**
index $(a \# l)\ c = ($*if* $a = c$ *then* 0 *else* $1+$*index* $l\ c)$

lemma *index-nth*:
$a \in set\ l \Longrightarrow l\ !\ (index\ l\ a) = a$
$\langle proof \rangle$

9.2 Invariants

We need the following property about updates: if there is a literal L with $-L$ in the trail, and L is not watched, then it stays unwatched; i.e., while updating with *rewatch* it does not get swap with a watched literal L' such that $-L'$ is in the trail.

primrec *watched-decided-most-recently* :: $('v, 'lvl, 'mark)$ *ann-literal list* $\Rightarrow 'v$ *clause twl-clause*
$\Rightarrow$ *bool*
where
watched-decided-most-recently M $(TWL\text{-}Clause\ W\ UW) \longleftrightarrow$
$(\forall L' \in\# W.\ \forall L \in\# UW.$
$-L' \in lits\text{-}of\ M \longrightarrow -L \in lits\text{-}of\ M \longrightarrow L \notin\# W \longrightarrow$
$index\ (map\ lit\text{-}of\ M)\ (-L') \leq index\ (map\ lit\text{-}of\ M)\ (-L))$

Here are the invariant strictly related to the 2-WL data structure.

primrec *wf-twl-cls* :: $('v, 'lvl, 'mark)$ *ann-literal list* $\Rightarrow 'v$ *clause twl-clause* $\Rightarrow$ *bool* **where**
wf-twl-cls M $(TWL\text{-}Clause\ W\ UW) \longleftrightarrow$
$distinct\text{-}mset\ W \wedge size\ W \leq 2 \wedge (size\ W < 2 \longrightarrow set\text{-}mset\ UW \subseteq set\text{-}mset\ W) \wedge$
$(\forall L \in\# W.\ -L \in lits\text{-}of\ M \longrightarrow (\forall L' \in\# UW.\ L' \notin\# W \longrightarrow -L' \in lits\text{-}of\ M)) \wedge$
watched-decided-most-recently M $(TWL\text{-}Clause\ W\ UW)$

lemma $-L \in lits\text{-}of\ M \Longrightarrow \{i.\ map\ lit\text{-}of\ M!i = -L\} \neq \{\}$
$\langle proof \rangle$

lemma *size-mset-2*: $size\ x1 = 2 \longleftrightarrow (\exists\ a\ b.\ x1 = \{\#a, b\#\})$
$\langle proof \rangle$

lemma *distinct-mset-size-2*: $distinct\text{-}mset\ \{\#a, b\#\} \longleftrightarrow a \neq b$
$\langle proof \rangle$

lemma *wf-twl-cls-annotation-indepnedant*:
assumes M: $map\ lit\text{-}of\ M = map\ lit\text{-}of\ M'$
shows *wf-twl-cls* M $(TWL\text{-}Clause\ W\ UW) \longleftrightarrow$ *wf-twl-cls* M' $(TWL\text{-}Clause\ W\ UW)$
$\langle proof \rangle$

lemma *wf-twl-cls-wf-twl-cls-tl*:
assumes *wf*: *wf-twl-cls* M C **and** *n-d*: *no-dup* M
shows *wf-twl-cls* $(tl\ M)$ C
$\langle proof \rangle$

definition *wf-twl-state* :: $('v, 'lvl, 'mark)$ *twl-state-abs* $\Rightarrow$ *bool* **where**
wf-twl-state $S \longleftrightarrow (\forall C \in\# clauses\ S.\ wf\text{-}twl\text{-}cls\ (trail\ S)\ C) \wedge no\text{-}dup\ (trail\ S)$

lemma *wf-candidates-propagate-sound*:
assumes *wf*: *wf-twl-state* S **and**
cand: $(L, C) \in candidates\text{-}propagate\ S$
shows $trail\ S \models as\ CNot\ (mset\text{-}set\ (set\text{-}mset\ C - \{L\})) \wedge undefined\text{-}lit\ (trail\ S)\ L$
$\langle proof \rangle$

lemma *wf-candidates-propagate-complete*:
  assumes *wf*: *wf-twl-state S* **and**
    c-mem: $C \in\# $ *raw-clauses S* **and**
    l-mem: $L \in\# C$ **and**
    unsat: *trail S* $\models$*as CNot* (*mset-set* (*set-mset* $C - \{L\}$)) **and**
    undef: *undefined-lit* (*trail S*) *L*
  shows $(L, C) \in$ *candidates-propagate S*
⟨*proof*⟩

lemma *wf-candidates-conflict-sound*:
  assumes *wf*: *wf-twl-state S* **and**
    cand: $C \in$ *candidates-conflict S*
  shows *trail S* $\models$*as CNot C* $\wedge$ $C \in\#$ *image-mset raw-clause* (*clauses S*)
⟨*proof*⟩

lemma *wf-candidates-conflict-complete*:
  assumes *wf*: *wf-twl-state S* **and**
    c-mem: $C \in\#$ *raw-clauses S* **and**
    unsat: *trail S* $\models$*as CNot C*
  shows $C \in$ *candidates-conflict S*
⟨*proof*⟩

typedef *'v wf-twl* = {*S*::(*'v*, *nat*, *'v clause*) *twl-state-abs*. *wf-twl-state S*}
morphisms *rough-state-of-twl twl-of-rough-state*
⟨*proof*⟩

lemma [*code abstype*]:
  twl-of-rough-state (*rough-state-of-twl S*) = *S*
  ⟨*proof*⟩

lemma *wf-twl-state-rough-state-of-twl*[*simp*]: *wf-twl-state* (*rough-state-of-twl S*)
  ⟨*proof*⟩

abbreviation *candidates-conflict-twl* :: *'v wf-twl* ⇒ *'v literal multiset set* **where**
candidates-conflict-twl S ≡ *candidates-conflict* (*rough-state-of-twl S*)

abbreviation *candidates-propagate-twl* :: *'v wf-twl* ⇒ (*'v literal* × *'v clause*) *set* **where**
candidates-propagate-twl S ≡ *candidates-propagate* (*rough-state-of-twl S*)

abbreviation *trail-twl* :: *'a wf-twl* ⇒ (*'a*, *nat*, *'a literal multiset*) *ann-literal list* **where**
trail-twl S ≡ *trail* (*rough-state-of-twl S*)

abbreviation *clauses-twl* :: *'a wf-twl* ⇒ *'a literal multiset multiset* **where**
clauses-twl S ≡ *raw-clauses* (*rough-state-of-twl S*)

abbreviation *init-clss-twl* :: *'a wf-twl* ⇒ *'a literal multiset multiset* **where**
init-clss-twl S ≡ *raw-init-clss* (*rough-state-of-twl S*)

abbreviation *learned-clss-twl* :: *'a wf-twl* ⇒ *'a literal multiset multiset* **where**
learned-clss-twl S ≡ *raw-learned-clss* (*rough-state-of-twl S*)

abbreviation *backtrack-lvl-twl* **where**
backtrack-lvl-twl S ≡ *backtrack-lvl* (*rough-state-of-twl S*)

abbreviation *conflicting-twl* **where**

conflicting-twl $S \equiv$ *conflicting* (*rough-state-of-twl* S)

lemma *wf-candidates-twl-conflict-complete*:
assumes
c-mem: $C \in\#$ *clauses-twl* S **and**
unsat: *trail-twl* $S \models as$ *CNot* C
shows $C \in$ *candidates-conflict-twl* S
⟨*proof*⟩

abbreviation *update-backtrack-lvl* **where**
update-backtrack-lvl k $S \equiv$
TWL-State (*trail* S) (*init-clss* S) (*learned-clss* S) k (*conflicting* S)

abbreviation *update-conflicting* **where**
update-conflicting C $S \equiv$ *TWL-State* (*trail* S) (*init-clss* S) (*learned-clss* S) (*backtrack-lvl* S) C

9.3 Abstract 2-WL

definition *tl-trail* **where**
tl-trail $S =$
TWL-State (*tl* (*trail* S)) (*init-clss* S) (*learned-clss* S) (*backtrack-lvl* S) (*conflicting* S)

locale *abstract-twl* $=$
fixes
watch :: ($'v$, *nat*, $'v$ *clause*) *twl-state-abs* $\Rightarrow$ $'v$ *clause* $\Rightarrow$ $'v$ *clause twl-clause* **and**
rewatch :: ($'v$, *nat*, $'v$ *literal multiset*) *ann-literal* $\Rightarrow$ ($'v$, *nat*, $'v$ *clause*) *twl-state-abs* $\Rightarrow$
$'v$ *clause twl-clause* $\Rightarrow$ $'v$ *clause twl-clause* **and**
linearize :: $'v$ *clauses* $\Rightarrow$ $'v$ *clause list* **and**
restart-learned :: ($'v$, *nat*, $'v$ *clause*) *twl-state-abs* $\Rightarrow$ $'v$ *clause twl-clause multiset*
assumes
clause-watch: *no-dup* (*trail* S) $\Longrightarrow$ *raw-clause* (*watch* S C) $= C$ **and**
wf-watch: *no-dup* (*trail* S) $\Longrightarrow$ *wf-twl-cls* (*trail* S) (*watch* S C) **and**
clause-rewatch: *raw-clause* (*rewatch* L S C') $=$ *raw-clause* C' **and**
wf-rewatch:
no-dup (*trail* S) $\Longrightarrow$ *undefined-lit* (*trail* S) (*lit-of* L) $\Longrightarrow$ *wf-twl-cls* (*trail* S) C' $\Longrightarrow$
wf-twl-cls (L # *trail* S) (*rewatch* L S C')
and
linearize: *mset* (*linearize* N) $= N$ **and**
restart-learned: *restart-learned* $S \subseteq\#$ *learned-clss* S
begin

lemma *linearize-mempty*[*simp*]: *linearize* $\{\#\} = []$
⟨*proof*⟩

definition
cons-trail :: ($'v$, *nat*, $'v$ *clause*) *ann-literal* $\Rightarrow$ ($'v$, *nat*, $'v$ *clause*) *twl-state-abs* $\Rightarrow$
($'v$, *nat*, $'v$ *clause*) *twl-state-abs*
where
cons-trail L $S =$
TWL-State (L # *trail* S) (*image-mset* (*rewatch* L S) (*init-clss* S))
(*image-mset* (*rewatch* L S) (*learned-clss* S)) (*backtrack-lvl* S) (*conflicting* S)

definition
add-init-cls :: $'v$ *clause* $\Rightarrow$ ($'v$, *nat*, $'v$ *clause*) *twl-state-abs* $\Rightarrow$
($'v$, *nat*, $'v$ *clause*) *twl-state-abs*
where

add-init-cls C S =
TWL-State (*trail S*) ({#*watch S C*#} + *init-clss S*) (*learned-clss S*) (*backtrack-lvl S*)
(*conflicting S*)

definition
add-learned-cls :: $'v$ *clause* ⇒ ($'v$, *nat*, $'v$ *clause*) *twl-state-abs* ⇒
($'v$, *nat*, $'v$ *clause*) *twl-state-abs*
where
add-learned-cls C S =
TWL-State (*trail S*) (*init-clss S*) ({#*watch S C*#} + *learned-clss S*) (*backtrack-lvl S*)
(*conflicting S*)

definition
remove-cls :: $'v$ *clause* ⇒ ($'v$, *nat*, $'v$ *clause*) *twl-state-abs* ⇒
($'v$, *nat*, $'v$ *clause*) *twl-state-abs*
where
remove-cls C S =
TWL-State (*trail S*) (*filter-mset* (λ*D*. *raw-clause D* ≠ *C*) (*init-clss S*))
(*filter-mset* (λ*D*. *raw-clause D* ≠ *C*) (*learned-clss S*)) (*backtrack-lvl S*)
(*conflicting S*)

definition *init-state* :: $'v$ *clauses* ⇒ ($'v$, *nat*, $'v$ *clause*) *twl-state-abs* **where**
init-state N = *fold add-init-cls* (*linearize N*) (*TWL-State* [] {#} {#} *0 None*)

lemma *unchanged-fold-add-init-cls*:
trail (*fold add-init-cls Cs* (*TWL-State M N U k C*)) = *M*
learned-clss (*fold add-init-cls Cs* (*TWL-State M N U k C*)) = *U*
backtrack-lvl (*fold add-init-cls Cs* (*TWL-State M N U k C*)) = *k*
conflicting (*fold add-init-cls Cs* (*TWL-State M N U k C*)) = *C*
⟨*proof*⟩

lemma *unchanged-init-state*[*simp*]:
trail (*init-state N*) = []
learned-clss (*init-state N*) = {#}
backtrack-lvl (*init-state N*) = *0*
conflicting (*init-state N*) = *None*
⟨*proof*⟩

lemma *clauses-init-fold-add-init*:
no-dup M ⟹
image-mset raw-clause (*init-clss* (*fold add-init-cls Cs* (*TWL-State M N U k C*))) =
mset Cs + *image-mset raw-clause N*
⟨*proof*⟩

lemma *init-clss-init-state*[*simp*]: *image-mset raw-clause* (*init-clss* (*init-state N*)) = *N*
⟨*proof*⟩

definition *restart′* **where**
restart′ S = *TWL-State* [] (*init-clss S*) (*restart-learned S*) *0 None*
end

9.4 Instanciation of the previous locale

definition *watch-nat* :: (*nat*, *nat*, *nat clause*) *twl-state-abs* ⇒ *nat clause* ⇒
nat clause twl-clause **where**
watch-nat S C =

(*let*
  C' = *remdups* (*sorted-list-of-set* (*set-mset* C));
  negation-not-assigned = *filter* ($\lambda L.\ -L \notin$ *lits-of* (*trail* S)) C';
  negation-assigned-sorted-by-trail = *filter* ($\lambda L.\ L \in\#\ C$) (*map* ($\lambda L.\ -$*lit-of* L) (*trail* S));
  W = *take* 2 (*negation-not-assigned* @ *negation-assigned-sorted-by-trail*);
  UW = *sorted-list-of-multiset* (C − *mset* W)
in *TWL-Clause* (*mset* W) (*mset* UW))

lemma *list-cases2*:
  fixes l :: $'a$ *list*
  assumes
    $l = [] \Longrightarrow P$ **and**
    $\bigwedge x.\ l = [x] \Longrightarrow P$ **and**
    $\bigwedge x\ y\ xs.\ l = x \# y \# xs \Longrightarrow P$
  shows P
  ⟨*proof*⟩

lemma *filter-in-list-prop-verifiedD*:
  assumes $[L \leftarrow P\ .\ Q\ L] = l$
  shows $\forall x \in$ *set* $l.\ x \in$ *set* $P \wedge Q\ x$
  ⟨*proof*⟩

lemma *no-dup-filter-diff*:
  assumes *n-d*: *no-dup* M **and** H: $[L \leftarrow$ *map* ($\lambda L.\ -$ *lit-of* L) $M.\ L \in\#\ C] = l$
  shows *distinct* l
  ⟨*proof*⟩

lemma *watch-nat-lists-disjointD*:
  assumes
    l: $[L \leftarrow$ *remdups* (*sorted-list-of-set* (*set-mset* C)) . $-\ L \notin$ *lits-of* (*trail* S)$] = l$ **and**
    l': $[L \leftarrow$ *map* ($\lambda L.\ -$ *lit-of* L) (*trail* S) . $L \in\#\ C] = l'$
  shows $\forall x \in$ *set* $l.\ \forall y \in$ *set* $l'.\ x \neq y$
  ⟨*proof*⟩

lemma *watch-nat-list-cases-witness*[*consumes 2*, *case-names nil-nil nil-single nil-other single-nil single-other other*]:
  fixes
    C :: $'v$ *literal multiset* **and**
    C' :: $'v$ *literal list* **and**
    S :: (($'v$, $'b$, $'c$) *ann-literal*, $'d$, $'e$, $'f$) *twl-state*
  defines
    $xs \equiv [L \leftarrow$ *remdups* $C'.\ -\ L \notin$ *lits-of* (*trail* S)$]$ **and**
    $ys \equiv [L \leftarrow$ *map* ($\lambda L.\ -$ *lit-of* L) (*trail* S) . $L \in\#\ C]$
  assumes
    n-d: *no-dup* (*trail* S) **and**
    C': *set* C' = *set-mset* C **and**
    nil-nil: $xs = [] \Longrightarrow ys = [] \Longrightarrow P$ **and**
    nil-single:
      $\bigwedge a.\ xs = [] \Longrightarrow ys = [a] \Longrightarrow\ a \in\#\ C \Longrightarrow P$ **and**
    nil-other: $\bigwedge a\ b\ ys'.\ xs = [] \Longrightarrow ys = a \# b \# ys' \Longrightarrow a \neq b \Longrightarrow P$ **and**
    single-nil: $\bigwedge a.\ xs = [a] \Longrightarrow ys = [] \Longrightarrow P$ **and**
    single-other: $\bigwedge a\ b\ ys'.\ xs = [a] \Longrightarrow ys = b \# ys' \Longrightarrow a \neq b \Longrightarrow P$ **and**
    other: $\bigwedge a\ b\ xs'.\ xs = a \# b \# xs' \Longrightarrow a \neq b \Longrightarrow P$
  shows P

⟨*proof*⟩

lemma *watch-nat-list-cases* [*consumes 1*, *case-names nil-nil nil-single nil-other single-nil single-other other*]:
fixes
C :: $'v$::*linorder literal multiset* **and**
S :: (($'v$, $'b$, $'c$) *ann-literal*, $'d$, $'e$, $'f$) *twl-state*
defines
$xs \equiv [L\leftarrow remdups\ (sorted\text{-}list\text{-}of\text{-}set\ (set\text{-}mset\ C))\ .\ -\ L \notin lits\text{-}of\ (trail\ S)]$ **and**
$ys \equiv [L\leftarrow map\ (\lambda L.\ -\ lit\text{-}of\ L)\ (trail\ S)\ .\ L \in\#\ C]$
assumes
n-d: *no-dup* (*trail S*) **and**
nil-nil: $xs = [] \Longrightarrow ys = [] \Longrightarrow P$ **and**
nil-single:
$\bigwedge a.\ xs = [] \Longrightarrow ys = [a] \Longrightarrow\ a \in\#\ C \Longrightarrow P$ **and**
nil-other: $\bigwedge a\ b\ ys'.\ xs = [] \Longrightarrow ys = a\ \#\ b\ \#\ ys' \Longrightarrow a \neq b \Longrightarrow P$ **and**
single-nil: $\bigwedge a.\ xs = [a] \Longrightarrow ys = [] \Longrightarrow P$ **and**
single-other: $\bigwedge a\ b\ ys'.\ xs = [a] \Longrightarrow ys = b\ \#\ ys' \Longrightarrow a \neq b \Longrightarrow P$ **and**
other: $\bigwedge a\ b\ xs'.\ xs = a\ \#\ b\ \#\ xs' \Longrightarrow a \neq b \Longrightarrow P$
shows P
⟨*proof*⟩

lemma *watch-nat-lists-set-union-witness*:
fixes
C :: $'v$ *literal multiset* **and**
C' :: $'v$ *literal list* **and**
S :: (($'v$, $'b$, $'c$) *ann-literal*, $'d$, $'e$, $'f$) *twl-state*
defines
$xs \equiv [L\leftarrow remdups\ C'.\ -\ L \notin lits\text{-}of\ (trail\ S)]$ **and**
$ys \equiv [L\leftarrow map\ (\lambda L.\ -\ lit\text{-}of\ L)\ (trail\ S)\ .\ L \in\#\ C]$
assumes *n-d*: *no-dup* (*trail S*) **and** C': $set\ C' = set\text{-}mset\ C$
shows $set\text{-}mset\ C = set\ xs \cup set\ ys$
⟨*proof*⟩

lemma *watch-nat-lists-set-union*:
fixes
C :: $'v$::*linorder literal multiset* **and**
S :: (($'v$, $'b$, $'c$) *ann-literal*, $'d$, $'e$, $'f$) *twl-state*
defines
$xs \equiv [L\leftarrow remdups\ (sorted\text{-}list\text{-}of\text{-}set\ (set\text{-}mset\ C)).\ -\ L \notin lits\text{-}of\ (trail\ S)]$ **and**
$ys \equiv [L\leftarrow map\ (\lambda L.\ -\ lit\text{-}of\ L)\ (trail\ S)\ .\ L \in\#\ C]$
assumes *n-d*: *no-dup* (*trail S*)
shows $set\text{-}mset\ C = set\ xs \cup set\ ys$
⟨*proof*⟩

lemma *mset-intersection-inclusion*: $A + (B - A) = B \longleftrightarrow A \subseteq\#\ B$
⟨*proof*⟩

lemma *clause-watch-nat*:
assumes *no-dup* (*trail S*)
shows *raw-clause* (*watch-nat S C*) = C
⟨*proof*⟩

lemma *set-mset-is-single-in-mset-is-single*:

$set\text{-}mset\ C = \{a\} \Longrightarrow x \in\#\ C \Longrightarrow x = a$
⟨proof⟩

lemma *index-uminus-index-map-uminus*:
$-a \in set\ L \Longrightarrow index\ L\ (-a) = index\ (map\ uminus\ L)\ (a{::}'a\ literal)$
⟨proof⟩

lemma *index-filter*:
$a \in set\ L \Longrightarrow b \in set\ L \Longrightarrow P\ a \Longrightarrow P\ b \Longrightarrow$
$index\ L\ a \le index\ L\ b \longleftrightarrow index\ (filter\ P\ L)\ a \le index\ (filter\ P\ L)\ b$
⟨proof⟩

lemma *wf-watch-witness*:
fixes $C :: 'a\ literal\ multiset$ **and** $C' :: 'a\ literal\ list$ **and**
$S :: (('a, 'b, 'c)\ ann\text{-}literal, 'd, 'e, 'f)\ twl\text{-}state$
defines
ass: $negation\text{-}not\text{-}assigned \equiv filter\ (\lambda L.\ -L \notin lits\text{-}of\ (trail\ S))\ (remdups\ C')$ **and**
tr: $negation\text{-}assigned\text{-}sorted\text{-}by\text{-}trail \equiv filter\ (\lambda L.\ L \in\#\ C)\ (map\ (\lambda L.\ -lit\text{-}of\ L)\ (trail\ S))$
defines
W: $W \equiv take\ 2\ (negation\text{-}not\text{-}assigned\ @\ negation\text{-}assigned\text{-}sorted\text{-}by\text{-}trail)$
assumes
$n\text{-}d[simp]$: $no\text{-}dup\ (trail\ S)$ **and**
C': $set\ C' = set\text{-}mset\ C$
shows $wf\text{-}twl\text{-}cls\ (trail\ S)\ (TWL\text{-}Clause\ (mset\ W)\ (C - mset\ W))$
⟨proof⟩

lemma *wf-watch-nat*: $no\text{-}dup\ (trail\ S) \Longrightarrow wf\text{-}twl\text{-}cls\ (trail\ S)\ (watch\text{-}nat\ S\ C)$
⟨proof⟩

definition
$rewatch\text{-}nat ::$
$(nat, nat, nat\ literal\ multiset)\ ann\text{-}literal \Rightarrow (nat, nat, nat\ clause)\ twl\text{-}state\text{-}abs \Rightarrow$
$nat\ clause\ twl\text{-}clause \Rightarrow nat\ clause\ twl\text{-}clause$
where
$rewatch\text{-}nat\ L\ S\ C =$
$(if\ -\ lit\text{-}of\ L \in\#\ watched\ C\ then$
$case\ filter\ (\lambda L'.\ L' \notin\#\ watched\ C \wedge -\ L' \notin lits\text{-}of\ (L\ \#\ trail\ S))$
$(sorted\text{-}list\text{-}of\text{-}multiset\ (unwatched\ C))\ of$
$[] \Rightarrow C$
$|\ L'\ \#\ \text{-} \Rightarrow$
$TWL\text{-}Clause\ (watched\ C - \{\#-\ lit\text{-}of\ L\#\} + \{\#L'\#\})\ (unwatched\ C - \{\#L'\#\} + \{\#-\ lit\text{-}of\ L\#\})$
$else$
$C)$

lemma *clause-rewatch-witness*:
fixes $UW :: 'a\ literal\ list$ **and**
$S :: (('a, 'b, 'c)\ ann\text{-}literal, 'd, 'e, 'f)\ twl\text{-}state$ **and**
$L :: ('a, 'b, 'c)\ ann\text{-}literal$ **and** $C :: 'a\ literal\ multiset\ twl\text{-}clause$
defines $C' \equiv (if\ -\ lit\text{-}of\ L \in\#\ watched\ C\ then$
$case\ filter\ (\lambda L'.\ L' \notin\#\ watched\ C \wedge -\ L' \notin lits\text{-}of\ (L\ \#\ trail\ S))\ UW\ of$
$[] \Rightarrow C$
$|\ L'\ \#\ \text{-} \Rightarrow$
$TWL\text{-}Clause\ (watched\ C - \{\#-\ lit\text{-}of\ L\#\} + \{\#L'\#\})\ (unwatched\ C - \{\#L'\#\} + \{\#-\ lit\text{-}of\ L\#\})$

else
C)
assumes
UW: *set UW* = *set-mset* (*unwatched C*)
shows *raw-clause C′* = *raw-clause C*
⟨*proof*⟩

lemma *clause-rewatch-nat*: *raw-clause* (*rewatch-nat L S C*) = *raw-clause C*
⟨*proof*⟩

lemma *filter-sorted-list-of-multiset-Nil*:
[*x* ← *sorted-list-of-multiset M*. *p x*] = [] ⟷ (∀ *x* ∈# *M*. ¬ *p x*)
⟨*proof*⟩

lemma *filter-sorted-list-of-multiset-ConsD*:
[*x* ← *sorted-list-of-multiset M*. *p x*] = *x* # *xs* ⟹ *p x*
⟨*proof*⟩

lemma *mset-minus-single-eq-mempty*:
a − {#*b*#} = {#} ⟷ *a* = {#*b*#} ∨ *a* = {#}
⟨*proof*⟩

lemma *size-mset-le-2-cases*:
assumes *size W* ≤ *2*
shows *W* = {#} ∨ (∃ *a*. *W* = {#*a*#}) ∨ (∃ *a b*. *W* = {#*a*,*b*#})
⟨*proof*⟩

lemma *filter-sorted-list-of-multiset-eqD*:
assumes [*x* ← *sorted-list-of-multiset A*. *p x*] = *x* # *xs* (**is** *?comp* = -)
shows *x* ∈# *A*
⟨*proof*⟩

lemma *clause-rewatch-witness′*:
fixes *UWC* :: *′a literal list* **and**
S :: ((*′a*, *′b*, *′c*) *ann-literal*, *′d*, *′e*, *′f*) *twl-state* **and**
L :: (*′a*, *′b*, *′c*) *ann-literal* **and** *C* :: *′a literal multiset twl-clause*
defines *C′* ≡ (*if* − *lit-of L* ∈# *watched C then*
case filter (λ*L′*. *L′* ∉# *watched C* ∧ − *L′* ∉ *lits-of* (*L* # *trail S*)) *UWC of*
[] ⇒ *C*
| *L′* # - ⇒
TWL-Clause (*watched C* − {#− *lit-of L*#} + {#*L′*#}) (*unwatched C* − {#*L′*#} + {#− *lit-of L*#})
else
C)
assumes
UWC: *set UWC* = *set-mset* (*unwatched C*) **and**
wf: *wf-twl-cls* (*trail S*) *C* **and**
n-d: *no-dup* (*trail S*) **and**
undef: *undefined-lit* (*trail S*) (*lit-of L*)
shows *wf-twl-cls* (*L* # *trail S*) *C′*
⟨*proof*⟩

lemma *wf-rewatch-nat′*:
assumes
wf: *wf-twl-cls* (*trail S*) *C* **and**

n-d: *no-dup* (*trail S*) **and**
undef: *undefined-lit* (*trail S*) (*lit-of L*)
shows *wf-twl-cls* (*L* # *trail S*) (*rewatch-nat L S C*)
⟨*proof*⟩

interpretation *twl*: *abstract-twl watch-nat rewatch-nat sorted-list-of-multiset learned-clss*
⟨*proof*⟩

9.5 Interpretation for $cdcl_W.cdcl_W$

context *abstract-twl*
begin

9.5.1 Direct Interpretation

interpretation *rough-cdcl*: $state_W$ *trail raw-init-clss raw-learned-clss backtrack-lvl conflicting cons-trail tl-trail add-init-cls add-learned-cls remove-cls update-backtrack-lvl update-conflicting init-state restart′*
⟨*proof*⟩

interpretation *rough-cdcl*: $cdcl_W$ *trail raw-init-clss raw-learned-clss backtrack-lvl conflicting cons-trail tl-trail add-init-cls add-learned-cls remove-cls update-backtrack-lvl update-conflicting init-state restart′*
⟨*proof*⟩

9.5.2 Opaque Type with Invariant

declare *rough-cdcl.state-simp*[*simp del*]

definition *cons-trail-twl* :: (′*v*, *nat*, ′*v literal multiset*) *ann-literal* ⇒ ′*v wf-twl* ⇒ ′*v wf-twl*
where
cons-trail-twl L S ≡ *twl-of-rough-state* (*cons-trail L* (*rough-state-of-twl S*))

lemma *wf-twl-state-cons-trail*:
undefined-lit (*trail S*) (*lit-of L*) ⟹ *wf-twl-state S* ⟹ *wf-twl-state* (*cons-trail L S*)
⟨*proof*⟩

lemma *rough-state-of-twl-cons-trail*:
undefined-lit (*trail-twl S*) (*lit-of L*) ⟹
rough-state-of-twl (*cons-trail-twl L S*) = *cons-trail L* (*rough-state-of-twl S*)
⟨*proof*⟩

abbreviation *add-init-cls-twl* **where**
add-init-cls-twl C S ≡ *twl-of-rough-state* (*add-init-cls C* (*rough-state-of-twl S*))

lemma *wf-twl-add-init-cls*: *wf-twl-state S* ⟹ *wf-twl-state* (*add-init-cls L S*)
⟨*proof*⟩

lemma *rough-state-of-twl-add-init-cls*:
rough-state-of-twl (*add-init-cls-twl L S*) = *add-init-cls L* (*rough-state-of-twl S*)
⟨*proof*⟩

abbreviation *add-learned-cls-twl* **where**
add-learned-cls-twl C S ≡ *twl-of-rough-state* (*add-learned-cls C* (*rough-state-of-twl S*))

lemma *wf-twl-add-learned-cls*: *wf-twl-state* $S \Longrightarrow$ *wf-twl-state* (*add-learned-cls* L S)
⟨*proof*⟩

lemma *rough-state-of-twl-add-learned-cls*:
rough-state-of-twl (*add-learned-cls-twl* L S) = *add-learned-cls* L (*rough-state-of-twl* S)
⟨*proof*⟩

abbreviation *remove-cls-twl* **where**
remove-cls-twl C S ≡ *twl-of-rough-state* (*remove-cls* C (*rough-state-of-twl* S))

lemma *wf-twl-remove-cls*: *wf-twl-state* $S \Longrightarrow$ *wf-twl-state* (*remove-cls* L S)
⟨*proof*⟩

lemma *rough-state-of-twl-remove-cls*:
rough-state-of-twl (*remove-cls-twl* L S) = *remove-cls* L (*rough-state-of-twl* S)
⟨*proof*⟩

abbreviation *init-state-twl* **where**
init-state-twl N ≡ *twl-of-rough-state* (*init-state* N)

lemma *wf-twl-state-wf-twl-state-fold-add-init-cls*:
assumes *wf-twl-state* S
shows *wf-twl-state* (*fold add-init-cls* N S)
⟨*proof*⟩

lemma *wf-twl-state-epsilon-state*[*simp*]:
wf-twl-state (*TWL-State* [] {#} {#} *0 None*)
⟨*proof*⟩

lemma *wf-twl-init-state*: *wf-twl-state* (*init-state* N)
⟨*proof*⟩

lemma *rough-state-of-twl-init-state*:
rough-state-of-twl (*init-state-twl* N) = *init-state* N
⟨*proof*⟩

abbreviation *tl-trail-twl* **where**
tl-trail-twl S ≡ *twl-of-rough-state* (*tl-trail* (*rough-state-of-twl* S))

lemma *wf-twl-state-tl-trail*: *wf-twl-state* $S \Longrightarrow$ *wf-twl-state* (*tl-trail* S)
⟨*proof*⟩

lemma *rough-state-of-twl-tl-trail*:
rough-state-of-twl (*tl-trail-twl* S) = *tl-trail* (*rough-state-of-twl* S)
⟨*proof*⟩

abbreviation *update-backtrack-lvl-twl* **where**
update-backtrack-lvl-twl k S ≡ *twl-of-rough-state* (*update-backtrack-lvl* k (*rough-state-of-twl* S))

lemma *wf-twl-state-update-backtrack-lvl*:
wf-twl-state $S \Longrightarrow$ *wf-twl-state* (*update-backtrack-lvl* k S)
⟨*proof*⟩

lemma *rough-state-of-twl-update-backtrack-lvl*:

rough-state-of-twl (*update-backtrack-lvl-twl* k S) = *update-backtrack-lvl* k (*rough-state-of-twl* S)
⟨*proof*⟩

abbreviation *update-conflicting-twl* **where**
update-conflicting-twl k S ≡ *twl-of-rough-state* (*update-conflicting* k (*rough-state-of-twl* S))

lemma *wf-twl-state-update-conflicting*:
wf-twl-state S ⟹ *wf-twl-state* (*update-conflicting* k S)
⟨*proof*⟩

lemma *rough-state-of-twl-update-conflicting*:
rough-state-of-twl (*update-conflicting-twl* k S) = *update-conflicting* k (*rough-state-of-twl* S)
⟨*proof*⟩

abbreviation *raw-clauses-twl* **where**
raw-clauses-twl S ≡ *raw-clauses* (*rough-state-of-twl* S)

abbreviation *restart-twl* **where**
restart-twl S ≡ *twl-of-rough-state* (*restart'* (*rough-state-of-twl* S))

lemma *wf-wf-restart'*: *wf-twl-state* S ⟹ *wf-twl-state* (*restart'* S)
⟨*proof*⟩

lemma *rough-state-of-twl-restart-twl*:
rough-state-of-twl (*restart-twl* S) = *restart'* (*rough-state-of-twl* S)
⟨*proof*⟩

interpretation $cdcl_W$*-twl-NOT*: *dpll-state*
λS. *convert-trail-from-W* (*trail-twl* S)
raw-clauses-twl
λL S. *cons-trail-twl* (*convert-ann-literal-from-NOT* L) S
λS. *tl-trail-twl* S
λC S. *add-learned-cls-twl* C S
λC S. *remove-cls-twl* C S
⟨*proof*⟩

interpretation $cdcl_W$*-twl*: $state_W$
trail-twl
init-clss-twl
learned-clss-twl
backtrack-lvl-twl
conflicting-twl
cons-trail-twl
tl-trail-twl
add-init-cls-twl
add-learned-cls-twl
remove-cls-twl
update-backtrack-lvl-twl
update-conflicting-twl
init-state-twl
restart-twl
⟨*proof*⟩

interpretation $cdcl_W$*-twl*: $cdcl_W$
trail-twl
init-clss-twl
learned-clss-twl
backtrack-lvl-twl
conflicting-twl
cons-trail-twl
tl-trail-twl
add-init-cls-twl
add-learned-cls-twl
remove-cls-twl
update-backtrack-lvl-twl
update-conflicting-twl
init-state-twl
restart-twl
⟨*proof*⟩

sublocale $cdcl_W$
trail-twl
init-clss-twl
learned-clss-twl
backtrack-lvl-twl
conflicting-twl
cons-trail-twl
tl-trail-twl
add-init-cls-twl
add-learned-cls-twl
remove-cls-twl
update-backtrack-lvl-twl
update-conflicting-twl
init-state-twl
restart-twl
⟨*proof*⟩

abbreviation *state-eq-twl* (**infix** $\sim TWL$ *51*) **where**
state-eq-twl S S' $\equiv$ *rough-cdcl.state-eq* (*rough-state-of-twl S*) (*rough-state-of-twl S'*)
notation $cdcl_W$*-twl.state-eq* (**infix** $\sim$ *51*)
declare $cdcl_W$*-twl.state-simp*[*simp del*]
$cdcl_W$*-twl-NOT.state-simp*$_{NOT}$[*simp del*]

To avoid ambiguities:

no-notation *state-eq-twl* (**infix** $\sim$ *51*)

definition *propagate-twl* **where**
propagate-twl S S' $\longleftrightarrow$
($\exists$ *L C*. (*L*, *C*) $\in$ *candidates-propagate-twl S*
$\wedge$ *S'* $\sim$ *cons-trail-twl* (*Propagated L C*) *S*
$\wedge$ *conflicting-twl S* = *None*)

lemma *propagate-twl-iff-propagate*:
assumes *inv*: $cdcl_W$*-twl.*$cdcl_W$*-all-struct-inv S*
shows $cdcl_W$*-twl.propagate S T* $\longleftrightarrow$ *propagate-twl S T* (**is** *?P* $\longleftrightarrow$ *?T*)
⟨*proof*⟩
no-notation *CDCL-Two-Watched-Literals.twl.state-eq-twl* (**infix** $\sim TWL$ *51*)

definition *conflict-twl* **where**
conflict-twl S $S' \longleftrightarrow$
($\exists$ C. $C \in$ *candidates-conflict-twl* S
$\wedge$ $S' \sim$ *update-conflicting-twl* (*Some* C) S
$\wedge$ *conflicting-twl* S = *None*)

lemma *conflict-twl-iff-conflict*:
shows $cdcl_W$-*twl.conflict* S $T \longleftrightarrow$ *conflict-twl* S T (**is** *?C* $\longleftrightarrow$ *?T*)
⟨*proof*⟩

inductive $cdcl_W$-*twl* :: $'v$ *wf-twl* $\Rightarrow$ $'v$ *wf-twl* $\Rightarrow$ *bool* **for** S :: $'v$ *wf-twl* **where**
propagate: *propagate-twl* S $S' \Longrightarrow cdcl_W$-*twl* S S' |
conflict: *conflict-twl* S $S' \Longrightarrow cdcl_W$-*twl* S S' |
other: $cdcl_W$-*twl.*$cdcl_W$-*o* S $S' \Longrightarrow cdcl_W$-*twl* S S'|
rf: $cdcl_W$-*twl.*$cdcl_W$-*rf* S $S' \Longrightarrow cdcl_W$-*twl* S S'

lemma $cdcl_W$-*twl-iff-*$cdcl_W$:
assumes $cdcl_W$-*twl.*$cdcl_W$-*all-struct-inv* S
shows $cdcl_W$-*twl* S $T \longleftrightarrow cdcl_W$-*twl.*$cdcl_W$ S T
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$-*twl-all-struct-inv-inv*:
assumes $cdcl_W$-*twl*** S T **and** $cdcl_W$-*twl.*$cdcl_W$-*all-struct-inv* S
shows $cdcl_W$-*twl.*$cdcl_W$-*all-struct-inv* T
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$-*twl-iff-rtranclp-*$cdcl_W$:
assumes $cdcl_W$-*twl.*$cdcl_W$-*all-struct-inv* S
shows $cdcl_W$-*twl*** S $T \longleftrightarrow cdcl_W$-*twl.*$cdcl_W{}^{**}$ S T (**is** *?T* $\longleftrightarrow$ *?W*)
⟨*proof*⟩

interpretation $cdcl_{NOT}$-*twl*: *backjumping-ops*
λS. *convert-trail-from-W* (*trail-twl* S)
abstract-twl.raw-clauses-twl
λL (S:: $'v$ *wf-twl*).
cons-trail-twl
(*convert-ann-literal-from-NOT* L) (S:: $'v$ *wf-twl*)
tl-trail-twl
add-learned-cls-twl
remove-cls-twl
λC - - (S:: $'v$ *wf-twl*) -. $C \in$ *candidates-conflict-twl* S
⟨*proof*⟩

lemma *reduce-trail-to*$_{NOT}$*-skip-beginning-twl*:
assumes *trail-twl* S = *convert-trail-from-NOT* (F' @ F)
shows *trail-twl* ($cdcl_W$-*twl.reduce-trail-to*$_{NOT}$ F S) = *convert-trail-from-NOT* F
⟨*proof*⟩

lemma *reduce-trail-to*$_{NOT}$*-trail-tl-trail-twl-decomp*[*simp*]:
trail-twl S = *convert-trail-from-NOT* (F' @ *Decided* K () # F) $\Longrightarrow$
trail-twl ($cdcl_W$-*twl.reduce-trail-to*$_{NOT}$ F (*tl-trail-twl* S)) = *convert-trail-from-NOT* F
⟨*proof*⟩

lemma *trail-twl-reduce-trail-to*$_{NOT}$*-drop*:
trail-twl ($cdcl_W$-*twl.reduce-trail-to*$_{NOT}$ F S) =

(*if length* (*trail-twl S*) $\geq$ *length F*
then drop (*length* (*trail-twl S*) $-$ *length F*) (*trail-twl S*)
else [])
⟨*proof*⟩

interpretation *cdcl*$_{NOT}$*-twl*: *dpll-with-backjumping-ops*
λS. *convert-trail-from-W* (*trail-twl S*)
abstract-twl.raw-clauses-twl
λL S.
cons-trail-twl
(*convert-ann-literal-from-NOT L*) *S*
tl-trail-twl
add-learned-cls-twl
remove-cls-twl
λL S. *lit-of L* $\in$ *fst* ' *candidates-propagate-twl S*
λS. *no-dup* (*trail-twl S*)
λC - - *S* -. *C* $\in$ *candidates-conflict-twl S*
⟨*proof*⟩

interpretation *cdcl*$_{NOT}$*-twl*: *dpll-with-backjumping*
λS. *convert-trail-from-W* (*trail-twl S*)
abstract-twl.raw-clauses-twl
λL (*S*:: '*v wf-twl*).
cons-trail-twl
(*convert-ann-literal-from-NOT L*) (*S*:: '*v wf-twl*)
tl-trail-twl
add-learned-cls-twl
remove-cls-twl
λL S. *lit-of L* $\in$ *fst* ' *candidates-propagate-twl S*
λS. *no-dup* (*trail-twl S*)
λC - - (*S*:: '*v wf-twl*) -. *C* $\in$ *candidates-conflict-twl S*
⟨*proof*⟩
end

end

10 Implementation for 2 Watched-Literals

theory *CDCL-Two-Watched-Literals-Implementation*
imports *CDCL-Two-Watched-Literals DPLL-CDCL-W-Implementation*
begin

type-synonym '*v conc-twl-state* =
(('*v*, *nat*, '*v literal list*) *ann-literal*, '*v literal list twl-clause list*, *nat*, '*v literal list*)
twl-state

fun *convert* :: ('*a*, '*b*, '*c list*) *ann-literal* $\Rightarrow$ ('*a*, '*b*, '*c multiset*) *ann-literal* **where**
convert (*Propagated L C*) = *Propagated L* (*mset C*) |
convert (*Decided K i*) = *Decided K i*

abbreviation *convert-tr* :: ('*a*, '*b*, '*c list*) *ann-literals* $\Rightarrow$ ('*a*, '*b*, '*c multiset*) *ann-literals*
where
convert-tr $\equiv$ *map convert*

abbreviation *convertC* :: '*a literal list option* $\Rightarrow$ '*a clause option* **where**

convertC ≡ *map-option mset*

fun *raw-clause-l* :: *'v list twl-clause* ⇒ *'v multiset twl-clause* **where**
raw-clause-l (*TWL-Clause UW W*) = *TWL-Clause* (*mset W*) (*mset UW*)

abbreviation *convert-clss* :: *'v literal list twl-clause list* ⇒ *'v clause twl-clause multiset* **where**
convert-clss S ≡ *mset* (*map raw-clause-l S*)

fun *raw-state-of-conc* :: *'v conc-twl-state* ⇒ (*'v*, *nat*, *'v clause*) *twl-state-abs* **where**
raw-state-of-conc (*TWL-State M N U k C*) =
TWL-State (*convert-tr M*) (*convert-clss N*) (*convert-clss U*) *k* (*map-option mset C*)

lemma
raw-state-of-conc (*tl-trail S*) = *tl-trail* (*raw-state-of-conc S*)
⟨*proof*⟩

typedef *'v conv-twl-state* = {*S*:: *'v conc-twl-state*. *wf-twl-state* (*raw-state-of-conc S*)}
morphisms *list-twl-state-of cls-twl-state*
⟨*proof*⟩
term *list-twl-state-of*

definition *watch-list* :: *'v conv-twl-state* ⇒ *'v literal list* ⇒ *'v literal list twl-clause* **where**
watch-list S' C =
(*let*
M = *trail* (*list-twl-state-of S'*);
C' = *remdups C*;
negation-not-assigned = *filter* ($\lambda L.\ -L \notin$ *lits-of M*) *C'*;
negation-assigned-sorted-by-trail = *filter* ($\lambda L.\ L \in$ *set C*) (*map* ($\lambda L.\ -$*lit-of L*) *M*);
W = *take 2* (*negation-not-assigned* @ *negation-assigned-sorted-by-trail*);
UW = *foldl* ($\lambda a\ l.$ *remove1 l a*) *C W*
in TWL-Clause W UW)

lemma *wf-watch-nat*: *no-dup* (*trail* (*list-twl-state-of S*)) ⟹
wf-twl-cls (*trail* (*list-twl-state-of S*)) (*raw-clause-l* (*watch-list S C*))
⟨*proof*⟩

end