Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

April 16, 2016

Contents

0.1 Partial Clausal Logic

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

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

begin

0.1.1 Decided Literals

Definition

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

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

lemma *is-decided-ex-Decided*:
$is\text{-}decided\ L \Longrightarrow (\bigwedge K.\ L = Decided\ K \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

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

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

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

lemma *lits-of-l-empty*[*simp*]:

$lits\text{-}of\ \{\} = \{\}$
⟨*proof*⟩

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

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

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

abbreviation *unmark* **where**
$unmark \equiv (\lambda a.\ \{\#lit\text{-}of\ a\#\})$

abbreviation *unmark-s* **where**
$unmark\text{-}s\ M \equiv unmark\ `\ M$

abbreviation *unmark-l* **where**
$unmark\text{-}l\ M \equiv unmark\text{-}s\ (set\ M)$

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

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

Entailment

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

definition *true-annots* :: $('a, 'm)\ ann\text{-}lits \Rightarrow 'a\ clauses \Rightarrow bool$ (**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*:
$I \models as \ CC \longleftrightarrow lits\text{-}of\text{-}l \ I \models s \ CC$
⟨*proof*⟩

lemma *in-lit-of-true-annot*:
$a \in lits\text{-}of\text{-}l \ M \longleftrightarrow M \models a \ \{\#a\#\}$
⟨*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*:
$I \models s \ unmark\text{-}l \ MLs \Longrightarrow lits\text{-}of\text{-}l \ MLs \subseteq I$
⟨*proof*⟩

lemma *true-annot-true-clss-cls*:
$MLs \models a \ \psi \Longrightarrow set \ (map \ unmark \ MLs) \models p \ \psi$
⟨*proof*⟩

lemma *true-annots-true-clss-cls*:
$MLs \models as \ \psi \Longrightarrow set \ (map \ unmark \ MLs) \models ps \ \psi$
⟨*proof*⟩

lemma *true-annots-decided-true-cls*[*iff*]:
$map \ Decided \ M \models as \ N \longleftrightarrow set \ M \models s \ N$
⟨*proof*⟩

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

lemma *true-annots-true-clss-clss*:
$A \models as \ \Psi \Longrightarrow unmark\text{-}l \ A \models ps \ \Psi$
⟨*proof*⟩

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

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

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

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

Defined and undefined literals

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

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

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

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

lemma *defined-lit-rev*[*simp*]:
defined-lit $(rev\ M)\ L \longleftrightarrow$ *defined-lit* $M\ L$
⟨*proof*⟩

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

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

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

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

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

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

lemma *Decided-Propagated-in-iff-in-lits-of-l*:
defined-lit I L $\longleftrightarrow$ ($L \in$ *lits-of-l I* $\vee$ $-L \in$ *lits-of-l I*)
⟨*proof*⟩

lemma *consistent-add-undefined-lit-consistent*[*simp*]:
assumes
consistent-interp (*lits-of-l Ls*) **and**
undefined-lit Ls L
shows *consistent-interp* (*insert L* (*lits-of-l Ls*))
⟨*proof*⟩

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

0.1.2 Backtracking

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

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

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

lemma *backtrack-split-list-eq*[*simp*]:
fst (*backtrack-split l*) @ (*snd* (*backtrack-split l*)) = *l*
⟨*proof*⟩

lemma *backtrack-snd-empty-not-decided*:
backtrack-split M = (M'', []) $\Longrightarrow$ $\forall$ *l*$\in$*set M*. $\neg$ *is-decided l*
⟨*proof*⟩

lemma *backtrack-split-some-is-decided-then-snd-has-hd*:
$\exists$ *l*$\in$*set M*. *is-decided l* $\Longrightarrow$ $\exists\, M'\ L'\ M''$. *backtrack-split M* = (M'', L' # M')
⟨*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-split M = (*takeWhile* (*Not o is-decided*) *M*, *dropWhile* (*Not o is-decided*) *M*)
⟨*proof*⟩

0.1.3 Decomposition with respect to the First Decided Literals

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

Definition

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

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

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

Now we can prove several simple properties about the function.

lemma *get-all-ann-decomposition-never-empty*[*iff*]:
  get-all-ann-decomposition M = [] $\longleftrightarrow$ *False*
  ⟨*proof*⟩

lemma *get-all-ann-decomposition-never-empty-sym*[*iff*]:
  [] = *get-all-ann-decomposition M* $\longleftrightarrow$ *False*
  ⟨*proof*⟩

lemma *get-all-ann-decomposition-decomp*:
  hd (*get-all-ann-decomposition S*) = (*a*, *c*) $\Longrightarrow$ *S* = *c* @ *a*
⟨*proof*⟩

lemma *get-all-ann-decomposition-backtrack-split*:
  backtrack-split S = (*M*, *M'*) $\longleftrightarrow$ *hd* (*get-all-ann-decomposition S*) = (*M'*, *M*)
⟨*proof*⟩

lemma *get-all-ann-decomposition-Nil-backtrack-split-snd-Nil*:
  get-all-ann-decomposition S = [([], *A*)] $\Longrightarrow$ *snd* (*backtrack-split S*) = []
  ⟨*proof*⟩

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

lemma *get-all-ann-decomposition-length-1-fst-empty-or-length-1*:
  assumes *get-all-ann-decomposition M* = (*a*, *b*) # []
  shows *a* = [] $\vee$ (*length a* = *1* $\wedge$ *is-decided* (*hd a*) $\wedge$ *hd a* $\in$ *set M*)
  ⟨*proof*⟩

lemma *get-all-ann-decomposition-fst-empty-or-hd-in-M*:
  assumes *get-all-ann-decomposition M* = (*a*, *b*) # *l*
  shows *a* = [] $\vee$ (*is-decided* (*hd a*) $\wedge$ *hd a* $\in$ *set M*)
  ⟨*proof*⟩

lemma *get-all-ann-decomposition-snd-not-decided*:
assumes $(a, b) \in set$ (*get-all-ann-decomposition M*)
and $L \in set\ b$
shows $\neg$*is-decided L*
⟨*proof*⟩

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

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

lemma *in-get-all-ann-decomposition-in-get-all-ann-decomposition-prepend*:
$(a, b) \in set$ (*get-all-ann-decomposition* M') $\Longrightarrow$
$\exists b'. (a, b'$ @ $b) \in set$ (*get-all-ann-decomposition* (M @ M'))
⟨*proof*⟩

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

lemma *get-all-ann-decomposition-remove-undecided-length*:
assumes $\forall l \in set\ M'. \neg$*is-decided l*
shows *length* (*get-all-ann-decomposition* (M' @ M'')) = *length* (*get-all-ann-decomposition* M'')
⟨*proof*⟩

lemma *get-all-ann-decomposition-not-is-decided-length*:
assumes $\forall l \in set\ M'. \neg$*is-decided l*
shows *1* + *length* (*get-all-ann-decomposition* (*Propagated* ($-L$) *P* # *M*))
= *length* (*get-all-ann-decomposition* (M' @ *Decided L* # *M*))
⟨*proof*⟩

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

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

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

lemma *get-all-ann-decomposition-exists-prepend*[*dest*]:

assumes $(a, b) \in set$ (*get-all-ann-decomposition* M)
shows $\exists c.\ M = c\ @\ b\ @\ a$
⟨*proof*⟩

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

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

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

lemma *Decided-cons-in-get-all-ann-decomposition-append-Decided-cons*:
$\exists$ *M1 M2*. (*Decided* K # *M1*, *M2*) $\in$ *set* (*get-all-ann-decomposition* (c @ *Decided* K # c'))
⟨*proof*⟩

lemma *fst-get-all-ann-decomposition-prepend-not-decided*:
assumes $\forall$ *m*$\in$*set MS*. $\neg$ *is-decided m*
shows *set* (*map fst* (*get-all-ann-decomposition* M))
= *set* (*map fst* (*get-all-ann-decomposition* (*MS* @ M)))
⟨*proof*⟩

Entailment of the Propagated by the Decided Literal

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

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

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

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

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

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

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

0.1.4 Negation of Clauses

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

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

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

lemma
shows
CNot-singleton[*simp*]: *CNot* $\{\#L\#\} = \{\{\#-L\#\}\}$ **and**
CNot-empty[*simp*]: *CNot* $\{\#\} = \{\}$ **and**
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-l* 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-l M*
⟨*proof*⟩

lemma *true-annots-CNot-all-uminus-atms-defined*:
assumes $M \models as\ CNot\ T$ **and** *a1*: $-L \in\#\ T$
shows *atm-of* $L \in$ *atm-of* ' *lits-of-l M*
⟨*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-l M*)
⟨*proof*⟩

lemma *true-annot-CNot-diff*:
$I \models as\ CNot\ C \Longrightarrow I \models as\ CNot\ (C - C')$
⟨*proof*⟩

lemma *CNot-mset-replicate*[*simp*]:
CNot (*mset* (*replicate n L*)) = (*if n* = *0 then* {} *else* $\{\{\#-L\#\}\}$)
⟨*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*: $atms\text{-}of\text{-}ms\ (CNot\ CC) = atms\text{-}of\text{-}ms\ \{CC\}$
⟨*proof*⟩

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

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

lemma *uminus-lit-swap*: $-(a::'a\ literal) = i \longleftrightarrow a = -i$
⟨*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** $\forall x \in atms\text{-}of\ D.\ x \notin atm\text{-}of\ `\ lits\text{-}of\text{-}l\ M$
shows $M' \models a\ D$
⟨*proof*⟩

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

lemma *all-variables-defined-not-imply-cnot*:
assumes
$\forall s \in atms\text{-}of\text{-}ms\ \{B\}.\ s \in atm\text{-}of\ `\ lits\text{-}of\text{-}l\ A$ **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*⟩

0.1.5 Other

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

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

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

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

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

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

0.1.6 Extending Entailments to multisets

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

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

abbreviation *true-clss-clss-m*:: *'v clause multiset* ⇒ *'v clause multiset* ⇒ *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
⟨*proof*⟩

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

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

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

abbreviation *atms-of-mm* :: *'a literal multiset multiset* ⇒ *'a set* **where**
atms-of-mm U ≡ *atms-of-ms* (*set-mset U*)

Other definition using *Union-mset*

lemma *atms-of-mm U* ≡ *set-mset* (⋃# *image-mset* (*image-mset atm-of*) *U*)
⟨*proof*⟩

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

abbreviation *true-clss-ext-m* (**infix** ⊨*sextm 49*) **where**
I ⊨*sextm C* ≡ *I* ⊨*sext set-mset C*

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

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

0.1.7 Abstract Clause Representation

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

We assume the following:

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

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

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

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

context
begin
interpretation *list-cls*: *raw-cls mset*
⟨*proof*⟩

interpretation *cls-cls*: *raw-cls id*

⟨*proof*⟩

interpretation *list-cls*: *raw-ccls-union mset*
union-mset-list remove1
⟨*proof*⟩

interpretation *cls-cls*: *raw-ccls-union id op* $\#\cup$ *remove1-mset*
⟨*proof*⟩
end

Over the abstract clauses, we have the following properties:

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

locale *raw-clss* =
raw-cls mset-cls
for
mset-cls :: $'cls \Rightarrow 'v$ *clause* +
fixes
mset-clss:: $'clss \Rightarrow 'v$ *clauses* **and**
union-clss :: $'clss \Rightarrow 'clss \Rightarrow 'clss$ **and**
in-clss :: $'cls \Rightarrow 'clss \Rightarrow bool$ **and**
insert-clss :: $'cls \Rightarrow 'clss \Rightarrow 'clss$ **and**
remove-from-clss :: $'cls \Rightarrow 'clss \Rightarrow 'clss$
assumes
insert-clss[*simp*]: *mset-clss* (*insert-clss* L C) = *mset-clss* C + {#*mset-cls* L#} **and**
union-clss[*simp*]: *mset-clss* (*union-clss* C D) = *mset-clss* C + *mset-clss* D **and**
mset-clss-union-clss[*simp*]: *mset-clss* (*insert-clss* C' D) = {#*mset-cls* C'#} + *mset-clss* D **and**
in-clss-mset-clss[*dest*]: *in-clss* a $C \Longrightarrow$ *mset-cls* $a \in\#$ *mset-clss* C **and**
in-mset-clss-exists-preimage: $b \in\#$ *mset-clss* $C \Longrightarrow \exists\, b'.$ *in-clss* b' $C \land$ *mset-cls* $b' = b$ **and**
remove-from-clss-mset-clss[*simp*]:
mset-clss (*remove-from-clss* a C) = *mset-clss* C − {#*mset-cls* a#} **and**
in-clss-union-clss[*simp*]:
in-clss a (*union-clss* C D) $\longleftrightarrow$ *in-clss* a $C \lor$ *in-clss* a D
begin

end

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

lemma *mset-map-mset-remove-first*:
mset (*map mset* (*remove-first* a C)) = *remove1-mset* (*mset* a) (*mset* (*map mset* C))
⟨*proof*⟩

interpretation *clss-clss*: *raw-clss id*
id op + *op* $\in\#$ $\lambda L\ C.\ C$ + {#L#} *remove1-mset*

⟨*proof*⟩

interpretation *list-clss*: *raw-clss mset*
$\lambda L.\ mset\ (map\ mset\ L)\ op\ @\ \lambda L\ C.\ L \in set\ C\ op\ \#$
remove-first
⟨*proof*⟩
end

end

Chapter 1

NOT's CDCL and DPLL

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

The organisation of the development is the following:

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

1.1 Measure

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

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

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

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

lemma μ_C*-single*[*simp*]:
$\mu_C\ s\ b\ [L] = L * b\ \hat{}\ (s - Suc\ 0)$
⟨*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*:
$\mu_C\ s\ b\ (L\ \#\ M) = L * b\ \hat{}\ (s - 1 - length\ M) + \mu_C\ s\ b\ M$
⟨*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*⟩

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

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

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

$(\forall\, i< \mathit{length}\ xs.\ xs\ !\ i < b) \wedge (\forall\, i< \mathit{length}\ ys.\ ys\ !\ i < b)\}$
⟨*proof*⟩

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

lemma *finite-νNOT*[*simp*]:
finite (*νNOT s base*)
⟨*proof*⟩

lemma *acyclic-νNOT*: *acyclic* (*νNOT s base*)
⟨*proof*⟩

lemma *wf-νNOT*: *wf* (*νNOT s base*)
⟨*proof*⟩

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

1.2 NOT's CDCL

1.2.1 Auxiliary Lemmas and Measure

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

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

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

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

lemma *atms-of-ms-single-image-atm-of-lit-of*:
$\mathit{atms\text{-}of\text{-}ms}\ (\mathit{unmark\text{-}s}\ A) = \mathit{atm\text{-}of}\ `\ (\mathit{lit\text{-}of}\ `\ A)$
⟨*proof*⟩

1.2.2 Initial definitions

The state

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

locale *dpll-state-ops* =
fixes
trail :: *'st* ⇒ (*'v*, *unit*) *ann-lits* **and**

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

end

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

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

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

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

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

Then we need several lemmas about the $reduce\text{-}trail\text{-}to_{NOT}$.

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

lemma $reduce\text{-}trail\text{-}to_{NOT}\text{-}length\text{-}ne$[*simp*]:

$length\ (trail\ S) \neq length\ F \Longrightarrow trail\ S \neq [] \Longrightarrow$
$reduce\text{-}trail\text{-}to_{NOT}\ F\ S = reduce\text{-}trail\text{-}to_{NOT}\ F\ (tl\text{-}trail\ S)$
⟨*proof*⟩

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

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

lemma *clauses-reduce-trail-to*$_{NOT}$*-Nil*:
$clauses_{NOT}\ (reduce\text{-}trail\text{-}to_{NOT}\ []\ S) = clauses_{NOT}\ S$
⟨*proof*⟩

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

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

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

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

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

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

lemma *reduce-trail-to*$_{NOT}$*-length*:
$length\ M = length\ M' \Longrightarrow reduce\text{-}trail\text{-}to_{NOT}\ M\ S = reduce\text{-}trail\text{-}to_{NOT}\ M'\ S$
⟨*proof*⟩

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

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

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

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

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

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

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

lemmas $state\text{-}simp_{NOT}[simp] = state\text{-}eq_{NOT}\text{-}trail\ state\text{-}eq_{NOT}\text{-}clauses$

lemma $reduce\text{-}trail\text{-}to_{NOT}\text{-}state\text{-}eq_{NOT}\text{-}compatible$:
assumes ST: $S \sim T$
shows $reduce\text{-}trail\text{-}to_{NOT}\ F\ S \sim reduce\text{-}trail\text{-}to_{NOT}\ F\ T$
⟨*proof*⟩

end

Definition of the operation

Each possible is in its own locale.

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

end

locale *decide-ops* =
  dpll-state trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
  for
    trail :: $'st \Rightarrow ('v,\ unit)$ *ann-lits* **and**
    $clauses_{NOT}$ :: $'st \Rightarrow 'v$ *clauses* **and**
    prepend-trail :: $('v,\ unit)$ *ann-lit* $\Rightarrow 'st \Rightarrow 'st$ **and**
    tl-trail :: $'st \Rightarrow 'st$ **and**
    $add\text{-}cls_{NOT}$ :: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$ **and**
    $remove\text{-}cls_{NOT}$ :: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$
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-mm* ($clauses_{NOT}$ 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 $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
  for
    trail :: $'st \Rightarrow ('v,\ unit)$ *ann-lits* **and**
    $clauses_{NOT}$ :: $'st \Rightarrow 'v$ *clauses* **and**
    prepend-trail :: $('v,\ unit)$ *ann-lit* $\Rightarrow 'st \Rightarrow 'st$ **and**
    tl-trail :: $'st \Rightarrow 'st$ **and**
    $add\text{-}cls_{NOT}$ :: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$ **and**
    $remove\text{-}cls_{NOT}$ :: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$ +
  fixes
    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)
   $\Longrightarrow C \in\#\ clauses_{NOT}\ S$
   $\Longrightarrow$ *trail* $S \models as$ *CNot* C
   $\Longrightarrow$ *undefined-lit* F L
   $\Longrightarrow$ *atm-of* $L \in$ *atms-of-mm* ($clauses_{NOT}$ S) $\cup$ *atm-of* ‘ (*lits-of-l* (*trail* S))
   $\Longrightarrow clauses_{NOT}\ S \models pm\ C' + \{\#L\#\}$
   $\Longrightarrow F \models as$ *CNot* C'
   $\Longrightarrow$ *backjump-conds* $C\ C'\ L\ S\ T$
   $\Longrightarrow$ *backjump* $S\ T$
inductive-cases *backjumpE*: *backjump* $S\ T$

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

end

1.2.3 DPLL with backjumping

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

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

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

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

Definition

We define dpll with backjumping:

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

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

$\Longrightarrow$ *clauses*$_{NOT}$ $S \models pm\ C' + \{\#L\#\}$
$\Longrightarrow$ $F \models as$ *CNot* C'
$\Longrightarrow$ $T \sim$ *prepend-trail* (*Propagated* L ()) (*reduce-trail-to*$_{NOT}$ F S)
$\Longrightarrow$ $P\ S\ T$
shows $P\ S\ T$
⟨*proof*⟩

Basic properties

First, some better suited induction principle **lemma** *dpll-bj-clauses*:
assumes *dpll-bj* S T **and** *inv* S
shows *clauses*$_{NOT}$ S = *clauses*$_{NOT}$ T
⟨*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 $I \models sm$ *clauses*$_{NOT}$ $S \longleftrightarrow I \models sm$ *clauses*$_{NOT}$ T
⟨*proof*⟩

Clauses **lemma** *dpll-bj-atms-of-ms-clauses-inv*:
assumes
dpll-bj S T **and**
inv S
shows *atms-of-mm* (*clauses*$_{NOT}$ S) = *atms-of-mm* (*clauses*$_{NOT}$ T)
⟨*proof*⟩

lemma *dpll-bj-atms-in-trail*:
assumes
dpll-bj S T **and**
inv S **and**
atm-of ‘ (*lits-of-l* (*trail* S)) $\subseteq$ *atms-of-mm* (*clauses*$_{NOT}$ S)
shows *atm-of* ‘ (*lits-of-l* (*trail* T)) $\subseteq$ *atms-of-mm* (*clauses*$_{NOT}$ S)
⟨*proof*⟩

lemma *dpll-bj-atms-in-trail-in-set*:
assumes *dpll-bj* S T**and**
inv S **and**
atms-of-mm (*clauses*$_{NOT}$ S) $\subseteq A$ **and**
atm-of ‘ (*lits-of-l* (*trail* S)) $\subseteq A$
shows *atm-of* ‘ (*lits-of-l* (*trail* T)) $\subseteq A$
⟨*proof*⟩

lemma *dpll-bj-all-decomposition-implies-inv*:
assumes
dpll-bj S T **and**
inv: *inv* S **and**
decomp: *all-decomposition-implies-m* (*clauses*$_{NOT}$ S) (*get-all-ann-decomposition* (*trail* S))
shows *all-decomposition-implies-m* (*clauses*$_{NOT}$ T) (*get-all-ann-decomposition* (*trail* T))
⟨*proof*⟩

Termination

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

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

⟨*proof*⟩

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

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

abbreviation *unassigned-lit* :: *'b literal multiset set* ⇒ *'a list* ⇒ *nat* **where**
unassigned-lit N M ≡ *card* (*atms-of-ms* N) − *length* M
lemma *dpll-bj-trail-mes-increasing-prop*:
fixes M :: (*'v*, *unit*) *ann-lits* **and** N :: *'v clauses*
assumes
dpll-bj S T **and**
inv S **and**
NA: *atms-of-mm* ($clauses_{NOT}$ S) ⊆ *atms-of-ms* A **and**
MA: *atm-of* ‘ *lits-of-l* (*trail* S) ⊆ *atms-of-ms* A **and**
n-d: *no-dup* (*trail* S) **and**
finite: *finite* A
shows μ_C (*1*+*card* (*atms-of-ms* A)) (*2*+*card* (*atms-of-ms* A)) (*trail-weight* T)
> μ_C (*1*+*card* (*atms-of-ms* A)) (*2*+*card* (*atms-of-ms* A)) (*trail-weight* S)
⟨*proof*⟩

lemma *dpll-bj-trail-mes-decreasing-prop*:
assumes *dpll*: *dpll-bj* S T **and** *inv*: *inv* S **and**
N-A: *atms-of-mm* ($clauses_{NOT}$ S) ⊆ *atms-of-ms* A **and**
M-A: *atm-of* ‘ *lits-of-l* (*trail* S) ⊆ *atms-of-ms* A **and**

nd: *no-dup* (*trail S*) **and**
fin-A: *finite A*
shows $(2+card\ (atms\text{-}of\text{-}ms\ A))\ \hat{}\ (1+card\ (atms\text{-}of\text{-}ms\ A))$
$- \mu_C\ (1+card\ (atms\text{-}of\text{-}ms\ A))\ (2+card\ (atms\text{-}of\text{-}ms\ A))\ (trail\text{-}weight\ T)$
$< (2+card\ (atms\text{-}of\text{-}ms\ A))\ \hat{}\ (1+card\ (atms\text{-}of\text{-}ms\ A))$
$- \mu_C\ (1+card\ (atms\text{-}of\text{-}ms\ A))\ (2+card\ (atms\text{-}of\text{-}ms\ A))\ (trail\text{-}weight\ S)$
⟨*proof*⟩

lemma *wf-dpll-bj*:
assumes *fin*: *finite A*
shows *wf* {(*T*, *S*). *dpll-bj S T*
∧ *atms-of-mm* ($clauses_{NOT}$ *S*) ⊆ *atms-of-ms A* ∧ *atm-of* ‘ *lits-of-l* (*trail S*) ⊆ *atms-of-ms A*
∧ *no-dup* (*trail S*) ∧ *inv S*}
(**is** *wf ?A*)
⟨*proof*⟩

Normal Forms

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

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

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

2. The assumption ¬ *M* ⊨*as N* implies that there is conflict.

3. There is at least one decision in the trail (otherwise, M would be a model of the set of clauses N).

4. Now if we build the clause with all the decision literals of the trail, we can apply the *backjump* rule.

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

theorem *dpll-backjump-final-state*:
fixes *A* :: *'v clause set* **and** *S T* :: *'st*
assumes
atms-of-mm ($clauses_{NOT}$ *S*) ⊆ *atms-of-ms A* **and**
atm-of ‘ *lits-of-l* (*trail S*) ⊆ *atms-of-ms A* **and**
no-dup (*trail S*) **and**
finite A **and**
inv: *inv S* **and**
n-s: *no-step dpll-bj S* **and**
decomp: *all-decomposition-implies-m* ($clauses_{NOT}$ *S*) (*get-all-ann-decomposition* (*trail S*))
shows *unsatisfiable* (*set-mset* ($clauses_{NOT}$ *S*))
∨ (*trail S* ⊨*asm* $clauses_{NOT}$ *S* ∧ *satisfiable* (*set-mset* ($clauses_{NOT}$ *S*)))
⟨*proof*⟩

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

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

for
  $trail :: 'st \Rightarrow ('v,\ unit)\ ann\text{-}lits$ **and**
  $clauses_{NOT} :: 'st \Rightarrow 'v\ clauses$ **and**
  $prepend\text{-}trail :: ('v,\ unit)\ ann\text{-}lit \Rightarrow 'st \Rightarrow 'st$ **and**
  $tl\text{-}trail :: 'st \Rightarrow 'st$ **and**
  $add\text{-}cls_{NOT} :: 'v\ clause \Rightarrow 'st \Rightarrow 'st$ **and**
  $remove\text{-}cls_{NOT} :: 'v\ clause \Rightarrow 'st \Rightarrow 'st$ **and**
  $inv :: 'st \Rightarrow bool$ **and**
  $backjump\text{-}conds :: 'v\ clause \Rightarrow 'v\ clause \Rightarrow 'v\ literal \Rightarrow 'st \Rightarrow 'st \Rightarrow bool$ **and**
  $propagate\text{-}conds :: ('v,\ unit)\ ann\text{-}lit \Rightarrow 'st \Rightarrow bool$
+
assumes *dpll-bj-inv*: $\bigwedge S\ T.\ dpll\text{-}bj\ S\ T \Longrightarrow inv\ S \Longrightarrow inv\ T$
begin

lemma *rtranclp-dpll-bj-inv*:
  assumes $dpll\text{-}bj^{**}\ S\ T$ **and** $inv\ S$
  shows $inv\ T$
  ⟨*proof*⟩

lemma *rtranclp-dpll-bj-no-dup*:
  assumes $dpll\text{-}bj^{**}\ S\ T$ **and** $inv\ S$
  and $no\text{-}dup\ (trail\ S)$
  shows $no\text{-}dup\ (trail\ T)$
  ⟨*proof*⟩

lemma *rtranclp-dpll-bj-atms-of-ms-clauses-inv*:
  assumes
    $dpll\text{-}bj^{**}\ S\ T$ **and** $inv\ S$
  shows $atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) = atms\text{-}of\text{-}mm\ (clauses_{NOT}\ T)$
  ⟨*proof*⟩

lemma *rtranclp-dpll-bj-atms-in-trail*:
  assumes
    $dpll\text{-}bj^{**}\ S\ T$ **and**
    $inv\ S$ **and**
    $atm\text{-}of\ `\ (lits\text{-}of\text{-}l\ (trail\ S)) \subseteq atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S)$
  shows $atm\text{-}of\ `\ (lits\text{-}of\text{-}l\ (trail\ T)) \subseteq atms\text{-}of\text{-}mm\ (clauses_{NOT}\ T)$
  ⟨*proof*⟩

lemma *rtranclp-dpll-bj-sat-iff*:
  assumes $dpll\text{-}bj^{**}\ S\ T$ **and** $inv\ S$
  shows $I \models sm\ clauses_{NOT}\ S \longleftrightarrow I \models sm\ clauses_{NOT}\ T$
  ⟨*proof*⟩

lemma *rtranclp-dpll-bj-atms-in-trail-in-set*:
  assumes
    $dpll\text{-}bj^{**}\ S\ T$ **and**
    $inv\ S$
    $atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \subseteq A$ **and**
    $atm\text{-}of\ `\ (lits\text{-}of\text{-}l\ (trail\ S)) \subseteq A$
  shows $atm\text{-}of\ `\ (lits\text{-}of\text{-}l\ (trail\ T)) \subseteq A$
  ⟨*proof*⟩

lemma *rtranclp-dpll-bj-all-decomposition-implies-inv*:
  assumes
    $dpll\text{-}bj^{**}\ S\ T$ **and**

inv S
all-decomposition-implies-m $(clauses_{NOT}\ S)$ (*get-all-ann-decomposition* (*trail S*))
shows *all-decomposition-implies-m* $(clauses_{NOT}\ T)$ (*get-all-ann-decomposition* (*trail T*))
⟨*proof*⟩

lemma *rtranclp-dpll-bj-inv-incl-dpll-bj-inv-trancl*:
$\{(T, S).\ dpll\text{-}bj^{++}\ S\ T$
$\wedge$ *atms-of-mm* $(clauses_{NOT}\ S) \subseteq$ *atms-of-ms A* $\wedge$ *atm-of* ‘ *lits-of-l* (*trail S*) $\subseteq$ *atms-of-ms A*
$\wedge$ *no-dup* (*trail S*) $\wedge$ *inv S*$\}$
$\subseteq \{(T, S).$ *dpll-bj S T* $\wedge$ *atms-of-mm* $(clauses_{NOT}\ S) \subseteq$ *atms-of-ms A*
$\wedge$ *atm-of* ‘ *lits-of-l* (*trail S*) $\subseteq$ *atms-of-ms A* $\wedge$ *no-dup* (*trail S*) $\wedge$ *inv S*$\}^+$
(**is** $?A \subseteq ?B^+$)
⟨*proof*⟩

lemma *wf-tranclp-dpll-bj*:
assumes *fin*: *finite A*
shows *wf* $\{(T, S).\ dpll\text{-}bj^{++}\ S\ T$
$\wedge$ *atms-of-mm* $(clauses_{NOT}\ S) \subseteq$ *atms-of-ms A* $\wedge$ *atm-of* ‘ *lits-of-l* (*trail S*) $\subseteq$ *atms-of-ms A*
$\wedge$ *no-dup* (*trail S*) $\wedge$ *inv S*$\}$
⟨*proof*⟩

lemma *dpll-bj-sat-ext-iff*:
dpll-bj S T $\Longrightarrow$ *inv S* $\Longrightarrow$ $I \models sextm\ clauses_{NOT}\ S \longleftrightarrow I \models sextm\ clauses_{NOT}\ T$
⟨*proof*⟩

lemma *rtranclp-dpll-bj-sat-ext-iff*:
$dpll\text{-}bj^{**}\ S\ T \Longrightarrow$ *inv S* $\Longrightarrow$ $I \models sextm\ clauses_{NOT}\ S \longleftrightarrow I \models sextm\ clauses_{NOT}\ T$
⟨*proof*⟩

theorem *full-dpll-backjump-final-state*:
fixes A :: *'v clause set* **and** $S\ T$ :: *'st*
assumes
full: *full dpll-bj S T* **and**
atms-S: *atms-of-mm* $(clauses_{NOT}\ S) \subseteq$ *atms-of-ms A* **and**
atms-trail: *atm-of* ‘ *lits-of-l* (*trail S*) $\subseteq$ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite A **and**
inv: *inv S* **and**
decomp: *all-decomposition-implies-m* $(clauses_{NOT}\ S)$ (*get-all-ann-decomposition* (*trail S*))
shows *unsatisfiable* (*set-mset* $(clauses_{NOT}\ S)$)
$\vee$ (*trail T* $\models asm\ clauses_{NOT}\ S \wedge$ *satisfiable* (*set-mset* $(clauses_{NOT}\ S)$))
⟨*proof*⟩

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

lemma *tranclp-dpll-bj-trail-mes-decreasing-prop*:
assumes *dpll*: $dpll\text{-}bj^{++}\ S\ T$ **and** *inv*: *inv S* **and**
N-A: *atms-of-mm* $(clauses_{NOT}\ S) \subseteq$ *atms-of-ms A* **and**

$M\text{-}A$: *atm-of* ‘ *lits-of-l* (*trail* S) $\subseteq$ *atms-of-ms* A **and**
$n\text{-}d$: *no-dup* (*trail* S) **and**
fin-A: *finite* A
shows $(2+card\ (atms\text{-}of\text{-}ms\ A))\ \hat{}\ (1+card\ (atms\text{-}of\text{-}ms\ A))$
$\quad - \mu_C\ (1+card\ (atms\text{-}of\text{-}ms\ A))\ (2+card\ (atms\text{-}of\text{-}ms\ A))\ (trail\text{-}weight\ T)$
$< (2+card\ (atms\text{-}of\text{-}ms\ A))\ \hat{}\ (1+card\ (atms\text{-}of\text{-}ms\ A))$
$\quad - \mu_C\ (1+card\ (atms\text{-}of\text{-}ms\ A))\ (2+card\ (atms\text{-}of\text{-}ms\ A))\ (trail\text{-}weight\ S)$
⟨*proof*⟩

end — End of *dpll-with-backjumping*

1.2.4 CDCL

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

Learn and Forget

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

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

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

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

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

$add\text{-}cls_{NOT}$ :: $'v$ *clause* ⇒ $'st$ ⇒ $'st$ **and**
$remove\text{-}cls_{NOT}$ :: $'v$ *clause* ⇒ $'st$ ⇒ $'st$ +
fixes
forget-cond :: $'v$ *clause* ⇒ $'st$ ⇒ *bool*
begin
inductive $forget_{NOT}$ :: $'st$ ⇒ $'st$ ⇒ *bool* **where**
$forget_{NOT}$:
removeAll-mset C(*clauses*$_{NOT}$ S) ⊨*pm* C ⟹
forget-cond C S ⟹
C ∈# *clauses*$_{NOT}$ 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$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$ *learn-cond* +
forget-ops trail clauses$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$ *forget-cond*
for
trail :: $'st$ ⇒ ($'v$, *unit*) *ann-lits* **and**
clauses$_{NOT}$ :: $'st$ ⇒ $'v$ *clauses* **and**
prepend-trail :: ($'v$, *unit*) *ann-lit* ⇒ $'st$ ⇒ $'st$ **and**
tl-trail :: $'st$ ⇒$'st$ **and**
add-cls$_{NOT}$ :: $'v$ *clause* ⇒ $'st$ ⇒ $'st$ **and**
remove-cls$_{NOT}$ :: $'v$ *clause* ⇒ $'st$ ⇒ $'st$ **and**
learn-cond forget-cond :: $'v$ *clause* ⇒ $'st$ ⇒ *bool*
begin
inductive *learn-and-forget*$_{NOT}$ :: $'st$ ⇒ $'st$ ⇒ *bool*
where
lf-learn: *learn* S T ⟹ *learn-and-forget*$_{NOT}$ S T |
lf-forget: $forget_{NOT}$ S T ⟹ *learn-and-forget*$_{NOT}$ S T
end

Definition of CDCL

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

begin

inductive $cdcl_{NOT}$ :: $'st \Rightarrow 'st \Rightarrow bool$ **for** S :: $'st$ **where**
c-dpll-bj: *dpll-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-bj* S $T \Longrightarrow P$ S T **and**
learning:
$\bigwedge C$ T. *clauses*$_{NOT}$ S $\models pm$ $C \Longrightarrow$
atms-of $C \subseteq$ *atms-of-mm* (*clauses*$_{NOT}$ S) $\cup$ *atm-of* ‘ (*lits-of-l* (*trail* S)) $\Longrightarrow$
$T \sim$ *add-cls*$_{NOT}$ C $S \Longrightarrow$
P S T **and**
forgetting: $\bigwedge C$ T. *removeAll-mset* C (*clauses*$_{NOT}$ S) $\models pm$ $C \Longrightarrow$
$C \in\#$ *clauses*$_{NOT}$ $S \Longrightarrow$
$T \sim$ *remove-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-l* (*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 means that some variable of the trail might not be present in the clauses anymore.

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

lemma $cdcl_{NOT}$*-atms-in-trail*:
assumes $cdcl_{NOT}$ S T**and** *inv* S **and** *no-dup* (*trail* S)
and *atm-of* ‘ (*lits-of-l* (*trail* S)) $\subseteq$ *atms-of-mm* (*clauses*$_{NOT}$ S)
shows *atm-of* ‘ (*lits-of-l* (*trail* T)) $\subseteq$ *atms-of-mm* (*clauses*$_{NOT}$ 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-mm (*clauses*$_{NOT}$ *S*) $\subseteq$ *A* **and**
atm-of ' (*lits-of-l* (*trail S*)) $\subseteq$ *A*
shows *atm-of* ' (*lits-of-l* (*trail T*)) $\subseteq$ *A*
⟨*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*$_{NOT}$ *S*) (*get-all-ann-decomposition* (*trail S*))
shows
all-decomposition-implies-m (*clauses*$_{NOT}$ *T*) (*get-all-ann-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*$_{NOT}$ *S* $\longleftrightarrow$ *I*$\models$*sextm clauses*$_{NOT}$ *T*
⟨*proof*⟩

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

CDCL with invariant

locale *conflict-driven-clause-learning* =
conflict-driven-clause-learning-ops +
assumes $cdcl_{NOT}$*-inv*: $\bigwedge S\ T.\ cdcl_{NOT}\ S\ T \Longrightarrow inv\ S \Longrightarrow inv\ T$
begin
sublocale *dpll-with-backjumping*
⟨*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-mm* (*clauses*$_{NOT}$ *S*) $\subseteq$ *A* **and**
atms-trail-S: *atm-of* '(*lits-of-l* (*trail S*)) $\subseteq$ *A*
shows *atm-of* ' (*lits-of-l* (*trail T*)) $\subseteq$ *A* $\wedge$ *atms-of-mm* (*clauses*$_{NOT}$ *T*) $\subseteq$ *A*
⟨*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*$_{NOT}$ *S*) (*get-all-ann-decomposition* (*trail S*))
shows
all-decomposition-implies-m (*clauses*$_{NOT}$ *T*) (*get-all-ann-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 $I \models sextm\ clauses_{NOT}\ S \longleftrightarrow I \models sextm\ clauses_{NOT}\ T$
⟨*proof*⟩

definition *cdcl*$_{NOT}$*-NOT-all-inv* **where**
$cdcl_{NOT}\text{-}NOT\text{-}all\text{-}inv\ A\ S \longleftrightarrow (finite\ A \wedge inv\ S \wedge atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \subseteq atms\text{-}of\text{-}ms\ A$
$\wedge\ atm\text{-}of\ \text{`}\ lits\text{-}of\text{-}l\ (trail\ S) \subseteq atms\text{-}of\text{-}ms\ A \wedge no\text{-}dup\ (trail\ S))$

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**
$learn\text{-}or\text{-}forget\ S\ T \equiv learn\ S\ T \vee forget_{NOT}\ S\ T$

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

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

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

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

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

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

lemma $cdcl_{NOT}$*-final-state*:
assumes
n-s: *no-step* $cdcl_{NOT}$ *S* **and**
inv: $cdcl_{NOT}$*-NOT-all-inv A S* **and**
decomp: *all-decomposition-implies-m* ($clauses_{NOT}$ *S*) (*get-all-ann-decomposition* (*trail S*))
shows *unsatisfiable* (*set-mset* ($clauses_{NOT}$ *S*))
∨ (*trail S* ⊨*asm* $clauses_{NOT}$ *S* ∧ *satisfiable* (*set-mset* ($clauses_{NOT}$ *S*)))
⟨*proof*⟩

lemma *full-*$cdcl_{NOT}$*-final-state*:
assumes
full: *full* $cdcl_{NOT}$ *S T* **and**
inv: $cdcl_{NOT}$*-NOT-all-inv A S* **and**
n-d: *no-dup* (*trail S*) **and**
decomp: *all-decomposition-implies-m* ($clauses_{NOT}$ *S*) (*get-all-ann-decomposition* (*trail S*))
shows *unsatisfiable* (*set-mset* ($clauses_{NOT}$ *T*))
∨ (*trail T* ⊨*asm* $clauses_{NOT}$ *T* ∧ *satisfiable* (*set-mset* ($clauses_{NOT}$ *T*)))
⟨*proof*⟩

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

Termination

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

Restricting learn and forget

locale *conflict-driven-clause-learning-learning-before-backjump-only-distinct-learnt* =
dpll-state trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ +
conflict-driven-clause-learning trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
inv backjump-conds propagate-conds
λ*C S*. *distinct-mset C* ∧ ¬*tautology C* ∧ *learn-restrictions C S* ∧
(∃ *F K d F′ C′ L*. *trail S* = *F′* @ *Decided K* # *F* ∧ *C* = *C′* + {#*L*#} ∧ *F* ⊨*as CNot C′*
∧ *C′* + {#*L*#} ∉# $clauses_{NOT}$ *S*)
λ*C S*. ¬(∃ *F′ F K d L*. *trail S* = *F′* @ *Decided K* # *F* ∧ *F* ⊨*as CNot* (*remove1-mset L C*))
∧ *forget-restrictions C S*
for
trail :: *′st* ⇒ (*′v*, *unit*) *ann-lits* **and**
$clauses_{NOT}$ :: *′st* ⇒ *′v clauses* **and**
prepend-trail :: (*′v*, *unit*) *ann-lit* ⇒ *′st* ⇒ *′st* **and**
tl-trail :: *′st* ⇒*′st* **and**
$add\text{-}cls_{NOT}$ :: *′v clause* ⇒ *′st* ⇒ *′st* **and**
$remove\text{-}cls_{NOT}$ :: *′v clause* ⇒ *′st* ⇒ *′st* **and**
inv :: *′st* ⇒ *bool* **and**
backjump-conds :: *′v clause* ⇒ *′v clause* ⇒ *′v literal* ⇒ *′st* ⇒ *′st* ⇒ *bool* **and**
propagate-conds :: (*′v*, *unit*) *ann-lit* ⇒ *′st* ⇒ *bool* **and**
learn-restrictions forget-restrictions :: *′v clause* ⇒ *′st* ⇒ *bool*

begin

lemma $cdcl_{NOT}$*-learn-all-induct*[*consumes 1*, *case-names dpll-bj learn forget*$_{NOT}$]:
fixes S T :: *'st*
assumes $cdcl_{NOT}$ S T **and**
dpll: $\bigwedge T.$ *dpll-bj* S $T \Longrightarrow P$ S T **and**
learning:
$\bigwedge C\ F\ K\ F'\ C'\ L\ T.$ $clauses_{NOT}$ $S \models pm$ $C \Longrightarrow$
atms-of $C \subseteq$ *atms-of-mm* ($clauses_{NOT}$ S) $\cup$ *atm-of* ‘ (*lits-of-l* (*trail* S)) $\Longrightarrow$
distinct-mset $C \Longrightarrow$
$\neg$ *tautology* $C \Longrightarrow$
learn-restrictions C $S \Longrightarrow$
trail $S = F'$ @ *Decided* K # $F \Longrightarrow$
$C = C' + \{\#L\#\} \Longrightarrow$
$F \models as$ *CNot* $C' \Longrightarrow$
$C' + \{\#L\#\} \notin\#$ $clauses_{NOT}$ $S \Longrightarrow$
$T \sim$ $add\text{-}cls_{NOT}$ C $S \Longrightarrow$
P S T **and**
forgetting: $\bigwedge C\ T.$ *removeAll-mset* C ($clauses_{NOT}$ S) $\models pm$ $C \Longrightarrow$
$C \in\#$ $clauses_{NOT}$ $S \Longrightarrow$
$\neg(\exists F'\ F\ K\ L.$ *trail* $S = F'$ @ *Decided* K # $F \wedge F \models as$ *CNot* $(C - \{\#L\#\})) \Longrightarrow$
$T \sim$ $remove\text{-}cls_{NOT}$ 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* ($clauses_{NOT}$ T $-$ $clauses_{NOT}$ S)
$\subseteq$ *simple-clss* (*atms-of-mm* ($clauses_{NOT}$ S) $\cup$ *atm-of* ‘ *lits-of-l* (*trail* S))
⟨*proof*⟩

definition *conflicting-bj-clss* $S \equiv$
$\{C+\{\#L\#\}\ |C\ L.\ C+\{\#L\#\} \in\#$ $clauses_{NOT}$ $S \wedge$ *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 ($remove\text{-}cls_{NOT}$ C S) = *conflicting-bj-clss* $S - \{C\}$
⟨*proof*⟩

lemma *conflicting-bj-clss-remove-cls*$_{NOT}$'[*simp*]:
$T \sim$ $remove\text{-}cls_{NOT}$ C $S \Longrightarrow$ *conflicting-bj-clss* T = *conflicting-bj-clss* $S - \{C\}$
⟨*proof*⟩

lemma *conflicting-bj-clss-add-cls*$_{NOT}$*-state-eq*:
assumes
T: $T \sim$ $add\text{-}cls_{NOT}$ C' S **and**
n-d: *no-dup* (*trail* S)

shows *conflicting-bj-clss* T
$= \textit{conflicting-bj-clss}\ S$
$\cup$ (*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)
$= \textit{conflicting-bj-clss}\ S$
$\cup$ (*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*$_{NOT}$ S)
⟨*proof*⟩

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

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

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

abbreviation μ_L :: *nat* $\Rightarrow$ *'st* $\Rightarrow$ *nat* $\times$ *nat* **where**
μ_L b S $\equiv$ (*conflicting-bj-clss-yet* b S, *card* (*set-mset* (*clauses*$_{NOT}$ S)))

lemma *remove1-mset-single-add-if*:
remove1-mset L $(C + \{\#L'\#\})$ = (*if* $L = L'$ *then* C *else remove1-mset* L $C + \{\#L'\#\})$
⟨*proof*⟩

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
⟨*proof*⟩

lemma *forget-*μ_L*-decrease*:
assumes *forget*$_{NOT}$: *forget*$_{NOT}$ S T
shows $(\mu_L\ b\ T, \mu_L\ b\ S) \in$ *less-than* <**lex**> *less-than*
⟨*proof*⟩

lemma *set-condition-or-split*:
$\{a.\ (a = b \vee Q\ a) \wedge S\ a\} = (\textit{if}\ S\ b\ \textit{then}\ \{b\}\ \textit{else}\ \{\}) \cup \{a.\ Q\ a \wedge S\ a\}$
⟨*proof*⟩

lemma *set-insert-neq*:
$A \neq \textit{insert}\ a\ A \longleftrightarrow a \notin A$
⟨*proof*⟩

lemma *learn-μ_L-decrease*:
assumes *learnST*: *learn S T* **and** *n-d*: *no-dup* (*trail S*) **and**
A: *atms-of-mm* (*clauses*$_{NOT}$ *S*) ∪ *atm-of* ‘ *lits-of-l* (*trail S*) ⊆ *A* **and**
fin-A: *finite A*
shows (μ_L (*card A*) *T*, μ_L (*card A*) *S*) ∈ *less-than* <∗*lex*∗> *less-than*
⟨*proof*⟩

We have to assume the following:

- *inv S*: the invariant holds in the inital state.
- *A* is a (finite *finite A*) superset of the literals in the trail *atm-of* ‘ *lits-of-l* (*trail S*) ⊆ *atms-of-ms A* and in the clauses *atms-of-mm* (*clauses*$_{NOT}$ *S*) ⊆ *atms-of-ms A*. This can the the set of all the literals in the starting set of clauses.
- *no-dup* (*trail S*): no duplicate in the trail. This is invariant along the path.

definition μ_{CDCL} **where**
μ_{CDCL} *A T* ≡ ((*2*+*card* (*atms-of-ms A*)) ^ (*1*+*card* (*atms-of-ms A*))
− μ_C (*1*+*card* (*atms-of-ms A*)) (*2*+*card* (*atms-of-ms A*)) (*trail-weight T*),
conflicting-bj-clss-yet (*card* (*atms-of-ms A*)) *T*, *card* (*set-mset* (*clauses*$_{NOT}$ *T*)))
lemma *cdcl*$_{NOT}$*-decreasing-measure*:
assumes
cdcl$_{NOT}$ *S T* **and**
inv: *inv S* **and**
atm-clss: *atms-of-mm* (*clauses*$_{NOT}$ *S*) ⊆ *atms-of-ms A* **and**
atm-lits: *atm-of* ‘ *lits-of-l* (*trail S*) ⊆ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
fin-A: *finite A*
shows (μ_{CDCL} *A T*, μ_{CDCL} *A S*)
∈ *less-than* <∗*lex*∗> (*less-than* <∗*lex*∗> *less-than*)
⟨*proof*⟩

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

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

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

lemma *cdcl*$_{NOT}$*-decreasing-measure'*:
assumes
cdcl$_{NOT}$ *S T* **and**

inv: *inv* S **and**
atms-clss: *atms-of-mm* ($clauses_{NOT}$ S) $\subseteq$ *atms-of-ms* A **and**
atms-trail: *atm-of* ‘ *lits-of-l* (*trail* S) $\subseteq$ *atms-of-ms* A **and**
n-d: *no-dup* (*trail* S) **and**
fin-A: *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-mm ($clauses_{NOT}$ S) $\subseteq$ A **and**
atm-of ‘(*lits-of-l* (*trail* S)) $\subseteq$ A **and**
n-d: *no-dup* (*trail* S) **and**
fin-A[*simp*]: *finite* A
shows *set-mset* ($clauses_{NOT}$ T) $\subseteq$ *set-mset* ($clauses_{NOT}$ S) $\cup$ *simple-clss* A
⟨*proof*⟩

lemma *rtranclp-*$cdcl_{NOT}$*-clauses-bound*:
assumes
$cdcl_{NOT}{}^{**}$ S T **and**
inv S **and**
atms-of-mm ($clauses_{NOT}$ S) $\subseteq$ A **and**
atm-of ‘(*lits-of-l* (*trail* S)) $\subseteq$ A **and**
n-d: *no-dup* (*trail* S) **and**
finite: *finite* A
shows *set-mset* ($clauses_{NOT}$ T) $\subseteq$ *set-mset* ($clauses_{NOT}$ S) $\cup$ *simple-clss* A
⟨*proof*⟩

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

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

definition $\mu_{CDCL}{}'$*-bound* :: *'v clause set* $\Rightarrow$ *'st* $\Rightarrow$ *nat* **where**
$\mu_{CDCL}{}'$*-bound A S* =
((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*))) $*$ (*1* + *3* ^ *card* (*atms-of-ms A*)) $*$ *2*
+ *2*$*$*3* ^ (*card* (*atms-of-ms A*))
+ *card* {*C*. *C* $\in\#$ $clauses_{NOT}$ *S* $\wedge$ (*tautology C* $\vee$ $\neg$*distinct-mset C*)} + *3* ^ (*card* (*atms-of-ms A*))

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

lemma *rtranclp-cdcl*$_{NOT}$*-*$\mu_{CDCL}{}'$*-bound-reduce-trail-to*$_{NOT}$:
assumes
$cdcl_{NOT}^{**}$ *S T* **and**
inv S **and**
atms-of-mm ($clauses_{NOT}$ *S*) $\subseteq$ *atms-of-ms A* **and**
atm-of ‘(*lits-of-l* (*trail S*)) $\subseteq$ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite* (*atms-of-ms A*) **and**
U: *U* $\sim$ *reduce-trail-to*$_{NOT}$ *M T*
shows $\mu_{CDCL}{}'$ *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-mm ($clauses_{NOT}$ *S*) $\subseteq$ *atms-of-ms A* **and**
atm-of ‘(*lits-of-l* (*trail S*)) $\subseteq$ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite* (*atms-of-ms A*)
shows $\mu_{CDCL}{}'$ *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-mm ($clauses_{NOT}$ *S*) $\subseteq$ *atms-of-ms A* **and**
atm-of ‘(*lits-of-l* (*trail S*)) $\subseteq$ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite[*simp*]: *finite* (*atms-of-ms A*)
shows $\mu_{CDCL}{}'$*-bound A T* $\leq$ $\mu_{CDCL}{}'$*-bound A S*
⟨*proof*⟩

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

1.2.5 CDCL with restarts

Definition

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

end

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

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

lemma $cdcl_{NOT}$*-*$cdcl_{NOT}$*-raw-restart*:
  $cdcl_{NOT}$ S $T \Longrightarrow$ *restart-ops.*$cdcl_{NOT}$*-raw-restart* $cdcl_{NOT}$ *restart* S T
  ⟨*proof*⟩
end

Increasing restarts

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

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

- an invariant on the states $cdcl_{NOT}$-*inv* that also holds after restarts.
- it is *not required* that the measure decrease with respect to restarts, but the measure has to be bound by some function μ-*bound* taking the same parameter as μ and the initial state of the considered $cdcl_{NOT}$ chain.

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

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

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

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

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

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

lemma *wf-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 $\mu\ A\ T \leq \mu\ A\ S$
  ⟨*proof*⟩

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

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

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

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

lemma *cdcl$_{NOT}$-restart-cdcl$_{NOT}$-raw-restart*:
  $cdcl_{NOT}\text{-}restart\ S\ T \Longrightarrow cdcl_{NOT}\text{-}raw\text{-}restart^{**}\ (fst\ S)\ (fst\ T)$
⟨*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* +
dpll-state trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
for
trail :: *'st* ⇒ (*'v*, *unit*) *ann-lits* **and**
$clauses_{NOT}$ :: *'st* ⇒ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
$add\text{-}cls_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
$remove\text{-}cls_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
f :: *nat* ⇒ *nat* **and**
restart :: *'st* ⇒ *'st* ⇒ *bool* **and**
bound-inv :: *'bound* ⇒ *'st* ⇒ *bool* **and**
μ :: *'bound* ⇒ *'st* ⇒ *nat* **and**
$cdcl_{NOT}$ :: *'st* ⇒ *'st* ⇒ *bool* **and**
$cdcl_{NOT}$*-inv* :: *'st* ⇒ *bool* **and**
μ*-bound* :: *'bound* ⇒ *'st* ⇒ *nat* +
assumes
measure-bound: $\bigwedge A\ T\ V\ n.$ $cdcl_{NOT}$*-inv* $T \Longrightarrow$ *bound-inv* A T
$\Longrightarrow$ $cdcl_{NOT}$*-restart* (T, n) (V, *Suc* n) $\Longrightarrow \mu\ A\ V \leq \mu$*-bound* A T **and**
$cdcl_{NOT}$*-raw-restart-*μ*-bound*:
$cdcl_{NOT}$*-restart* (T, a) (V, b) $\Longrightarrow$ $cdcl_{NOT}$*-inv* $T \Longrightarrow$ *bound-inv* A T
$\Longrightarrow \mu$*-bound* A $V \leq \mu$*-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
$\Longrightarrow \mu$*-bound* A $V \leq \mu$*-bound* A T

$\langle proof \rangle$

lemma *cdcl$_{NOT}$-raw-restart-measure-bound*:
cdcl$_{NOT}$-restart $(T,\ a)$ $(V,\ b) \Longrightarrow$ *cdcl$_{NOT}$-inv* $T \Longrightarrow$ *bound-inv* A T
$\Longrightarrow \mu\ A\ V \leq \mu$*-bound* A T
$\langle proof \rangle$

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
$\Longrightarrow \mu\ A\ V \leq \mu$*-bound* A T
$\langle proof \rangle$

lemma *wf-cdcl$_{NOT}$-restart*:
wf $\{(T,\ S).$ *cdcl$_{NOT}$-restart* S $T \wedge$ *cdcl$_{NOT}$-inv* (*fst* S)$\}$ (**is** *wf ?A*)
$\langle proof \rangle$

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) $> \mu$*-bound* A (*fst* S)
shows *full1 cdcl$_{NOT}$* (*fst* S) (*fst* T)
$\langle proof \rangle$

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$_{NOT}$-inv* $S \wedge$ *bound-inv* A $S)^{**}$ S $T \longleftrightarrow$ *cdcl$_{NOT}$*** S T
(**is** $?A^{**}$ S $T \longleftrightarrow$ $?B^{**}$ S T)
$\langle proof \rangle$

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)
$\langle proof \rangle$

end

1.2.6 Merging backjump and learning

locale *cdcl$_{NOT}$-merge-bj-learn-ops* =
decide-ops trail clauses$_{NOT}$ prepend-trail tl-trail add-cls$_{NOT}$ remove-cls$_{NOT}$ +
forget-ops trail clauses$_{NOT}$ prepend-trail tl-trail add-cls$_{NOT}$ remove-cls$_{NOT}$ forget-cond +
propagate-ops trail clauses$_{NOT}$ prepend-trail tl-trail add-cls$_{NOT}$ remove-cls$_{NOT}$ propagate-conds
for
trail :: $'st \Rightarrow ('v,\ unit)$ *ann-lits* **and**
clauses$_{NOT}$ :: $'st \Rightarrow 'v$ *clauses* **and**
prepend-trail :: $('v,\ unit)$ *ann-lit* $\Rightarrow 'st \Rightarrow 'st$ **and**
tl-trail :: $'st \Rightarrow 'st$ **and**
add-cls$_{NOT}$ :: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$ **and**
remove-cls$_{NOT}$ :: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$ **and**

propagate-conds :: ('*v*, *unit*) *ann-lit* ⇒ '*st* ⇒ *bool* **and**
forget-cond :: '*v clause* ⇒ '*st* ⇒ *bool* +
fixes *backjump-l-cond* :: '*v clause* ⇒ '*v clause* ⇒ '*v literal* ⇒ '*st* ⇒ '*st* ⇒ *bool*
begin

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

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

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

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

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

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

locale *cdcl*$_{NOT}$*-merge-bj-learn-proxy* =
cdcl$_{NOT}$*-merge-bj-learn-ops trail clauses*$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$
propagate-conds forget-cond
λ*C* C' L' *S T*. *backjump-l-cond C* C' L' *S T*
∧ *distinct-mset* (C' + {#L'#}) ∧ ¬*tautology* (C' + {#L'#})
for
trail :: '*st* ⇒ ('*v*, *unit*) *ann-lits* **and**
clauses$_{NOT}$ :: '*st* ⇒ '*v clauses* **and**
prepend-trail :: ('*v*, *unit*) *ann-lit* ⇒ '*st* ⇒ '*st* **and**
tl-trail :: '*st* ⇒'*st* **and**
add-cls$_{NOT}$ :: '*v clause* ⇒ '*st* ⇒ '*st* **and**
remove-cls$_{NOT}$ :: '*v clause* ⇒ '*st* ⇒ '*st* **and**
propagate-conds :: ('*v*, *unit*) *ann-lit* ⇒ '*st* ⇒ *bool* **and**
forget-cond :: '*v clause* ⇒ '*st* ⇒ *bool* **and**
backjump-l-cond :: '*v clause* ⇒ '*v clause* ⇒ '*v literal* ⇒ '*st* ⇒ '*st* ⇒ *bool* +
fixes
inv :: '*st* ⇒ *bool*

assumes
bj-merge-can-jump:
$\bigwedge S\ C\ F'\ K\ F\ L.$
inv S
$\Longrightarrow$ *trail S* $=$ *F'* @ *Decided K* # *F*
$\Longrightarrow$ $C \in\#$ $clauses_{NOT}$ *S*
$\Longrightarrow$ *trail S* $\models$*as CNot C*
$\Longrightarrow$ *undefined-lit F L*
$\Longrightarrow$ *atm-of L* $\in$ *atms-of-mm* ($clauses_{NOT}$ *S*) $\cup$ *atm-of* ' (*lits-of-l* (*F'* @ *Decided K* # *F*))
$\Longrightarrow$ $clauses_{NOT}$ *S* $\models$*pm C'* + {#*L*#}
$\Longrightarrow$ *F* $\models$*as CNot C'*
$\Longrightarrow$ $\neg$*no-step backjump-l S* **and**
cdcl-merged-inv: $\bigwedge S\ T.$ $cdcl_{NOT}$*-merged-bj-learn S T* $\Longrightarrow$ *inv S* $\Longrightarrow$ *inv T*
begin

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

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

sublocale *dpll-with-backjumping-ops trail* $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
inv backjump-conds propagate-conds
⟨*proof*⟩

end

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

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

locale $cdcl_{NOT}$*-merge-bj-learn* =
$cdcl_{NOT}$*-merge-bj-learn-proxy2 trail* $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
propagate-conds forget-cond backjump-l-cond inv
for
trail :: *'st* $\Rightarrow$ (*'v*, *unit*) *ann-lits* **and**
$clauses_{NOT}$ :: *'st* $\Rightarrow$ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*) *ann-lit* $\Rightarrow$ *'st* $\Rightarrow$ *'st* **and**

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

sublocale
$conflict\text{-}driven\text{-}clause\text{-}learning\ trail\ clauses_{NOT}\ prepend\text{-}trail\ tl\text{-}trail\ add\text{-}cls_{NOT}\ remove\text{-}cls_{NOT}$
$inv\ backjump\text{-}conds\ propagate\text{-}conds$
$\lambda C\ \text{-}.\ distinct\text{-}mset\ C \wedge \neg tautology\ C$
$forget\text{-}cond$
$\langle proof \rangle$

lemma $backjump\text{-}l\text{-}learn\text{-}backjump$:
assumes bt: $backjump\text{-}l\ S\ T$ **and** inv: $inv\ S$ **and** $n\text{-}d$: $no\text{-}dup\ (trail\ S)$
shows $\exists\ C'\ L\ D.\ learn\ S\ (add\text{-}cls_{NOT}\ D\ S)$
$\wedge\ D = (C' + \{\#L\#\})$
$\wedge\ backjump\ (add\text{-}cls_{NOT}\ D\ S)\ T$
$\wedge\ atms\text{-}of\ (C' + \{\#L\#\}) \subseteq atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \cup atm\text{-}of\ `\ (lits\text{-}of\text{-}l\ (trail\ S))$
$\langle proof \rangle$

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

lemma $rtranclp\text{-}cdcl_{NOT}\text{-}merged\text{-}bj\text{-}learn\text{-}is\text{-}rtranclp\text{-}cdcl_{NOT}\text{-}and\text{-}inv$:
$cdcl_{NOT}\text{-}merged\text{-}bj\text{-}learn^{**}\ S\ T \Longrightarrow inv\ S \Longrightarrow no\text{-}dup\ (trail\ S) \Longrightarrow {cdcl_{NOT}}^{**}\ S\ T \wedge inv\ T$
$\langle proof \rangle$

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

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

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

definition $\mu_{CDCL}{}'\text{-}merged :: 'v\ clause\ set \Rightarrow 'st \Rightarrow nat$ **where**
$\mu_{CDCL}{}'\text{-}merged\ A\ T \equiv$
$((2{+}card\ (atms\text{-}of\text{-}ms\ A))\ \hat{}\ (1{+}card\ (atms\text{-}of\text{-}ms\ A)) - \mu_C{}'\ A\ T) * 2 + card\ (set\text{-}mset\ (clauses_{NOT}\ T))$

lemma $cdcl_{NOT}\text{-}decreasing\text{-}measure'$:
assumes
$cdcl_{NOT}\text{-}merged\text{-}bj\text{-}learn\ S\ T$ **and**
inv: $inv\ S$ **and**
$atm\text{-}clss$: $atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \subseteq atms\text{-}of\text{-}ms\ A$ **and**

atm-trail: *atm-of ' lits-of-l* (*trail S*) ⊆ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
fin-A: *finite A*
shows μ_{CDCL}*'-merged A T* < μ_{CDCL}*'-merged A S*
⟨*proof*⟩

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

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

lemma *backjump-no-step-backjump-l*:
backjump S T ⟹ *inv S* ⟹ ¬*no-step backjump-l S*
⟨*proof*⟩

lemma *cdcl*$_{NOT}$*-merged-bj-learn-final-state*:
fixes *A* :: *'v clause set* **and** *S T* :: *'st*
assumes
n-s: *no-step cdcl*$_{NOT}$*-merged-bj-learn S* **and**
atms-S: *atms-of-mm* (*clauses*$_{NOT}$ *S*) ⊆ *atms-of-ms A* **and**
atms-trail: *atm-of ' lits-of-l* (*trail S*) ⊆ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite A **and**
inv: *inv S* **and**
decomp: *all-decomposition-implies-m* (*clauses*$_{NOT}$ *S*) (*get-all-ann-decomposition* (*trail S*))
shows *unsatisfiable* (*set-mset* (*clauses*$_{NOT}$ *S*))
∨ (*trail S* ⊨*asm clauses*$_{NOT}$ *S* ∧ *satisfiable* (*set-mset* (*clauses*$_{NOT}$ *S*)))
⟨*proof*⟩

lemma *full-cdcl*$_{NOT}$*-merged-bj-learn-final-state*:

fixes A :: $'v$ *clause set* **and** S T :: $'st$
assumes
full: *full* $cdcl_{NOT}$*-merged-bj-learn* S T **and**
atms-S: *atms-of-mm* ($clauses_{NOT}$ S) $\subseteq$ *atms-of-ms* A **and**
atms-trail: *atm-of* ‘ *lits-of-l* (*trail* S) $\subseteq$ *atms-of-ms* A **and**
n-d: *no-dup* (*trail* S) **and**
finite A **and**
inv: *inv* S **and**
decomp: *all-decomposition-implies-m* ($clauses_{NOT}$ S) (*get-all-ann-decomposition* (*trail* S))
shows *unsatisfiable* (*set-mset* ($clauses_{NOT}$ T))
$\vee$ (*trail* T $\models$*asm* $clauses_{NOT}$ T $\wedge$ *satisfiable* (*set-mset* ($clauses_{NOT}$ T)))
⟨*proof*⟩

end

1.2.7 Instantiations

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

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

lemma *bound-inv-inv*:
assumes
inv S **and**
n-d: *no-dup* (*trail* S) **and**
atms-clss-S-A: *atms-of-mm* ($clauses_{NOT}$ S) $\subseteq$ *atms-of-ms* A **and**
atms-trail-S-A:*atm-of* ‘ *lits-of-l* (*trail* S) $\subseteq$ *atms-of-ms* A **and**
finite A **and**
$cdcl_{NOT}$: $cdcl_{NOT}$ S T
shows
atms-of-mm ($clauses_{NOT}$ T) $\subseteq$ *atms-of-ms* A **and**
atm-of ‘ *lits-of-l* (*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-mm (clauses$_{NOT}$ S) $\subseteq$ atms-of-ms A $\wedge$ atm-of ‘ lits-of-l (trail S) $\subseteq$ atms-of-ms A $\wedge$
finite A
$\mu_{CDCL}{}'$ $\lambda S.$ inv S $\wedge$ no-dup (trail S)
$\mu_{CDCL}{}'$-bound
$\langle proof \rangle$

lemma cdcl$_{NOT}$-with-restart-$\mu_{CDCL}{}'$-le-$\mu_{CDCL}{}'$-bound:
assumes
cdcl$_{NOT}$: cdcl$_{NOT}$-restart (T, a) (V, b) **and**
cdcl$_{NOT}$-inv:
inv T
no-dup (trail T) **and**
bound-inv:
atms-of-mm (clauses$_{NOT}$ T) $\subseteq$ atms-of-ms A
atm-of ‘ lits-of-l (trail T) $\subseteq$ atms-of-ms A
finite A
shows $\mu_{CDCL}{}'$ A V $\leq$ $\mu_{CDCL}{}'$-bound A T
$\langle proof \rangle$

lemma cdcl$_{NOT}$-with-restart-$\mu_{CDCL}{}'$-bound-le-$\mu_{CDCL}{}'$-bound:
assumes
cdcl$_{NOT}$: cdcl$_{NOT}$-restart (T, a) (V, b) **and**
cdcl$_{NOT}$-inv:
inv T
no-dup (trail T) **and**
bound-inv:
atms-of-mm (clauses$_{NOT}$ T) $\subseteq$ atms-of-ms A
atm-of ‘ lits-of-l (trail T) $\subseteq$ atms-of-ms A
finite A
shows $\mu_{CDCL}{}'$-bound A V $\leq$ $\mu_{CDCL}{}'$-bound A T
$\langle proof \rangle$

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

lemma cdcl$_{NOT}$-restart-all-decomposition-implies:
assumes cdcl$_{NOT}$-restart S T **and**
inv (fst S) **and**
no-dup (trail (fst S))
all-decomposition-implies-m (clauses$_{NOT}$ (fst S)) (get-all-ann-decomposition (trail (fst S)))
shows
all-decomposition-implies-m (clauses$_{NOT}$ (fst T)) (get-all-ann-decomposition (trail (fst T)))
$\langle proof \rangle$

lemma rtranclp-cdcl$_{NOT}$-restart-all-decomposition-implies:
assumes cdcl$_{NOT}$-restart** S T **and**
inv: inv (fst S) **and**
n-d: no-dup (trail (fst S)) **and**

decomp:
all-decomposition-implies-m ($clauses_{NOT}$ (*fst S*)) (*get-all-ann-decomposition* (*trail* (*fst S*)))
shows
all-decomposition-implies-m ($clauses_{NOT}$ (*fst T*)) (*get-all-ann-decomposition* (*trail* (*fst T*)))
⟨*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* ⊨*sextm* $clauses_{NOT}$ (*fst S*) ⟷ *I* ⊨*sextm* $clauses_{NOT}$ (*fst T*)
⟨*proof*⟩

lemma *rtranclp-*$cdcl_{NOT}$*-restart-sat-ext-iff*:
fixes *S T* :: *'st* × *nat*
assumes
st: $cdcl_{NOT}$*-restart*** *S T* **and**
n-d: *no-dup* (*trail* (*fst S*)) **and**
inv: *inv* (*fst S*)
shows *I* ⊨*sextm* $clauses_{NOT}$ (*fst S*) ⟷ *I* ⊨*sextm* $clauses_{NOT}$ (*fst T*)
⟨*proof*⟩

theorem *full-*$cdcl_{NOT}$*-restart-backjump-final-state*:
fixes *A* :: *'v clause set* **and** *S T* :: *'st*
assumes
full: *full* $cdcl_{NOT}$*-restart* (*S*, *n*) (*T*, *m*) **and**
atms-S: *atms-of-mm* ($clauses_{NOT}$ *S*) ⊆ *atms-of-ms A* **and**
atms-trail: *atm-of* ' *lits-of-l* (*trail S*) ⊆ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
fin-A[*simp*]: *finite A* **and**
inv: *inv S* **and**
decomp: *all-decomposition-implies-m* ($clauses_{NOT}$ *S*) (*get-all-ann-decomposition* (*trail S*))
shows *unsatisfiable* (*set-mset* ($clauses_{NOT}$ *S*))
∨ (*lits-of-l* (*trail T*) ⊨*sextm* $clauses_{NOT}$ *S* ∧ *satisfiable* (*set-mset* ($clauses_{NOT}$ *S*)))
⟨*proof*⟩
end — end of $cdcl_{NOT}$*-with-backtrack-and-restarts* locale

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

locale $cdcl_{NOT}$*-merge-bj-learn-with-backtrack-restarts* =
$cdcl_{NOT}$*-merge-bj-learn trail* $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
λ*C C' L' S T*. *distinct-mset* (*C'* + {#*L'*#}) ∧ *backjump-l-cond C C' L' S T*
propagate-conds forget-conds inv
for
trail :: *'st* ⇒ (*'v*, *unit*) *ann-lits* **and**
$clauses_{NOT}$ :: *'st* ⇒ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
$add\text{-}cls_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
$remove\text{-}cls_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
propagate-conds :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *bool* **and**
inv :: *'st* ⇒ *bool* **and**
forget-conds :: *'v clause* ⇒ *'st* ⇒ *bool* **and**
backjump-l-cond :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool*
+

fixes f :: *nat* $\Rightarrow$ *nat*
assumes
unbounded: *unbounded* f **and** *f-ge-1*: $\bigwedge n.\ n \geq 1 \Longrightarrow f\ n \geq 1$ **and**
inv-restart: $\bigwedge S\ T.\ inv\ S \Longrightarrow T \sim$ *reduce-trail-to*$_{NOT}$ [] $S \Longrightarrow inv\ T$
begin

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

lemma *simple-clss-or-not-simplified-cls*:
assumes *atms-of-mm* (*clauses*$_{NOT}$ S) $\subseteq$ *atms-of-ms* A **and**
$x \in\#$ *clauses*$_{NOT}$ S **and** *finite* A
shows $x \in$ *simple-clss* (*atms-of-ms* A) $\vee$ $x \in\#$ *not-simplified-cls* (*clauses*$_{NOT}$ S)
⟨*proof*⟩

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

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

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

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

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

lemma *rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-clauses-bound-card*:
assumes
cdcl$_{NOT}$*-merged-bj-learn*** S T **and**

inv S **and**
atms-of-mm (*clauses*$_{NOT}$ *S*) $\subseteq$ *atms-of-ms A* **and**
atm-of ‘(*lits-of-l* (*trail S*)) $\subseteq$ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite A*
shows μ_{CDCL}*'-merged A T* $\leq$ μ_{CDCL}*'-bound A S*
⟨*proof*⟩

sublocale *cdcl*$_{NOT}$*-increasing-restarts-ops* λS *T*. *T* $\sim$ *reduce-trail-to*$_{NOT}$ ([]::*'a list*) *S*
cdcl$_{NOT}$*-merged-bj-learn f*
λA *S*. *atms-of-mm* (*clauses*$_{NOT}$ *S*) $\subseteq$ *atms-of-ms A*
$\wedge$ *atm-of* ‘ *lits-of-l* (*trail S*) $\subseteq$ *atms-of-ms A* $\wedge$ *finite A*
μ_{CDCL}*'-merged*
λS. *inv S* $\wedge$ *no-dup* (*trail S*)
μ_{CDCL}*'-bound*
⟨*proof*⟩

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

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

sublocale *cdcl*$_{NOT}$*-increasing-restarts* - - - - - - *f*
λS *T*. *T* $\sim$ *reduce-trail-to*$_{NOT}$ ([]::*'a list*) *S*
λA *S*. *atms-of-mm* (*clauses*$_{NOT}$ *S*) $\subseteq$ *atms-of-ms A*
$\wedge$ *atm-of* ‘ *lits-of-l* (*trail S*) $\subseteq$ *atms-of-ms A* $\wedge$ *finite A*
μ_{CDCL}*'-merged cdcl*$_{NOT}$*-merged-bj-learn*
λS. *inv S* $\wedge$ *no-dup* (*trail S*)
λA *T*. ((*2*+*card* (*atms-of-ms A*)) ^ (*1*+*card* (*atms-of-ms A*))) $*$ *2*
\+ *card* (*set-mset* (*not-simplified-cls*(*clauses*$_{NOT}$ *T*)))
\+ *3* ^ *card* (*atms-of-ms A*)
⟨*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*$_{NOT}$ (*fst S*) $\longleftrightarrow$ *I* $\models$*sextm clauses*$_{NOT}$ (*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$_{NOT}$* (*fst S*) $\longleftrightarrow$ $I \models$*sextm clauses$_{NOT}$* (*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$_{NOT}$* (*fst S*))
      (*get-all-ann-decomposition* (*trail* (*fst S*)))
  shows *all-decomposition-implies-m* (*clauses$_{NOT}$* (*fst T*))
      (*get-all-ann-decomposition* (*trail* (*fst T*)))
  ⟨*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$_{NOT}$* (*fst S*))
      (*get-all-ann-decomposition* (*trail* (*fst S*)))
  shows *all-decomposition-implies-m* (*clauses$_{NOT}$* (*fst T*))
      (*get-all-ann-decomposition* (*trail* (*fst T*)))
  ⟨*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$_{NOT}$* (*fst S*))
      (*get-all-ann-decomposition* (*trail* (*fst S*))) **and**
    atms-cls: *atms-of-mm* (*clauses$_{NOT}$* (*fst S*)) $\subseteq$ *atms-of-ms A* **and**
    atms-trail: *atm-of ‘ lits-of-l* (*trail* (*fst S*)) $\subseteq$ *atms-of-ms A* **and**
    fin: *finite A*
  shows *unsatisfiable* (*set-mset* (*clauses$_{NOT}$* (*fst S*)))
    $\vee$ *lits-of-l* (*trail* (*fst T*)) $\models$*sextm clauses$_{NOT}$* (*fst S*) $\wedge$ *satisfiable* (*set-mset* (*clauses$_{NOT}$* (*fst S*)))
⟨*proof*⟩

corollary *full-cdcl$_{NOT}$-restart-normal-form-init-state*:
  assumes
    init-state: *trail S* = [] *clauses$_{NOT}$ S* = *N* **and**
    full: *full cdcl$_{NOT}$-restart* (*S*, *0*) *T* **and**
    inv: *inv S*
  shows *unsatisfiable* (*set-mset N*)
    $\vee$ *lits-of-l* (*trail* (*fst T*)) $\models$*sextm N* $\wedge$ *satisfiable* (*set-mset N*)
  ⟨*proof*⟩

end

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

1.3 DPLL as an instance of NOT

1.3.1 DPLL with simple backtrack

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

locale *dpll-with-backtrack*
begin
inductive *backtrack* :: $('v,\ unit)\ ann\text{-}lits \times 'v\ clauses$
  $\Rightarrow ('v,\ unit)\ ann\text{-}lits \times 'v\ clauses \Rightarrow bool$ **where**
$backtrack\text{-}split\ (fst\ S) = (M',\ L\ \#\ M) \Longrightarrow is\text{-}decided\ L \Longrightarrow D \in\#\ snd\ S$
  $\Longrightarrow fst\ S \models as\ CNot\ D \Longrightarrow backtrack\ S\ (Propagated\ (-\ (lit\text{-}of\ L))\ ()\ \#\ M,\ snd\ S)$

inductive-cases *backtrackE*[*elim*]: $backtrack\ (M,\ N)\ (M',\ N')$
lemma *backtrack-is-backjump*:
  fixes $M\ M' :: ('v,\ unit)\ ann\text{-}lits$
  assumes
    backtrack: $backtrack\ (M,\ N)\ (M',\ N')$ **and**
    no-dup: $(no\text{-}dup \circ fst)\ (M,\ N)$ **and**
    decomp: $all\text{-}decomposition\text{-}implies\text{-}m\ N\ (get\text{-}all\text{-}ann\text{-}decomposition\ M)$
    shows
      $\exists\, C\ F'\ K\ F\ L\ l\ C'.$
        $M = F'\ @\ Decided\ K\ \#\ F\ \wedge$
        $M' = Propagated\ L\ l\ \#\ F \wedge N = N' \wedge C \in\#\ N \wedge F'\ @\ Decided\ K\ \#\ F \models as\ CNot\ C\ \wedge$
        $undefined\text{-}lit\ F\ L \wedge atm\text{-}of\ L \in atms\text{-}of\text{-}mm\ N \cup atm\text{-}of\ `\ lits\text{-}of\text{-}l\ (F'\ @\ Decided\ K\ \#\ F)\ \wedge$
        $N \models pm\ C' + \{\#L\#\} \wedge F \models as\ CNot\ C'$
⟨*proof*⟩

lemma *backtrack-is-backjump'*:
  fixes $M\ M' :: ('v,\ unit)\ ann\text{-}lits$
  assumes
    backtrack: $backtrack\ S\ T$ **and**
    no-dup: $(no\text{-}dup \circ fst)\ S$ **and**
    decomp: $all\text{-}decomposition\text{-}implies\text{-}m\ (snd\ S)\ (get\text{-}all\text{-}ann\text{-}decomposition\ (fst\ S))$
    shows
      $\exists\, C\ F'\ K\ F\ L\ l\ C'.$
        $fst\ S = F'\ @\ Decided\ K\ \#\ F\ \wedge$
        $T = (Propagated\ L\ l\ \#\ F,\ snd\ S) \wedge C \in\#\ snd\ S \wedge fst\ S \models as\ CNot\ C$
        $\wedge\ undefined\text{-}lit\ F\ L \wedge atm\text{-}of\ L \in atms\text{-}of\text{-}mm\ (snd\ S) \cup atm\text{-}of\ `\ lits\text{-}of\text{-}l\ (fst\ S)\ \wedge$
        $snd\ S \models pm\ C' + \{\#L\#\} \wedge F \models as\ CNot\ C'$
  ⟨*proof*⟩

sublocale *dpll-state*
  $fst\ snd\ \lambda L\ (M,\ N).\ (L\ \#\ M,\ N)\ \lambda(M,\ N).\ (tl\ M,\ N)$
  $\lambda C\ (M,\ N).\ (M,\ \{\#C\#\} + N)\ \lambda C\ (M,\ N).\ (M,\ removeAll\text{-}mset\ C\ N)$
  ⟨*proof*⟩

sublocale *backjumping-ops*
  $fst\ snd\ \lambda L\ (M,\ N).\ (L\ \#\ M,\ N)\ \lambda(M,\ N).\ (tl\ M,\ N)$
  $\lambda C\ (M,\ N).\ (M,\ \{\#C\#\} + N)\ \lambda C\ (M,\ N).\ (M,\ removeAll\text{-}mset\ C\ N)\ \lambda\text{-}\ \text{-}\ \text{-}\ S\ T.\ backtrack\ S\ T$
  ⟨*proof*⟩
thm $reduce\text{-}trail\text{-}to_{NOT}\text{-}clauses$

lemma $reduce\text{-}trail\text{-}to_{NOT}$:
  $reduce\text{-}trail\text{-}to_{NOT}\ F\ S =$
    $(if\ length\ (fst\ S) \geq length\ F$

then drop (*length* (*fst S*) − *length F*) (*fst S*)
else [],
snd S) (**is** *?R* = *?C*)
⟨*proof*⟩

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

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

end

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

sublocale *dpll-with-backtrack* ⊆ *dpll-with-backjumping*
fst snd λ*L* (*M*, *N*). (*L* # *M*, *N*)
λ(*M*, *N*). (*tl M*, *N*) λ*C* (*M*, *N*). (*M*, {#*C*#} + *N*) λ*C* (*M*, *N*). (*M*, *removeAll-mset C N*)
λ(*M*, *N*). *no-dup M* ∧ *all-decomposition-implies-m N* (*get-all-ann-decomposition M*)
λ- - - *S T*. *backtrack S T*
λ- -. *True*
⟨*proof*⟩

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

corollary *full-dpll-final-state-conclusive*:
fixes M M' :: ($'v$, *unit*) *ann-lits*
assumes
full: *full dpll-bj* ([], *N*) (M', N')
shows *unsatisfiable* (*set-mset N*) ∨ (M' ⊨*asm N* ∧ *satisfiable* (*set-mset N*))
⟨*proof*⟩

corollary *full-dpll-normal-form-from-init-state*:

fixes $M\ M' :: ('v,\ unit)\ ann\text{-}lits$
assumes
full: $full\ dpll\text{-}bj\ ([],\ N)\ (M',\ N')$
shows $M' \models asm\ N \longleftrightarrow satisfiable\ (set\text{-}mset\ N)$
⟨*proof*⟩

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

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

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

Another proof of termination:

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

1.3.2 Adding restarts

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

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

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

begin

1.4 Weidenbach's DPLL

1.4.1 Rules

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

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

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

1.4.2 Invariants

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

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

lemma $dpll_W$*-vars-in-snd-inv*:
assumes $dpll_W$ S S'
and *atm-of* ` (*lits-of-l* (*trail* S)) $\subseteq$ *atms-of-mm* (*clauses* S)
shows *atm-of* ` (*lits-of-l* (*trail* S')) $\subseteq$ *atms-of-mm* (*clauses* S')
⟨*proof*⟩

lemma *atms-of-ms-lit-of-atms-of*: *atms-of-ms* (($\lambda a.$ {#*lit-of* a#}) ` $c$) = *atm-of* ` *lit-of* ` c
⟨*proof*⟩

theorem 2.8.2 page 73 of Weidenbach's book

lemma $dpll_W$*-propagate-is-conclusion*:
assumes $dpll_W$ S S'
and *all-decomposition-implies-m* (*clauses* S) (*get-all-ann-decomposition* (*trail* S))
and *atm-of* ` *lits-of-l* (*trail* S) $\subseteq$ *atms-of-mm* (*clauses* S)
shows *all-decomposition-implies-m* (*clauses* S') (*get-all-ann-decomposition* (*trail* S'))
⟨*proof*⟩

theorem 2.8.3 page 73 of Weidenbach's book

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

theorem 2.8.4 page 73 of Weidenbach's book

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

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-ann-decomposition* (*trail* S))
  and *atm-incl*: *atm-of* ' *lits-of-l* (*trail* S) $\subseteq$ *atms-of-mm* (*clauses* S)
  and *consistent-interp* (*lits-of-l* (*trail* S))
  and *no-dup* (*trail* S)
  shows *all-decomposition-implies-m* (*clauses* S') (*get-all-ann-decomposition* (*trail* S'))
  and *atm-of* ' *lits-of-l* (*trail* S') $\subseteq$ *atms-of-mm* (*clauses* S')
  and *clauses* S = *clauses* S'
  and *consistent-interp* (*lits-of-l* (*trail* S'))
  and *no-dup* (*trail* S')
  ⟨*proof*⟩

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

lemma $dpll_W$*-all-inv-dest*[*dest*]:
  assumes $dpll_W$*-all-inv* S
  shows *all-decomposition-implies-m* (*clauses* S) (*get-all-ann-decomposition* (*trail* S))
  and *atm-of* ' *lits-of-l* (*trail* S) $\subseteq$ *atms-of-mm* (*clauses* S)
  and *consistent-interp* (*lits-of-l* (*trail* S)) $\wedge$ *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-mm N*
shows *rtranclp* $dpll_W$ ([], *N*) (*map Decided 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*)))

theorem 2.8.6 page 74 of Weidenbach's book

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

theorem 2.8.5 page 73 of Weidenbach's book

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

1.4.3 Termination

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

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

lemma *distinctcard-atm-of-lit-of-eq-length*:
assumes *no-dup S*
shows *card* (*atm-of* ‘ *lits-of-l 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) $\leq$ *card vars*
shows ($dpll_W$*-mes* (*trail* S') (*card vars*), $dpll_W$*-mes* (*trail* S) (*card vars*))
$\in$ *lexn* $\{(a, b).\ a < b\}$ (*card vars*)
⟨*proof*⟩

theorem 2.8.7 page 74 of Weidenbach's book

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

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

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

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

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

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

1.4.4 Final States

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

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

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

1.4.5 Link with NOT's DPLL

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

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

Level of literals and clauses

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

abbreviation *count-decided* :: ('*v*, '*m*) *ann-lits* ⇒ *nat* **where**
count-decided l ≡ *length* (*filter is-decided* l)

abbreviation *get-level* :: ('*v*, '*m*) *ann-lits* ⇒ '*v literal* ⇒ *nat* **where**
get-level S L ≡ *length* (*filter is-decided* (*dropWhile* (λS. *atm-of* (*lit-of* S) ≠ *atm-of* L) S))

lemma *get-level-uminus*: *get-level* M $(-L)$ = *get-level* M L
⟨*proof*⟩

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

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

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

lemma *get-rev-level-skip*[*simp*]:

assumes $\mathit{atm\text{-}of}\ L \notin \mathit{atm\text{-}of}\ `\ \mathit{lits\text{-}of\text{-}l}\ M$
shows $\mathit{get\text{-}level}\ (M\ @\ M')\ L = \mathit{get\text{-}level}\ M'\ L$
$\langle \mathit{proof} \rangle$

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

lemma *get-rev-level-skip-end*[*simp*]:
assumes $\mathit{atm\text{-}of}\ L \in \mathit{atm\text{-}of}\ `\ \mathit{lits\text{-}of\text{-}l}\ M$
shows $\mathit{get\text{-}level}\ (M\ @\ M')\ L = \mathit{get\text{-}level}\ M\ L + \mathit{length}\ (\mathit{filter}\ \mathit{is\text{-}decided}\ M')$
$\langle \mathit{proof} \rangle$

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

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

lemma *get-level-skip-in-all-not-decided*:
fixes $M :: ('a, 'b)\ \mathit{ann\text{-}lits}$ **and** $L :: 'a\ \mathit{literal}$
assumes $\forall m \in \mathit{set}\ M.\ \neg\ \mathit{is\text{-}decided}\ m$
and $\mathit{atm\text{-}of}\ L \in \mathit{atm\text{-}of}\ `\ \mathit{lits\text{-}of\text{-}l}\ M$
shows $\mathit{get\text{-}level}\ M\ L = \mathit{0}$
$\langle \mathit{proof} \rangle$

lemma *get-level-skip-all-not-decided*[*simp*]:
fixes M
assumes $\forall m \in \mathit{set}\ M.\ \neg\ \mathit{is\text{-}decided}\ m$
shows $\mathit{get\text{-}level}\ M\ L = \mathit{0}$
$\langle \mathit{proof} \rangle$

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

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

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

lemma *get-maximum-level-ge-get-level*:
$L \in\#\ D \Longrightarrow \mathit{get\text{-}maximum\text{-}level}\ M\ D \geq \mathit{get\text{-}level}\ M\ L$
$\langle \mathit{proof} \rangle$

lemma *get-maximum-level-empty*[*simp*]:
$\mathit{get\text{-}maximum\text{-}level}\ M\ \{\#\} = \mathit{0}$
$\langle \mathit{proof} \rangle$

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

lemma *get-maximum-level-empty-list*[*simp*]:
$\mathit{get\text{-}maximum\text{-}level}\ []\ D = \mathit{0}$
$\langle \mathit{proof} \rangle$

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 ∃ *L* ∈#*D*. *get-level M L* = *n*
⟨*proof*⟩

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

lemma *get-maximum-level-skip-beginning*:
assumes *DH*: ∀ *x* ∈ *atms-of D*. *x* ∉ *atm-of* ‘ *lits-of-l c*
shows *get-maximum-level* (*c* @ *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-un-decided-not-present*:
assumes
∀ *L*∈#*D*. *atm-of L* ∉ *atm-of* ‘ *lits-of-l M* **and**
∀ *m*∈*set M*. ¬ *is-decided m*
shows *get-maximum-level* (*M* @ *aa*) *D* = *get-maximum-level aa D*
⟨*proof*⟩

lemma *get-maximum-level-union-mset*:
get-maximum-level M (*A* #∪ *B*) = *get-maximum-level M* (*A* + *B*)
⟨*proof*⟩

lemma *count-decided-rev*[*simp*]:
count-decided (*rev M*) = *count-decided M*
⟨*proof*⟩

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

lemma *count-decided-ge-get-maximum-level*:
count-decided 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*⟩

Properties about the levels

lemma *atm-lit-of-set-lits-of-l*:
$(\lambda l.$ *atm-of* (*lit-of* l)) ‘ *set* xs = *atm-of* ‘ *lits-of-l* xs
⟨*proof*⟩

lemma *le-count-decided-decomp*:
assumes *no-dup* M
shows $i <$ *count-decided* $M \longleftrightarrow (\exists\, c\ K\ c'.\ M = c\ @\ Decided\ K\ \#\ c' \wedge$ *get-level* $M\ K = Suc\ i)$
(**is** $?A \longleftrightarrow ?B$)
⟨*proof*⟩

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

begin

Chapter 2

Weidenbach's CDCL

The organisation of the development is the following:

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

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

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

2.1 Weidenbach's CDCL with Multisets

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

2.1.1 The State

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

locale $state_W$*-ops* =

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

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

init-state :: *'v clauses* ⇒ *'st* **and**
restart-state :: *'st* ⇒ *'st*
begin
abbreviation *hd-trail* :: *'st* ⇒ (*'v*, *'v clause*) *ann-lit* **where**
hd-trail S ≡ *hd* (*trail S*)

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

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

end

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

Weidenbach state is a five-tuple composed of:

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

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

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

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

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

— Some specific states:
init-state
restart-state
for
trail :: $'st \Rightarrow ('v, 'v\ clause)\ ann\text{-}lits$ **and**
init-clss :: $'st \Rightarrow 'v\ clauses$ **and**
learned-clss :: $'st \Rightarrow 'v\ clauses$ **and**
backtrack-lvl :: $'st \Rightarrow nat$ **and**
conflicting :: $'st \Rightarrow 'v\ clause\ option$ **and**

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

init-state :: $'v\ clauses \Rightarrow 'st$ **and**
restart-state :: $'st \Rightarrow 'st$ +
assumes
trail-cons-trail[*simp*]:
$\bigwedge L\ st.\ trail\ (cons\text{-}trail\ L\ st) = L\ \#\ trail\ st$ **and**
trail-tl-trail[*simp*]: $\bigwedge st.\ trail\ (tl\text{-}trail\ st) = tl\ (trail\ st)$ **and**
trail-add-learned-cls[*simp*]:
$\bigwedge C\ st.\ trail\ (add\text{-}learned\text{-}cls\ C\ st) = trail\ st$ **and**
trail-remove-cls[*simp*]:
$\bigwedge C\ st.\ trail\ (remove\text{-}cls\ C\ st) = trail\ st$ **and**
trail-update-backtrack-lvl[*simp*]: $\bigwedge st\ C.\ trail\ (update\text{-}backtrack\text{-}lvl\ C\ st) = trail\ st$ **and**
trail-update-conflicting[*simp*]: $\bigwedge C\ st.\ trail\ (update\text{-}conflicting\ C\ st) = trail\ st$ **and**

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

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

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

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

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

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

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

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

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

$clauses\ (add\text{-}learned\text{-}cls\ U\ S) = \{\#U\#\} + learned\text{-}clss\ S + init\text{-}clss\ S$
and
clauses-update-backtrack-lvl[*simp*]: $clauses\ (update\text{-}backtrack\text{-}lvl\ k\ S) = clauses\ S$ **and**
clauses-update-conflicting[*simp*]: $clauses\ (update\text{-}conflicting\ D\ S) = clauses\ S$ **and**
clauses-remove-cls[*simp*]:
$clauses\ (remove\text{-}cls\ C\ S) = removeAll\text{-}mset\ C\ (clauses\ S)$ **and**
clauses-add-learned-cls[*simp*]:
$clauses\ (add\text{-}learned\text{-}cls\ C\ S) = \{\#C\#\} + clauses\ S$ **and**
clauses-restart[*simp*]: $clauses\ (restart\text{-}state\ S) \subseteq\# clauses\ S$ **and**
clauses-init-state[*simp*]: $clauses\ (init\text{-}state\ N) = N$
⟨*proof*⟩

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

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

definition $state\text{-}eq :: 'st \Rightarrow 'st \Rightarrow bool$ (**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: $S \sim T \Longrightarrow init\text{-}clss\ S = init\text{-}clss\ T$ **and**
state-eq-learned-clss: $S \sim T \Longrightarrow learned\text{-}clss\ S = learned\text{-}clss\ T$ **and**
state-eq-backtrack-lvl: $S \sim T \Longrightarrow backtrack\text{-}lvl\ S = backtrack\text{-}lvl\ T$ **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: $S \sim T \Longrightarrow undefined\text{-}lit\ (trail\ S)\ L = undefined\text{-}lit\ (trail\ T)\ L$
⟨*proof*⟩

lemma *state-eq-conflicting-None*:
$S \sim T \Longrightarrow conflicting\ T = None \Longrightarrow conflicting\ S = None$
⟨*proof*⟩

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

lemmas *state-simp*[*simp*] = *state-eq-trail state-eq-init-clss state-eq-learned-clss state-eq-backtrack-lvl state-eq-conflicting state-eq-clauses state-eq-undefined-lit state-eq-conflicting-None*

lemma *atms-of-ms-learned-clss-restart-state-in-atms-of-ms-learned-clssI*[*intro*]:
$x \in$ *atms-of-mm* (*learned-clss* (*restart-state* S)) $\Longrightarrow x \in$ *atms-of-mm* (*learned-clss* S)
⟨*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-trail*[*simp*]:
conflicting (*reduce-trail-to* F S) = *conflicting* S
⟨*proof*⟩

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

lemma *init-clss-update-trail*[*simp*]:

$init\text{-}clss\ (reduce\text{-}trail\text{-}to\ F\ S) = init\text{-}clss\ S$
⟨proof⟩

lemma *learned-clss-update-trail*[*simp*]:
$learned\text{-}clss\ (reduce\text{-}trail\text{-}to\ F\ S) = learned\text{-}clss\ S$
⟨proof⟩

lemma *conflicting-reduce-trail-to*[*simp*]:
$conflicting\ (reduce\text{-}trail\text{-}to\ F\ S) = None \longleftrightarrow conflicting\ S = None$
⟨proof⟩

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

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

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

lemma *reduce-trail-to-add-learned-cls*[*simp*]:
$trail\ (reduce\text{-}trail\text{-}to\ F\ (add\text{-}learned\text{-}cls\ C\ S)) = trail\ (reduce\text{-}trail\text{-}to\ F\ S)$
⟨proof⟩

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

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

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

lemma *reduce-trail-to-length*:
$length\ M = length\ M' \Longrightarrow reduce\text{-}trail\text{-}to\ M\ S = reduce\text{-}trail\text{-}to\ M'\ S$
⟨proof⟩

lemma *trail-reduce-trail-to-drop*:
$trail\ (reduce\text{-}trail\text{-}to\ F\ S) =$
$(if\ length\ (trail\ S) \geq length\ F$
$then\ drop\ (length\ (trail\ S) - length\ F)\ (trail\ S)$
$else\ [])$
⟨proof⟩

lemma *in-get-all-ann-decomposition-trail-update-trail*[*simp*]:
assumes H: $(L\ \#\ M1,\ M2) \in set\ (get\text{-}all\text{-}ann\text{-}decomposition\ (trail\ S))$
shows $trail\ (reduce\text{-}trail\text{-}to\ M1\ S) = M1$
⟨proof⟩

lemma *conflicting-cons-trail-conflicting*[*simp*]:
  assumes *undefined-lit* (*trail S*) (*lit-of L*)
  shows
    conflicting (*cons-trail L S*) = *None* ⟷ *conflicting S* = *None*
  ⟨*proof*⟩

lemma *conflicting-add-learned-cls-conflicting*[*simp*]:
  conflicting (*add-learned-cls C S*) = *None* ⟷ *conflicting S* = *None*
  ⟨*proof*⟩

lemma *conflicting-update-backtracl-lvl*[*simp*]:
  conflicting (*update-backtrack-lvl k S*) = *None* ⟷ *conflicting S* = *None*
  ⟨*proof*⟩

end — end of $state_W$ locale

2.1.2 CDCL Rules

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

locale $conflict\text{-}driven\text{-}clause\text{-}learning_W$ =
  $state_W$
    — functions for the state:
      — access functions:
    trail init-clss learned-clss backtrack-lvl conflicting
      — changing state:
    cons-trail tl-trail add-learned-cls remove-cls update-backtrack-lvl
    update-conflicting

      — get state:
    init-state
    restart-state
  for
    trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **and**
    init-clss :: *'st* ⇒ *'v clauses* **and**
    learned-clss :: *'st* ⇒ *'v clauses* **and**
    backtrack-lvl :: *'st* ⇒ *nat* **and**
    conflicting :: *'st* ⇒ *'v clause option* **and**

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

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

inductive *propagate* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
propagate-rule: *conflicting S* = *None* ⟹
  E ∈# *clauses S* ⟹
  L ∈# *E* ⟹
  trail S ⊨*as CNot* (*E* − {#*L*#}) ⟹
  undefined-lit (*trail S*) *L* ⟹

$T \sim$ *cons-trail* (*Propagated L E*) $S \Longrightarrow$
propagate S T

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

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

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

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

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

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

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

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

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

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

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

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

inductive *restart* :: *'st* ⇒ *'st* ⇒ *bool* **for** S :: *'st* **where**
restart: *state* S = (M, N, U, k, *None*) ⟹ ¬M ⊨*asm clauses* S
  ⟹ T ∼ *restart-state* S
  ⟹ *restart* S T

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

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

inductive *forget* :: *'st* ⇒ *'st* ⇒ *bool* **where**
forget-rule:
  conflicting S = *None* ⟹
  C ∈# *learned-clss* S ⟹
  ¬(*trail* S) ⊨*asm clauses* S ⟹
  C ∉ *set* (*get-all-mark-of-propagated* (*trail* S)) ⟹
  C ∉# *init-clss* S ⟹
  T ∼ *remove-cls* C S ⟹
  forget S T

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

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

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

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

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

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

lemma *rtranclp-propagate-is-rtranclp-cdcl$_W$*:
propagate** S S' $\Longrightarrow$ *cdcl$_W$*** S S'
⟨*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: $\bigwedge T$. *propagate* S T $\Longrightarrow$ P S T **and**
conflict: $\bigwedge T$. *conflict* S T $\Longrightarrow$ P S T **and**
forget: $\bigwedge T$. *forget* S T $\Longrightarrow$ P S T **and**
restart: $\bigwedge T$. *restart* S T $\Longrightarrow$ P S T **and**
decide: $\bigwedge T$. *decide* S T $\Longrightarrow$ P S T **and**
skip: $\bigwedge T$. *skip* S T $\Longrightarrow$ P S T **and**
resolve: $\bigwedge T$. *resolve* S T $\Longrightarrow$ P S T **and**
backtrack: $\bigwedge T$. *backtrack* S T $\Longrightarrow$ P S T
shows P S S'
⟨*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: $\bigwedge C$ L T. *conflicting* S = *None* $\Longrightarrow$
C $\in\#$ *clauses* S $\Longrightarrow$
L $\in\#$ C $\Longrightarrow$
trail S $\models$*as* *CNot* (*remove1-mset* L C) $\Longrightarrow$
undefined-lit (*trail* S) L $\Longrightarrow$
T $\sim$ *cons-trail* (*Propagated* L C) S $\Longrightarrow$
P S T **and**
conflictH: $\bigwedge D$ T. *conflicting* S = *None* $\Longrightarrow$
D $\in\#$ *clauses* S $\Longrightarrow$
trail S $\models$*as* *CNot* D $\Longrightarrow$
T $\sim$ *update-conflicting* (*Some* D) S $\Longrightarrow$
P S T **and**
forgetH: $\bigwedge C$ T. *conflicting* S = *None* $\Longrightarrow$
C $\in\#$ *learned-clss* S $\Longrightarrow$
$\neg$(*trail* S) $\models$*asm* *clauses* S $\Longrightarrow$
C $\notin$ *set* (*get-all-mark-of-propagated* (*trail* S)) $\Longrightarrow$
C $\notin\#$ *init-clss* S $\Longrightarrow$
T $\sim$ *remove-cls* C S $\Longrightarrow$
P S T **and**
restartH: $\bigwedge T$. $\neg$*trail* S $\models$*asm* *clauses* S $\Longrightarrow$
conflicting S = *None* $\Longrightarrow$
T $\sim$ *restart-state* S $\Longrightarrow$
P S T **and**
decideH: $\bigwedge L$ T. *conflicting* S = *None* $\Longrightarrow$
undefined-lit (*trail* S) L $\Longrightarrow$
atm-of L $\in$ *atms-of-mm* (*init-clss* S) $\Longrightarrow$
T $\sim$ *cons-trail* (*Decided* L) (*incr-lvl* S) $\Longrightarrow$
P S T **and**
skipH: $\bigwedge L$ C' M E T.
trail S = *Propagated* L C' # M $\Longrightarrow$

$conflicting\ S = Some\ E \Longrightarrow$
$-L \notin\# E \Longrightarrow E \neq \{\#\} \Longrightarrow$
$T \sim tl\text{-}trail\ S \Longrightarrow$
$P\ S\ T$ **and**
$resolveH$: $\bigwedge L\ E\ M\ D\ T.$
$trail\ S = Propagated\ L\ E\ \#\ M \Longrightarrow$
$L \in\# E \Longrightarrow$
$hd\text{-}trail\ S = Propagated\ L\ E \Longrightarrow$
$conflicting\ S = Some\ D \Longrightarrow$
$-L \in\# D \Longrightarrow$
$get\text{-}maximum\text{-}level\ (trail\ S)\ ((remove1\text{-}mset\ (-L)\ D)) = backtrack\text{-}lvl\ S \Longrightarrow$
$T \sim update\text{-}conflicting$
$(Some\ (resolve\text{-}cls\ L\ D\ E))\ (tl\text{-}trail\ S) \Longrightarrow$
$P\ S\ T$ **and**
$backtrackH$: $\bigwedge L\ D\ K\ i\ M1\ M2\ T.$
$conflicting\ S = Some\ D \Longrightarrow$
$L \in\# D \Longrightarrow$
$(Decided\ K\ \#\ M1,\ M2) \in set\ (get\text{-}all\text{-}ann\text{-}decomposition\ (trail\ S)) \Longrightarrow$
$get\text{-}level\ (trail\ S)\ L = backtrack\text{-}lvl\ S \Longrightarrow$
$get\text{-}level\ (trail\ S)\ L = get\text{-}maximum\text{-}level\ (trail\ S)\ D \Longrightarrow$
$get\text{-}maximum\text{-}level\ (trail\ S)\ (remove1\text{-}mset\ L\ D) \equiv i \Longrightarrow$
$get\text{-}level\ (trail\ S)\ K = i+1 \Longrightarrow$
$T \sim cons\text{-}trail\ (Propagated\ L\ D)$
$(reduce\text{-}trail\text{-}to\ M1$
$(add\text{-}learned\text{-}cls\ D$
$(update\text{-}backtrack\text{-}lvl\ i$
$(update\text{-}conflicting\ None\ S)))) \Longrightarrow$
$P\ S\ T$
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**
$decideH$: $\bigwedge L\ T.\ conflicting\ S = None \Longrightarrow undefined\text{-}lit\ (trail\ S)\ L$
$\Longrightarrow atm\text{-}of\ L \in atms\text{-}of\text{-}mm\ (init\text{-}clss\ S)$
$\Longrightarrow T \sim cons\text{-}trail\ (Decided\ L)\ (incr\text{-}lvl\ S)$
$\Longrightarrow P\ S\ T$ **and**
$skipH$: $\bigwedge L\ C'\ M\ E\ T.$
$trail\ S = Propagated\ L\ C'\ \#\ M \Longrightarrow$
$conflicting\ S = Some\ E \Longrightarrow$
$-L \notin\# E \Longrightarrow E \neq \{\#\} \Longrightarrow$
$T \sim tl\text{-}trail\ S \Longrightarrow$
$P\ S\ T$ **and**
$resolveH$: $\bigwedge L\ E\ M\ D\ T.$
$trail\ S = Propagated\ L\ E\ \#\ M \Longrightarrow$
$L \in\# E \Longrightarrow$
$hd\text{-}trail\ S = Propagated\ L\ E \Longrightarrow$
$conflicting\ S = Some\ D \Longrightarrow$
$-L \in\# D \Longrightarrow$
$get\text{-}maximum\text{-}level\ (trail\ S)\ ((remove1\text{-}mset\ (-L)\ D)) = backtrack\text{-}lvl\ S \Longrightarrow$
$T \sim update\text{-}conflicting$
$(Some\ (resolve\text{-}cls\ L\ D\ E))\ (tl\text{-}trail\ S) \Longrightarrow$
$P\ S\ T$ **and**
$backtrackH$: $\bigwedge L\ D\ K\ i\ M1\ M2\ T.$
$conflicting\ S = Some\ D \Longrightarrow$

$L \in\# D \Longrightarrow$
$(Decided\ K\ \#\ M1,\ M2) \in set\ (get\text{-}all\text{-}ann\text{-}decomposition\ (trail\ S)) \Longrightarrow$
$get\text{-}level\ (trail\ S)\ L = backtrack\text{-}lvl\ S \Longrightarrow$
$get\text{-}level\ (trail\ S)\ L = get\text{-}maximum\text{-}level\ (trail\ S)\ D \Longrightarrow$
$get\text{-}maximum\text{-}level\ (trail\ S)\ (remove1\text{-}mset\ L\ D) \equiv i \Longrightarrow$
$get\text{-}level\ (trail\ S)\ K = i + 1 \Longrightarrow$
$T \sim cons\text{-}trail\ (Propagated\ L\ D)$
$(reduce\text{-}trail\text{-}to\ M1$
$(add\text{-}learned\text{-}cls\ D$
$(update\text{-}backtrack\text{-}lvl\ i$
$(update\text{-}conflicting\ None\ S)))) \Longrightarrow$
$P\ S\ T$
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
$cdcl_W\text{-}o\ S\ T$ **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
$cdcl_W\text{-}o\ S\ T$ **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*⟩

2.1.3 Structural Invariants

Properties of the trail

We here establish that:

- the consistency of the trail;
- the fact that there is no duplicate in the trail.

lemma *backtrack-lit-skiped*:
assumes
L: $get\text{-}level\ (trail\ S)\ L = backtrack\text{-}lvl\ S$ **and**
M1: $(Decided\ K\ \#\ M1,\ M2) \in set\ (get\text{-}all\text{-}ann\text{-}decomposition\ (trail\ S))$ **and**
no-dup: $no\text{-}dup\ (trail\ S)$ **and**
bt-l: $backtrack\text{-}lvl\ S = length\ (filter\ is\text{-}decided\ (trail\ S))$ **and**
lev-K: $get\text{-}level\ (trail\ S)\ K = i + 1$
shows $atm\text{-}of\ L \notin atm\text{-}of\ `\ lits\text{-}of\text{-}l\ M1$

⟨*proof*⟩

lemma $cdcl_W$*-distinctinv-1*:
assumes
$cdcl_W$ S S' **and**
no-dup (*trail* S) **and**
bt-lev: *backtrack-lvl* S = *count-decided* (*trail* S)
shows *no-dup* (*trail* S')
⟨*proof*⟩

Item 1 page 81 of Weidenbach's book

lemma $cdcl_W$*-consistent-inv-2*:
assumes
$cdcl_W$ S S' **and**
no-dup (*trail* S) **and**
backtrack-lvl S = *count-decided* (*trail* S)
shows *consistent-interp* (*lits-of-l* (*trail* S'))
⟨*proof*⟩

lemma $cdcl_W$*-o-bt*:
assumes
$cdcl_W$*-o* S S' **and**
backtrack-lvl S = *count-decided* (*trail* S) **and**
n-d[*simp*]: *no-dup* (*trail* S)
shows *backtrack-lvl* S' = *count-decided* (*trail* S')
⟨*proof*⟩

lemma $cdcl_W$*-rf-bt*:
assumes
$cdcl_W$*-rf* S S' **and**
backtrack-lvl S = *count-decided* (*trail* S)
shows *backtrack-lvl* S' = *count-decided* (*trail* S')
⟨*proof*⟩

Item 7 page 81 of Weidenbach's book

lemma $cdcl_W$*-bt*:
assumes
$cdcl_W$ S S' **and**
backtrack-lvl S = *count-decided* (*trail* S) **and**
no-dup (*trail* S)
shows *backtrack-lvl* S' = *count-decided* (*trail* S')
⟨*proof*⟩

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

definition $cdcl_W$*-M-level-inv* :: *'st* ⇒ *bool* **where**
$cdcl_W$*-M-level-inv* S ⟷
consistent-interp (*lits-of-l* (*trail* S))
∧ *no-dup* (*trail* S)
∧ *backtrack-lvl* S = *count-decided* (*trail* S)

lemma $cdcl_W$*-M-level-inv-decomp*:
assumes $cdcl_W$*-M-level-inv* S
shows
consistent-interp (*lits-of-l* (*trail* S)) **and**

no-dup (*trail* S)
⟨*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) $L \leq$ *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 $\exists K\ M1\ M2.$ (*Decided* K # $M1$, $M2$) $\in$ *set* (*get-all-ann-decomposition* (*trail* S)) $\wedge$
get-level (*trail* S) $K =$ *Suc* i
⟨*proof*⟩

Compatibility with *op* ∼

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

lemma *conflict-state-eq-compatible*:
assumes
confl: *conflict* S T **and**
TT': $T \sim T'$ **and**

SS': $S \sim S'$
shows *conflict* S' T'
⟨*proof*⟩

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

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: *skip* S T **and**
SS': $S \sim S'$ **and**
TT': $T \sim T'$
shows *skip* S' T'
⟨*proof*⟩

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

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

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

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

$cdcl_W$-*bj* S T **and** $cdcl_W$-*M-level-inv* S
$T \sim T'$
shows $cdcl_W$-*bj* S T'
⟨*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*⟩

Conservation of some Properties

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

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

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

lemma $cdcl_W$*-init-clss*:
assumes
$cdcl_W$ S T **and**
inv: $cdcl_W$-*M-level-inv* S
shows *init-clss* S = *init-clss* T
⟨*proof*⟩

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

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

Learned Clause

This invariant shows that:

- the learned clauses are entailed by the initial set of clauses.

- the conflicting clause is entailed by the initial set of clauses.
- the marks are entailed by the clauses.

definition $\mathit{cdcl}_W\text{-}\mathit{learned}\text{-}\mathit{clause}$ $(S :: {}'st) \longleftrightarrow$
$(\mathit{init}\text{-}\mathit{clss}\ S \models psm\ \mathit{learned}\text{-}\mathit{clss}\ S$
$\wedge (\forall T.\ \mathit{conflicting}\ S = \mathit{Some}\ T \longrightarrow \mathit{init}\text{-}\mathit{clss}\ S \models pm\ T)$
$\wedge\ \mathit{set}\ (\mathit{get}\text{-}\mathit{all}\text{-}\mathit{mark}\text{-}\mathit{of}\text{-}\mathit{propagated}\ (\mathit{trail}\ S)) \subseteq \mathit{set}\text{-}\mathit{mset}\ (\mathit{clauses}\ S))$

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

lemma $\mathit{cdcl}_W\text{-}\mathit{learned}\text{-}\mathit{clause}\text{-}S0\text{-}\mathit{cdcl}_W[\mathit{simp}]$:
$\mathit{cdcl}_W\text{-}\mathit{learned}\text{-}\mathit{clause}\ (\mathit{init}\text{-}\mathit{state}\ N)$
$\langle \mathit{proof} \rangle$

Item 4 page 81 of Weidenbach's book

lemma $\mathit{cdcl}_W\text{-}\mathit{learned}\text{-}\mathit{clss}$:
assumes
$\mathit{cdcl}_W\ S\ S'$ **and**
$\mathit{learned}$: $\mathit{cdcl}_W\text{-}\mathit{learned}\text{-}\mathit{clause}\ S$ **and**
$\mathit{lev}\text{-}\mathit{inv}$: $\mathit{cdcl}_W\text{-}M\text{-}\mathit{level}\text{-}\mathit{inv}\ S$
shows $\mathit{cdcl}_W\text{-}\mathit{learned}\text{-}\mathit{clause}\ S'$
$\langle \mathit{proof} \rangle$

lemma $\mathit{rtranclp}\text{-}\mathit{cdcl}_W\text{-}\mathit{learned}\text{-}\mathit{clss}$:
assumes
$\mathit{cdcl}_W^{**}\ S\ S'$ **and**
$\mathit{cdcl}_W\text{-}M\text{-}\mathit{level}\text{-}\mathit{inv}\ S$
$\mathit{cdcl}_W\text{-}\mathit{learned}\text{-}\mathit{clause}\ S$
shows $\mathit{cdcl}_W\text{-}\mathit{learned}\text{-}\mathit{clause}\ S'$
$\langle \mathit{proof} \rangle$

No alien atom in the state

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

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

lemma $\mathit{no}\text{-}\mathit{strange}\text{-}\mathit{atm}\text{-}\mathit{decomp}$:
assumes $\mathit{no}\text{-}\mathit{strange}\text{-}\mathit{atm}\ S$
shows $\mathit{conflicting}\ S = \mathit{Some}\ T \Longrightarrow \mathit{atms}\text{-}\mathit{of}\ T \subseteq \mathit{atms}\text{-}\mathit{of}\text{-}\mathit{mm}\ (\mathit{init}\text{-}\mathit{clss}\ S)$
and $(\forall L\ \mathit{mark}.\ \mathit{Propagated}\ L\ \mathit{mark} \in \mathit{set}\ (\mathit{trail}\ S)$
$\longrightarrow \mathit{atms}\text{-}\mathit{of}\ \mathit{mark} \subseteq \mathit{atms}\text{-}\mathit{of}\text{-}\mathit{mm}\ (\mathit{init}\text{-}\mathit{clss}\ S))$
and $\mathit{atms}\text{-}\mathit{of}\text{-}\mathit{mm}\ (\mathit{learned}\text{-}\mathit{clss}\ S) \subseteq \mathit{atms}\text{-}\mathit{of}\text{-}\mathit{mm}\ (\mathit{init}\text{-}\mathit{clss}\ S)$
and $\mathit{atm}\text{-}\mathit{of}\ `\ (\mathit{lits}\text{-}\mathit{of}\text{-}l\ (\mathit{trail}\ S)) \subseteq \mathit{atms}\text{-}\mathit{of}\text{-}\mathit{mm}\ (\mathit{init}\text{-}\mathit{clss}\ S)$
$\langle \mathit{proof} \rangle$

lemma $\mathit{no}\text{-}\mathit{strange}\text{-}\mathit{atm}\text{-}S0$ $[\mathit{simp}]$: $\mathit{no}\text{-}\mathit{strange}\text{-}\mathit{atm}\ (\mathit{init}\text{-}\mathit{state}\ N)$
$\langle \mathit{proof} \rangle$

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

lemma *propagate-no-strange-atm-inv*:
assumes
propagate S T **and**
alien: *no-strange-atm S*
shows *no-strange-atm T*
⟨*proof*⟩

lemma *in-atms-of-remove1-mset-in-atms-of*:
$x \in \textit{atms-of } (\textit{remove1-mset } L\ C) \Longrightarrow x \in \textit{atms-of } C$
⟨*proof*⟩

lemma $\textit{cdcl}_W$*-no-strange-atm-explicit*:
assumes
$\textit{cdcl}_W\ S\ S'$ **and**
lev: $\textit{cdcl}_W$*-M-level-inv S* **and**
conf: $\forall T.\ \textit{conflicting } S = \textit{Some } T \longrightarrow \textit{atms-of } T \subseteq \textit{atms-of-mm } (\textit{init-clss } S)$ **and**
decided: $\forall L\ \textit{mark}.\ \textit{Propagated } L\ \textit{mark} \in \textit{set } (\textit{trail } S)$
$\longrightarrow \textit{atms-of mark} \subseteq \textit{atms-of-mm } (\textit{init-clss } S)$ **and**
learned: $\textit{atms-of-mm } (\textit{learned-clss } S) \subseteq \textit{atms-of-mm } (\textit{init-clss } S)$ **and**
trail: $\textit{atm-of } `\ (\textit{lits-of-l } (\textit{trail } S)) \subseteq \textit{atms-of-mm } (\textit{init-clss } S)$
shows
$(\forall T.\ \textit{conflicting } S' = \textit{Some } T \longrightarrow \textit{atms-of } T \subseteq \textit{atms-of-mm } (\textit{init-clss } S')) \land$
$(\forall L\ \textit{mark}.\ \textit{Propagated } L\ \textit{mark} \in \textit{set } (\textit{trail } S')$
$\longrightarrow \textit{atms-of mark} \subseteq \textit{atms-of-mm } (\textit{init-clss } S')) \land$
$\textit{atms-of-mm } (\textit{learned-clss } S') \subseteq \textit{atms-of-mm } (\textit{init-clss } S') \land$
$\textit{atm-of } `\ (\textit{lits-of-l } (\textit{trail } S')) \subseteq \textit{atms-of-mm } (\textit{init-clss } S')$
(**is** $?C\ S' \land ?M\ S' \land ?U\ S' \land ?V\ S'$)
⟨*proof*⟩

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

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

No Duplicates all Around

This invariant shows that there is no duplicate (no literal appearing twice in the formula). The last part could be proven using the previous invariant also. Remark that we will show later that there cannot be duplicate *clause*.

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

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* $(S ::'st)$ **and** *conflicting* S = *Some* T
shows *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**
lev-inv: *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*⟩

Conflicts and Annotations

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

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

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

lemma *backtrack-atms-of-D-in-M1*:
fixes *M1* :: $('v, 'v$ *clause*) *ann-lits*
assumes
inv: *cdcl$_W$-M-level-inv* S **and**
i: *get-maximum-level* (*trail* S) ((*remove1-mset* L D)) $\equiv i$ **and**
decomp: (*Decided* K # *M1*, *M2*)
$\in$ *set* (*get-all-ann-decomposition* (*trail* S)) **and**
S-lvl: *backtrack-lvl* S = *get-maximum-level* (*trail* S) D **and**

S-confl: *conflicting S* = *Some D* **and**
lev-K: *get-level* (*trail S*) *K* = *Suc i* **and**
T: *T* ∼ *cons-trail* (*Propagated L D*)
(*reduce-trail-to M1*
(*add-learned-cls D*
(*update-backtrack-lvl i*
(*update-conflicting None S*)))) **and**
confl: ∀ *T*. *conflicting S* = *Some T* ⟶ *trail S* ⊨*as CNot T*
shows *atms-of* ((*remove1-mset L D*)) ⊆ *atm-of* ‘ *lits-of-l* (*tl* (*trail T*))
⟨*proof*⟩

lemma *distinct-atms-of-incl-not-in-other*:
assumes
a1: *no-dup* (*M* @ *M′*) **and**
a2: *atms-of D* ⊆ *atm-of* ‘ *lits-of-l M′* **and**
a3: *x* ∈ *atms-of D*
shows *x* ∉ *atm-of* ‘ *lits-of-l M*
⟨*proof*⟩

Item 5 page 81 of Weidenbach's book

lemma *cdcl$_W$-propagate-is-conclusion*:
assumes
cdcl$_W$ S S′ **and**
inv: *cdcl$_W$-M-level-inv S* **and**
decomp: *all-decomposition-implies-m* (*init-clss S*) (*get-all-ann-decomposition* (*trail S*)) **and**
learned: *cdcl$_W$-learned-clause S* **and**
confl: ∀ *T*. *conflicting S* = *Some T* ⟶ *trail S* ⊨*as CNot T* **and**
alien: *no-strange-atm S*
shows *all-decomposition-implies-m* (*init-clss S′*) (*get-all-ann-decomposition* (*trail S′*))
⟨*proof*⟩

lemma *cdcl$_W$-propagate-is-false*:
assumes
cdcl$_W$ S S′ **and**
lev: *cdcl$_W$-M-level-inv S* **and**
learned: *cdcl$_W$-learned-clause S* **and**
decomp: *all-decomposition-implies-m* (*init-clss S*) (*get-all-ann-decomposition* (*trail S*)) **and**
confl: ∀ *T*. *conflicting S* = *Some T* ⟶ *trail S* ⊨*as CNot T* **and**
alien: *no-strange-atm S* **and**
mark-confl: *every-mark-is-a-conflict S*
shows *every-mark-is-a-conflict S′*
⟨*proof*⟩

lemma *cdcl$_W$-conflicting-is-false*:
assumes
cdcl$_W$ S S′ **and**
M-lev: *cdcl$_W$-M-level-inv S* **and**
confl-inv: ∀ *T*. *conflicting S* = *Some T* ⟶ *trail S* ⊨*as CNot T* **and**
decided-confl: ∀ *L mark a b*. *a* @ *Propagated L mark* # *b* = (*trail S*)
⟶ (*b* ⊨*as CNot* (*mark* − {#*L*#}) ∧ *L* ∈# *mark*) **and**
dist: *distinct-cdcl$_W$-state S*
shows ∀ *T*. *conflicting S′* = *Some T* ⟶ *trail S′* ⊨*as CNot T*
⟨*proof*⟩

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

Putting all the invariants together

lemma $cdcl_W$*-all-inv*:
assumes
$cdcl_W$: $cdcl_W$ S S' **and**
1: *all-decomposition-implies-m* (*init-clss* S) (*get-all-ann-decomposition* (*trail* S)) **and**
2: $cdcl_W$*-learned-clause* S **and**
4: $cdcl_W$*-M-level-inv* S **and**
5: *no-strange-atm* S **and**
7: *distinct-*$cdcl_W$*-state* S **and**
8: $cdcl_W$*-conflicting* S
shows
all-decomposition-implies-m (*init-clss* S') (*get-all-ann-decomposition* (*trail* S')) **and**
$cdcl_W$*-learned-clause* S' **and**
$cdcl_W$*-M-level-inv* S' **and**
no-strange-atm S' **and**
distinct-$cdcl_W$*-state* S' **and**
$cdcl_W$*-conflicting* S'
$\langle proof \rangle$

lemma *rtranclp-*$cdcl_W$*-all-inv*:
assumes
$cdcl_W$: *rtranclp* $cdcl_W$ S S' **and**
1: *all-decomposition-implies-m* (*init-clss* S) (*get-all-ann-decomposition* (*trail* S)) **and**
2: $cdcl_W$*-learned-clause* S **and**
4: $cdcl_W$*-M-level-inv* S **and**
5: *no-strange-atm* S **and**
7: *distinct-*$cdcl_W$*-state* S **and**
8: $cdcl_W$*-conflicting* S
shows
all-decomposition-implies-m (*init-clss* S') (*get-all-ann-decomposition* (*trail* S')) **and**
$cdcl_W$*-learned-clause* S' **and**
$cdcl_W$*-M-level-inv* S' **and**
no-strange-atm S' **and**
distinct-$cdcl_W$*-state* S' **and**
$cdcl_W$*-conflicting* S'
$\langle proof \rangle$

lemma *all-invariant-S0-*$cdcl_W$:
assumes *distinct-mset-mset* N
shows
all-decomposition-implies-m (*init-clss* (*init-state* N))

(*get-all-ann-decomposition* (*trail* (*init-state* N))) **and**
cdcl$_W$*-learned-clause* (*init-state* N) **and**
$\forall T$. *conflicting* (*init-state* N) = *Some* $T \longrightarrow$ (*trail* (*init-state* N))$\models$*as CNot* T **and**
no-strange-atm (*init-state* N) **and**
consistent-interp (*lits-of-l* (*trail* (*init-state* N))) **and**
$\forall L$ *mark a b*. *a* @ *Propagated L mark* # *b* = *trail* (*init-state* N) $\longrightarrow$
(*b* $\models$*as CNot* (*mark* $-$ {#*L*#}) $\wedge$ *L* $\in$# *mark*) **and**
distinct-cdcl$_W$*-state* (*init-state* N)
⟨*proof*⟩

Item 6 page 81 of Weidenbach's book

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-ann-decomposition M*) **and**
state: *state S* = (*M*, *N*, *U*, *k*, *C*) **and**
learned-cl: *cdcl*$_W$*-learned-clause S* **and**
atm-incl: *no-strange-atm S*
shows *unsatisfiable* (*set-mset N*)
⟨*proof*⟩

Item 5 page 81 of Weidenbach's book

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

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

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

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

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

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

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: $\forall s \in\# \textit{learned-clss } S.\ \neg\textit{tautology } s$
shows $\forall s \in\# \textit{learned-clss } S'.\ \neg\textit{tautology } s$
⟨*proof*⟩

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-mm* (*init-clss* S). $L \in$ *atm-of* ‘ *lits-of-l* (*trail* S))
$\wedge$ ($\exists C \in\#$ *init-clss* S. *trail* S $\models$*as CNot* C)))

2.1.4 CDCL Strong Completeness

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

theorem 2.9.11 page 84 of Weidenbach's book

lemma $cdcl_W$*-strong-completeness*:
assumes
MN: *set* M $\models$*sm* N **and**
cons: *consistent-interp* (*set* M) **and**
dist: *distinct* M **and**
atm: *atm-of* ‘ (*set* M) $\subseteq$ *atms-of-mm* N
obtains S **where**
state S = (*map* ($\lambda L.$ *Decided* L) M, N, {#}, *length* M, *None*) **and**
rtranclp $cdcl_W$ (*init-state* N) S **and**
final-$cdcl_W$*-state* S
⟨*proof*⟩

2.1.5 Higher level strategy

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

Definition

lemma *tranclp-conflict*:
tranclp conflict S S' $\Longrightarrow$ *conflict* S S'
⟨*proof*⟩

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

$\langle proof \rangle$

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 $rtranclp\text{-}cdcl_W\text{-}cp\text{-}rtranclp\text{-}cdcl_W$:
$cdcl_W\text{-}cp^{**}\ S\ T \Longrightarrow cdcl_W^{**}\ S\ T$
$\langle proof \rangle$

lemma $cdcl_W\text{-}cp\text{-}state\text{-}eq\text{-}compatible$:
assumes
$cdcl_W\text{-}cp\ S\ T$ **and**
$S \sim S'$ **and**
$T \sim T'$
shows $cdcl_W\text{-}cp\ S'\ T'$
$\langle proof \rangle$

lemma $tranclp\text{-}cdcl_W\text{-}cp\text{-}state\text{-}eq\text{-}compatible$:
assumes
$cdcl_W\text{-}cp^{++}\ S\ T$ **and**
$S \sim S'$ **and**
$T \sim T'$
shows $cdcl_W\text{-}cp^{++}\ S'\ T'$
$\langle proof \rangle$

lemma $option\text{-}full\text{-}cdcl_W\text{-}cp$:
$conflicting\ S \neq None \Longrightarrow full\ cdcl_W\text{-}cp\ S\ S$
$\langle proof \rangle$

lemma $skip\text{-}unique$:
$skip\ S\ T \Longrightarrow skip\ S\ T' \Longrightarrow T \sim T'$
$\langle proof \rangle$

lemma $resolve\text{-}unique$:
$resolve\ S\ T \Longrightarrow resolve\ S\ T' \Longrightarrow T \sim T'$
$\langle proof \rangle$

lemma $cdcl_W\text{-}cp\text{-}no\text{-}more\text{-}clauses$:
assumes $cdcl_W\text{-}cp\ S\ S'$
shows $clauses\ S = clauses\ S'$
$\langle proof \rangle$

lemma $tranclp\text{-}cdcl_W\text{-}cp\text{-}no\text{-}more\text{-}clauses$:
assumes $cdcl_W\text{-}cp^{++}\ S\ S'$
shows $clauses\ S = clauses\ S'$
$\langle proof \rangle$

lemma $rtranclp\text{-}cdcl_W\text{-}cp\text{-}no\text{-}more\text{-}clauses$:
assumes $cdcl_W\text{-}cp^{**}\ S\ S'$
shows $clauses\ S = clauses\ S'$
$\langle proof \rangle$

lemma $no\text{-}conflict\text{-}after\text{-}conflict$:
$conflict\ S\ T \Longrightarrow \neg conflict\ T\ U$
$\langle proof \rangle$

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* $D \Longrightarrow \neg$*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 $S' \Longrightarrow$ *no-step cdcl*$_W$*-cp* $S \Longrightarrow$ *full cdcl*$_W$*-cp* S' $S'' \Longrightarrow$ *cdcl*$_W$*-stgy* S S''

Invariants

These are the same invariants as before, but lifted

lemma *cdcl*$_W$*-cp-learned-clause-inv*:
  assumes *cdcl*$_W$*-cp* S S'
  shows *learned-clss* S = *learned-clss* S'
  ⟨*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** ($\forall$ l $\in$ *set* M. $\neg$*is-decided* l)
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-cp-dropWhile-trail'*:
assumes $cdcl_W$*-cp*** S S'
obtains M :: ('*v*, '*v clause*) *ann-lits* **where**
trail S' = M @ *trail* S **and** $\forall$ l $\in$ *set* M. $\neg$*is-decided* l
⟨*proof*⟩

lemma $cdcl_W$*-cp-dropWhile-trail*:

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

lemma *rtranclp-*$cdcl_W$*-cp-dropWhile-trail*:
assumes $cdcl_W$*-cp*** S S'
shows $\exists M.$ *trail* $S' = M$ @ *trail* $S \wedge (\forall l \in$ *set* $M.$ $\neg$*is-decided* $l)$
⟨*proof*⟩

This theorem can be seen a a termination theorem for $cdcl_W$*-cp*.

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

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

lemma $cdcl_W$*-cp-wf*: *wf* $\{(b, a).$ ($cdcl_W$*-M-level-inv* $a \wedge$ *no-strange-atm* a) $\wedge$ $cdcl_W$*-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 (λa $b.$ ($cdcl_W$*-M-level-inv* $a \wedge$ *no-strange-atm* a) $\wedge$ $cdcl_W$*-cp* a b)** S T
$\longleftrightarrow$ $cdcl_W$*-cp*** S T
(**is** *?I* S T $\longleftrightarrow$ *?C* S T)
⟨*proof*⟩

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

Literal of highest level in conflicting clauses

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

abbreviation *no-clause-is-false* :: $'st \Rightarrow bool$ **where**
no-clause-is-false $\equiv$
$\lambda S.$ (*conflicting* S = *None* $\longrightarrow$ ($\forall D \in\#$ *clauses* S. $\neg$*trail* $S \models as$ *CNot* D))

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

lemma *not-conflict-not-any-negated-init-clss*:
assumes $\forall S'. \neg$*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$_W$-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*⟩

Literal of highest level in decided literals

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

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

lemma *propagate-no-more-propagation-to-do*:
  assumes *propagate*: *propagate S S'*
  and *H*: *no-more-propagation-to-do S*
  and *lev-inv*: *cdcl$_W$-M-level-inv S*
  shows *no-more-propagation-to-do S'*
  ⟨*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 ∃ *S'*. *cdcl$_W$-stgy S S'*
⟨*proof*⟩

lemma *backtrack-no-decomp*:
  assumes
    S: *conflicting S* = *Some E* **and**

LE: $L \in\# E$ **and**
L: *get-level* (*trail S*) $L =$ *backtrack-lvl S* **and**
D: *get-maximum-level* (*trail S*) (*remove1-mset L E*) $<$ *backtrack-lvl S* **and**
bt: *backtrack-lvl S* $=$ *get-maximum-level* (*trail S*) *E* **and**
M-L: $cdcl_W$*-M-level-inv S*
shows $\exists S'$. $cdcl_W$*-o S S'*
⟨*proof*⟩

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

lemma $cdcl_W$*-cp-tranclp-*$cdcl_W$:
$cdcl_W$*-cp S S'* $\Longrightarrow$ $cdcl_W{}^{++}$ *S S'*
⟨*proof*⟩

lemma *tranclp-*$cdcl_W$*-cp-tranclp-*$cdcl_W$:
$cdcl_W$*-cp*$^{++}$ *S S'* $\Longrightarrow$ $cdcl_W{}^{++}$ *S S'*
⟨*proof*⟩

lemma $cdcl_W$*-stgy-tranclp-*$cdcl_W$:
$cdcl_W$*-stgy S S'* $\Longrightarrow$ $cdcl_W{}^{++}$ *S S'*
⟨*proof*⟩

lemma *tranclp-*$cdcl_W$*-stgy-tranclp-*$cdcl_W$:
$cdcl_W$*-stgy*$^{++}$ *S S'* $\Longrightarrow$ $cdcl_W{}^{++}$ *S S'*
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-stgy-rtranclp-*$cdcl_W$:
$cdcl_W$*-stgy*** *S S'* $\Longrightarrow$ $cdcl_W{}^{**}$ *S S'*
⟨*proof*⟩

lemma *not-empty-get-maximum-level-exists-lit*:
assumes *n*: $D \neq$ {#}
and *max*: *get-maximum-level M D* $= n$
shows $\exists L \in\# D$. *get-level M L* $= n$
⟨*proof*⟩

lemma $cdcl_W$*-o-conflict-is-false-with-level-inv*:
assumes
$cdcl_W$*-o S S'* **and**
lev: $cdcl_W$*-M-level-inv S* **and**
confl-inv: *conflict-is-false-with-level S* **and**
n-d: *distinct-*$cdcl_W$*-state S* **and**
conflicting: $cdcl_W$*-conflicting S*
shows *conflict-is-false-with-level S'*

⟨*proof*⟩

Strong completeness

lemma $cdcl_W$*-cp-propagate-confl*:
assumes $cdcl_W$*-cp S T*
shows $propagate^{**}\ S\ T \lor (\exists S'.\ propagate^{**}\ S\ S' \land conflict\ S'\ T)$
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-cp-propagate-confl*:
assumes $cdcl_W\text{-}cp^{**}\ S\ T$
shows $propagate^{**}\ S\ T \lor (\exists S'.\ propagate^{**}\ S\ S' \land conflict\ S'\ T)$
⟨*proof*⟩

lemma *propagate-high-levelE*:
assumes *propagate S T*
obtains $M'\ N'\ U\ k\ L\ C$ **where**
$state\ S = (M', N', U, k, None)$ **and**
$state\ T = (Propagated\ L\ (C + \{\#L\#\})\ \#\ M', N', U, k, None)$ **and**
$C + \{\#L\#\} \in\#\ local.clauses\ S$ **and**
$M' \models as\ CNot\ C$ **and**
undefined-lit (*trail S*) *L*
⟨*proof*⟩

lemma $cdcl_W$*-cp-propagate-completeness*:
assumes *MN*: $set\ M \models s\ set\text{-}mset\ N$ **and**
cons: *consistent-interp* (*set M*) **and**
tot: *total-over-m* (*set M*) (*set-mset N*) **and**
lits-of-l (*trail S*) $\subseteq$ *set M* **and**
init-clss S = *N* **and**
$propagate^{**}\ S\ S'$ **and**
learned-clss S = {#}
shows $length\ (trail\ S) \le length\ (trail\ S') \land lits\text{-}of\text{-}l\ (trail\ S') \subseteq set\ M$
⟨*proof*⟩

lemma
assumes $propagate^{**}\ S\ X$
shows
rtranclp-propagate-init-clss: *init-clss X* = *init-clss S* **and**
rtranclp-propagate-learned-clss: *learned-clss X* = *learned-clss S*
⟨*proof*⟩

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-interp* (*set M*)
and *tot*: *total-over-m* (*set M*) (*set-mset N*)
and *alien*: *no-strange-atm S*
and *learned*: *learned-clss S* = {#}
and *clsS*[*simp*]: *init-clss S* = *N*
and *lits*: *lits-of-l* (*trail S*) $\subseteq$ *set M*
shows $\exists S'.\ propagate^{**}\ S\ S' \land full\ cdcl_W\text{-}cp\ S\ S'$
⟨*proof*⟩

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*:

$propagate^{**}\ S\ T \Longrightarrow \exists\, c.\ trail\ T = c\ @\ trail\ S$
⟨*proof*⟩

lemma *rtranclp-propagate-is-update-trail*:
$propagate^{**}\ S\ T \Longrightarrow cdcl_W\text{-}M\text{-}level\text{-}inv\ S \Longrightarrow$
$init\text{-}clss\ S = init\text{-}clss\ T \wedge learned\text{-}clss\ S = learned\text{-}clss\ T \wedge backtrack\text{-}lvl\ S = backtrack\text{-}lvl\ T$
$\wedge\ conflicting\ S = conflicting\ T$
⟨*proof*⟩

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: $atm\text{-}of\ `\ (set\ M) \subseteq atms\text{-}of\text{-}mm\ N$ **and**
distM: $distinct\ M$ **and**
length: $n \leq length\ M$
shows
$\exists\, M'\ k\ S.\ length\ M' \geq n\ \wedge$
$lits\text{-}of\text{-}l\ M' \subseteq set\ M\ \wedge$
$no\text{-}dup\ M'\ \wedge$
$state\ S = (M',\ N,\ \{\#\},\ k,\ None)\ \wedge$
$cdcl_W\text{-}stgy^{**}\ (init\text{-}state\ N)\ S$
⟨*proof*⟩

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

lemma $cdcl_W$*-stgy-strong-completeness*:
assumes
MN: $set\ M \models s\ set\text{-}mset\ N$ **and**
cons: $consistent\text{-}interp\ (set\ M)$ **and**
tot: $total\text{-}over\text{-}m\ (set\ M)\ (set\text{-}mset\ N)$ **and**
atm-incl: $atm\text{-}of\ `\ (set\ M) \subseteq atms\text{-}of\text{-}mm\ N$ **and**
distM: $distinct\ M$
shows
$\exists\, M'\ k\ S.$
$lits\text{-}of\text{-}l\ M' = set\ M\ \wedge$
$state\ S = (M',\ N,\ \{\#\},\ k,\ None)\ \wedge$
$cdcl_W\text{-}stgy^{**}\ (init\text{-}state\ N)\ S\ \wedge$
$final\text{-}cdcl_W\text{-}state\ S$
⟨*proof*⟩

No conflict with only variables of level less than backtrack level

This invariant is stronger than the previous argument in the sense that it is a property about all possible conflicts.

definition *no-smaller-confl* $(S ::'st) \equiv$
$(\forall\, M\ K\ M'\ D.\ M'\ @\ Decided\ K\ \#\ M = trail\ S \longrightarrow D \in\#\ clauses\ S$
$\longrightarrow \neg M \models as\ CNot\ D)$

lemma *no-smaller-confl-init-sate*[*simp*]:
$no\text{-}smaller\text{-}confl\ (init\text{-}state\ N)$ ⟨*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*: $cdcl_W\text{-}cp^{**}$ *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*: $cdcl_W\text{-}cp^{++}$ *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'*

$\langle proof \rangle$

lemma *is-conflicting-exists-conflict*:
assumes $\neg(\forall D{\in}\#init\text{-}clss\ S' + learned\text{-}clss\ S'.\ \neg\ trail\ S' \models as\ CNot\ D)$
and $conflicting\ S' = None$
shows $\exists S''.\ conflict\ S'\ S''$
$\langle proof \rangle$

lemma *$cdcl_W$-o-conflict-is-no-clause-is-false*:
fixes $S\ S' :: 'st$
assumes
$cdcl_W\text{-}o\ S\ S'$ **and**
lev: $cdcl_W\text{-}M\text{-}level\text{-}inv\ S$ **and**
$max\text{-}lev$: $conflict\text{-}is\text{-}false\text{-}with\text{-}level\ S$ **and**
$no\text{-}f$: $no\text{-}clause\text{-}is\text{-}false\ S$ **and**
$no\text{-}l$: $no\text{-}smaller\text{-}confl\ S$
shows $no\text{-}clause\text{-}is\text{-}false\ S'$
$\vee$ $(conflicting\ S' = None$
$\longrightarrow (\forall D \in\#\ clauses\ S'.\ trail\ S' \models as\ CNot\ D$
$\longrightarrow (\exists L.\ L \in\#\ D \wedge get\text{-}level\ (trail\ S')\ L = backtrack\text{-}lvl\ S')))$
$\langle proof \rangle$

lemma *full1-$cdcl_W$-cp-exists-conflict-decompose*:
assumes
$confl$: $\exists D{\in}\#clauses\ S.\ trail\ S \models as\ CNot\ D$ **and**
$full$: $full\ cdcl_W\text{-}cp\ S\ U$ **and**
$no\text{-}confl$: $conflicting\ S = None$ **and**
lev: $cdcl_W\text{-}M\text{-}level\text{-}inv\ S$
shows $\exists T.\ propagate^{**}\ S\ T \wedge conflict\ T\ U$
$\langle proof \rangle$

lemma *full1-$cdcl_W$-cp-exists-conflict-full1-decompose*:
assumes
$confl$: $\exists D{\in}\#clauses\ S.\ trail\ S \models as\ CNot\ D$ **and**
$full$: $full\ cdcl_W\text{-}cp\ S\ U$ **and**
$no\text{-}confl$: $conflicting\ S = None$**and**
lev: $cdcl_W\text{-}M\text{-}level\text{-}inv\ S$
shows $\exists T\ D.\ propagate^{**}\ S\ T \wedge conflict\ T\ U$
$\wedge\ trail\ T \models as\ CNot\ D \wedge conflicting\ U = Some\ D \wedge D \in\#\ clauses\ S$
$\langle proof \rangle$

lemma *$cdcl_W$-stgy-no-smaller-confl*:
assumes
$cdcl_W\text{-}stgy\ S\ S'$ **and**
$n\text{-}l$: $no\text{-}smaller\text{-}confl\ S$ **and**
$conflict\text{-}is\text{-}false\text{-}with\text{-}level\ S$ **and**
$cdcl_W\text{-}M\text{-}level\text{-}inv\ S$ **and**
$no\text{-}clause\text{-}is\text{-}false\ S$ **and**
$distinct\text{-}cdcl_W\text{-}state\ S$ **and**
$cdcl_W\text{-}conflicting\ S$
shows $no\text{-}smaller\text{-}confl\ S'$
$\langle proof \rangle$

lemma *$cdcl_W$-stgy-ex-lit-of-max-level*:
assumes
$cdcl_W\text{-}stgy\ S\ S'$ **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-ann-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'
⟨*proof*⟩

Final States are Conclusive

lemma *full-*$cdcl_W$*-stgy-final-state-conclusive-non-false*:
fixes S' :: *'st*
assumes *full*: *full* $cdcl_W$*-stgy* (*init-state N*) S'
and *no-d*: *distinct-mset-mset N*
and *no-empty*: $\forall D \in\# N.\ D \neq \{\#\}$
shows (*conflicting* S' = *Some* $\{\#\}$ $\wedge$ *unsatisfiable* (*set-mset* (*init-clss* S')))
$\vee$ (*conflicting* S' = *None* $\wedge$ *trail* S' $\models$*asm init-clss* S')
⟨*proof*⟩

lemma *conflict-is-full1-*$cdcl_W$*-cp*:
assumes *cp*: *conflict S* S'
shows *full1* $cdcl_W$*-cp S* S'
⟨*proof*⟩

lemma $cdcl_W$*-cp-fst-empty-conflicting-false*:
assumes
$cdcl_W$*-cp S* S' **and**
trail S = [] **and**
conflicting S $\neq$ *None*
shows *False*
⟨*proof*⟩

lemma $cdcl_W$*-o-fst-empty-conflicting-false*:
assumes $cdcl_W$*-o S* S'
and *trail S* = []
and *conflicting S* $\neq$ *None*
shows *False*
⟨*proof*⟩

lemma $cdcl_W$*-stgy-fst-empty-conflicting-false*:
assumes $cdcl_W$*-stgy* S S'
and *trail* $S = []$
and *conflicting* $S \neq$ *None*
shows *False*
⟨*proof*⟩
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*
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-cp-conflicting-is-false*:
$cdcl_W$*-cp*$^{++}$ S $S' \Longrightarrow$ *conflicting* $S =$ *Some* $\{\#\} \Longrightarrow$ *False*
⟨*proof*⟩

lemma $cdcl_W$*-o-conflicting-is-false*:
$cdcl_W$*-o* S $S' \Longrightarrow$ *conflicting* $S =$ *Some* $\{\#\} \Longrightarrow$ *False*
⟨*proof*⟩

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 $S' \Longrightarrow$ *conflicting* $S =$ *Some* $\{\#\} \Longrightarrow S' = S$
⟨*proof*⟩

lemma *full-*$cdcl_W$*-init-clss-with-false-normal-form*:
assumes
$\forall$ $m \in$ *set* M. $\neg$*is-decided* m **and**
$E =$ *Some* D **and**
state $S = (M, N, U, 0, E)$
full $cdcl_W$*-stgy* S S' **and**
all-decomposition-implies-m (*init-clss* S) (*get-all-ann-decomposition* (*trail* S))
$cdcl_W$*-learned-clause* S
$cdcl_W$*-M-level-inv* S
no-strange-atm S
distinct-$cdcl_W$*-state* S
$cdcl_W$*-conflicting* S
shows $\exists M''$. *state* $S' = (M'', N, U, 0,$ *Some* $\{\#\})$
⟨*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*: $\{\#\} \in\# N$
shows *conflicting* $S' =$ *Some* $\{\#\} \wedge$ *unsatisfiable* (*set-mset* (*init-clss* S'))
⟨*proof*⟩

theorem 2.9.9 page 83 of Weidenbach's book

lemma *full-*$cdcl_W$*-stgy-final-state-conclusive*:
fixes S' :: *'st*
assumes *full*: *full* $cdcl_W$*-stgy* (*init-state* N) S' **and** *no-d*: *distinct-mset-mset* N
shows (*conflicting* $S' =$ *Some* $\{\#\} \wedge$ *unsatisfiable* (*set-mset* (*init-clss* S')))

$\vee$ (*conflicting* S' = *None* $\wedge$ *trail* S' $\models$*asm init-clss* S')
⟨*proof*⟩

theorem 2.9.9 page 83 of Weidenbach's book

lemma *full-cdcl*$_W$*-stgy-final-state-conclusive-from-init-state*:
fixes S' :: *'st*
assumes *full*: *full cdcl*$_W$*-stgy* (*init-state* N) S'
and *no-d*: *distinct-mset-mset* N
shows (*conflicting* S' = *Some* {#} $\wedge$ *unsatisfiable* (*set-mset* N))
$\vee$ (*conflicting* S' = *None* $\wedge$ *trail* S' $\models$*asm* N $\wedge$ *satisfiable* (*set-mset* N))
⟨*proof*⟩

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

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

2.1.6 Termination

The condition that no learned clause is a tautology is overkill (in the sense that the no-duplicate condition is enough), but we can reuse *simple-clss*.

The invariant contains all the structural invariants that holds,

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

lemma *cdcl*$_W$*-all-struct-inv-inv*:
assumes *cdcl*$_W$ S S' **and** *cdcl*$_W$*-all-struct-inv* S
shows *cdcl*$_W$*-all-struct-inv* S'
⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-all-struct-inv-inv*:
assumes *cdcl*$_W$** S S' **and** *cdcl*$_W$*-all-struct-inv* S
shows *cdcl*$_W$*-all-struct-inv* S'
⟨*proof*⟩

lemma *cdcl*$_W$*-stgy-cdcl*$_W$*-all-struct-inv*:
cdcl$_W$*-stgy* S T $\Longrightarrow$ *cdcl*$_W$*-all-struct-inv* S $\Longrightarrow$ *cdcl*$_W$*-all-struct-inv* T
⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-stgy-cdcl*$_W$*-all-struct-inv*:
cdcl$_W$*-stgy*** S T $\Longrightarrow$ *cdcl*$_W$*-all-struct-inv* S $\Longrightarrow$ *cdcl*$_W$*-all-struct-inv* T
⟨*proof*⟩

No Relearning of a clause

lemma *$cdcl_W$-o-new-clause-learned-is-backtrack-step*:
assumes *learned*: $D \in\#$ *learned-clss* T **and**
new: $D \notin\#$ *learned-clss* S **and**
$cdcl_W$: *$cdcl_W$-o* S T **and**
lev: *$cdcl_W$-M-level-inv* S
shows *backtrack* S T $\wedge$ *conflicting* S = *Some* D
⟨*proof*⟩

lemma *$cdcl_W$-cp-new-clause-learned-has-backtrack-step*:
assumes *learned*: $D \in\#$ *learned-clss* T **and**
new: $D \notin\#$ *learned-clss* S **and**
$cdcl_W$: *$cdcl_W$-stgy* S T **and**
lev: *$cdcl_W$-M-level-inv* S
shows $\exists S'$. *backtrack* S S' $\wedge$ *$cdcl_W$-stgy*** S' T $\wedge$ *conflicting* S = *Some* D
⟨*proof*⟩

lemma *rtranclp-$cdcl_W$-cp-new-clause-learned-has-backtrack-step*:
assumes *learned*: $D \in\#$ *learned-clss* T **and**
new: $D \notin\#$ *learned-clss* S **and**
$cdcl_W$: *$cdcl_W$-stgy*** S T **and**
lev: *$cdcl_W$-M-level-inv* S
shows $\exists S'\ S''$. *$cdcl_W$-stgy*** S S' $\wedge$ *backtrack* S' S'' $\wedge$ *conflicting* S' = *Some* D $\wedge$
$cdcl_W$-stgy** S'' T
⟨*proof*⟩

lemma *propagate-no-more-Decided-lit*:
assumes *propagate* S S'
shows *Decided* $K \in$ *set* (*trail* S) $\longleftrightarrow$ *Decided* $K \in$ *set* (*trail* S')
⟨*proof*⟩

lemma *conflict-no-more-Decided-lit*:
assumes *conflict* S S'
shows *Decided* $K \in$ *set* (*trail* S) $\longleftrightarrow$ *Decided* $K \in$ *set* (*trail* S')
⟨*proof*⟩

lemma *$cdcl_W$-cp-no-more-Decided-lit*:
assumes *$cdcl_W$-cp* S S'
shows *Decided* $K \in$ *set* (*trail* S) $\longleftrightarrow$ *Decided* $K \in$ *set* (*trail* S')
⟨*proof*⟩

lemma *rtranclp-$cdcl_W$-cp-no-more-Decided-lit*:
assumes *$cdcl_W$-cp*** S S'
shows *Decided* $K \in$ *set* (*trail* S) $\longleftrightarrow$ *Decided* $K \in$ *set* (*trail* S')
⟨*proof*⟩

lemma *$cdcl_W$-o-no-more-Decided-lit*:
assumes *$cdcl_W$-o* S S' **and** *lev*: *$cdcl_W$-M-level-inv* S **and** $\neg$*decide* S S'
shows *Decided* $K \in$ *set* (*trail* S') $\longrightarrow$ *Decided* $K \in$ *set* (*trail* S)
⟨*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 # M **and**

$trail\ S = M$
shows $\exists\, T.\ decide\ S\ T \wedge no\text{-}step\ cdcl_W\text{-}cp\ S$
⟨proof⟩

lemma $cdcl_W\text{-}o\text{-}is\text{-}decide$:
assumes $cdcl_W\text{-}o\ S\ T$ **and** lev: $cdcl_W\text{-}M\text{-}level\text{-}inv\ S$
$trail\ T = drop\ (length\ M_0)\ M' @ Decided\ L \# H @ M$**and**
$\neg\ (\exists\, M'.\ trail\ S = M' @ Decided\ L \# H @ M)$
shows $decide\ S\ T$
⟨proof⟩

lemma $rtranclp\text{-}cdcl_W\text{-}new\text{-}decided\text{-}at\text{-}beginning\text{-}is\text{-}decide$:
assumes $cdcl_W\text{-}stgy^{**}\ R\ U$ **and**
$trail\ U = M' @ Decided\ L \# H @ M$ **and**
$trail\ R = M$ **and**
$cdcl_W\text{-}M\text{-}level\text{-}inv\ R$
shows
$\exists\, S\ T\ T'.\ cdcl_W\text{-}stgy^{**}\ R\ S \wedge decide\ S\ T \wedge cdcl_W\text{-}stgy^{**}\ T\ U \wedge cdcl_W\text{-}stgy^{**}\ S\ U \wedge$
$no\text{-}step\ cdcl_W\text{-}cp\ S \wedge trail\ T = Decided\ L \# H @ M \wedge trail\ S = H @ M \wedge cdcl_W\text{-}stgy\ S\ T' \wedge$
$cdcl_W\text{-}stgy^{**}\ T'\ U$
⟨proof⟩

lemma $rtranclp\text{-}cdcl_W\text{-}new\text{-}decided\text{-}at\text{-}beginning\text{-}is\text{-}decide'$:
assumes $cdcl_W\text{-}stgy^{**}\ R\ U$ **and**
$trail\ U = M' @ Decided\ L \# H @ M$ **and**
$trail\ R = M$ **and**
$cdcl_W\text{-}M\text{-}level\text{-}inv\ R$
shows $\exists\, y\ y'.\ cdcl_W\text{-}stgy^{**}\ R\ y \wedge cdcl_W\text{-}stgy\ y\ y' \wedge \neg\ (\exists\, c.\ trail\ y = c @ Decided\ L \# H @ M)$
$\wedge\ (\lambda a\ b.\ cdcl_W\text{-}stgy\ a\ b \wedge (\exists\, c.\ trail\ a = c @ Decided\ L \# H @ M))^{**}\ y'\ U$
⟨proof⟩

lemma $beginning\text{-}not\text{-}decided\text{-}invert$:
assumes A: $M @ A = M' @ Decided\ K \# H$ **and**
nm: $\forall\, m{\in}set\ M.\ \neg is\text{-}decided\ m$
shows $\exists\, M.\ A = M @ Decided\ K \# H$
⟨proof⟩

lemma $cdcl_W\text{-}stgy\text{-}trail\text{-}has\text{-}new\text{-}decided\text{-}is\text{-}decide\text{-}step$:
assumes $cdcl_W\text{-}stgy\ S\ T$
$\neg\ (\exists\, c.\ trail\ S = c @ Decided\ L \# H @ M)$ **and**
$(\lambda a\ b.\ cdcl_W\text{-}stgy\ a\ b \wedge (\exists\, c.\ trail\ a = c @ Decided\ L \# H @ M))^{**}\ T\ U$ **and**
$\exists\, M'.\ trail\ U = M' @ Decided\ L \# H @ M$ **and**
lev: $cdcl_W\text{-}M\text{-}level\text{-}inv\ S$
shows $\exists\, S'.\ decide\ S\ S' \wedge full\ cdcl_W\text{-}cp\ S'\ T \wedge no\text{-}step\ cdcl_W\text{-}cp\ S$
⟨proof⟩

lemma $rtranclp\text{-}cdcl_W\text{-}stgy\text{-}with\text{-}trail\text{-}end\text{-}has\text{-}trail\text{-}end$:
assumes $(\lambda a\ b.\ cdcl_W\text{-}stgy\ a\ b \wedge (\exists\, c.\ trail\ a = c @ Decided\ L \# H @ M))^{**}\ T\ U$ **and**
$\exists\, M'.\ trail\ U = M' @ Decided\ L \# H @ M$
shows $\exists\, M'.\ trail\ T = M' @ Decided\ L \# H @ M$
⟨proof⟩

lemma $remove1\text{-}mset\text{-}eq\text{-}remove1\text{-}mset\text{-}same$:
$remove1\text{-}mset\ L\ D = remove1\text{-}mset\ L'\ D \Longrightarrow L \in\#\ D \Longrightarrow L = L'$
⟨proof⟩

lemma *cdcl*$_W$*-o-cannot-learn*:
  assumes
    cdcl$_W$*-o* y z **and**
    lev: *cdcl*$_W$*-M-level-inv* y **and**
    M: *trail* y = c @ *Decided Kh* # H **and**
    DL: D ∉# *learned-clss* y **and**
    LD: L ∈# D **and**
    DH: *atms-of* (*remove1-mset* L D) ⊆ *atm-of* ‘ *lits-of-l* H **and**
    LH: *atm-of* L ∉ *atm-of* ‘ *lits-of-l* H **and**
    learned: ∀ T. *conflicting* y = *Some* T ⟶ *trail* y ⊨*as CNot* T **and**
    z: *trail* z = c' @ *Decided Kh* # H
  shows D ∉# *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* # H **and**
    D ∉# *learned-clss* y **and**
    LD: L ∈# D **and**
    DH: *atms-of* (*remove1-mset* L D) ⊆ *atm-of* ‘ *lits-of-l* H **and**
    LH: *atm-of* L ∉ *atm-of* ‘ *lits-of-l* H **and**
    ∀ T. *conflicting* y = *Some* T ⟶ *trail* y ⊨*as CNot* T **and**
    trail z = c' @ *Decided Kh* # H
  shows D ∉# *learned-clss* z
  ⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-stgy-with-trail-end-has-not-been-learned*:
  assumes
    (λa b. *cdcl*$_W$*-stgy* a b ∧ (∃ c. *trail* a = c @ *Decided* K# H @ []))** S z **and**
    cdcl$_W$*-all-struct-inv* S **and**
    trail S = c @ *Decided* K # H **and**
    D ∉# *learned-clss* S **and**
    LD: L ∈# D **and**
    DH: *atms-of* (*remove1-mset* L D) ⊆ *atm-of* ‘ *lits-of-l* H **and**
    LH: *atm-of* L ∉ *atm-of* ‘ *lits-of-l* H **and**
    ∃ c'. *trail* z = c' @ *Decided* K # H
  shows D ∉# *learned-clss* z
  ⟨*proof*⟩

lemma *cdcl*$_W$*-stgy-new-learned-clause*:
  assumes *cdcl*$_W$*-stgy* S T **and**
    lev: *cdcl*$_W$*-M-level-inv* S **and**
    E ∉# *learned-clss* S **and**
    E ∈# *learned-clss* T
  shows ∃ S'. *backtrack* S S' ∧ *conflicting* S = *Some* E ∧ *full cdcl*$_W$*-cp* S' T
  ⟨*proof*⟩

theorem 2.9.7 page 83 of Weidenbach's book

lemma *cdcl*$_W$*-stgy-no-relearned-clause*:
  assumes
    invR: *cdcl*$_W$*-all-struct-inv* R **and**
    st': *cdcl*$_W$*-stgy*** R S **and**
    bt: *backtrack* S T **and**
    confl: *conflicting* S = *Some* E **and**

already-learned: $E \in\# $ *clauses* S **and**
R: *trail* $R = []$
shows *False*
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-stgy-distinct-mset-clauses*:
assumes
invR: *cdcl$_W$-all-struct-inv* R **and**
st: *cdcl$_W$-stgy*** R S **and**
dist: *distinct-mset* (*clauses* R) **and**
R: *trail* $R = []$
shows *distinct-mset* (*clauses* S)
⟨*proof*⟩

lemma *cdcl$_W$-stgy-distinct-mset-clauses*:
assumes
st: *cdcl$_W$-stgy*** (*init-state* N) S **and**
no-duplicate-clause: *distinct-mset* N **and**
no-duplicate-in-clause: *distinct-mset-mset* N
shows *distinct-mset* (*clauses* S)
⟨*proof*⟩

Decrease of a Measure

fun *cdcl$_W$-measure* **where**
cdcl$_W$-measure S =
[(*3*::*nat*) ^ (*card* (*atms-of-mm* (*init-clss* S))) − *card* (*set-mset* (*learned-clss* S)),
if conflicting S = *None then 1 else 0*,
if conflicting S = *None then card* (*atms-of-mm* (*init-clss* S)) − *length* (*trail* S)
else length (*trail* S)
]

lemma *length-model-le-vars-all-inv*:
assumes *cdcl$_W$-all-struct-inv* S
shows *length* (*trail* S) ≤ *card* (*atms-of-mm* (*init-clss* S))
⟨*proof*⟩
end

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

lemma *learned-clss-less-upper-bound*:
fixes S :: *'st*
assumes
distinct-cdcl$_W$-state S **and**
∀ $s \in\#$ *learned-clss* S. ¬*tautology* s
shows *card*(*set-mset* (*learned-clss* S)) ≤ *3* ^ *card* (*atms-of-mm* (*learned-clss* S))
⟨*proof*⟩

lemma *cdcl$_W$-measure-decreasing*:
fixes S :: *'st*
assumes
cdcl$_W$ S S' **and**
no-restart:
¬(*learned-clss* $S \subseteq\#$ *learned-clss* S' ∧ [] = *trail* S' ∧ *conflicting* S' = *None*)

and
no-forget: *learned-clss* $S \subseteq\#$ *learned-clss* S' **and**
no-relearn: $\bigwedge S'$. *backtrack* S $S' \Longrightarrow \forall T$. *conflicting* S = *Some* $T \longrightarrow T \notin\#$ *learned-clss* S
and
alien: *no-strange-atm* S **and**
M-level: $cdcl_W$*-M-level-inv* S **and**
no-taut: $\forall s \in\#$ *learned-clss* S. $\neg$*tautology* s **and**
no-dup: *distinct-*$cdcl_W$*-state* S **and**
confl: $cdcl_W$*-conflicting* S
shows ($cdcl_W$*-measure* S', $cdcl_W$*-measure* S) $\in$ *lexn less-than 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) $\in$ *lexn less-than 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) $\in$ *lexn less-than 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 less-than 3*
⟨*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 less-than 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 less-than 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 less-than 3*
⟨*proof*⟩

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

lemma *tranclp-*$cdcl_W$*-stgy-decreasing*:
fixes R S T :: *'st*
assumes $cdcl_W$*-stgy*$^{++}$ R S
trail R = [] **and**

cdcl$_W$*-all-struct-inv R*
shows (*cdcl*$_W$*-measure S*, *cdcl*$_W$*-measure R*) $\in$ *lexn less-than 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 less-than 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*⟩

lemma *cdcl*$_W$*-cp-wf-all-inv*:
wf {(*S'*, *S*). *cdcl*$_W$*-all-struct-inv S* $\wedge$ *cdcl*$_W$*-cp S S'*}
(**is** *wf ?R*)
⟨*proof*⟩

end

end

2.2 Merging backjump rules

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

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

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

2.2.1 Inclusion of the states

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

lemma *backtrack-no-cdcl*$_W$*-bj*:
assumes *cdcl*: *cdcl*$_W$*-bj T U* **and** *inv*: *cdcl*$_W$*-M-level-inv V*
shows $\neg$*backtrack V T*
⟨*proof*⟩

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

inductive *skip-or-resolve* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
s-or-r-skip[*intro*]: *skip S T* $\Longrightarrow$ *skip-or-resolve S T* |

s-or-r-resolve[*intro*]: *resolve S T* $\Longrightarrow$ *skip-or-resolve S T*

lemma *rtranclp-cdcl$_W$-bj-skip-or-resolve-backtrack*:
assumes *cdcl$_W$-bj** S U* **and** *inv*: *cdcl$_W$-M-level-inv S*
shows *skip-or-resolve** S U* $\vee$ ($\exists$ *T*. *skip-or-resolve** S T* $\wedge$ *backtrack T U*)
⟨*proof*⟩

lemma *rtranclp-skip-or-resolve-rtranclp-cdcl$_W$*:
skip-or-resolve** S T $\Longrightarrow$ *cdcl$_W$** S T*
⟨*proof*⟩

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

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

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

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

2.2.2 More lemmas conflict–propagate and backjumping

Termination

lemma *cdcl$_W$-cp-normalized-element-all-inv*:
assumes *inv*: *cdcl$_W$-all-struct-inv S*
obtains *T* **where** *full cdcl$_W$-cp S T*
⟨*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*)
init-clss S = init-clss T
learned-clss S = learned-clss T
backtrack-lvl S = backtrack-lvl T
conflicting S = conflicting T
⟨*proof*⟩

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*⟩

See also ⟦$skip^{**}$ *?S ?T*; *backtrack ?T ?W*; $cdcl_W$*-all-struct-inv ?S*⟧ ⟹ *backtrack ?S ?W*

lemma *rtranclp-skip-backtrack-backtrack-end*:
 assumes
  skip: $skip^{**}$ *S T* **and**
  bt: *backtrack S W* **and**
  inv: $cdcl_W$*-all-struct-inv S*
 shows *backtrack T W*
 ⟨*proof*⟩

lemma $cdcl_W$*-bj-decomp-resolve-skip-and-bj*:
 assumes $cdcl_W$*-bj*** *S T* **and** *inv*: $cdcl_W$*-M-level-inv S*
 shows (*skip-or-resolve*** *S T*
  ∨ (∃ *U*. *skip-or-resolve*** *S U* ∧ *backtrack U T*))
 ⟨*proof*⟩

lemma *resolve-skip-deterministic*:
 resolve S T ⟹ *skip S U* ⟹ *False*
 ⟨*proof*⟩

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

lemma *backtrack-unique*:
 assumes
  bt-T: *backtrack S T* **and**
  bt-U: *backtrack S U* **and**
  inv: $cdcl_W$*-all-struct-inv S*
 shows *T* ∼ *U*
⟨*proof*⟩

lemma *if-can-apply-backtrack-no-more-resolve*:
 assumes
  skip: $skip^{**}$ *S U* **and**
  bt: *backtrack S T* **and**
  inv: $cdcl_W$*-all-struct-inv S*
 shows ¬*resolve U V*
⟨*proof*⟩

lemma *if-can-apply-resolve-no-more-backtrack*:
 assumes
  skip: $skip^{**}$ *S U* **and**
  resolve: *resolve S T* **and**

inv: $cdcl_W$*-all-struct-inv S*
shows $\neg$*backtrack U V*
⟨*proof*⟩

lemma *if-can-apply-backtrack-skip-or-resolve-is-skip*:
assumes
bt: *backtrack S T* **and**
skip: *skip-or-resolve*** *S U* **and**
inv: $cdcl_W$*-all-struct-inv S*
shows *skip*** *S U*
⟨*proof*⟩

lemma $cdcl_W$*-bj-bj-decomp*:
assumes $cdcl_W$*-bj*** *S W* **and** $cdcl_W$*-all-struct-inv S*
shows
$(\exists\, T\ U\ V.\ (\lambda S\ T.\ \textit{skip-or-resolve}\ S\ T \wedge \textit{no-step backtrack}\ S)^{**}\ S\ T$
$\wedge\ (\lambda T\ U.\ \textit{resolve}\ T\ U \wedge \textit{no-step backtrack}\ T)\ T\ U$
$\wedge\ \textit{skip}^{**}\ U\ V \wedge \textit{backtrack}\ V\ W)$
$\vee\ (\exists\, T\ U.\ (\lambda S\ T.\ \textit{skip-or-resolve}\ S\ T \wedge \textit{no-step backtrack}\ S)^{**}\ S\ T$
$\wedge\ (\lambda T\ U.\ \textit{resolve}\ T\ U \wedge \textit{no-step backtrack}\ T)\ T\ U \wedge \textit{skip}^{**}\ U\ W)$
$\vee\ (\exists\, T.\ \textit{skip}^{**}\ S\ T \wedge \textit{backtrack}\ T\ W)$
$\vee\ \textit{skip}^{**}\ S\ W$ (**is** *?RB S W* $\vee$ *?R S W* $\vee$ *?SB S W* $\vee$ *?S S W*)
⟨*proof*⟩

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 $T \sim V \vee cdcl_W\text{-}bj^{**}\ T\ V$
⟨*proof*⟩

lemma $cdcl_W$*-bj-unique-normal-form*:
assumes
ST: $cdcl_W$*-bj*** *S T* **and** *SU*: $cdcl_W$*-bj*** *S U* **and**
n-s-U: *no-step* $cdcl_W$*-bj U* **and**
n-s-T: *no-step* $cdcl_W$*-bj T* **and**
inv: $cdcl_W$*-all-struct-inv S*
shows $T \sim U$
⟨*proof*⟩

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 $T \sim U$
⟨*proof*⟩

2.2.3 CDCL FW

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

fw-r-rf: $cdcl_W$*-rf* S S' $\Longrightarrow$ $cdcl_W$*-merge-restart* S S'

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

lemma $cdcl_W$*-merge-restart-*$cdcl_W$:
  assumes $cdcl_W$*-merge-restart* S T
  shows $cdcl_W{}^{**}$ S T
  ⟨*proof*⟩

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
  ⟨*proof*⟩

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

lemma $cdcl_W$*-merge-*$cdcl_W$*-merge-restart*:
  $cdcl_W$*-merge* S T $\Longrightarrow$ $cdcl_W$*-merge-restart* S T
  ⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-merge-tranclp-*$cdcl_W$*-merge-restart*:
  $cdcl_W$*-merge*** S T $\Longrightarrow$ $cdcl_W$*-merge-restart*** S T
  ⟨*proof*⟩

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

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

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

lemma $cdcl_W$*-all-struct-inv-tranclp-*$cdcl_W$*-merge-tranclp-*$cdcl_W$*-merge-*$cdcl_W$*-all-struct-inv*:
  assumes
    inv: $cdcl_W$*-all-struct-inv* b
    $cdcl_W$*-merge*$^{++}$ b a
  shows (λS T. $cdcl_W$*-all-struct-inv* S $\wedge$ $cdcl_W$*-merge* S T)$^{++}$ b a
  ⟨*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 ($cdcl_W$*-merge-restart*** S V $\wedge$ *conflicting* V = *None*)

$\vee$ ($\exists$ T U. $cdcl_W$-$merge$-$restart^{**}$ S T $\wedge$ $conflicting$ V $\neq$ $None$ $\wedge$ $conflict$ T U $\wedge$ $cdcl_W$-bj^{**} U V)
⟨*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 *cdcl*$_W$*-merge-restart-no-step-cdcl*$_W$*-bj*:
assumes
cdcl$_W$*-merge-restart* S T
shows *no-step cdcl*$_W$*-bj* T
⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-merge-restart-no-step-cdcl*$_W$*-bj*:
assumes
cdcl$_W$*-merge-restart*** S T **and**
conflicting S = *None*
shows *no-step cdcl*$_W$*-bj* T
⟨*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-state* N) V $\longleftrightarrow$ *full cdcl*$_W$*-merge-restart* (*init-state* N) V
⟨*proof*⟩

2.2.4 FW with strategy

The intermediate step

inductive *cdcl*$_W$*-s'* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
conflict': *full1 cdcl*$_W$*-cp* S S' $\Longrightarrow$ *cdcl*$_W$*-s'* S S' |
decide': *decide* S S' $\Longrightarrow$ *no-step cdcl*$_W$*-cp* S $\Longrightarrow$ *full cdcl*$_W$*-cp* S' S'' $\Longrightarrow$ *cdcl*$_W$*-s'* S S'' |
bj': *full1 cdcl*$_W$*-bj* S S' $\Longrightarrow$ *no-step cdcl*$_W$*-cp* S $\Longrightarrow$ *full cdcl*$_W$*-cp* S' S'' $\Longrightarrow$ *cdcl*$_W$*-s'* S S''

inductive-cases *cdcl*$_W$*-s'E*: *cdcl*$_W$*-s'* S T

lemma *rtranclp-cdcl*$_W$*-bj-full1-cdclp-cdcl*$_W$*-stgy*:
cdcl$_W$*-bj*** S S' $\Longrightarrow$ *full cdcl*$_W$*-cp* S' S'' $\Longrightarrow$ *cdcl*$_W$*-stgy*** S S''
⟨*proof*⟩

lemma *cdcl*$_W$*-s'-is-rtranclp-cdcl*$_W$*-stgy*:

$cdcl_W\text{-}s'\ S\ T \Longrightarrow cdcl_W\text{-}stgy^{**}\ S\ T$
⟨proof⟩

lemma *cdcl$_W$-cp-cdcl$_W$-bj-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$_W$-cp-cdcl$_W$-bj-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$_W$-stgy-cdcl$_W$-s$'$-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$_W$-stgy-cdcl$_W$-s$'$-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$_W$-stgy-cdcl$_W$-s$'$-no-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-cdcl$_W$-stgy-connected-to-rtranclp-cdcl$_W$-s$'$*:
assumes $cdcl_W\text{-}stgy^{**}\ S\ U$ **and** *inv*: $cdcl_W\text{-}M\text{-}level\text{-}inv\ S$
shows $cdcl_W\text{-}s'^{**}\ S\ U \vee (\exists\ T.\ cdcl_W\text{-}s'^{**}\ S\ T \wedge cdcl_W\text{-}bj^{++}\ T\ U \wedge conflicting\ U \neq None)$
⟨proof⟩

lemma *n-step-cdcl$_W$-stgy-iff-no-step-cdcl$_W$-cl-cdcl$_W$-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** $?S'\ S \longleftrightarrow\ ?C\ S \wedge\ ?O\ S$)
⟨proof⟩

lemma *cdcl$_W$-s$'$-tranclp-cdcl$_W$*:
$cdcl_W\text{-}s'\ S\ S' \Longrightarrow cdcl_W^{++}\ S\ S'$
⟨proof⟩

lemma *tranclp-cdcl$_W$-s'-tranclp-cdcl$_W$*:
cdcl$_W$-s'$^{++}$ S S' $\Longrightarrow$ *cdcl$_W$$^{++}$ S S'*
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-s'-rtranclp-cdcl$_W$*:
cdcl$_W$-s'** S S' $\Longrightarrow$ *cdcl$_W$** S S'*
⟨*proof*⟩

lemma *full-cdcl$_W$-stgy-iff-full-cdcl$_W$-s'*:
assumes *inv*: *cdcl$_W$-all-struct-inv S*
shows *full cdcl$_W$-stgy S T* $\longleftrightarrow$ *full cdcl$_W$-s' S T* (**is** *?S* $\longleftrightarrow$ *?S'*)
⟨*proof*⟩

lemma *conflict-step-cdcl$_W$-stgy-step*:
assumes
conflict S T
cdcl$_W$-all-struct-inv S
shows $\exists$ *T. cdcl$_W$-stgy S T*
⟨*proof*⟩

lemma *decide-step-cdcl$_W$-stgy-step*:
assumes
decide S T
cdcl$_W$-all-struct-inv S
shows $\exists$ *T. cdcl$_W$-stgy S T*
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-cp-conflicting-Some*:
cdcl$_W$-cp** S T $\Longrightarrow$ *conflicting S = Some D* $\Longrightarrow$ *S = T*
⟨*proof*⟩

inductive *cdcl$_W$-merge-cp* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
conflict': *conflict S T* $\Longrightarrow$ *full cdcl$_W$-bj T U* $\Longrightarrow$ *cdcl$_W$-merge-cp S U* |
propagate': *propagate^{++} S S'* $\Longrightarrow$ *cdcl$_W$-merge-cp S S'*

lemma *cdcl$_W$-merge-restart-cases*[*consumes 1*, *case-names conflict propagate*]:
assumes
cdcl$_W$-merge-cp S U **and**
$\bigwedge$*T. conflict S T* $\Longrightarrow$ *full cdcl$_W$-bj T U* $\Longrightarrow$ *P* **and**
propagate^{++} S U $\Longrightarrow$ *P*
shows *P*
⟨*proof*⟩

lemma *cdcl$_W$-merge-cp-tranclp-cdcl$_W$-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*⟩

Full Transformation

inductive $cdcl_W$*-s'-without-decide* **where**
conflict'-without-decide[intro]: *full1* $cdcl_W$*-cp* S S' $\Longrightarrow$ $cdcl_W$*-s'-without-decide* S S' |
bj'-without-decide[intro]: *full1* $cdcl_W$*-bj* S S' $\Longrightarrow$ *no-step* $cdcl_W$*-cp* S $\Longrightarrow$ *full* $cdcl_W$*-cp* S' S''
$\Longrightarrow$ $cdcl_W$*-s'-without-decide* S S''

lemma *rtranclp-*$cdcl_W$*-s'-without-decide-rtranclp-*$cdcl_W$:
$cdcl_W$*-s'-without-decide*** S T $\Longrightarrow$ $cdcl_W{}^{**}$ S T
⟨*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* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
conflict': *full1 cdcl*$_W$*-s'-without-decide* S $S' \Longrightarrow$ *cdcl*$_W$*-s'-w* S S' |
decide': *decide* S $S' \Longrightarrow$ *no-step cdcl*$_W$*-s'-without-decide* $S \Longrightarrow$ *full cdcl*$_W$*-s'-without-decide* S' S''
  $\Longrightarrow$ *cdcl*$_W$*-s'-w* S S''

lemma *cdcl*$_W$*-s'-w-rtranclp-cdcl*$_W$:
  cdcl$_W$*-s'-w* S $T \Longrightarrow$ *cdcl*$_W$** S T
  ⟨*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 $S = T \vee$ *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*⟩
end

Termination and full Equivalence

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

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

lemma *wf-tranclp-cdcl*$_W$*-merge*: *wf* $\{(T, S).$ *cdcl*$_W$*-all-struct-inv* $S \wedge$ *cdcl*$_W$*-merge*$^{++}$ $S\ T\}$
⟨*proof*⟩

lemma *wf-cdcl*$_W$*-merge-cp*:
wf$\{(T, S).$ *cdcl*$_W$*-all-struct-inv* $S \wedge$ *cdcl*$_W$*-merge-cp* $S\ T\}$
⟨*proof*⟩

lemma *wf-cdcl*$_W$*-merge-stgy*:
wf$\{(T, S).$ *cdcl*$_W$*-all-struct-inv* $S \wedge$ *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*⟩

end

end
theory *CDCL-WNOT*

imports *CDCL-NOT CDCL-W-Termination CDCL-W-Merge*
begin

2.3 Link between Weidenbach's and NOT's CDCL

2.3.1 Inclusion of the states

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

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

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

lemma *lits-of-l-convert-trail-from-W*[*simp*]:
lits-of-l (*convert-trail-from-W M*) = *lits-of-l M*
⟨*proof*⟩

lemma *lit-of-convert-trail-from-W*[*simp*]:
lit-of (*convert-ann-lit-from-W L*) = *lit-of L*
⟨*proof*⟩

lemma *no-dup-convert-from-W*[*simp*]:
no-dup (*convert-trail-from-W M*) ⟷ *no-dup M*
⟨*proof*⟩

lemma *convert-trail-from-W-true-annots*[*simp*]:
convert-trail-from-W M ⊨*as C* ⟷ *M* ⊨*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.

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

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

lemma *undefined-lit-convert-trail-from-NOT*[*simp*]:
undefined-lit (*convert-trail-from-NOT F*) *L* ⟷ *undefined-lit F L*
⟨*proof*⟩

lemma *lits-of-l-convert-trail-from-NOT*:
lits-of-l (*convert-trail-from-NOT F*) = *lits-of-l 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-lit-from-W (*convert-ann-lit-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-lit-from-NOT*[*iff*]:
lit-of (*convert-ann-lit-from-NOT L*) = *lit-of L*
⟨*proof*⟩

sublocale $state_W$ ⊆ *dpll-state-ops*
λ*S*. *convert-trail-from-W* (*trail S*)
clauses
λ*L S*. *cons-trail* (*convert-ann-lit-from-NOT L*) *S*
λ*S*. *tl-trail S*
λ*C S*. *add-learned-cls C S*
λ*C S*. *remove-cls C S*
⟨*proof*⟩

sublocale $state_W$ ⊆ *dpll-state*
λ*S*. *convert-trail-from-W* (*trail S*)
clauses
λ*L S*. *cons-trail* (*convert-ann-lit-from-NOT L*) *S*
λ*S*. *tl-trail S*
λ*C S*. *add-learned-cls C S*
λ*C S*. *remove-cls C S*
⟨*proof*⟩

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

sublocale $conflict\text{-}driven\text{-}clause\text{-}learning_W$ ⊆ $cdcl_{NOT}$*-merge-bj-learn-ops*
λ*S*. *convert-trail-from-W* (*trail S*)
clauses
λ*L S*. *cons-trail* (*convert-ann-lit-from-NOT L*) *S*
λ*S*. *tl-trail S*
λ*C S*. *add-learned-cls C S*
λ*C S*. *remove-cls C S*
λ- -. *True*
λ- *S*. *conflicting S* = *None*
λ*C C' L' S T*. *backjump-l-cond C C' L' S T*
∧ *distinct-mset* (*C'* + {#*L'*#}) ∧ ¬*tautology* (*C'* + {#*L'*#})
⟨*proof*⟩

thm $cdcl_{NOT}$*-merge-bj-learn-proxy.axioms*
sublocale *conflict-driven-clause-learning*$_W$ $\subseteq$ $cdcl_{NOT}$*-merge-bj-learn-proxy*
  $\lambda S.$ *convert-trail-from-W* (*trail* S)
  clauses
  $\lambda L\ S.$ *cons-trail* (*convert-ann-lit-from-NOT* L) S
  $\lambda S.$ *tl-trail* S
  $\lambda C\ S.$ *add-learned-cls* C S
  $\lambda C\ S.$ *remove-cls* C S

  λ- -. *True*
  λ- $S.$ *conflicting* S = *None*
  backjump-l-cond
  inv_{NOT}
⟨*proof*⟩

sublocale *conflict-driven-clause-learning*$_W$ $\subseteq$ $cdcl_{NOT}$*-merge-bj-learn-proxy2*
  $\lambda S.$ *convert-trail-from-W* (*trail* S)
  clauses
  $\lambda L\ S.$ *cons-trail* (*convert-ann-lit-from-NOT* L) S
  $\lambda S.$ *tl-trail* S
  $\lambda C\ S.$ *add-learned-cls* C S
  $\lambda C\ S.$ *remove-cls* C S
  λ- -. *True*
  λ- $S.$ *conflicting* S = *None* *backjump-l-cond* inv_{NOT}
  ⟨*proof*⟩

sublocale *conflict-driven-clause-learning*$_W$ $\subseteq$ $cdcl_{NOT}$*-merge-bj-learn*
  $\lambda S.$ *convert-trail-from-W* (*trail* S)
  clauses
  $\lambda L\ S.$ *cons-trail* (*convert-ann-lit-from-NOT* L) S
  $\lambda S.$ *tl-trail* S
  $\lambda C\ S.$ *add-learned-cls* C S
  $\lambda C\ S.$ *remove-cls* C S
  backjump-l-cond
  λ- -. *True*
  λ- $S.$ *conflicting* S = *None* inv_{NOT}
  ⟨*proof*⟩

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

Notations are lost while proving locale inclusion:

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

2.3.2 Additional Lemmas between NOT and W states

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

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*:

$reduce\text{-}trail\text{-}to_{NOT}\ C\ S = reduce\text{-}trail\text{-}to\ (convert\text{-}trail\text{-}from\text{-}NOT\ C)\ S$
⟨proof⟩

lemma *reduce-trail-to-map*[*simp*]:
$reduce\text{-}trail\text{-}to\ (map\ f\ M)\ S = reduce\text{-}trail\text{-}to\ M\ S$
⟨proof⟩

lemma $reduce\text{-}trail\text{-}to_{NOT}\text{-}map$[*simp*]:
$reduce\text{-}trail\text{-}to_{NOT}\ (map\ f\ M)\ S = reduce\text{-}trail\text{-}to_{NOT}\ M\ S$
⟨proof⟩

lemma *skip-or-resolve-state-change*:
assumes $skip\text{-}or\text{-}resolve^{**}\ S\ T$
shows
$\exists M.\ trail\ S = M\ @\ trail\ T \wedge (\forall m \in set\ M.\ \neg is\text{-}decided\ m)$
$clauses\ S = clauses\ T$
$backtrack\text{-}lvl\ S = backtrack\text{-}lvl\ T$
⟨proof⟩

2.3.3 More lemmas conflict–propagate and backjumping

2.3.4 CDCL FW

lemma $cdcl_W\text{-}merge\text{-}is\text{-}cdcl_{NOT}\text{-}merged\text{-}bj\text{-}learn$:
assumes
inv: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$ **and**
$cdcl_W$: $cdcl_W\text{-}merge\ S\ T$
shows $cdcl_{NOT}\text{-}merged\text{-}bj\text{-}learn\ S\ T$
$\vee\ (no\text{-}step\ cdcl_W\text{-}merge\ T \wedge conflicting\ T \neq None)$
⟨proof⟩

abbreviation $cdcl_{NOT}\text{-}restart$ **where**
$cdcl_{NOT}\text{-}restart \equiv restart\text{-}ops.cdcl_{NOT}\text{-}raw\text{-}restart\ cdcl_{NOT}\ restart$

lemma $cdcl_W\text{-}merge\text{-}restart\text{-}is\text{-}cdcl_{NOT}\text{-}merged\text{-}bj\text{-}learn\text{-}restart\text{-}no\text{-}step$:
assumes
inv: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$ **and**
$cdcl_W$: $cdcl_W\text{-}merge\text{-}restart\ S\ T$
shows $cdcl_{NOT}\text{-}restart^{**}\ S\ T \vee (no\text{-}step\ cdcl_W\text{-}merge\ T \wedge conflicting\ T \neq None)$
⟨proof⟩

abbreviation μ_{FW} :: $'st \Rightarrow nat$ **where**
$\mu_{FW}\ S \equiv (if\ no\text{-}step\ cdcl_W\text{-}merge\ S\ then\ 0\ else\ 1+\mu_{CDCL}'\text{-}merged\ (set\text{-}mset\ (init\text{-}clss\ S))\ S)$

lemma $cdcl_W\text{-}merge\text{-}\mu_{FW}\text{-}decreasing$:
assumes
inv: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$ **and**
fw: $cdcl_W\text{-}merge\ S\ T$
shows $\mu_{FW}\ T < \mu_{FW}\ S$
⟨proof⟩

lemma $wf\text{-}cdcl_W\text{-}merge$: $wf\ \{(T, S).\ cdcl_W\text{-}all\text{-}struct\text{-}inv\ S \wedge cdcl_W\text{-}merge\ S\ T\}$
⟨proof⟩

sublocale $conflict\text{-}driven\text{-}clause\text{-}learning_W\text{-}termination$
⟨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* $\longleftrightarrow$ *full cdcl$_W$-merge-stgy R V* (**is** *?s'* $\longleftrightarrow$ *?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* $\longleftrightarrow$ *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* {#} $\wedge$ *unsatisfiable* (*set-mset N*))
$\vee$ (*conflicting S' = None* $\wedge$ *trail S'* $\models$*asm N* $\wedge$ *satisfiable* (*set-mset N*))
⟨*proof*⟩
end

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

2.4 Incremental SAT solving

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

— Some specific states:
init-state
restart-state
for
trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **and**
init-clss :: *'st* ⇒ *'v clauses* **and**
learned-clss :: *'st* ⇒ *'v clauses* **and**
backtrack-lvl :: *'st* ⇒ *nat* **and**
conflicting :: *'st* ⇒ *'v clause option* **and**

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

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

init-state :: *'v clauses* ⇒ *'st* **and**
restart-state :: *'st* ⇒ *'st* +
fixes
add-init-cls :: *'v clause* ⇒ *'st* ⇒ *'st*
assumes
trail-add-init-cls[*simp*]:
$\bigwedge$*st C*. *trail* (*add-init-cls C st*) = *trail st* **and**
init-clss-add-init-cls[*simp*]:
$\bigwedge$*st C*. *init-clss* (*add-init-cls C st*) = {#*C*#} + *init-clss st*
and
learned-clss-add-init-cls[*simp*]:
$\bigwedge$*st C*. *learned-clss* (*add-init-cls C st*) = *learned-clss st* **and**
backtrack-lvl-add-init-cls[*simp*]:
$\bigwedge$*st C*. *no-dup* (*trail st*) $\Longrightarrow$ *backtrack-lvl* (*add-init-cls C st*) = *backtrack-lvl st* **and**
conflicting-add-init-cls[*simp*]:
$\bigwedge$*st C*. *conflicting* (*add-init-cls C st*) = *conflicting st*
begin
lemma *clauses-add-init-cls*[*simp*]:
clauses (*add-init-cls N S*) = {#*N*#} + *init-clss S* + *learned-clss S*
⟨*proof*⟩

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

lemma *conflicting-add-init-cls-iff-conflicting*[*simp*]:
conflicting (*add-init-cls C S*) = *None* $\longleftrightarrow$ *conflicting S* = *None*
⟨*proof*⟩
end

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

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

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

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

tl-trail :: *'st* ⇒ *'st* **and**
add-learned-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
remove-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
update-backtrack-lvl :: *nat* ⇒ *'st* ⇒ *'st* **and**
update-conflicting :: *'v clause option* ⇒ *'st* ⇒ *'st* **and**

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

sublocale *conflict-driven-clause-learning*$_W$
⟨*proof*⟩

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* ∈# *C then S*
else cut-trail-wrt-clause C M (*tl-trail S*))

definition *add-new-clause-and-update* :: $'v$ *clause* $\Rightarrow$ $'st$ $\Rightarrow$ $'st$ **where**
add-new-clause-and-update C S =
(*if trail* S $\models$*as 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*:
$\exists M.$ *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 = *count-decided* (*trail* T)
shows
backtrack-lvl (*cut-trail-wrt-clause* C (*trail* T) T) =
count-decided (*trail* (*cut-trail-wrt-clause* C (*trail* T) T))
⟨*proof*⟩

lemma *cut-trail-wrt-clause-CNot-trail*:
assumes *trail* T $\models$*as CNot* C
shows
(*trail* ((*cut-trail-wrt-clause* C (*trail* T) T))) $\models$*as CNot* C
⟨*proof*⟩

lemma *cut-trail-wrt-clause-hd-trail-in-or-empty-trail*:
(($\forall L \in\# C.$ $-L \notin$ *lits-of-l* (*trail* T)) $\wedge$ *trail* (*cut-trail-wrt-clause* C (*trail* T) T) = [])
$\vee$ ($-$*lit-of* (*hd* (*trail* (*cut-trail-wrt-clause* C (*trail* T) T))) $\in\#$ C
$\wedge$ *length* (*trail* (*cut-trail-wrt-clause* C (*trail* T) T)) $\geq$ *1*)
⟨*proof*⟩

We can fully run $cdcl_W$*-s* or add a clause. Remark that we use $cdcl_W$*-s* to avoid an explicit *skip*, *resolve*, and *backtrack* normalisation to get rid of the conflict C if possible.

inductive *incremental-*$cdcl_W$ :: $'st$ $\Rightarrow$ $'st$ $\Rightarrow$ *bool* **for** S **where**
add-confl:
trail S $\models$*asm init-clss* S $\Longrightarrow$ *distinct-mset* C $\Longrightarrow$ *conflicting* S = *None* $\Longrightarrow$
trail S $\models$*as CNot* C $\Longrightarrow$
full $cdcl_W$*-stgy*

($update\text{-}conflicting$ ($Some$ C)
($add\text{-}init\text{-}cls$ C ($cut\text{-}trail\text{-}wrt\text{-}clause$ C ($trail$ S) S))) T $\Longrightarrow$
$incremental\text{-}cdcl_W$ S T |
add-no-confl:
$trail$ S $\models asm$ $init\text{-}clss$ S $\Longrightarrow$ $distinct\text{-}mset$ C $\Longrightarrow$ $conflicting$ S = $None$ $\Longrightarrow$
$\neg trail$ S $\models as$ $CNot$ C $\Longrightarrow$
$full$ $cdcl_W\text{-}stgy$ ($add\text{-}init\text{-}cls$ C S) T $\Longrightarrow$
$incremental\text{-}cdcl_W$ S T

lemma $cdcl_W\text{-}all\text{-}struct\text{-}inv\text{-}add\text{-}new\text{-}clause\text{-}and\text{-}update\text{-}cdcl_W\text{-}all\text{-}struct\text{-}inv$:
assumes
inv-T: $cdcl_W\text{-}all\text{-}struct\text{-}inv$ T **and**
tr-T-N[*simp*]: $trail$ T $\models asm$ N **and**
tr-C[*simp*]: $trail$ T $\models as$ $CNot$ C **and**
[*simp*]: $distinct\text{-}mset$ C
shows $cdcl_W\text{-}all\text{-}struct\text{-}inv$ ($add\text{-}new\text{-}clause\text{-}and\text{-}update$ C T) (**is** $cdcl_W\text{-}all\text{-}struct\text{-}inv$ $?T'$)
⟨*proof*⟩

lemma $cdcl_W\text{-}all\text{-}struct\text{-}inv\text{-}add\text{-}new\text{-}clause\text{-}and\text{-}update\text{-}cdcl_W\text{-}stgy\text{-}inv$:
assumes
inv-s: $cdcl_W\text{-}stgy\text{-}invariant$ T **and**
inv: $cdcl_W\text{-}all\text{-}struct\text{-}inv$ T **and**
tr-T-N[*simp*]: $trail$ T $\models asm$ N **and**
tr-C[*simp*]: $trail$ T $\models as$ $CNot$ C **and**
[*simp*]: $distinct\text{-}mset$ C
shows $cdcl_W\text{-}stgy\text{-}invariant$ ($add\text{-}new\text{-}clause\text{-}and\text{-}update$ C T)
(**is** $cdcl_W\text{-}stgy\text{-}invariant$ $?T'$)
⟨*proof*⟩

lemma $full\text{-}cdcl_W\text{-}stgy\text{-}inv\text{-}normal\text{-}form$:
assumes
full: $full$ $cdcl_W\text{-}stgy$ S T **and**
inv-s: $cdcl_W\text{-}stgy\text{-}invariant$ S **and**
inv: $cdcl_W\text{-}all\text{-}struct\text{-}inv$ S
shows $conflicting$ T = $Some$ {#} $\wedge$ $unsatisfiable$ ($set\text{-}mset$ ($init\text{-}clss$ S))
$\vee$ $conflicting$ T = $None$ $\wedge$ $trail$ T $\models asm$ $init\text{-}clss$ S $\wedge$ $satisfiable$ ($set\text{-}mset$ ($init\text{-}clss$ S))
⟨*proof*⟩

lemma $incremental\text{-}cdcl_W\text{-}inv$:
assumes
inc: $incremental\text{-}cdcl_W$ S T **and**
inv: $cdcl_W\text{-}all\text{-}struct\text{-}inv$ S **and**
s-inv: $cdcl_W\text{-}stgy\text{-}invariant$ S
shows
$cdcl_W\text{-}all\text{-}struct\text{-}inv$ T **and**
$cdcl_W\text{-}stgy\text{-}invariant$ T
⟨*proof*⟩

lemma $rtranclp\text{-}incremental\text{-}cdcl_W\text{-}inv$:
assumes
inc: $incremental\text{-}cdcl_W^{**}$ S T **and**
inv: $cdcl_W\text{-}all\text{-}struct\text{-}inv$ S **and**
s-inv: $cdcl_W\text{-}stgy\text{-}invariant$ S
shows
$cdcl_W\text{-}all\text{-}struct\text{-}inv$ T **and**
$cdcl_W\text{-}stgy\text{-}invariant$ 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*⟩

end

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

2.4.1 Adding Restarts

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

— get state:
init-state
restart-state
for
trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **and**
init-clss :: *'st* ⇒ *'v clauses* **and**
learned-clss :: *'st* ⇒ *'v clauses* **and**
backtrack-lvl :: *'st* ⇒ *nat* **and**
conflicting :: *'st* ⇒ *'v clause option* **and**

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

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

restart-state :: *'st* ⇒ *'st* +
fixes *f* :: *nat* ⇒ *nat*
assumes *f*: *unbounded f*
begin

The condition of the differences of cardinality has to be strict. Otherwise, you could be in a strange state, where nothing remains to do, but a restart is done. See the proof of well-foundedness.

inductive $cdcl_W$*-merge-with-restart* **where**
restart-step:
($cdcl_W$*-merge-stgy*^^(*card* (*set-mset* (*learned-clss T*)) − *card* (*set-mset* (*learned-clss S*)))) *S T*
⟹ *card* (*set-mset* (*learned-clss T*)) − *card* (*set-mset* (*learned-clss S*)) > *f n*
⟹ *restart T U* ⟹ $cdcl_W$*-merge-with-restart* (*S*, *n*) (*U*, *Suc n*) |
restart-full: *full1* $cdcl_W$*-merge-stgy S T* ⟹ $cdcl_W$*-merge-with-restart* (*S*, *n*) (*T*, *Suc n*)

lemma $cdcl_W$*-merge-with-restart S T* ⟹ $cdcl_W$*-merge-restart*** (*fst S*) (*fst T*)
⟨*proof*⟩

lemma $cdcl_W$*-merge-with-restart-rtranclp-*$cdcl_W$:
$cdcl_W$*-merge-with-restart S T* ⟹ $cdcl_W{}^{**}$ (*fst S*) (*fst T*)
⟨*proof*⟩

lemma $cdcl_W$*-merge-with-restart-increasing-number*:
$cdcl_W$*-merge-with-restart S T* ⟹ *snd T* = *1* + *snd S*
⟨*proof*⟩

lemma *full1* $cdcl_W$*-merge-stgy S T* ⟹ $cdcl_W$*-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*) ⊆ *simple-clss* (*atms-of-mm* (*init-clss S*))
⟨*proof*⟩

lemma $cdcl_W$*-merge-with-restart-init-clss*:
$cdcl_W$*-merge-with-restart S T* ⟹ $cdcl_W$*-M-level-inv* (*fst S*) ⟹
init-clss (*fst S*) = *init-clss* (*fst T*)
⟨*proof*⟩

lemma
wf {(*T*, *S*). $cdcl_W$*-all-struct-inv* (*fst S*) ∧ $cdcl_W$*-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* ⟹
card (*set-mset* (*learned-clss T*)) − *card* (*set-mset* (*learned-clss S*)) > *f n* ⟹

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*⟩

function *natlog2* :: *nat* ⇒ *nat* **where**
natlog2 n = (*if n* = *0 then 0 else 1* + *natlog2* (*n div 2*))
⟨*proof*⟩
termination ⟨*proof*⟩

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

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

lemma *two-pover-n-eq-two-power-n′-eq*:
assumes *H*: $(2::nat)$ ^ $(k::nat) - 1 = 2$ ^ $k' - 1$
shows $k' = k$
⟨*proof*⟩

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 ^ $(k - Suc\ 0) \le i$ **and**
k′: $i < 2$ ^ $k' - Suc\ 0$ **and**
k-k′: $k > k'$
shows *False*
⟨*proof*⟩

lemma *luby-sequence-core-not-two-power-minus-one*:
assumes
k-i: 2 ^ $(k - 1) \le i$ **and**
i-k: $i < 2$^$k - 1$
shows *luby-sequence-core i* = *luby-sequence-core* $(i - 2$ ^ $(k - 1) + 1)$
⟨*proof*⟩

lemma *unbounded-luby-sequence-core*: *unbounded luby-sequence-core*
⟨*proof*⟩

abbreviation *luby-sequence* :: *nat* ⇒ *nat* **where**
luby-sequence n ≡ *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* ≥ *1*
⟨*proof*⟩
end

locale *luby-sequence-restart* =
luby-sequence ur +
conflict-driven-clause-learning$_W$ — functions for clauses:
— functions for the state:

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

— get state:
init-state
restart-state
for
ur :: *nat* **and**
trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **and**
hd-trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lit* **and**
init-clss :: *'st* ⇒ *'v clauses* **and**
learned-clss :: *'st* ⇒ *'v clauses* **and**
backtrack-lvl :: *'st* ⇒ *nat* **and**
conflicting :: *'st* ⇒ *'v clause option* **and**

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

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

sublocale $cdcl_W$*-restart* - - - - - - - - - - - - - *luby-sequence*
⟨*proof*⟩

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

Chapter 3

Implementation of DPLL and CDCL

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

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

3.1 Simple Implementation of the DPLL and CDCL

3.1.1 Common Rules

Propagation

The following theorem holds:

lemma *lits-of-l-unfold*[*iff*]:
$(\forall\, c \in \mathit{set}\ C.\ -c \in \mathit{lits\text{-}of\text{-}l}\ \mathit{Ms}) \longleftrightarrow \mathit{Ms} \models\!\mathit{as}\ \mathit{CNot}\ (\mathit{mset}\ C)$
$\langle \mathit{proof} \rangle$

The right-hand version is written at a high-level, but only the left-hand side is executable.

definition *is-unit-clause* :: $'a\ \mathit{literal}\ \mathit{list} \Rightarrow ('a,\ 'b)\ \mathit{ann\text{-}lits} \Rightarrow\ 'a\ \mathit{literal}\ \mathit{option}$
where
$\mathit{is\text{-}unit\text{-}clause}\ l\ M =$
$(\mathit{case}\ \mathit{List.filter}\ (\lambda a.\ \mathit{atm\text{-}of}\ a \notin \mathit{atm\text{-}of}\ `\ \mathit{lits\text{-}of\text{-}l}\ M)\ l\ \mathit{of}$
$a \# [] \Rightarrow \mathit{if}\ M \models\!\mathit{as}\ \mathit{CNot}\ (\mathit{mset}\ l - \{\#a\#\})\ \mathit{then}\ \mathit{Some}\ a\ \mathit{else}\ \mathit{None}$
$|\ \text{-} \Rightarrow \mathit{None})$

definition *is-unit-clause-code* :: $'a\ \mathit{literal}\ \mathit{list} \Rightarrow ('a,\ 'b)\ \mathit{ann\text{-}lits}$
$\Rightarrow\ 'a\ \mathit{literal}\ \mathit{option}$ **where**
$\mathit{is\text{-}unit\text{-}clause\text{-}code}\ l\ M =$
$(\mathit{case}\ \mathit{List.filter}\ (\lambda a.\ \mathit{atm\text{-}of}\ a \notin \mathit{atm\text{-}of}\ `\ \mathit{lits\text{-}of\text{-}l}\ M)\ l\ \mathit{of}$
$a \# [] \Rightarrow \mathit{if}\ (\forall\, c \in \mathit{set}\ (\mathit{remove1}\ a\ l).\ -c \in \mathit{lits\text{-}of\text{-}l}\ M)\ \mathit{then}\ \mathit{Some}\ a\ \mathit{else}\ \mathit{None}$
$|\ \text{-} \Rightarrow \mathit{None})$

lemma *is-unit-clause-is-unit-clause-code*[*code*]:
$\mathit{is\text{-}unit\text{-}clause}\ l\ M = \mathit{is\text{-}unit\text{-}clause\text{-}code}\ l\ M$
$\langle \mathit{proof} \rangle$

lemma *is-unit-clause-some-undef*:
assumes $\mathit{is\text{-}unit\text{-}clause}\ l\ M = \mathit{Some}\ a$
shows $\mathit{undefined\text{-}lit}\ M\ a$

$\langle proof \rangle$

lemma *is-unit-clause-some-CNot*: $is\text{-}unit\text{-}clause\ l\ M = Some\ a \Longrightarrow M \models as\ CNot\ (mset\ l - \{\#a\#\})$
$\langle proof \rangle$

lemma *is-unit-clause-some-in*: $is\text{-}unit\text{-}clause\ l\ M = Some\ a \Longrightarrow a \in set\ l$
$\langle proof \rangle$

lemma *is-unit-clause-Nil*[*simp*]: $is\text{-}unit\text{-}clause\ []\ M = None$
$\langle proof \rangle$

Unit propagation for all clauses

Finding the first clause to propagate

fun *find-first-unit-clause* :: $'a\ literal\ list\ list \Rightarrow ('a, 'b)\ ann\text{-}lits$
$\Rightarrow ('a\ literal \times 'a\ literal\ list)\ option$ **where**
$find\text{-}first\text{-}unit\text{-}clause\ (a\ \#\ l)\ M =$
$(case\ is\text{-}unit\text{-}clause\ a\ M\ of$
$None \Rightarrow find\text{-}first\text{-}unit\text{-}clause\ l\ M$
$|\ Some\ L \Rightarrow Some\ (L, a))\ |$
$find\text{-}first\text{-}unit\text{-}clause\ []\ \text{-} = None$

lemma *find-first-unit-clause-some*:
$find\text{-}first\text{-}unit\text{-}clause\ l\ M = Some\ (a, c)$
$\Longrightarrow c \in set\ l \land\ M \models as\ CNot\ (mset\ c - \{\#a\#\}) \land undefined\text{-}lit\ M\ a \land a \in set\ c$
$\langle proof \rangle$

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 $is\text{-}unit\text{-}clause\ c\ M \neq None$
$\langle proof \rangle$

lemma *find-first-unit-clause-none*:
$distinct\ c \Longrightarrow c \in set\ l \Longrightarrow\ M \models as\ CNot\ (mset\ c - \{\#a\#\}) \Longrightarrow undefined\text{-}lit\ M\ a \Longrightarrow a \in set\ c$
$\Longrightarrow find\text{-}first\text{-}unit\text{-}clause\ l\ M \neq None$
$\langle proof \rangle$

Decide

fun *find-first-unused-var* :: $'a\ literal\ list\ list \Rightarrow 'a\ literal\ set \Rightarrow 'a\ literal\ option$ **where**
$find\text{-}first\text{-}unused\text{-}var\ (a\ \#\ l)\ M =$
$(case\ List.find\ (\lambda lit.\ lit \notin M \land -lit \notin M)\ a\ of$
$None \Rightarrow find\text{-}first\text{-}unused\text{-}var\ l\ M$
$|\ Some\ a \Rightarrow Some\ a)\ |$
$find\text{-}first\text{-}unused\text{-}var\ []\ \text{-} = None$

lemma *find-none*[*iff*]:
$List.find\ (\lambda lit.\ lit \notin M \land -lit \notin M)\ a = None \longleftrightarrow\ atm\text{-}of\ `\ set\ a \subseteq atm\text{-}of\ `\ M$
$\langle proof \rangle$

lemma *find-some*: $List.find\ (\lambda lit.\ lit \notin M \land -lit \notin M)\ a = Some\ b \Longrightarrow b \in set\ a \land b \notin M \land -b \notin M$
$\langle proof \rangle$

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-l* *Ms*) = *Some* $a \Longrightarrow$ *undefined-lit* *Ms* a
⟨*proof*⟩

3.1.2 CDCL specific functions

Level

fun *maximum-level-code*:: '*a literal list* $\Rightarrow$ ('*a*, '*b*) *ann-lits* $\Rightarrow$ *nat*
where
maximum-level-code [] - = *0* |
maximum-level-code (L # *Ls*) M = *max* (*get-level* M L) (*maximum-level-code* *Ls* M)

lemma *maximum-level-code-eq-get-maximum-level*[*simp*]:
maximum-level-code D M = *get-maximum-level* M (*mset* D)
⟨*proof*⟩

lemma [*code*]:
fixes M :: ('*a*, '*b*) *ann-lits*
shows *get-maximum-level* M (*mset* D) = *maximum-level-code* D M
⟨*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 # Ls*) = (*if count-decided Ls* = *i then Some* (*Decided K # Ls*) *else bt-cut i Ls*) |
bt-cut i [] = *None*

lemma *bt-cut-some-decomp*:
  assumes *no-dup M* **and** *bt-cut i M* = *Some M′*
  shows ∃ *K M2 M1*. *M* = *M2* @ *M′* ∧ *M′* = *Decided K # M1* ∧ *get-level M K* = (*i*+*1*)
  ⟨*proof*⟩

lemma *bt-cut-not-none*:
  assumes *no-dup M* **and** *M* = *M2* @ *Decided K # M′* **and** *get-level M K* = (*i*+*1*)
  shows *bt-cut i M* ≠ *None*
  ⟨*proof*⟩

lemma *get-all-ann-decomposition-ex*:
  ∃ *N*. (*Decided K # M′*, *N*) ∈ *set* (*get-all-ann-decomposition* (*M2*@*Decided K # M′*))
  ⟨*proof*⟩

lemma *bt-cut-in-get-all-ann-decomposition*:
  assumes *no-dup M* **and** *bt-cut i M* = *Some M′*
  shows ∃ *M2*. (*M′*, *M2*) ∈ *set* (*get-all-ann-decomposition M*)
  ⟨*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*

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

3.1.3 Simple Implementation of DPLL

Combining the propagate and decide: a DPLL step

definition *DPLL-step* :: *int* $dpll_W$*-ann-lits* × *int literal list list*
  ⇒ *int* $dpll_W$*-ann-lits* × *int literal list list* **where**
DPLL-step = (λ(*Ms*, *N*).
  (*case find-first-unit-clause N Ms of*
    Some (*L*, -) ⇒ (*Propagated L* () # *Ms*, *N*)
  | - ⇒
    if ∃ *C* ∈ *set N*. (∀ *c* ∈ *set C*. −*c* ∈ *lits-of-l Ms*)
    then
      (*case backtrack-split Ms of*
        (-, *L # M*) ⇒ (*Propagated* (− (*lit-of L*)) () # *M*, *N*)
      | (-, -) ⇒ (*Ms*, *N*)
)
    else

(*case find-first-unused-var N* (*lits-of-l Ms*) *of*
Some a ⇒ (*Decided a* # *Ms*, *N*)
| *None* ⇒ (*Ms*, *N*))))

Example of propagation:

value *DPLL-step* ([*Decided* (*Neg 1*)], [[*Pos* (*1*::*int*), *Neg 2*]])

We define the conversion function between the states as defined in *Prop-DPLL* (with multisets) and here (with lists).

abbreviation *toS* ≡ λ(*Ms*::(*int*, *unit*) *ann-lits*)
(*N*:: *int literal list list*). (*Ms*, *mset* (*map mset N*))
abbreviation *toS'* ≡ λ(*Ms*::(*int*, *unit*) *ann-lits*,
N:: *int literal list list*). (*Ms*, *mset* (*map mset N*))

Proof of correctness of *DPLL-step*

lemma *DPLL-step-is-a-dpll*$_W$*-step*:
assumes *step*: (*Ms'*, *N'*) = *DPLL-step* (*Ms*, *N*)
and *neq*: (*Ms*, *N*) ≠ (*Ms'*, *N'*)
shows *dpll*$_W$ (*toS Ms N*) (*toS Ms' N'*)
⟨*proof*⟩

lemma *DPLL-step-stuck-final-state*:
assumes *step*: (*Ms*, *N*) = *DPLL-step* (*Ms*, *N*)
shows *conclusive-dpll*$_W$*-state* (*toS Ms N*)
⟨*proof*⟩

Adding invariants

Invariant tested in the function **function** *DPLL-ci* :: *int dpll*$_W$*-ann-lits* ⇒ *int literal list list* ⇒ *int dpll*$_W$*-ann-lits* × *int literal list list* **where**
DPLL-ci Ms N =
(*if* ¬*dpll*$_W$*-all-inv* (*Ms*, *mset* (*map mset N*))
then (*Ms*, *N*)
else
let (*Ms'*, *N'*) = *DPLL-step* (*Ms*, *N*) *in*
if (*Ms'*, *N'*) = (*Ms*, *N*) *then* (*Ms*, *N*) *else DPLL-ci Ms' N*)
⟨*proof*⟩
termination
⟨*proof*⟩

No invariant tested **function** (*domintros*) *DPLL-part*:: *int dpll*$_W$*-ann-lits* ⇒ *int literal list list* ⇒ *int dpll*$_W$*-ann-lits* × *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*) *ann-lits* **and**
N :: *int literal list list*
assumes *inv*: *dpll*$_W$*-all-inv* (*Ms*, *mset* (*map mset N*))
and *MsN*: *DPLL-part Ms N* = (*Ms*′, *N*)
shows *conclusive-dpll*$_W$*-state* (*toS Ms*′ *N*) ∧ $dpll_W{}^{**}$ (*toS Ms N*) (*toS Ms*′ *N*)
⟨*proof*⟩

Embedding the invariant into the type

Defining the type **typedef** *dpll*$_W$*-state* =
{(*M*::(*int*, *unit*) *ann-lits*, *N*::*int literal list list*).
dpll$_W$*-all-inv* (*toS M N*)}
morphisms *rough-state-of state-of*
⟨*proof*⟩

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*⟩

Code export

A conversion to *DPLL-W-Implementation.dpll$_W$-state* **definition** *Con* :: (*int*, *unit*) *ann-lits* × *int literal list list*
  ⇒ *dpll$_W$-state* **where**
  Con xs = *state-of* (*if dpll$_W$-all-inv* (*toS* (*fst xs*) (*snd xs*)) *then xs else* ([], []))
lemma [*code abstype*]:
  Con (*rough-state-of S*) = *S*
  ⟨*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-l* (M)), M))

One version of the generated SML code is here, but not included in the generated document. The only differences are:

- export *'a literal* from the SML Module *Clausal-Logic*;
- export the constructor *Con* from *DPLL-W-Implementation*;
- export the *int* constructor from *Arith*.

  All these allows to test on the code on some examples.

end