Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

April 20, 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 $('v, 'm)\ ann\text{-}lits = ('v, 'm)\ ann\text{-}lit\ list$

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

abbreviation $lits\text{-}of\text{-}l :: ('a, 'b)\ ann\text{-}lits \Rightarrow 'a\ literal\ set$ **where**
$lits\text{-}of\text{-}l\ Ls \equiv lits\text{-}of\ (set\ Ls)$

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

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

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

Entailment

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

definition $true\text{-}annots :: ('a, 'm)\ ann\text{-}lits \Rightarrow 'a\ clauses \Rightarrow bool$ (**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*:
$MLs \models a\ \psi \Longrightarrow set\ (map\ unmark\ MLs) \models p\ \psi$

$\langle proof \rangle$

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

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

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

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

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

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

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

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

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

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

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

lemma *atm-imp-decided-or-proped*:
assumes $x \in set\ I$
shows
$(Decided\ (-\ lit\text{-}of\ x) \in set\ I)$
$\lor (Decided\ (lit\text{-}of\ x) \in set\ I)$

$\vee$ ($\exists$ *l*. *Propagated* ($-$ *lit-of* *x*) *l* $\in$ *set* *I*)
$\vee$ ($\exists$ *l*. *Propagated* (*lit-of* *x*) *l* $\in$ *set* *I*)
⟨*proof*⟩

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

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

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

lemma *defined-lit-map*:
defined-lit *Ls* *L* $\longleftrightarrow$ *atm-of* *L* $\in$ (λ*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'', []) $\Longrightarrow$ ∀ *l*∈*set M*. ¬ *is-decided l*
⟨*proof*⟩

lemma *backtrack-split-some-is-decided-then-snd-has-hd*:
∃ *l*∈*set M*. *is-decided l* $\Longrightarrow$ ∃ 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*
⇒ (('*a*, '*m*) *ann-lits* × ('*a*, '*m*) *ann-lits*) *list* **where**
get-all-ann-decomposition (*Decided L* # *Ls*) =
(*Decided L* # *Ls*, []) # *get-all-ann-decomposition Ls* |
get-all-ann-decomposition (*Propagated L P*# *Ls*) =
(*apsnd* ((*op* #) (*Propagated L P*)) (*hd* (*get-all-ann-decomposition Ls*)))
tl (*get-all-ann-decomposition Ls*) |
get-all-ann-decomposition [] = [([], [])]

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

Now we can prove several simple properties about the function.

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

lemma *get-all-ann-decomposition-never-empty-sym*[*iff*]:
[] = *get-all-ann-decomposition M* ⟷ *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') ⟷ *hd* (*get-all-ann-decomposition S*) = (M', *M*)
⟨*proof*⟩

lemma *get-all-ann-decomposition-Nil-backtrack-split-snd-Nil*:
$get\text{-}all\text{-}ann\text{-}decomposition\ S = [([],\ A)] \Longrightarrow snd\ (backtrack\text{-}split\ S) = []$
⟨*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 $get\text{-}all\text{-}ann\text{-}decomposition\ M = (a,\ b)\ \#\ []$
shows $a = [] \vee (length\ a = 1 \wedge is\text{-}decided\ (hd\ a) \wedge hd\ a \in set\ M)$
⟨*proof*⟩

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

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

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

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

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

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

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

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

lemma *get-all-ann-decomposition-last-choice*:
assumes $tl\ (get\text{-}all\text{-}ann\text{-}decomposition\ (M'\ @\ Decided\ L\ \#\ M)) \neq []$

and $\forall\, l \in$ *set* M'. $\neg$*is-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 = $M0'$ @ $M0$ $\wedge$ *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)) $\cup$ $\{L \mid L.$ *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* ∪ *N* ⊨*ps unmark-l seen*
  ⟨*proof*⟩

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

lemma *all-decomposition-implies-cons-single*[*iff*]:
  all-decomposition-implies N (*l* # *S'*) $\longleftrightarrow$
    (*unmark-l* (*fst l*) ∪ *N* ⊨*ps unmark-l* (*snd l*) ∧
      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* ∪ {*unmark L* |*L*. *is-decided L* ∧ *L* ∈ *set M*}
    ⊨*ps unmark* ‘ ⋃(*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* ∪ {*unmark L* |*L*. *is-decided L* ∧ *L* ∈ *set M*} ⊨*ps unmark-l M*
    (**is** *?I* ⊨*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* ⇒ *'v clauses* **where**
CNot ψ = { {#−*L*#} | *L*. *L* ∈# ψ }

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

lemma
  shows
    CNot-singleton[*simp*]: *CNot* {#*L*#} = {{#−*L*#}} **and**
    CNot-empty[*simp*]: *CNot* {#} = {} **and**

$CNot\text{-}plus[simp]$: $CNot\ (A + B) = CNot\ A \cup CNot\ B$
⟨*proof*⟩

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

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

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

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

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

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

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

lemma $atms\text{-}of\text{-}ms\text{-}CNot\text{-}atms\text{-}of[simp]$:
$atms\text{-}of\text{-}ms\ (CNot\ C) = atms\text{-}of\ C$
⟨*proof*⟩

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

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

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

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

$\langle proof \rangle$

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

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

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

lemma *consistent-CNot-not-tautology*:
$consistent\text{-}interp\ M \Longrightarrow M \models s\ CNot\ D \Longrightarrow \neg tautology\ D$
$\langle proof \rangle$

lemma *atms-of-ms-CNot-atms-of-ms*: $atms\text{-}of\text{-}ms\ (CNot\ CC) = atms\text{-}of\text{-}ms\ \{CC\}$
$\langle proof \rangle$

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

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*: $-(a::'a\ literal) = i \longleftrightarrow a = -i$
$\langle proof \rangle$

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

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 $M \models as\ CNot\ A$
$\langle proof \rangle$

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

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 $M' \models a\ D$
$\langle proof \rangle$

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 $M' \models a\ D$
$\langle proof \rangle$

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*]:
$CNot\ (A \mathbin{\#\cup} B) = CNot\ A \cup CNot\ B$
⟨*proof*⟩

0.1.5 Other

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

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

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

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

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

lemma *true-annots-lit-of-notin-skip*:
assumes $L \# M \models as\ CNot\ A$
and $-lit\text{-}of\ L \notin\# A$
and $no\text{-}dup\ (L \# M)$
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**
$I \models asm\ C \equiv I \models as\ (set\text{-}mset\ C)$

abbreviation *true-clss-clss-m*:: $'v\ clause\ multiset \Rightarrow\ 'v\ clause\ multiset \Rightarrow bool$ (**infix** $\models psm$ *50*)

where
$I \models psm\ C \equiv set\text{-}mset\ I \models ps\ (set\text{-}mset\ C)$

Analog of theorem *true-clss-clss-subsetE*

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* $\Rightarrow$ $'a$ *clause* $\Rightarrow$ *bool* (**infix** $\models pm$ *50*) **where**
$I \models pm\ C \equiv set\text{-}mset\ I \models p\ C$

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

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

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

Other definition using *Union-mset*

lemma $atms\text{-}of\text{-}mm\ U \equiv set\text{-}mset\ (\bigcup\#\ image\text{-}mset\ (image\text{-}mset\ atm\text{-}of)\ U)$
⟨*proof*⟩

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

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

type-synonym $'v$ *clauses* = $'v$ *clause multiset*
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* $\Rightarrow$ *nat* $\Rightarrow$ *nat list* $\Rightarrow$ *nat* **where**
$\mu_C\ s\ b\ M \equiv (\sum i{=}0..{<}length\ M.\ M!i * b\hat{}\ (s + i - length\ M))$

lemma μ_C-*Nil*[*simp*]:
$\mu_C\ s\ b\ [] = 0$
⟨*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* $\{(ys, xs).\ length\ xs < s \wedge length\ ys < s \wedge$

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

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

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 *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*:
$\textit{no-dup } M \Longrightarrow -\ \textit{lit-of xa} = \textit{lit-of } x \Longrightarrow x \in \textit{set } M \Longrightarrow xa \notin \textit{set } M$
⟨*proof*⟩

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

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

lemma *atms-of-ms-single-image-atm-of-lit-of*:
$\textit{atms-of-ms}\ (\textit{unmark-s } A) = \textit{atm-of}\ `\ (\textit{lit-of}\ `\ A)$
⟨*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 :: $'st \Rightarrow ('v, \textit{unit})$ *ann-lits* **and**
$\textit{clauses}_{NOT}$ :: $'st \Rightarrow\ 'v$ *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*
begin
abbreviation $state_{NOT}$ :: '*st* ⇒ ('*v*, *unit*) *ann-lit list* × '*v clauses* **where**
$state_{NOT}\ S \equiv (trail\ S,\ clauses_{NOT}\ S)$
end

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

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

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

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

function $reduce\text{-}trail\text{-}to_{NOT}$ :: '*a list* ⇒ '*st* ⇒ '*st* **where**
$reduce\text{-}trail\text{-}to_{NOT}\ F\ S =$
(*if length* (*trail S*) = *length F* ∨ *trail S* = [] *then S else* $reduce\text{-}trail\text{-}to_{NOT}$ *F* (*tl-trail S*))
⟨*proof*⟩
termination ⟨*proof*⟩

declare *reduce-trail-to*$_{NOT}$*.simps*[*simp del*]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

end

Definition of the operation

Each possible is in its own locale.

locale *propagate-ops* =
  dpll-state trail clauses$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$
  for
    trail :: *'st* ⇒ (*'v*, *unit*) *ann-lits* **and**
    clauses$_{NOT}$ :: *'st* ⇒ *'v clauses* **and**
    prepend-trail :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
    tl-trail :: *'st* ⇒*'st* **and**
    add-cls$_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
    remove-cls$_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* +
  fixes
    propagate-cond :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *bool*

begin
inductive $propagate_{NOT}$ :: $'st \Rightarrow 'st \Rightarrow bool$ **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* =
$dpll\text{-}state\ trail\ clauses_{NOT}\ prepend\text{-}trail\ tl\text{-}trail\ add\text{-}cls_{NOT}\ remove\text{-}cls_{NOT}$
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**
$remove\text{-}cls_{NOT} :: 'v\ clause \Rightarrow 'st \Rightarrow 'st$
begin
inductive $decide_{NOT}$ :: $'st \Rightarrow 'st \Rightarrow bool$ **where**
$decide_{NOT}[intro]$: $undefined\text{-}lit\ (trail\ S)\ L \Longrightarrow atm\text{-}of\ L \in atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S)$
$\Longrightarrow T \sim prepend\text{-}trail\ (Decided\ L)\ S$
$\Longrightarrow decide_{NOT}\ S\ T$

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

locale *backjumping-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
$backjump\text{-}conds :: 'v\ clause \Rightarrow 'v\ clause \Rightarrow 'v\ literal \Rightarrow 'st \Rightarrow 'st \Rightarrow bool$
begin

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

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

end

1.2.3 DPLL with backjumping

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

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

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

Definition

We define dpll with backjumping:

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

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

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

Basic properties

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

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

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

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

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) $\leq$ *1* + *length* M
⟨*proof*⟩

lemma *length-in-get-all-ann-decomposition-bounded*:
assumes *i*:$i \in$ *set* (*trail-weight* S)
shows $i \leq$ *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) $\subseteq$ *atms-of-ms* A **and**

MA: *atm-of* ' *lits-of-l* (*trail S*) $\subseteq$ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite A*
shows μ_C (*1+card* (*atms-of-ms A*)) (*2+card* (*atms-of-ms A*)) (*trail-weight T*)
$>$ μ_C (*1+card* (*atms-of-ms A*)) (*2+card* (*atms-of-ms A*)) (*trail-weight S*)
⟨*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*) $\subseteq$ *atms-of-ms A* **and**
M-A: *atm-of* ' *lits-of-l* (*trail S*) $\subseteq$ *atms-of-ms A* **and**
nd: *no-dup* (*trail S*) **and**
fin-A: *finite A*
shows (*2+card* (*atms-of-ms A*)) ^ (*1+card* (*atms-of-ms A*))
$-$ μ_C (*1+card* (*atms-of-ms A*)) (*2+card* (*atms-of-ms A*)) (*trail-weight T*)
$<$ (*2+card* (*atms-of-ms A*)) ^ (*1+card* (*atms-of-ms A*))
$-$ μ_C (*1+card* (*atms-of-ms A*)) (*2+card* (*atms-of-ms A*)) (*trail-weight S*)
⟨*proof*⟩

lemma *wf-dpll-bj*:
assumes *fin*: *finite A*
shows *wf* {(*T*, *S*). *dpll-bj S T*
$\wedge$ *atms-of-mm* (*clauses*$_{NOT}$ *S*) $\subseteq$ *atms-of-ms A* $\wedge$ *atm-of* ' *lits-of-l* (*trail S*) $\subseteq$ *atms-of-ms A*
$\wedge$ *no-dup* (*trail S*) $\wedge$ *inv S*}
(**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, $\neg$ M $\models$*as* N and there is no remaining step is incompatible.

1. The *decide* rule tells us that every variable in N has a value.
2. The assumption $\neg$ M $\models$*as* N implies that there is conflict.
3. There is at least one decision in the trail (otherwise, M would be a model of the set of clauses N).
4. Now if we build the clause with all the decision literals of the trail, we can apply the *backjump* rule.

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

theorem *dpll-backjump-final-state*:
fixes *A* :: *'v clause set* **and** *S T* :: *'st*
assumes
atms-of-mm (*clauses*$_{NOT}$ *S*) $\subseteq$ *atms-of-ms A* **and**
atm-of ' *lits-of-l* (*trail S*) $\subseteq$ *atms-of-ms A* **and**
no-dup (*trail S*) **and**
finite A **and**
inv: *inv S* **and**

n-s: *no-step dpll-bj S* **and**
decomp: *all-decomposition-implies-m* ($clauses_{NOT}$ *S*) (*get-all-ann-decomposition* (*trail S*))
shows *unsatisfiable* (*set-mset* ($clauses_{NOT}$ *S*))
∨ (*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* ⇒ (*'v*, *unit*) *ann-lits* **and**
$clauses_{NOT}$ :: *'st* ⇒ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
$add\text{-}cls_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
$remove\text{-}cls_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
inv :: *'st* ⇒ *bool* **and**
backjump-conds :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool* **and**
propagate-conds :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *bool*
+
assumes *dpll-bj-inv*: ⋀*S T*. *dpll-bj S T* ⟹ *inv S* ⟹ *inv T*
begin

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

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

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

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

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

lemma *rtranclp-dpll-bj-atms-in-trail-in-set*:

assumes
  $dpll\text{-}bj^{**}$ S T **and**
  inv S
  atms-of-mm ($clauses_{NOT}$ S) $\subseteq$ A **and**
  atm-of ‘ (*lits-of-l* (*trail S*)) $\subseteq$ A
shows *atm-of* ‘ (*lits-of-l* (*trail T*)) $\subseteq$ A
⟨*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
  ∧ *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*}
   ⊆ {(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** *?A* ⊆ $?B^+$)
⟨*proof*⟩

lemma *wf-tranclp-dpll-bj*:
assumes *fin*: *finite A*
shows *wf* {(T, S). $dpll\text{-}bj^{++}$ 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*}
⟨*proof*⟩

lemma *dpll-bj-sat-ext-iff*:
dpll-bj S T ⟹ *inv S* ⟹ *I*⊨*sextm* $clauses_{NOT}$ S ⟷ *I*⊨*sextm* $clauses_{NOT}$ T
⟨*proof*⟩

lemma *rtranclp-dpll-bj-sat-ext-iff*:
$dpll\text{-}bj^{**}$ S T ⟹ *inv S* ⟹ *I*⊨*sextm* $clauses_{NOT}$ S ⟷ *I*⊨*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) ⊆ *atms-of-ms A* **and**
  atms-trail: *atm-of* ‘ *lits-of-l* (*trail S*) ⊆ *atms-of-ms A* **and**
  n-d: *no-dup* (*trail S*) **and**
  finite A **and**
  inv: *inv S* **and**
  decomp: *all-decomposition-implies-m* ($clauses_{NOT}$ S) (*get-all-ann-decomposition* (*trail S*))
shows *unsatisfiable* (*set-mset* ($clauses_{NOT}$ S))
∨ (*trail T* ⊨*asm* $clauses_{NOT}$ S ∧ *satisfiable* (*set-mset* ($clauses_{NOT}$ S)))
⟨*proof*⟩

corollary *full-dpll-backjump-final-state-from-init-state*:
fixes *A* :: *'v clause set* **and** *S T* :: *'st*

assumes
  full: *full dpll-bj S T* **and**
  trail S = [] **and**
  $clauses_{NOT}$ *S* = *N* **and**
  inv S
shows *unsatisfiable* (*set-mset N*) ∨ (*trail T* ⊨*asm N* ∧ *satisfiable* (*set-mset N*))
⟨*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*) ⊆ *atms-of-ms A* **and**
  M-A: *atm-of* ' *lits-of-l* (*trail S*) ⊆ *atms-of-ms A* **and**
  n-d: *no-dup* (*trail S*) **and**
  fin-A: *finite A*
  shows (*2*+*card* (*atms-of-ms A*)) ^ (*1*+*card* (*atms-of-ms A*))
      − μ_C (*1*+*card* (*atms-of-ms A*)) (*2*+*card* (*atms-of-ms A*)) (*trail-weight T*)
    < (*2*+*card* (*atms-of-ms A*)) ^ (*1*+*card* (*atms-of-ms A*))
      − μ_C (*1*+*card* (*atms-of-ms A*)) (*2*+*card* (*atms-of-ms A*)) (*trail-weight S*)
⟨*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* ⇒ (*'v*, *unit*) *ann-lits* **and**
    $clauses_{NOT}$ :: *'st* ⇒ *'v clauses* **and**
    prepend-trail :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
    tl-trail :: *'st* ⇒*'st* **and**
    $add\text{-}cls_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
    $remove\text{-}cls_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* +
  fixes
    learn-cond :: *'v clause* ⇒ *'st* ⇒ *bool*
begin
inductive *learn* :: *'st* ⇒ *'st* ⇒ *bool* **where**
$learn_{NOT}$*-rule*: $clauses_{NOT}$ *S* ⊨*pm C* ⟹
  atms-of C ⊆ *atms-of-mm* ($clauses_{NOT}$ *S*) ∪ *atm-of* ' (*lits-of-l* (*trail S*)) ⟹
  learn-cond C S ⟹
  T ∼ $add\text{-}cls_{NOT}$ *C S* ⟹
  learn S T
inductive-cases $learn_{NOT}E$: *learn S T*

lemma *learn-*μ_C*-stable*:
  assumes *learn S T* **and** *no-dup* (*trail S*)
  shows μ_C *A B* (*trail-weight S*) = μ_C *A B* (*trail-weight T*)
  ⟨*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-lits* **and**
    $clauses_{NOT}$ :: $'st \Rightarrow 'v$ *clauses* **and**
    prepend-trail :: $('v, unit)$ *ann-lit* $\Rightarrow 'st \Rightarrow 'st$ **and**
    tl-trail :: $'st \Rightarrow 'st$ **and**
    $add\text{-}cls_{NOT}$ :: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$ **and**
    $remove\text{-}cls_{NOT}$ :: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$ +
  fixes
    forget-cond :: $'v$ *clause* $\Rightarrow 'st \Rightarrow bool$
begin
inductive $forget_{NOT}$ :: $'st \Rightarrow 'st \Rightarrow bool$ **where**
$forget_{NOT}$:
  removeAll-mset C($clauses_{NOT}$ S) $\models pm$ $C \Longrightarrow$
  forget-cond C $S \Longrightarrow$
  $C \in\#$ $clauses_{NOT}$ $S \Longrightarrow$
  $T \sim$ $remove\text{-}cls_{NOT}$ C $S \Longrightarrow$
  $forget_{NOT}$ S T
inductive-cases $forget_{NOT}E$: $forget_{NOT}$ S T

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

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

Definition of CDCL

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

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**
  $propagate\text{-}conds$ :: $('v,\ unit)\ ann\text{-}lit \Rightarrow\ 'st \Rightarrow bool$ **and**
  $learn\text{-}cond\ forget\text{-}cond$ :: $'v\ clause \Rightarrow\ 'st \Rightarrow bool$
begin

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

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

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

Consistency of the trail **lemma** $cdcl_{NOT}\text{-}consistent$:
  assumes
  $cdcl_{NOT}\ S\ T$ **and**
  $inv\ S$ **and**
  $no\text{-}dup\ (trail\ S)$
  shows $consistent\text{-}interp\ (lits\text{-}of\text{-}l\ (trail\ T))$
  ⟨*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 $cdcl_{NOT}\text{-}atms\text{-}of\text{-}ms\text{-}clauses\text{-}decreasing$:
  assumes $cdcl_{NOT}\ S\ T$**and** $inv\ S$ **and** $no\text{-}dup\ (trail\ S)$

shows *atms-of-mm* (*clauses*$_{NOT}$ *T*) ⊆ *atms-of-mm* (*clauses*$_{NOT}$ *S*) ∪ *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*)) ⊆ *atms-of-mm* (*clauses*$_{NOT}$ *S*)
shows *atm-of* ‘ (*lits-of-l* (*trail T*)) ⊆ *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*) ⊆ *A* **and**
atm-of ‘ (*lits-of-l* (*trail S*)) ⊆ *A*
shows *atm-of* ‘ (*lits-of-l* (*trail T*)) ⊆ *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*⊨*sextm clauses*$_{NOT}$ *S* ⟷ *I*⊨*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*: ⋀*S T*. *cdcl*$_{NOT}$ *S T* ⟹ *inv S* ⟹ *inv T*
begin
sublocale *dpll-with-backjumping*
⟨*proof*⟩

lemma *rtranclp-cdcl*$_{NOT}$*-inv*:
cdcl$_{NOT}$** *S T* ⟹ *inv S* ⟹ *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*) ⊆ *A* **and**

atms-trail-S: $atm\text{-}of$ ‘$(lits\text{-}of\text{-}l\ (trail\ S)) \subseteq A$
shows $atm\text{-}of$ ‘ $(lits\text{-}of\text{-}l\ (trail\ T)) \subseteq A \wedge atms\text{-}of\text{-}mm\ (clauses_{NOT}\ T) \subseteq A$
⟨*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$_{NOT}$-NOT-all-inv $A\ S \longleftrightarrow (finite\ A \wedge inv\ S \wedge atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \subseteq atms\text{-}of\text{-}ms\ A$
$\wedge\ atm\text{-}of$ ‘ $lits\text{-}of\text{-}l\ (trail\ S) \subseteq atms\text{-}of\text{-}ms\ A \wedge no\text{-}dup\ (trail\ S))$

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-or-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 $(2+card\ (atms\text{-}of\text{-}ms\ A))\ \hat{}\ (1+card\ (atms\text{-}of\text{-}ms\ A))$
$-\ \mu_C\ (1+card\ (atms\text{-}of\text{-}ms\ A))\ (2+card\ (atms\text{-}of\text{-}ms\ A))\ (trail\text{-}weight\ U)$
$<\ (2+card\ (atms\text{-}of\text{-}ms\ A))\ \hat{}\ (1+card\ (atms\text{-}of\text{-}ms\ A))$
$-\ \mu_C\ (1+card\ (atms\text{-}of\text{-}ms\ A))\ (2+card\ (atms\text{-}of\text{-}ms\ A))\ (trail\text{-}weight\ S)$
(**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 $\exists j.\ \forall i \ge j.\ learn\text{-}or\text{-}forget\ (f\ i)\ (f\ (Suc\ i))$
⟨*proof*⟩

lemma *wf-cdcl$_{NOT}$-no-learn-and-forget-infinite-chain*:
assumes
no-infinite-lf: $\bigwedge f\ j.\ \neg\ (\forall i \ge j.\ learn\text{-}or\text{-}forget\ (f\ i)\ (f\ (Suc\ i)))$
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
$\lambda C\ S.\ distinct\text{-}mset\ C \wedge \neg tautology\ C \wedge learn\text{-}restrictions\ C\ S\ \wedge$
$(\exists\ F\ K\ d\ F'\ C'\ L.\ trail\ S = F'\ @\ Decided\ K\ \#\ F \wedge C = C' + \{\#L\#\} \wedge F \models as\ CNot\ C'$
$\wedge\ C' + \{\#L\#\} \notin\#\ clauses_{NOT}\ S)$
$\lambda C\ S.\ \neg(\exists\ F'\ F\ K\ d\ L.\ trail\ S = F'\ @\ Decided\ K\ \#\ F \wedge F \models as\ CNot\ (remove1\text{-}mset\ L\ C))$
$\wedge\ forget\text{-}restrictions\ C\ S$

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*
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*]:

$conflicting\text{-}bj\text{-}clss\ (remove\text{-}cls_{NOT}\ C\ S) = conflicting\text{-}bj\text{-}clss\ S - \{C\}$
⟨proof⟩

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

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

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

lemma *conflicting-bj-clss-incl-clauses*:
$conflicting\text{-}bj\text{-}clss\ S \subseteq set\text{-}mset\ (clauses_{NOT}\ S)$
⟨proof⟩

lemma *finite-conflicting-bj-clss*[*simp*]:
$finite\ (conflicting\text{-}bj\text{-}clss\ S)$
⟨proof⟩

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

abbreviation $conflicting\text{-}bj\text{-}clss\text{-}yet\ b\ S \equiv$
$3\ \widehat{}\ b - card\ (conflicting\text{-}bj\text{-}clss\ S)$

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

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

lemma *do-not-forget-before-backtrack-rule-clause-learned-clause-untouched*:
assumes $forget_{NOT}\ S\ T$
shows $conflicting\text{-}bj\text{-}clss\ S = conflicting\text{-}bj\text{-}clss\ T$
⟨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*:
$\{a.\ (a = b \lor Q\ a) \land S\ a\} = (\mathit{if}\ S\ b\ \mathit{then}\ \{b\}\ \mathit{else}\ \{\}) \cup \{a.\ Q\ a \land S\ a\}$
⟨*proof*⟩

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

lemma *learn-μ_L-decrease*:
assumes *learnST*: *learn S T* **and** *n-d*: *no-dup* (*trail S*) **and**
A: *atms-of-mm* ($\mathit{clauses}_{NOT}$ *S*) ∪ *atm-of* ' *lits-of-l* (*trail S*) ⊆ *A* **and**
fin-A: *finite A*
shows $(\mu_L\ (\mathit{card}\ A)\ T,\ \mu_L\ (\mathit{card}\ A)\ S) \in$ *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* ($\mathit{clauses}_{NOT}$ *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**
$\mu_{CDCL}\ A\ T \equiv ((2+\mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A))\ \widehat{}\ (1+\mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A))$
$\quad - \mu_C\ (1+\mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A))\ (2+\mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A))\ (\mathit{trail\text{-}weight}\ T),$
$\quad \mathit{conflicting\text{-}bj\text{-}clss\text{-}yet}\ (\mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A))\ T,\ \mathit{card}\ (\mathit{set\text{-}mset}\ (\mathit{clauses}_{NOT}\ T)))$
lemma *cdcl_{NOT}-decreasing-measure*:
assumes
cdcl_{NOT} *S T* **and**
inv: *inv S* **and**
atm-clss: *atms-of-mm* ($\mathit{clauses}_{NOT}$ *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 $(\mu_{CDCL}\ A\ T,\ \mu_{CDCL}\ A\ S)$
∈ *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* ($\mathit{clauses}_{NOT}$ *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**
$\mu_C{}'\ A\ T \equiv \mu_C\ (1+\mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A))\ (2+\mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A))\ (\mathit{trail\text{-}weight}\ T)$

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

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

lemma $cdcl_{NOT}$*-clauses-bound*:
assumes
$cdcl_{NOT}$ *S T* **and**
inv S **and**
atms-of-mm ($clauses_{NOT}$ *S*) ⊆ *A* **and**
atm-of ‘(*lits-of-l* (*trail S*)) ⊆ *A* **and**
n-d: *no-dup* (*trail S*) **and**
fin-A[*simp*]: *finite A*
shows *set-mset* ($clauses_{NOT}$ *T*) ⊆ *set-mset* ($clauses_{NOT}$ *S*) ∪ *simple-clss A*
⟨*proof*⟩

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

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

lemma *rtranclp-*$cdcl_{NOT}$*-card-clauses-bound'*:
assumes
$cdcl_{NOT}{}^{**}$ *S T* **and**
inv S **and**

atms-of-mm (*clauses*$_{NOT}$ *S*) ⊆ *A* **and**
atm-of ‘(*lits-of-l* (*trail S*)) ⊆ *A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite A*
shows *card* {*C*|*C*. *C* ∈# *clauses*$_{NOT}$ *T* ∧ (*tautology C* ∨ ¬*distinct-mset C*)}
≤ *card* {*C*|*C*. *C*∈# *clauses*$_{NOT}$ *S* ∧ (*tautology C* ∨ ¬*distinct-mset C*)} + *3* ^ (*card A*)
(**is** *card ?T* ≤ *card ?S* + -)
⟨*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*) ⊆ *A* **and**
MA: *atm-of* ‘ (*lits-of-l* (*trail S*)) ⊆ *A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite A*
shows *card* (*set-mset* (*clauses*$_{NOT}$ *T*))
≤ *card* {*C*. *C* ∈# *clauses*$_{NOT}$ *S* ∧ (*tautology C* ∨ ¬*distinct-mset C*)} + *3* ^ (*card A*)
(**is** *card ?T* ≤ *card ?S* + -)
⟨*proof*⟩

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

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

lemma *rtranclp-cdcl*$_{NOT}$*-*μ_{CDCL}*'-bound-reduce-trail-to*$_{NOT}$:
assumes
cdcl$_{NOT}$** *S T* **and**
inv S **and**
atms-of-mm (*clauses*$_{NOT}$ *S*) ⊆ *atms-of-ms A* **and**
atm-of ‘(*lits-of-l* (*trail S*)) ⊆ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite* (*atms-of-ms A*) **and**
U: *U* ∼ *reduce-trail-to*$_{NOT}$ *M T*
shows μ_{CDCL}*'* *A U* ≤ μ_{CDCL}*'-bound A S*
⟨*proof*⟩

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

lemma *rtranclp-*μ_{CDCL}*'-bound-decreasing*:

assumes
$cdcl_{NOT}{}^{**}$ S T **and**
inv S **and**
atms-of-mm (*clauses*$_{NOT}$ S) $\subseteq$ *atms-of-ms* A **and**
atm-of ‘(*lits-of-l* (*trail* S)) $\subseteq$ *atms-of-ms* A **and**
n-d: *no-dup* (*trail* S) **and**
finite[*simp*]: *finite* (*atms-of-ms* A)
shows $\mu_{CDCL}{}'$*-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-cls*$_{NOT}$ *remove-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-cls$_{NOT}$ :: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$ **and**
remove-cls$_{NOT}$ :: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$ **and**
inv :: $'st \Rightarrow bool$ **and**
backjump-conds :: $'v$ *clause* $\Rightarrow 'v$ *clause* $\Rightarrow 'v$ *literal* $\Rightarrow 'st \Rightarrow 'st \Rightarrow bool$ **and**
propagate-conds :: $('v,\ unit)$ *ann-lit* $\Rightarrow 'st \Rightarrow bool$ **and**
learn-cond forget-cond :: $'v$ *clause* $\Rightarrow 'st \Rightarrow bool$
begin

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

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 :: $'st \Rightarrow 'st \Rightarrow bool$ **and**
    $cdcl_{NOT}$ :: $'st \Rightarrow 'st \Rightarrow bool$ +
  fixes
    f :: $nat \Rightarrow nat$ **and**
    bound-inv :: $'bound \Rightarrow 'st \Rightarrow bool$ **and**
    μ :: $'bound \Rightarrow 'st \Rightarrow nat$ **and**
    $cdcl_{NOT}$*-inv* :: $'st \Rightarrow bool$ **and**
    μ*-bound* :: $'bound \Rightarrow 'st \Rightarrow nat$
  assumes
    f: *unbounded* f **and**
    f-ge-1: $\bigwedge n.\ n \geq 1 \Longrightarrow f\ n \neq 0$ **and**
    bound-inv: $\bigwedge A\ S\ T.$ $cdcl_{NOT}$*-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
    $(cdcl_{NOT}\widehat{\ \ }n)\ S\ T$ **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
$(cdcl_{NOT}\widehat{}\widehat{}(Suc\ n))$ *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 $\neg(cdcl_{NOT}\ \widehat{}\widehat{}\ m)$ *S T*
⟨*proof*⟩

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

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

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

lemma $cdcl_{NOT}$*-restart-*$cdcl_{NOT}$*-raw-restart*:
$cdcl_{NOT}$*-restart* S $T \Longrightarrow$ $cdcl_{NOT}$*-raw-restart*** (*fst* S) (*fst* T)
⟨*proof*⟩

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* $A\ T$
⟨*proof*⟩

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

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

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

lemma $cdcl_{NOT}$-*restart-steps-bigger-than-bound*:
assumes
$cdcl_{NOT}$-*restart* $S\ T$ **and**
bound-inv A $(fst\ S)$ **and**
$cdcl_{NOT}$-*inv* $(fst\ S)$ **and**
$f\ (snd\ S) > \mu$-*bound* $A\ (fst\ S)$
shows *full1* $cdcl_{NOT}$ $(fst\ S)\ (fst\ T)$
⟨*proof*⟩

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

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

end

1.2.6 Merging backjump and learning

locale $cdcl_{NOT}$*-merge-bj-learn-ops* =
decide-ops trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ +
forget-ops trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ *forget-cond* +
propagate-ops trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ *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**
$add\text{-}cls_{NOT}$ :: $'v$ *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* $\Rightarrow 'st \Rightarrow bool$ +
fixes *backjump-l-cond* :: $'v$ *clause* $\Rightarrow 'v$ *clause* $\Rightarrow 'v$ *literal* $\Rightarrow 'st \Rightarrow 'st \Rightarrow bool$
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
$\Longrightarrow$ *no-dup* (*trail* S)
$\Longrightarrow$ $T \sim$ *prepend-trail* (*Propagated* L ()) ($reduce\text{-}trail\text{-}to_{NOT}$ F ($add\text{-}cls_{NOT}$ C'' 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))
$\Longrightarrow$ $clauses_{NOT}$ $S \models pm$ $C' + \{\#L\#\}$
$\Longrightarrow$ $C'' = C' + \{\#L\#\}$
$\Longrightarrow$ $F \models as$ *CNot* C'
$\Longrightarrow$ *backjump-l-cond* C C' L S T
$\Longrightarrow$ *backjump-l* S T

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

declare $reduce\text{-}trail\text{-}to_{NOT}$*-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 $reduce\text{-}trail\text{-}to_{NOT}$*-length-ne*[*simp*] *Set.Un-iff*[*simp*] *Set.insert-iff*[*simp*]

inductive $cdcl_{NOT}$*-merged-bj-learn* :: $'st \Rightarrow 'st \Rightarrow bool$ **for** S :: $'st$ **where**
$cdcl_{NOT}$*-merged-bj-learn-decide*$_{NOT}$: $decide_{NOT}$ S S' $\Longrightarrow$ $cdcl_{NOT}$*-merged-bj-learn* S S' |
$cdcl_{NOT}$*-merged-bj-learn-propagate*$_{NOT}$: $propagate_{NOT}$ S S' $\Longrightarrow$ $cdcl_{NOT}$*-merged-bj-learn* S S' |
$cdcl_{NOT}$*-merged-bj-learn-backjump-l*: *backjump-l* S S' $\Longrightarrow$ $cdcl_{NOT}$*-merged-bj-learn* S S' |
$cdcl_{NOT}$*-merged-bj-learn-forget*$_{NOT}$: $forget_{NOT}$ S S' $\Longrightarrow$ $cdcl_{NOT}$*-merged-bj-learn* S S'

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

locale $cdcl_{NOT}$*-merge-bj-learn-proxy* =
$cdcl_{NOT}$*-merge-bj-learn-ops trail* $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
propagate-conds forget-cond
$\lambda C\ C'\ L'\ S\ T.$ *backjump-l-cond* C C' L' S T
$\wedge$ *distinct-mset* $(C' + \{\#L'\#\}) \wedge \neg$*tautology* $(C' + \{\#L'\#\})$

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

abbreviation *backjump-conds* :: $'v$ *clause* $\Rightarrow\ 'v$ *clause* $\Rightarrow\ 'v$ *literal* $\Rightarrow\ 'st \Rightarrow\ 'st \Rightarrow$ *bool*
  where
backjump-conds $\equiv \lambda C\ C'\ L'\ S\ T.$ *distinct-mset* $(C' + \{\#L'\#\}) \land \neg$*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-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-l-cond :: $'v$ *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**
forget-cond :: $'v$ *clause* $\Rightarrow 'st \Rightarrow$ *bool* **and**
inv :: $'st \Rightarrow$ *bool* +
assumes
dpll-merge-bj-inv: $\bigwedge S\ T.$ *dpll-bj* $S\ T \Longrightarrow$ *inv* $S \Longrightarrow$ *inv* T **and**
learn-inv: $\bigwedge S\ T.$ *learn* $S\ T \Longrightarrow$ *inv* $S \Longrightarrow$ *inv* T
begin

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

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

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

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

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

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

definition $\mu_C{}'$ :: $'v$ *clause set* $\Rightarrow 'st \Rightarrow$ *nat* **where**

$\mu_C{}'$ A $T \equiv \mu_C$ (*1+card* (*atms-of-ms* A)) (*2+card* (*atms-of-ms* A)) (*trail-weight* T)

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

lemma *cdcl*$_{NOT}$*-decreasing-measure'*:
assumes
cdcl$_{NOT}$*-merged-bj-learn* S T **and**
inv: *inv* S **and**
atm-clss: *atms-of-mm* (*clauses*$_{NOT}$ S) $\subseteq$ *atms-of-ms* A **and**
atm-trail: *atm-of* ‘ *lits-of-l* (*trail* S) $\subseteq$ *atms-of-ms* A **and**
n-d: *no-dup* (*trail* S) **and**
fin-A: *finite* A
shows $\mu_{CDCL}{}'$*-merged* A T < $\mu_{CDCL}{}'$*-merged* A S
⟨*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) $\subseteq$ *atms-of-ms* A ∧ *atm-of* ‘ *lits-of-l* (*trail* S) $\subseteq$ *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) $\subseteq$ *atms-of-ms* A **and**
atm-trail: *atm-of* ‘ *lits-of-l* (*trail* S) $\subseteq$ *atms-of-ms* A **and**
n-d: *no-dup* (*trail* S) **and**
fin-A[*simp*]: *finite* A
shows (T, S) ∈ {(T, S).
(*inv* S ∧ *atms-of-mm* (*clauses*$_{NOT}$ S) $\subseteq$ *atms-of-ms* A ∧ *atm-of* ‘ *lits-of-l* (*trail* S) $\subseteq$ *atms-of-ms* A
∧ *no-dup* (*trail* S))
∧ *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) $\subseteq$ *atms-of-ms* A ∧ *atm-of* ‘ *lits-of-l* (*trail* S) $\subseteq$ *atms-of-ms* A
∧ *no-dup* (*trail* S))
∧ *cdcl*$_{NOT}$*-merged-bj-learn*$^{++}$ S T}
⟨*proof*⟩

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

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

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* ⇒ (*'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*
+
fixes *f* :: *nat* ⇒ *nat*
assumes
unbounded: *unbounded f* **and** *f-ge-1*: ⋀*n*. *n* ≥ *1* ⟹ *f n* ≥ *1* **and**
inv-restart:⋀*S T*. *inv S* ⟹ *T* ∼ *reduce-trail-to*$_{NOT}$ ([]::*'a list*) *S* ⟹ *inv T*
begin

lemma *bound-inv-inv*:
assumes

inv S **and**
n-d: *no-dup* (*trail S*) **and**
atms-clss-S-A: *atms-of-mm* ($clauses_{NOT}$ *S*) ⊆ *atms-of-ms A* **and**
atms-trail-S-A:*atm-of* ‘ *lits-of-l* (*trail S*) ⊆ *atms-of-ms A* **and**
finite A **and**
$cdcl_{NOT}$: $cdcl_{NOT}$ *S T*
shows
atms-of-mm ($clauses_{NOT}$ *T*) ⊆ *atms-of-ms A* **and**
atm-of ‘ *lits-of-l* (*trail T*) ⊆ *atms-of-ms A* **and**
finite A
⟨*proof*⟩

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

lemma $cdcl_{NOT}$*-with-restart-*μ_{CDCL}'*-le-*μ_{CDCL}'*-bound*:
assumes
$cdcl_{NOT}$: $cdcl_{NOT}$*-restart* (*T*, *a*) (*V*, *b*) **and**
$cdcl_{NOT}$*-inv*:
inv T
no-dup (*trail T*) **and**
bound-inv:
atms-of-mm ($clauses_{NOT}$ *T*) ⊆ *atms-of-ms A*
atm-of ‘ *lits-of-l* (*trail T*) ⊆ *atms-of-ms A*
finite A
shows μ_{CDCL}' *A V* ≤ μ_{CDCL}'*-bound A T*
⟨*proof*⟩

lemma $cdcl_{NOT}$*-with-restart-*μ_{CDCL}'*-bound-le-*μ_{CDCL}'*-bound*:
assumes
$cdcl_{NOT}$: $cdcl_{NOT}$*-restart* (*T*, *a*) (*V*, *b*) **and**
$cdcl_{NOT}$*-inv*:
inv T
no-dup (*trail T*) **and**
bound-inv:
atms-of-mm ($clauses_{NOT}$ *T*) ⊆ *atms-of-ms A*
atm-of ‘ *lits-of-l* (*trail T*) ⊆ *atms-of-ms A*
finite A
shows μ_{CDCL}'*-bound A V* ≤ μ_{CDCL}'*-bound A T*
⟨*proof*⟩

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

lemma $cdcl_{NOT}$*-restart-all-decomposition-implies*:

assumes $cdcl_{NOT}$*-restart* S T **and**
inv (*fst* S) **and**
no-dup (*trail* (*fst* S))
all-decomposition-implies-m ($clauses_{NOT}$ (*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*:
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 $\models$*sextm* $clauses_{NOT}$ (*fst* S) $\longleftrightarrow$ I $\models$*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 $\models$*sextm* $clauses_{NOT}$ (*fst* S) $\longleftrightarrow$ I $\models$*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) $\models$*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* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
$\lambda C\ C'\ L'\ S\ T.$ *distinct-mset* (C' + {#L'#}) ∧ *backjump-l-cond* C C' L' S T
propagate-conds *forget-conds* *inv*

for
trail :: $'st \Rightarrow ('v, unit)$ *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**
inv :: $'st \Rightarrow bool$ **and**
forget-conds :: $'v$ *clause* $\Rightarrow 'st \Rightarrow bool$ **and**
backjump-l-cond :: $'v$ *clause* $\Rightarrow 'v$ *clause* $\Rightarrow 'v$ *literal* $\Rightarrow 'st \Rightarrow 'st \Rightarrow bool$
+
fixes f :: $nat \Rightarrow nat$
assumes
unbounded: *unbounded* f **and** *f-ge-1*: $\bigwedge n.\ n \geq 1 \Longrightarrow f\ n \geq 1$ **and**
inv-restart: $\bigwedge S\ T.\ inv\ S \Longrightarrow T \sim reduce\text{-}trail\text{-}to_{NOT}\ []\ S \Longrightarrow inv\ T$
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 $atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \subseteq atms\text{-}of\text{-}ms\ A$ **and**
$x \in\# clauses_{NOT}\ S$ **and** *finite* A
shows $x \in simple\text{-}clss\ (atms\text{-}of\text{-}ms\ A) \vee x \in\# not\text{-}simplified\text{-}cls\ (clauses_{NOT}\ S)$
⟨*proof*⟩

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

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

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

lemma *rtranclp-*$cdcl_{NOT}$*-merged-bj-learn-clauses-bound*:
assumes
$cdcl_{NOT}$*-merged-bj-learn*** $S\ T$ **and**
inv S **and**
$atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \subseteq atms\text{-}of\text{-}ms\ A$ **and**
$atm\text{-}of\ `(lits\text{-}of\text{-}l\ (trail\ S)) \subseteq atms\text{-}of\text{-}ms\ A$ **and**
n-d: *no-dup* (*trail* S) **and**
finite[*simp*]: *finite* A

shows *set-mset* ($clauses_{NOT}$ T) ⊆ *set-mset* (*not-simplified-cls* ($clauses_{NOT}$ S))
∪ *simple-clss* (*atms-of-ms* A)
⟨*proof*⟩

abbreviation μ_{CDCL}'*-bound* **where**
μ_{CDCL}'*-bound* A T ≡ ((2+*card* (*atms-of-ms* A)) ^ (1+*card* (*atms-of-ms* A))) ∗ 2
+ *card* (*set-mset* (*not-simplified-cls*($clauses_{NOT}$ T)))
+ 3 ^ *card* (*atms-of-ms* A)

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

sublocale $cdcl_{NOT}$*-increasing-restarts-ops* λS T. T ∼ *reduce-trail-to*$_{NOT}$ ([]::'*a list*) S
$cdcl_{NOT}$*-merged-bj-learn* f
λA S. *atms-of-mm* ($clauses_{NOT}$ S) ⊆ *atms-of-ms* A
∧ *atm-of* ' *lits-of-l* (*trail* S) ⊆ *atms-of-ms* A ∧ *finite* A
μ_{CDCL}'*-merged*
λS. *inv* S ∧ *no-dup* (*trail* S)
μ_{CDCL}'*-bound*
⟨*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)) ⊆ *atms-of-ms* A **and**
atm-of ' *lits-of-l* (*trail* (*fst* T)) ⊆ *atms-of-ms* A **and**
finite A
shows μ_{CDCL}'*-merged* A (*fst* V) ≤ μ_{CDCL}'*-bound* A (*fst* T)
⟨*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) ≤ μ_{CDCL}'*-bound* A (*fst* T)
⟨*proof*⟩

sublocale $cdcl_{NOT}$*-increasing-restarts* - - - - - - f
λS T. T ∼ *reduce-trail-to*$_{NOT}$ ([]::'*a list*) S
λA S. *atms-of-mm* ($clauses_{NOT}$ S) ⊆ *atms-of-ms* A
∧ *atm-of* ' *lits-of-l* (*trail* S) ⊆ *atms-of-ms* A ∧ *finite* A
μ_{CDCL}'*-merged* $cdcl_{NOT}$*-merged-bj-learn*
λS. *inv* S ∧ *no-dup* (*trail* S)

$\lambda A\ T.\ ((2+card\ (atms\text{-}of\text{-}ms\ A))\ \widehat{}\ (1+card\ (atms\text{-}of\text{-}ms\ A)))*2$
$+\ card\ (set\text{-}mset\ (not\text{-}simplified\text{-}cls(clauses_{NOT}\ T)))$
$+\ 3\ \widehat{}\ card\ (atms\text{-}of\text{-}ms\ A)$
⟨*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)
∨ *lits-of-l* (*trail* (*fst* T)) ⊨*sextm* N ∧ *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-lits* × '*v clauses*
⇒ ('*v*, *unit*) *ann-lits* × '*v clauses* ⇒ *bool* **where**
backtrack-split (*fst* S) = (M', L # M) ⟹ *is-decided* L ⟹ D ∈# *snd* S
⟹ *fst* S ⊨*as* *CNot* D ⟹ *backtrack* S (*Propagated* (− (*lit-of* L)) () # M, *snd* S)

inductive-cases *backtrackE*[*elim*]: *backtrack* (M, N) (M', N')
lemma *backtrack-is-backjump*:
fixes M M' :: ('*v*, *unit*) *ann-lits*
assumes
backtrack: *backtrack* (M, N) (M', N') **and**
no-dup: (*no-dup* ∘ *fst*) (M, N) **and**
decomp: *all-decomposition-implies-m* N (*get-all-ann-decomposition* M)
shows
∃ C F' K F L l C'.
M = F' @ *Decided* K # F ∧
M' = *Propagated* L l # F ∧ N = N' ∧ C ∈# N ∧ F' @ *Decided* K # F ⊨*as* *CNot* C ∧
undefined-lit F L ∧ *atm-of* L ∈ *atms-of-mm* N ∪ *atm-of* ‘ *lits-of-l* (F' @ *Decided* K # F) ∧
N ⊨*pm* C' + {#L#} ∧ F ⊨*as* *CNot* 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
∃ C F' K F L l C'.
fst S = F' @ *Decided* K # F ∧
T = (*Propagated* L l # F, *snd* S) ∧ C ∈# *snd* S ∧ *fst* S ⊨*as* *CNot* C
∧ *undefined-lit* F L ∧ *atm-of* L ∈ *atms-of-mm* (*snd* S) ∪ *atm-of* ‘ *lits-of-l* (*fst* S) ∧
snd S ⊨*pm* C' + {#L#} ∧ F ⊨*as* *CNot* C'
⟨*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 $\lambda L\ (M,\ N).\ (L\ \#\ M,\ N)\ \lambda(M,\ N).\ (tl\ M,\ N)$
$\lambda C\ (M,\ N).\ (M,\ \{\#C\#\}\ +\ N)\ \lambda C\ (M,\ N).\ (M,\ removeAll\text{-}mset\ C\ N)\ \lambda\text{- - -}\ S\ T.\ backtrack\ S\ T$
⟨*proof*⟩
thm *reduce-trail-to*$_{NOT}$*-clauses*

lemma *reduce-trail-to*$_{NOT}$:
reduce-trail-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 $\lambda L\ (M,\ N).\ (L\ \#\ M,\ N)$
$\lambda(M,\ N).\ (tl\ M,\ N)\ \lambda C\ (M,\ N).\ (M,\ \{\#C\#\}\ +\ N)\ \lambda C\ (M,\ N).\ (M,\ removeAll\text{-}mset\ C\ N)$
$\lambda(M,\ N).\ no\text{-}dup\ M \wedge all\text{-}decomposition\text{-}implies\text{-}m\ N\ (get\text{-}all\text{-}ann\text{-}decomposition\ M)$
$\lambda\text{- - -}\ S\ T.\ backtrack\ S\ T$
$\lambda\text{- -}.\ True$
⟨*proof*⟩

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

context *dpll-with-backtrack*
begin

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

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

corollary *full-dpll-normal-form-from-init-state*:
  fixes $M\ M' :: ('v,\ unit)\ ann\text{-}lits$
  assumes
    full: *full dpll-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 $cdcl_{NOT}$*-is-dpll*:
  $cdcl_{NOT}\ S\ T \longleftrightarrow dpll\text{-}bj\ S\ T$
  ⟨*proof*⟩

Another proof of termination:

lemma *wf* $\{(T,\ S).\ dpll\text{-}bj\ S\ T \wedge cdcl_{NOT}\text{-}NOT\text{-}all\text{-}inv\ A\ S\}$
  ⟨*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 $cdcl_{NOT}$*-increasing-restarts*
  $fst\ snd\ \lambda L\ (M,\ N).\ (L\ \#\ M,\ N)\ \lambda(M,\ N).\ (tl\ M,\ N)$

$\lambda C\ (M,\ N).\ (M,\ \{\#C\#\} + N)\ \lambda C\ (M,\ N).\ (M,\ \textit{removeAll-mset}\ C\ N)\ f\ \lambda(\text{-},\ N)\ S.\ S = ([],\ N)$
$\lambda A\ (M,\ N).\ \textit{atms-of-mm}\ N \subseteq \textit{atms-of-ms}\ A \wedge \textit{atm-of}\ \text{`}\ \textit{lits-of-l}\ M \subseteq \textit{atms-of-ms}\ A \wedge \textit{finite}\ A$
$\wedge\ \textit{all-decomposition-implies-m}\ N\ (\textit{get-all-ann-decomposition}\ M)$
$\lambda A\ T.\ (2+\textit{card}\ (\textit{atms-of-ms}\ A))\ \widehat{}\ (1+\textit{card}\ (\textit{atms-of-ms}\ A))$
$-\ \mu_C\ (1+\textit{card}\ (\textit{atms-of-ms}\ A))\ (2+\textit{card}\ (\textit{atms-of-ms}\ A))\ (\textit{trail-weight}\ T)\ \textit{dpll-bj}$
$\lambda(M,\ N).\ \textit{no-dup}\ M \wedge \textit{all-decomposition-implies-m}\ N\ (\textit{get-all-ann-decomposition}\ M)$
$\lambda A\ \text{-}.\ (2+\textit{card}\ (\textit{atms-of-ms}\ A))\ \widehat{}\ (1+\textit{card}\ (\textit{atms-of-ms}\ A))$
⟨*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 $'a\ \textit{dpll}_W\textit{-ann-lit} = ('a,\ \textit{unit})\ \textit{ann-lit}$
type-synonym $'a\ \textit{dpll}_W\textit{-ann-lits} = ('a,\ \textit{unit})\ \textit{ann-lits}$
type-synonym $'v\ \textit{dpll}_W\textit{-state} = 'v\ \textit{dpll}_W\textit{-ann-lits} \times 'v\ \textit{clauses}$

abbreviation $\textit{trail} :: 'v\ \textit{dpll}_W\textit{-state} \Rightarrow 'v\ \textit{dpll}_W\textit{-ann-lits}$ **where**
$\textit{trail} \equiv \textit{fst}$
abbreviation $\textit{clauses} :: 'v\ \textit{dpll}_W\textit{-state} \Rightarrow 'v\ \textit{clauses}$ **where**
$\textit{clauses} \equiv \textit{snd}$

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

1.4.2 Invariants

lemma $\textit{dpll}_W\textit{-distinct-inv}$:
assumes $\textit{dpll}_W\ S\ S'$
and $\textit{no-dup}\ (\textit{trail}\ S)$
shows $\textit{no-dup}\ (\textit{trail}\ S')$
⟨*proof*⟩

lemma $\textit{dpll}_W\textit{-consistent-interp-inv}$:
assumes $\textit{dpll}_W\ S\ S'$
and $\textit{consistent-interp}\ (\textit{lits-of-l}\ (\textit{trail}\ S))$
and $\textit{no-dup}\ (\textit{trail}\ S)$
shows $\textit{consistent-interp}\ (\textit{lits-of-l}\ (\textit{trail}\ S'))$
⟨*proof*⟩

lemma $\textit{dpll}_W\textit{-vars-in-snd-inv}$:
assumes $\textit{dpll}_W\ S\ S'$
and $\textit{atm-of}\ \text{`}\ (\textit{lits-of-l}\ (\textit{trail}\ S)) \subseteq \textit{atms-of-mm}\ (\textit{clauses}\ S)$
shows $\textit{atm-of}\ \text{`}\ (\textit{lits-of-l}\ (\textit{trail}\ S')) \subseteq \textit{atms-of-mm}\ (\textit{clauses}\ S')$

⟨*proof*⟩

lemma *atms-of-ms-lit-of-atms-of*: *atms-of-ms* $((\lambda a.\ \{\#\mathit{lit\text{-}of}\ a\#\})$ ‘ $c)$ = *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*) ⊆ *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*) ⊆ *atms-of-mm* (*clauses S*)
shows *set-mset* (*clauses S'*) ∪ {{#*lit-of L*#} |*L. is-decided L* ∧ *L* ∈ *set* (*trail S'*)}
⊨*ps* (λ*a*. {#*lit-of a*#}) ‘ ⋃(*set* ‘ *snd* ‘ *set* (*get-all-ann-decomposition* (*trail S'*)))
⟨*proof*⟩

theorem 2.8.4 page 73 of Weidenbach's book

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

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*) ⊆ *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'*) ⊆ *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* ≡
(*all-decomposition-implies-m* (*clauses S*) (*get-all-ann-decomposition* (*trail S*))
∧ *atm-of* ‘ *lits-of-l* (*trail S*) ⊆ *atms-of-mm* (*clauses S*)
∧ *consistent-interp* (*lits-of-l* (*trail S*))
∧ *no-dup* (*trail S*))

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

definition *conclusive-*$dpll_W$*-state* (S:: $'v$ $dpll_W$*-state*) $\longleftrightarrow$
(*trail* S $\models$*asm clauses* S $\vee$ (($\forall L \in$ *set* (*trail* S). $\neg$*is-decided* L)
$\wedge$ ($\exists C \in\#$ *clauses* S. *trail* S $\models$*as CNot* C)))

theorem 2.8.6 page 74 of Weidenbach's book

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

theorem 2.8.5 page 73 of Weidenbach's book

lemma $dpll_W$*-sound*:
assumes
rtranclp $dpll_W$ ([], N) (M, N) **and**
$\forall S$. $\neg dpll_W$ (M, N) S
shows M $\models$*asm* N $\longleftrightarrow$ *satisfiable* (*set-mset* N) (**is** *?A* $\longleftrightarrow$ *?B*)
⟨*proof*⟩

1.4.3 Termination

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

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

theorem 2.8.7 page 74 of Weidenbach's book

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

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

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

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

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

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

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 $M \models asm\ N \longleftrightarrow$ *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* $\Rightarrow$ *nat* **where**
count-decided $l \equiv$ *length* (*filter is-decided* l)

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

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

lemma *atm-of-notin-get-rev-level-eq-0*[*simp*]:
assumes $atm\text{-}of\ L \notin atm\text{-}of\ `\ lits\text{-}of\text{-}l\ M$
shows $get\text{-}level\ M\ L = 0$
⟨*proof*⟩

lemma *get-level-ge-0-atm-of-in*:
assumes $get\text{-}level\ M\ L > n$
shows $atm\text{-}of\ L \in atm\text{-}of\ `\ lits\text{-}of\text{-}l\ M$
⟨*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 $atm\text{-}of\ L \notin atm\text{-}of\ `\ lits\text{-}of\text{-}l\ M$
shows $get\text{-}level\ (M\ @\ M')\ L = get\text{-}level\ M'\ L$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *get-maximum-level-skip-beginning*:
assumes *DH*: $\forall x \in$ *atms-of D*. $x \notin$ *atm-of* ‘ *lits-of-l c*
shows *get-maximum-level* (*c* @ *H*) *D* = *get-maximum-level H D*
⟨*proof*⟩

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
$\forall L \in\# D$. *atm-of L* $\notin$ *atm-of* ‘ *lits-of-l M* **and**
$\forall m \in$ *set M*. $\neg$ *is-decided m*
shows *get-maximum-level* (*M* @ *aa*) *D* = *get-maximum-level aa D*
⟨*proof*⟩

lemma *get-maximum-level-union-mset*:
get-maximum-level M $(A \mathbin{\#\cup} B)$ = *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 $\geq$ *get-level M L*
⟨*proof*⟩

lemma *count-decided-ge-get-maximum-level*:
count-decided M $\geq$ *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*:
(λl. *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 *List-More CDCL-W-Level Wellfounded-More Partial-Annotated-Clausal-Logic*

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*
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*

abbreviation *state* :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* × *'v clauses* × *'v clauses*
  × *nat* × *'v clause option* **where**
state S ≡ (*trail S*, *init-clss S*, *learned-clss S*, *backtrack-lvl S*, *conflicting S*)
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 Partial-Clausal-Logic.clause*, standing for conflicting clause) and one for the initial and learned clauses (*'v Partial-Clausal-Logic.clause*, standing for clause). The representation of the clauses annotating literals in the trail is slightly different: being able to convert it to *'v Partial-Clausal-Logic.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
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 :: $'v\ clauses \Rightarrow 'st$ +
assumes
cons-trail:
$\bigwedge S'.\ state\ st = (M,\ S') \Longrightarrow$
$state\ (cons\text{-}trail\ L\ st) = (L\ \#\ M,\ S')$ **and**

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

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

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

update-backtrack-lvl:
$\bigwedge S'.\ state\ st = (M,\ N,\ U,\ k,\ S') \Longrightarrow$
$state\ (update\text{-}backtrack\text{-}lvl\ k'\ st) = (M,\ N,\ U,\ k',\ S')$ **and**

update-conflicting:
$state\ st = (M,\ N,\ U,\ k,\ D) \Longrightarrow$
$state\ (update\text{-}conflicting\ E\ st) = (M,\ N,\ U,\ k,\ E)$ **and**

init-state:
$state\ (init\text{-}state\ N) = ([],\ N,\ \{\#\},\ 0,\ None)$
begin
lemma
trail-cons-trail[*simp*]:

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

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

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

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

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

conflicting-remove-cls[*simp*]:
conflicting (*remove-cls C st*) = *conflicting st* **and**
conflicting-update-backtrack-lvl[*simp*]:
conflicting (*update-backtrack-lvl k st*) = *conflicting st* **and**
conflicting-update-conflicting[*simp*]:
conflicting (*update-conflicting E st*) = *E* **and**

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

⟨*proof*⟩

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

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

abbreviation *incr-lvl* :: *'st* ⇒ *'st* **where**
incr-lvl S ≡ *update-backtrack-lvl* (*backtrack-lvl S* + *1*) *S*

definition *state-eq* :: *'st* ⇒ *'st* ⇒ *bool* (**infix** ∼ *50*) **where**
S ∼ *T* ⟷ *state S* = *state T*

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

lemma *state-eq-sym*:
S ∼ *T* ⟷ *T* ∼ *S*
⟨*proof*⟩

lemma *state-eq-trans*:
S ∼ *T* ⟹ *T* ∼ *U* ⟹ *S* ∼ *U*
⟨*proof*⟩

lemma
shows
state-eq-trail: *S* ∼ *T* ⟹ *trail S* = *trail T* **and**
state-eq-init-clss: *S* ∼ *T* ⟹ *init-clss S* = *init-clss T* **and**
state-eq-learned-clss: *S* ∼ *T* ⟹ *learned-clss S* = *learned-clss T* **and**

$state\text{-}eq\text{-}backtrack\text{-}lvl$: $S \sim T \Longrightarrow backtrack\text{-}lvl\ S = backtrack\text{-}lvl\ T$ **and**
$state\text{-}eq\text{-}conflicting$: $S \sim T \Longrightarrow conflicting\ S = conflicting\ T$ **and**
$state\text{-}eq\text{-}clauses$: $S \sim T \Longrightarrow clauses\ S = clauses\ T$ **and**
$state\text{-}eq\text{-}undefined\text{-}lit$: $S \sim T \Longrightarrow undefined\text{-}lit\ (trail\ S)\ L = undefined\text{-}lit\ (trail\ T)\ L$
⟨*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* ∼ 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*

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

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

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

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

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

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

lemma *clauses-reduce-trail-to-Nil*:
$clauses\ (reduce\text{-}trail\text{-}to\ []\ S) = clauses\ S$
⟨*proof*⟩

lemma *reduce-trail-to-skip-beginning*:
assumes $trail\ S = F'\ @\ F$
shows $trail\ (reduce\text{-}trail\text{-}to\ F\ S) = F$
⟨*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-clss (*reduce-trail-to F S*) = *init-clss S*
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

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
  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**

$update\text{-}conflicting :: {}'v\ clause\ option \Rightarrow {}'st \Rightarrow {}'st$ **and**

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

inductive $propagate :: {}'st \Rightarrow {}'st \Rightarrow bool$ **for** $S :: {}'st$ **where**
$propagate\text{-}rule$: $conflicting\ S = None \Longrightarrow$
$E \in\# clauses\ S \Longrightarrow$
$L \in\# E \Longrightarrow$
$trail\ S \models as\ CNot\ (E - \{\#L\#\}) \Longrightarrow$
$undefined\text{-}lit\ (trail\ S)\ L \Longrightarrow$
$T \sim cons\text{-}trail\ (Propagated\ L\ E)\ S \Longrightarrow$
$propagate\ S\ T$

inductive-cases $propagateE$: $propagate\ S\ T$

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

inductive-cases $conflictE$: $conflict\ S\ T$

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

inductive-cases $backtrackE$: $backtrack\ S\ T$
thm $backtrackE$

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

inductive-cases $decideE$: $decide\ S\ T$

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

skip-rule:
$trail\ S = Propagated\ L\ C' \#\ M \Longrightarrow$
$conflicting\ S = Some\ E \Longrightarrow$
$-L \notin\#\ E \Longrightarrow$
$E \neq \{\#\} \Longrightarrow$
$T \sim tl\text{-}trail\ S \Longrightarrow$
$skip\ S\ T$

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

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

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

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

inductive *restart* :: $'st \Rightarrow 'st \Rightarrow bool$ **for** $S :: 'st$ **where**
restart: $state\ S = (M, N, U, k, None) \Longrightarrow$
$\neg M \models asm\ clauses\ S \Longrightarrow$
$U' \subseteq\#\ U \Longrightarrow$
$state\ T = ([], N, U', 0, None) \Longrightarrow$
$restart\ S\ T$

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

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

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

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

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

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

backtrack: *backtrack* S S' $\Longrightarrow$ $cdcl_W$*-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' $\Longrightarrow$ $cdcl_W$*-o* S S' |
bj: $cdcl_W$*-bj* S S' $\Longrightarrow$ $cdcl_W$*-o* S S'

inductive $cdcl_W$ :: *'st* ⇒ *'st* ⇒ *bool* **for** S :: *'st* **where**
propagate: *propagate* S S' $\Longrightarrow$ $cdcl_W$ S S' |
conflict: *conflict* S S' $\Longrightarrow$ $cdcl_W$ S S' |
other: $cdcl_W$*-o* S S' $\Longrightarrow$ $cdcl_W$ S S'|
rf: $cdcl_W$*-rf* S S' $\Longrightarrow$ $cdcl_W$ 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 ∈# *clauses* S $\Longrightarrow$
      L ∈# C $\Longrightarrow$
      trail S ⊨*as* *CNot* (*remove1-mset* L C) $\Longrightarrow$
      undefined-lit (*trail* S) L $\Longrightarrow$
      T ∼ *cons-trail* (*Propagated* L C) S $\Longrightarrow$
      P S T **and**
    conflictH: $\bigwedge D$ T. *conflicting* S = *None* $\Longrightarrow$
      D ∈# *clauses* S $\Longrightarrow$
      trail S ⊨*as* *CNot* D $\Longrightarrow$
      T ∼ *update-conflicting* (*Some* D) S $\Longrightarrow$
      P S T **and**
    forgetH: $\bigwedge C$ T. *conflicting* S = *None* $\Longrightarrow$
      C ∈# *learned-clss* S $\Longrightarrow$
      ¬(*trail* S) ⊨*asm* *clauses* S $\Longrightarrow$
      C ∉ *set* (*get-all-mark-of-propagated* (*trail* S)) $\Longrightarrow$
      C ∉# *init-clss* S $\Longrightarrow$

$T \sim$ *remove-cls* $C\ S \Longrightarrow$
$P\ S\ T$ **and**
restartH: $\bigwedge T\ U.\ \neg$*trail* $S \models$*asm clauses* $S \Longrightarrow$
conflicting $S =$ *None* $\Longrightarrow$
state $T = ([],$ *init-clss* $S,\ U,\ 0,$ *None*$) \Longrightarrow$
$U \subseteq\#$ *learned-clss* $S \Longrightarrow$
$P\ S\ T$ **and**
decideH: $\bigwedge L\ T.$ *conflicting* $S =$ *None* $\Longrightarrow$
undefined-lit (*trail* S) $L \Longrightarrow$
atm-of $L \in$ *atms-of-mm* (*init-clss* S) $\Longrightarrow$
$T \sim$ *cons-trail* (*Decided* L) (*incr-lvl* S) $\Longrightarrow$
$P\ S\ T$ **and**
skipH: $\bigwedge L\ C'\ M\ E\ T.$
trail $S =$ *Propagated* $L\ C'\ \#\ M \Longrightarrow$
conflicting $S =$ *Some* $E \Longrightarrow$
$-L \notin\#\ E \Longrightarrow E \neq \{\#\} \Longrightarrow$
$T \sim$ *tl-trail* $S \Longrightarrow$
$P\ S\ T$ **and**
resolveH: $\bigwedge L\ E\ M\ D\ T.$
trail $S =$ *Propagated* $L\ E\ \#\ M \Longrightarrow$
$L \in\#\ E \Longrightarrow$
hd-trail $S =$ *Propagated* $L\ E \Longrightarrow$
conflicting $S =$ *Some* $D \Longrightarrow$
$-L \in\#\ D \Longrightarrow$
get-maximum-level (*trail* S) ((*remove1-mset* $(-L)\ D$)) $=$ *backtrack-lvl* $S \Longrightarrow$
$T \sim$ *update-conflicting*
(*Some* (*resolve-cls* $L\ D\ E$)) (*tl-trail* S) $\Longrightarrow$
$P\ S\ T$ **and**
backtrackH: $\bigwedge L\ D\ K\ i\ M1\ M2\ T.$
conflicting $S =$ *Some* $D \Longrightarrow$
$L \in\#\ D \Longrightarrow$
(*Decided* $K\ \#\ M1,\ M2) \in$ *set* (*get-all-ann-decomposition* (*trail* S)) $\Longrightarrow$
get-level (*trail* S) $L =$ *backtrack-lvl* $S \Longrightarrow$
get-level (*trail* S) $L =$ *get-maximum-level* (*trail* S) $D \Longrightarrow$
get-maximum-level (*trail* S) (*remove1-mset* $L\ D) \equiv i \Longrightarrow$
get-level (*trail* S) $K = i+1 \Longrightarrow$
$T \sim$ *cons-trail* (*Propagated* $L\ D$)
(*reduce-trail-to* $M1$
(*add-learned-cls* D
(*update-backtrack-lvl* i
(*update-conflicting* *None* S)))) $\Longrightarrow$
$P\ S\ T$
shows $P\ S\ S'$
⟨*proof*⟩

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

$T \sim$ *tl-trail* $S \Longrightarrow$
$P\ S\ T$ **and**
resolveH: $\bigwedge L\ E\ M\ D\ T$.
trail S = *Propagated* $L\ E\ \#\ M \Longrightarrow$
$L \in\#\ E \Longrightarrow$
hd-trail S = *Propagated* $L\ E \Longrightarrow$
conflicting S = *Some* $D \Longrightarrow$
$-L \in\#\ D \Longrightarrow$
get-maximum-level (*trail* S) ((*remove1-mset* $(-L)$ D)) = *backtrack-lvl* $S \Longrightarrow$
$T \sim$ *update-conflicting*
(*Some* (*resolve-cls* $L\ D\ E$)) (*tl-trail* S) $\Longrightarrow$
$P\ S\ T$ **and**
backtrackH: $\bigwedge L\ D\ K\ i\ M1\ M2\ T$.
conflicting S = *Some* $D \Longrightarrow$
$L \in\#\ D \Longrightarrow$
(*Decided* $K\ \#\ M1$, $M2$) $\in$ *set* (*get-all-ann-decomposition* (*trail* S)) $\Longrightarrow$
get-level (*trail* S) L = *backtrack-lvl* $S \Longrightarrow$
get-level (*trail* S) L = *get-maximum-level* (*trail* S) $D \Longrightarrow$
get-maximum-level (*trail* S) (*remove1-mset* $L\ D$) $\equiv i \Longrightarrow$
get-level (*trail* S) $K = i + 1 \Longrightarrow$
$T \sim$ *cons-trail* (*Propagated* $L\ D$)
(*reduce-trail-to* $M1$
(*add-learned-cls* D
(*update-backtrack-lvl* i
(*update-conflicting None* S)))) $\Longrightarrow$
$P\ S\ T$
shows $P\ S\ T$
⟨*proof*⟩

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

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

2.1.3 Structural Invariants

Properties of the trail

We here establish that:

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

lemma *backtrack-lit-skiped*:
assumes
L: *get-level* (*trail S*) *L* = *backtrack-lvl S* **and**
M1: (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*)) **and**
no-dup: *no-dup* (*trail S*) **and**
bt-l: *backtrack-lvl S* = *length* (*filter is-decided* (*trail S*)) **and**
lev-K: *get-level* (*trail S*) *K* = *i* + *1*
shows *atm-of L* ∉ *atm-of* ‘ *lits-of-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**
$backtrack\text{-}lvl$ S = $count\text{-}decided$ ($trail$ S) **and**
$no\text{-}dup$ ($trail$ S)
shows $backtrack\text{-}lvl$ S' = $count\text{-}decided$ ($trail$ 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* ≤ *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 $\exists K\ M1\ M2.\ (Decided\ K\ \#\ M1,\ M2) \in set\ (get\text{-}all\text{-}ann\text{-}decomposition\ (trail\ S)) \wedge$
$get\text{-}level\ (trail\ S)\ K = Suc\ i$
⟨*proof*⟩

lemma *backtrack-lvl-backtrack-decrease*:
assumes *inv*: $cdcl_W$*-M-level-inv S* **and** *bt*: *backtrack S T*
shows *backtrack-lvl T* < *backtrack-lvl S*
⟨*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** $\neg$*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 $cdcl_W$*-learned-clause* (S :: *'st*) $\longleftrightarrow$
(*init-clss* S ⊨*psm learned-clss* S
∧ (∀ T. *conflicting* S = *Some* $T \longrightarrow$ *init-clss* S ⊨*pm* T)
∧ *set* (*get-all-mark-of-propagated* (*trail* S)) ⊆ *set-mset* (*clauses* S))

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

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

Item 4 page 81 of Weidenbach's book

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

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

No alien atom in the state

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

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

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

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

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 *atms-of-ms-learned-clss-restart-state-in-atms-of-ms-learned-clssI*:
  $\textit{atms-of-mm}\ (\textit{learned-clss}\ S) \subseteq \textit{atms-of-mm}\ (\textit{init-clss}\ S) \Longrightarrow$
   $x \in \textit{atms-of-mm}\ (\textit{learned-clss}\ T) \Longrightarrow$
   $\textit{learned-clss}\ T \subseteq\#\ \textit{learned-clss}\ S \Longrightarrow$
   $x \in \textit{atms-of-mm}\ (\textit{init-clss}\ S)$
  ⟨*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: $\forall\, L\ \textit{mark}.\ \textit{Propagated}\ L\ \textit{mark} \in \textit{set}\ (\textit{trail}\ S)$
     $\longrightarrow \textit{atms-of}\ \textit{mark} \subseteq \textit{atms-of-mm}\ (\textit{init-clss}\ S)$ **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

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

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

lemma $\textit{rtranclp-cdcl}_W\textit{-no-strange-atm-inv}$:
assumes $\textit{cdcl}_W^{**}\ S\ S'$ **and** *no-strange-atm S* **and** $\textit{cdcl}_W\textit{-M-level-inv}\ S$
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 $\textit{distinct-cdcl}_W\textit{-state}\ (S :: 'st)$
$\longleftrightarrow ((\forall T.\ \textit{conflicting}\ S = \textit{Some}\ T \longrightarrow \textit{distinct-mset}\ T)$
$\wedge\ \textit{distinct-mset-mset}\ (\textit{learned-clss}\ S)$
$\wedge\ \textit{distinct-mset-mset}\ (\textit{init-clss}\ S)$
$\wedge\ (\forall L\ \textit{mark}.\ (\textit{Propagated}\ L\ \textit{mark} \in \textit{set}\ (\textit{trail}\ S) \longrightarrow \textit{distinct-mset}\ \textit{mark})))$

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

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

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

lemma $\textit{distinct-cdcl}_W\textit{-state-inv}$:
assumes
$\textit{cdcl}_W\ S\ S'$ **and**
lev-inv: $\textit{cdcl}_W\textit{-M-level-inv}\ S$ **and**
$\textit{distinct-cdcl}_W\textit{-state}\ S$
shows $\textit{distinct-cdcl}_W\textit{-state}\ S'$
⟨*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* ⇒ *bool* **where**
every-mark-is-a-conflict S ≡
∀ L *mark* a b. a @ *Propagated* L *mark* # b = (*trail* S)
  ⟶ (b ⊨*as* *CNot* (*mark* − {#L#}) ∧ L ∈# *mark*)

definition *cdcl$_W$-conflicting* S ⟷
  (∀ T. *conflicting* S = *Some* T ⟶ *trail* S ⊨*as* *CNot* T)
  ∧ *every-mark-is-a-conflict* S

lemma *backtrack-atms-of-D-in-M1*:
  fixes *M1* :: (*'v*, *'v clause*) *ann-lits*
  assumes
    inv: *cdcl$_W$-M-level-inv* S **and**
    i: *get-maximum-level* (*trail* S) ((*remove1-mset* L D)) ≡ i **and**
    decomp: (*Decided* K # *M1*, *M2*)
      ∈ *set* (*get-all-ann-decomposition* (*trail* S)) **and**
    S-lvl: *backtrack-lvl* S = *get-maximum-level* (*trail* S) D **and**
    S-confl: *conflicting* S = *Some* D **and**
    lev-K: *get-level* (*trail* S) K = *Suc* i **and**
    T: T ∼ *cons-trail* (*Propagated* L D)
            (*reduce-trail-to* *M1*
              (*add-learned-cls* D
                (*update-backtrack-lvl* i
                  (*update-conflicting* *None* S)))) **and**
    confl: ∀ T. *conflicting* S = *Some* T ⟶ *trail* S ⊨*as* *CNot* T
  shows *atms-of* ((*remove1-mset* L D)) ⊆ *atm-of* ‘ *lits-of-l* (*tl* (*trail* T))
⟨*proof*⟩

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: $\forall T.$ *conflicting* S = *Some* $T \longrightarrow$ *trail* $S \models as$ *CNot* T **and**
alien: *no-strange-atm* S
shows *all-decomposition-implies-m* (*init-clss* S') (*get-all-ann-decomposition* (*trail* S'))
⟨*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: $\forall T.$ *conflicting* S = *Some* $T \longrightarrow$ *trail* $S \models as$ *CNot* T **and**
alien: *no-strange-atm* S **and**
mark-confl: *every-mark-is-a-conflict* S
shows *every-mark-is-a-conflict* S'
⟨*proof*⟩

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

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

lemma *cdcl*$_W$*-conflicting-S0-cdcl*$_W$[*simp*]:
cdcl$_W$*-conflicting* (*init-state* N)
⟨*proof*⟩

Putting all the invariants together

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

all-decomposition-implies-m (*init-clss* S') (*get-all-ann-decomposition* (*trail* S')) **and**
$cdcl_W$*-learned-clause* S' **and**
$cdcl_W$*-M-level-inv* S' **and**
no-strange-atm S' **and**
distinct-$cdcl_W$*-state* S' **and**
$cdcl_W$*-conflicting* S'
⟨*proof*⟩

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'
⟨*proof*⟩

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-propagated-lits-are-implied*, that show that the only choices we made are decided in the formula

lemma
 assumes *all-decomposition-implies-m* N (*get-all-ann-decomposition* M)
 and $\forall m \in$ *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: $\forall s \in\#$ *learned-clss* S. $\neg$*tautology* s
 shows $\forall s \in\#$ *learned-clss* S'. $\neg$*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'
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *cdcl$_W$-cp-conflicting-not-empty*[*simp*]: *conflicting* S = *Some* $D \Longrightarrow \neg$*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 (λS. *card* (*atms-of-mm* (*init-clss* S)) $-$ *length* (*trail* S)
$+$ (*if conflicting* S = *None then 1 else 0*)) S
$>$ (λS. *card* (*atms-of-mm* (*init-clss* S)) $-$ *length* (*trail* S)

$+$ (*if conflicting* S = *None then 1 else 0*)) T
⟨*proof*⟩

lemma *cdcl$_W$-cp-wf*: *wf* {(b, a). (*cdcl$_W$-M-level-inv* a ∧ *no-strange-atm* a) ∧ *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 ∧ *no-strange-atm* a) ∧ *cdcl$_W$-cp* a b)** S T
⟷ *cdcl$_W$-cp*** S T
(**is** *?I* S T ⟷ *?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 ∃ 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* ⇒ *bool* **where**
no-clause-is-false ≡
λS. (*conflicting* S = *None* ⟶ (∀ D ∈# *clauses* S. ¬*trail* S ⊨*as CNot* D))

abbreviation *conflict-is-false-with-level* :: *'st* ⇒ *bool* **where**
conflict-is-false-with-level S ≡ ∀ D. *conflicting* S = *Some* D ⟶ D ≠ {#}
⟶ (∃ L ∈# D. *get-level* (*trail* S) L = *backtrack-lvl* S)

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

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' ≡
$\forall D\ M1\ M2\ L.\ M1\ @\ Propagated\ L\ D\ \#\ M2 = trail\ S' \longrightarrow D - \{\#L\#\} \neq \{\#\}$
$\longrightarrow (\exists L.\ L \in\#\ D \land get\text{-}level\ (trail\ S')\ L = count\text{-}decided\ M1)$

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

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 $\exists S'.$ $cdcl_W$*-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* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss* S)))
∨ (*conflicting* S = *None* ∧ *trail* S ⊨*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 $\vee$ ($\exists S'$. *propagate*** S S' $\wedge$ *conflict* S' T)
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-cp-propagate-confl*:
assumes *cdcl$_W$-cp*** S T
shows *propagate*** S T $\vee$ ($\exists S'$. *propagate*** S S' $\wedge$ *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-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**

$init\text{-}clss\ S = N$ **and**
$propagate^{**}\ S\ S'$ **and**
$learned\text{-}clss\ S = \{\#\}$
shows $length\ (trail\ S) \leq length\ (trail\ S') \wedge lits\text{-}of\text{-}l\ (trail\ S') \subseteq set\ M$
$\langle proof \rangle$

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

lemma *completeness-is-a-full1-propagation*:
fixes $S :: 'st$ **and** $M :: 'v\ literal\ list$
assumes MN: $set\ M \models s\ set\text{-}mset\ N$
and $cons$: $consistent\text{-}interp\ (set\ M)$
and tot: $total\text{-}over\text{-}m\ (set\ M)\ (set\text{-}mset\ N)$
and $alien$: $no\text{-}strange\text{-}atm\ S$
and $learned$: $learned\text{-}clss\ S = \{\#\}$
and $clsS[simp]$: $init\text{-}clss\ S = N$
and $lits$: $lits\text{-}of\text{-}l\ (trail\ S) \subseteq set\ M$
shows $\exists S'.\ propagate^{**}\ S\ S' \wedge full\ cdcl_W\text{-}cp\ S\ S'$
$\langle proof \rangle$

See also *rtranclp-cdcl$_W$-cp-dropWhile-trail*

lemma *rtranclp-propagate-is-trail-append*:
$propagate^{**}\ S\ T \Longrightarrow \exists c.\ trail\ T = c\ @\ trail\ S$
$\langle proof \rangle$

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

lemma *cdcl$_W$-stgy-strong-completeness-n*:
assumes
MN: $set\ M \models s\ set\text{-}mset\ N$ **and**
$cons$: $consistent\text{-}interp\ (set\ M)$ **and**
tot: $total\text{-}over\text{-}m\ (set\ M)\ (set\text{-}mset\ N)$ **and**
$atm\text{-}incl$: $atm\text{-}of\ `\ (set\ M) \subseteq atms\text{-}of\text{-}mm\ N$ **and**
$distM$: $distinct\ M$ **and**
$length$: $n \leq length\ M$
shows
$\exists M'\ k\ S.\ length\ M' \geq n\ \wedge$
$lits\text{-}of\text{-}l\ M' \subseteq set\ M\ \wedge$
$no\text{-}dup\ M'\ \wedge$
$state\ S = (M', N, \{\#\}, k, None)\ \wedge$
$cdcl_W\text{-}stgy^{**}\ (init\text{-}state\ N)\ S$
$\langle proof \rangle$

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

No conflict with only variables of level less than backtrack level

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

definition *no-smaller-confl* (*S* ::'*st*) ≡
(∀ *M K M' D*. *M'* @ *Decided K* # *M* = *trail S* ⟶ *D* ∈# *clauses S*
⟶ ¬*M* ⊨*as CNot D*)

lemma *no-smaller-confl-init-sate*[*simp*]:
no-smaller-confl (*init-state N*) ⟨*proof*⟩

lemma $cdcl_W$*-o-no-smaller-confl-inv*:
fixes *S S'* :: '*st*
assumes
$cdcl_W$*-o S S'* **and**
lev: $cdcl_W$*-M-level-inv S* **and**
max-lev: *conflict-is-false-with-level S* **and**
smaller: *no-smaller-confl S* **and**
no-f: *no-clause-is-false S*
shows *no-smaller-confl S'*
⟨*proof*⟩

lemma *conflict-no-smaller-confl-inv*:
assumes *conflict S S'*
and *no-smaller-confl S*
shows *no-smaller-confl S'*
⟨*proof*⟩

lemma *propagate-no-smaller-confl-inv*:
assumes *propagate*: *propagate S S'*
and *n-l*: *no-smaller-confl S*
shows *no-smaller-confl S'*
⟨*proof*⟩

lemma $cdcl_W$*-cp-no-smaller-confl-inv*:
assumes *propagate*: $cdcl_W$*-cp S S'*
and *n-l*: *no-smaller-confl S*
shows *no-smaller-confl S'*
⟨*proof*⟩

lemma *rtrancp-*$cdcl_W$*-cp-no-smaller-confl-inv*:
assumes *propagate*: $cdcl_W$*-cp*** *S S'*
and *n-l*: *no-smaller-confl S*

shows *no-smaller-confl* S'
⟨*proof*⟩

lemma *trancp-cdcl$_W$-cp-no-smaller-confl-inv*:
assumes *propagate*: *cdcl$_W$-cp*$^{++}$ $S\ S'$
and *n-l*: *no-smaller-confl* S
shows *no-smaller-confl* S'
⟨*proof*⟩

lemma *full-cdcl$_W$-cp-no-smaller-confl-inv*:
assumes *full cdcl$_W$-cp* $S\ S'$
and *n-l*: *no-smaller-confl* S
shows *no-smaller-confl* S'
⟨*proof*⟩

lemma *full1-cdcl$_W$-cp-no-smaller-confl-inv*:
assumes *full1 cdcl$_W$-cp* $S\ S'$
and *n-l*: *no-smaller-confl* S
shows *no-smaller-confl* S'
⟨*proof*⟩

lemma *cdcl$_W$-stgy-no-smaller-confl-inv*:
assumes *cdcl$_W$-stgy* $S\ S'$
and *n-l*: *no-smaller-confl* S
and *conflict-is-false-with-level* S
and *cdcl$_W$-M-level-inv* S
shows *no-smaller-confl* S'
⟨*proof*⟩

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

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

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

⟨*proof*⟩

lemma *full1-cdcl$_W$-cp-exists-conflict-full1-decompose*:
assumes
confl: $\exists$ *D*$\in$#*clauses S*. *trail S* $\models$*as CNot D* **and**
full: *full cdcl$_W$-cp S U* **and**
no-confl: *conflicting S* = *None***and**
lev: *cdcl$_W$-M-level-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*
⟨*proof*⟩

lemma *cdcl$_W$-stgy-no-smaller-confl*:
assumes
cdcl$_W$-stgy S S' **and**
n-l: *no-smaller-confl S* **and**
conflict-is-false-with-level S **and**
cdcl$_W$-M-level-inv S **and**
no-clause-is-false S **and**
distinct-cdcl$_W$-state S **and**
cdcl$_W$-conflicting S
shows *no-smaller-confl S'*
⟨*proof*⟩

lemma *cdcl$_W$-stgy-ex-lit-of-max-level*:
assumes
cdcl$_W$-stgy S S' **and**
n-l: *no-smaller-confl S* **and**
conflict-is-false-with-level S **and**
cdcl$_W$-M-level-inv S **and**
no-clause-is-false S **and**
distinct-cdcl$_W$-state S **and**
cdcl$_W$-conflicting S
shows *conflict-is-false-with-level S'*
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-stgy-no-smaller-confl-inv*:
assumes
cdcl$_W$-stgy** *S S'* **and**
n-l: *no-smaller-confl S* **and**
cls-false: *conflict-is-false-with-level S* **and**
lev: *cdcl$_W$-M-level-inv S* **and**
no-f: *no-clause-is-false S* **and**
dist: *distinct-cdcl$_W$-state S* **and**
conflicting: *cdcl$_W$-conflicting S* **and**
decomp: *all-decomposition-implies-m* (*init-clss S*) (*get-all-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
  $\forall\ m\in\ set\ M.\ \neg is\text{-}decided\ m$ **and**
  $E = Some\ D$ **and**
  $state\ S = (M,\ N,\ U,\ 0,\ E)$
  $full\ cdcl_W\text{-}stgy\ S\ S'$ **and**
  $all\text{-}decomposition\text{-}implies\text{-}m\ (init\text{-}clss\ S)\ (get\text{-}all\text{-}ann\text{-}decomposition\ (trail\ S))$
  $cdcl_W\text{-}learned\text{-}clause\ S$
  $cdcl_W\text{-}M\text{-}level\text{-}inv\ S$
  $no\text{-}strange\text{-}atm\ S$
  $distinct\text{-}cdcl_W\text{-}state\ S$
  $cdcl_W\text{-}conflicting\ S$
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*: $full\ cdcl_W\text{-}stgy\ (init\text{-}state\ N)\ S'$
  and *no-d*: $distinct\text{-}mset\text{-}mset\ N$
  and *empty*: $\{\#\} \in\#\ N$
  shows $conflicting\ S' = Some\ \{\#\} \wedge unsatisfiable\ (set\text{-}mset\ (init\text{-}clss\ S'))$
⟨*proof*⟩

theorem 2.9.9 page 83 of Weidenbach's book

lemma *full-cdcl*$_W$*-stgy-final-state-conclusive*:
  fixes $S' :: 'st$
  assumes *full*: $full\ cdcl_W\text{-}stgy\ (init\text{-}state\ N)\ S'$ **and** *no-d*: $distinct\text{-}mset\text{-}mset\ N$
  shows $(conflicting\ S' = Some\ \{\#\} \wedge unsatisfiable\ (set\text{-}mset\ (init\text{-}clss\ S')))$
    $\vee\ (conflicting\ S' = None \wedge trail\ S' \models asm\ init\text{-}clss\ S')$
  ⟨*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*: $full\ cdcl_W\text{-}stgy\ (init\text{-}state\ N)\ S'$
  and *no-d*: $distinct\text{-}mset\text{-}mset\ N$
  shows $(conflicting\ S' = Some\ \{\#\} \wedge unsatisfiable\ (set\text{-}mset\ N))$
   $\vee\ (conflicting\ S' = None \wedge trail\ S' \models asm\ N \wedge satisfiable\ (set\text{-}mset\ N))$
⟨*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 $\wedge$
$(\forall s \in\# \textit{ learned-clss } S. \neg\textit{tautology } s)$ $\wedge$
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-step* $cdcl_W$*-cp S*
⟨*proof*⟩

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

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

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

$\langle proof \rangle$

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

lemma *cdcl$_W$-stgy-trail-has-new-decided-is-decide-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 $\exists S'.\ decide\ S\ S' \wedge full\ cdcl_W\text{-}cp\ S'\ T \wedge no\text{-}step\ cdcl_W\text{-}cp\ S$
$\langle proof \rangle$

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

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

lemma *cdcl$_W$-o-cannot-learn*:
assumes
$cdcl_W\text{-}o\ y\ z$ **and**
lev: $cdcl_W\text{-}M\text{-}level\text{-}inv\ y$ **and**
M: $trail\ y = c\ @\ Decided\ Kh\ \#\ H$ **and**
DL: $D \notin\#\ learned\text{-}clss\ y$ **and**
LD: $L \in\#\ D$ **and**
DH: $atms\text{-}of\ (remove1\text{-}mset\ L\ D) \subseteq atm\text{-}of\ `\ lits\text{-}of\text{-}l\ H$ **and**
LH: $atm\text{-}of\ L \notin atm\text{-}of\ `\ lits\text{-}of\text{-}l\ H$ **and**
$learned$: $\forall T.\ conflicting\ y = Some\ T \longrightarrow trail\ y \models as\ CNot\ T$ **and**
z: $trail\ z = c'\ @\ Decided\ Kh\ \#\ H$
shows $D \notin\#\ learned\text{-}clss\ z$
$\langle proof \rangle$

lemma *cdcl$_W$-stgy-with-trail-end-has-not-been-learned*:
assumes
$cdcl_W\text{-}stgy\ y\ z$ **and**
$cdcl_W\text{-}M\text{-}level\text{-}inv\ y$ **and**
$trail\ y = c\ @\ Decided\ Kh\ \#\ H$ **and**
$D \notin\#\ learned\text{-}clss\ y$ **and**
LD: $L \in\#\ D$ **and**
DH: $atms\text{-}of\ (remove1\text{-}mset\ L\ D) \subseteq atm\text{-}of\ `\ lits\text{-}of\text{-}l\ H$ **and**
LH: $atm\text{-}of\ L \notin atm\text{-}of\ `\ lits\text{-}of\text{-}l\ H$ **and**
$\forall T.\ conflicting\ y = Some\ T \longrightarrow trail\ y \models as\ CNot\ T$ **and**
$trail\ z = c'\ @\ Decided\ Kh\ \#\ H$
shows $D \notin\#\ learned\text{-}clss\ z$
$\langle proof \rangle$

lemma *rtranclp-cdcl$_W$-stgy-with-trail-end-has-not-been-learned*:

assumes
  $(\lambda a\ b.\ cdcl_W\text{-}stgy\ a\ b \wedge (\exists\, c.\ trail\ a = c\ @\ Decided\ K\#\ H\ @\ []))^{**}\ S\ z$ **and**
  $cdcl_W$*-all-struct-inv* S **and**
  trail $S = c$ @ *Decided* K # H **and**
  $D \notin\#$ *learned-clss* S **and**
  LD: $L \in\# D$ **and**
  DH: *atms-of* (*remove1-mset* L D) $\subseteq$ *atm-of* ‘ *lits-of-l* H **and**
  LH: *atm-of* $L \notin$ *atm-of* ‘ *lits-of-l* H **and**
  $\exists\, c'$. *trail* $z = c'$ @ *Decided* K # H
shows $D \notin\#$ *learned-clss* z
⟨*proof*⟩

lemma $cdcl_W$*-stgy-new-learned-clause*:
assumes $cdcl_W$*-stgy* S T **and**
  lev: $cdcl_W$*-M-level-inv* S **and**
  $E \notin\#$ *learned-clss* S **and**
  $E \in\#$ *learned-clss* T
shows $\exists\, S'$. *backtrack* S S' $\wedge$ *conflicting* S = *Some* E $\wedge$ *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*) $\leq$ *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**
$\forall s \in\#$ *learned-clss S*. $\neg$*tautology s*
shows *card*(*set-mset* (*learned-clss S*)) $\leq$ *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:
$\neg$(*learned-clss S* $\subseteq\#$ *learned-clss S'* $\wedge$ [] = *trail S'* $\wedge$ *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
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 List-Based Implementation of the DPLL and CDCL

The idea of the list-based implementation is to test the stack: the theories about the calculi, adapting the theorems to a simple implementation and the code exportation. The implementation are very simple ans simply iterate over-and-over on lists.

3.1.1 Common Rules

Propagation

The following theorem holds:

lemma *lits-of-l-unfold*[*iff*]:
$(\forall\, c \in set\ C.\ -c \in lits\text{-}of\text{-}l\ Ms) \longleftrightarrow Ms \models as\ CNot\ (mset\ C)$
$\langle proof \rangle$

The right-hand version is written at a high-level, but only the left-hand side is executable.

definition *is-unit-clause* :: $'a$ *literal list* $\Rightarrow$ $('a,\ 'b)$ *ann-lits* $\Rightarrow$ $'a$ *literal option*
where
is-unit-clause l M =
$(case\ List.filter\ (\lambda a.\ atm\text{-}of\ a \notin atm\text{-}of\ `\ lits\text{-}of\text{-}l\ M)\ l\ of$
$a\ \#\ [] \Rightarrow if\ M \models as\ CNot\ (mset\ l - \{\#a\#\})\ then\ Some\ a\ else\ None$
$|\ \text{-} \Rightarrow None)$

definition *is-unit-clause-code* :: $'a$ *literal list* $\Rightarrow$ $('a,\ 'b)$ *ann-lits*
$\Rightarrow$ $'a$ *literal option* **where**
is-unit-clause-code l M =
$(case\ List.filter\ (\lambda a.\ atm\text{-}of\ a \notin atm\text{-}of\ `\ lits\text{-}of\text{-}l\ M)\ l\ of$
$a\ \#\ [] \Rightarrow if\ (\forall\, c \in set\ (remove1\ a\ l).\ -c \in lits\text{-}of\text{-}l\ M)\ then\ Some\ a\ else\ None$
$|\ \text{-} \Rightarrow None)$

lemma *is-unit-clause-is-unit-clause-code*[*code*]:
is-unit-clause l M = *is-unit-clause-code l M*

⟨*proof*⟩

lemma *is-unit-clause-some-undef*:
  assumes *is-unit-clause l M = Some a*
  shows *undefined-lit M a*
⟨*proof*⟩

lemma *is-unit-clause-some-CNot*: *is-unit-clause l M = Some a* $\Longrightarrow$ *M* $\models$*as CNot* (*mset l* − {#*a*#})
  ⟨*proof*⟩

lemma *is-unit-clause-some-in*: *is-unit-clause l M = Some a* $\Longrightarrow$ *a* $\in$ *set l*
  ⟨*proof*⟩

lemma *is-unit-clause-Nil*[*simp*]: *is-unit-clause* [] *M = None*
  ⟨*proof*⟩

Unit propagation for all clauses

Finding the first clause to propagate

fun *find-first-unit-clause* :: *'a literal list list* ⇒ (*'a*, *'b*) *ann-lits*
  ⇒ (*'a literal* × *'a literal list*) *option* **where**
find-first-unit-clause (*a* # *l*) *M* =
  (*case is-unit-clause a M of*
    None ⇒ *find-first-unit-clause l M*
  | *Some L* ⇒ *Some* (*L*, *a*)) |
find-first-unit-clause [] - = *None*

lemma *find-first-unit-clause-some*:
  find-first-unit-clause l M = Some (*a*, *c*)
  $\Longrightarrow$ *c* $\in$ *set l* $\wedge$ *M* $\models$*as CNot* (*mset c* − {#*a*#}) $\wedge$ *undefined-lit M a* $\wedge$ *a* $\in$ *set c*
  ⟨*proof*⟩

lemma *propagate-is-unit-clause-not-None*:
  assumes *dist*: *distinct c* **and**
  M: *M* $\models$*as CNot* (*mset c* − {#*a*#}) **and**
  undef: *undefined-lit M a* **and**
  ac: *a* $\in$ *set c*
  shows *is-unit-clause c M* $\neq$ *None*
⟨*proof*⟩

lemma *find-first-unit-clause-none*:
  distinct c $\Longrightarrow$ *c* $\in$ *set l* $\Longrightarrow$ *M* $\models$*as CNot* (*mset c* − {#*a*#}) $\Longrightarrow$ *undefined-lit M a* $\Longrightarrow$ *a* $\in$ *set c*
  $\Longrightarrow$ *find-first-unit-clause l M* $\neq$ *None*
  ⟨*proof*⟩

Decide

fun *find-first-unused-var* :: *'a literal list list* ⇒ *'a literal set* ⇒ *'a literal option* **where**
find-first-unused-var (*a* # *l*) *M* =
  (*case List.find* (λ*lit. lit* $\notin$ *M* $\wedge$ −*lit* $\notin$ *M*) *a of*
    None ⇒ *find-first-unused-var l M*
  | *Some a* ⇒ *Some a*) |
find-first-unused-var [] - = *None*

lemma *find-none*[*iff*]:

$List.find\ (\lambda lit.\ lit \notin M \wedge -lit \notin M)\ a = None \longleftrightarrow\ atm\text{-}of\ `\ set\ a \subseteq atm\text{-}of\ `\ M$
⟨proof⟩

lemma *find-some*: $List.find\ (\lambda lit.\ lit \notin M \wedge -lit \notin M)\ a = Some\ b \Longrightarrow b \in set\ a \wedge b \notin M \wedge -b \notin M$
⟨proof⟩

lemma *find-first-unused-var-None*[*iff*]:
$find\text{-}first\text{-}unused\text{-}var\ l\ M = None \longleftrightarrow (\forall a \in set\ l.\ atm\text{-}of\ `\ set\ a \subseteq atm\text{-}of\ `\ M)$
⟨proof⟩

lemma *find-first-unused-var-Some-not-all-incl*:
assumes $find\text{-}first\text{-}unused\text{-}var\ l\ M = Some\ c$
shows $\neg(\forall a \in set\ l.\ atm\text{-}of\ `\ set\ a \subseteq atm\text{-}of\ `\ M)$
⟨proof⟩

lemma *find-first-unused-var-Some*:
$find\text{-}first\text{-}unused\text{-}var\ l\ M = Some\ a \Longrightarrow (\exists m \in set\ l.\ a \in set\ m \wedge a \notin M \wedge -a \notin M)$
⟨proof⟩

lemma *find-first-unused-var-undefined*:
$find\text{-}first\text{-}unused\text{-}var\ l\ (lits\text{-}of\text{-}l\ Ms) = Some\ a \Longrightarrow undefined\text{-}lit\ Ms\ a$
⟨proof⟩

3.1.2 CDCL specific functions

Level

fun *maximum-level-code*:: $'a\ literal\ list \Rightarrow ('a, 'b)\ ann\text{-}lits \Rightarrow nat$
where
$maximum\text{-}level\text{-}code\ []\ \text{-} = 0\ |$
$maximum\text{-}level\text{-}code\ (L\ \#\ Ls)\ M = max\ (get\text{-}level\ M\ L)\ (maximum\text{-}level\text{-}code\ Ls\ M)$

lemma *maximum-level-code-eq-get-maximum-level*[*simp*]:
$maximum\text{-}level\text{-}code\ D\ M = get\text{-}maximum\text{-}level\ M\ (mset\ D)$
⟨proof⟩

lemma [*code*]:
fixes $M :: ('a, 'b)\ ann\text{-}lits$
shows $get\text{-}maximum\text{-}level\ M\ (mset\ D) = maximum\text{-}level\text{-}code\ D\ M$
⟨proof⟩

Backjumping

fun *find-level-decomp* **where**
$find\text{-}level\text{-}decomp\ M\ []\ D\ k = None\ |$
$find\text{-}level\text{-}decomp\ M\ (L\ \#\ Ls)\ D\ k =$
$(case\ (get\text{-}level\ M\ L,\ maximum\text{-}level\text{-}code\ (D\ @\ Ls)\ M)\ of$
$(i, j) \Rightarrow if\ i = k \wedge j < i\ then\ Some\ (L, j)\ else\ find\text{-}level\text{-}decomp\ M\ Ls\ (L\#D)\ k$
$)$

lemma *find-level-decomp-some*:
assumes $find\text{-}level\text{-}decomp\ M\ Ls\ D\ k = Some\ (L, j)$
shows $L \in set\ Ls \wedge get\text{-}maximum\text{-}level\ M\ (mset\ (remove1\ L\ (Ls\ @\ D))) = j \wedge get\text{-}level\ M\ L = k$
⟨proof⟩

lemma *find-level-decomp-none*:

assumes *find-level-decomp M Ls E k = None* **and** *mset (L#D) = mset (Ls @ E)*
shows ¬(*L* ∈ *set Ls* ∧ *get-maximum-level M (mset D) < k* ∧ *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 *CDCL-W-Implementation*
imports *DPLL-CDCL-W-Implementation CDCL-W-Termination*
begin

3.1.3 List-based CDCL Implementation

We here have a very simple implementation of Weidenbach's CDCL, based on the same principle as the implementation of DPLL: iterating over-and-over on lists. We do not use any fancy data-structure (see the two-watched literals for a better suited data-structure).

The goal was (as for DPLL) to test the infrastructure and see if an important lemma was missing to prove the correctness and the termination of a simple implementation.

Types and Instantiation

notation *image-mset* (**infixr** *‘# 90*)

type-synonym $'a$ $cdcl_W$*-mark* = $'a$ *clause*

type-synonym $'v$ $cdcl_W$*-ann-lit* = ($'v$, $'v$ $cdcl_W$*-mark*) *ann-lit*
type-synonym $'v$ $cdcl_W$*-ann-lits* = ($'v$, $'v$ $cdcl_W$*-mark*) *ann-lits*
type-synonym $'v$ $cdcl_W$*-state* =
$'v$ $cdcl_W$*-ann-lits* × $'v$ *clauses* × $'v$ *clauses* × *nat* × $'v$ *clause option*

abbreviation *raw-trail* :: $'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'a$ **where**
raw-trail ≡ (λ(M, -). M)

abbreviation *raw-cons-trail* :: $'a \Rightarrow 'a$ *list* $\times 'b \times 'c \times 'd \times 'e \Rightarrow 'a$ *list* $\times 'b \times 'c \times 'd \times 'e$
where
raw-cons-trail ≡ (λL (M, S). ($L\#M$, S))

abbreviation *raw-tl-trail* :: $'a$ *list* $\times 'b \times 'c \times 'd \times 'e \Rightarrow 'a$ *list* $\times 'b \times 'c \times 'd \times 'e$ **where**
raw-tl-trail ≡ (λ(M, S). (*tl* M, S))

abbreviation *raw-init-clss* :: $'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'b$ **where**
raw-init-clss ≡ λ(M, N, -). N

abbreviation *raw-learned-clss* :: $'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'c$ **where**
raw-learned-clss ≡ λ(M, N, U, -). U

abbreviation *raw-backtrack-lvl* :: $'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'd$ **where**
raw-backtrack-lvl ≡ λ(M, N, U, k, -). k

abbreviation *raw-update-backtrack-lvl* :: $'d \Rightarrow 'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'a \times 'b \times 'c \times 'd \times 'e$
where
raw-update-backtrack-lvl ≡ λk (M, N, U, -, S). (M, N, U, k, S)

abbreviation *raw-conflicting* :: $'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'e$ **where**
raw-conflicting ≡ λ(M, N, U, k, D). D

abbreviation *raw-update-conflicting* :: $'e \Rightarrow 'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'a \times 'b \times 'c \times 'd \times 'e$
where
raw-update-conflicting ≡ λS (M, N, U, k, -). (M, N, U, k, S)

abbreviation *S0-*$cdcl_W$ N ≡ (([], N, {#}, *0*, *None*):: $'v$ $cdcl_W$*-state*)

abbreviation *raw-add-learned-clss* **where**
raw-add-learned-clss ≡ λC (M, N, U, S). (M, N, {#C#} + U, S)

abbreviation *raw-remove-cls* **where**
raw-remove-cls ≡ λC (M, N, U, S). (M, *removeAll-mset* C N, *removeAll-mset* C U, S)

lemma *raw-trail-conv*: *raw-trail* (M, N, U, k, D) = M **and**
clauses-conv: *raw-init-clss* (M, N, U, k, D) = N **and**
raw-learned-clss-conv: *raw-learned-clss* (M, N, U, k, D) = U **and**
raw-conflicting-conv: *raw-conflicting* (M, N, U, k, D) = D **and**
raw-backtrack-lvl-conv: *raw-backtrack-lvl* (M, N, U, k, D) = k
⟨*proof*⟩

lemma *state-conv*:
S = (*raw-trail* S, *raw-init-clss* S, *raw-learned-clss* S, *raw-backtrack-lvl* S, *raw-conflicting* S)
⟨*proof*⟩

interpretation $state_W$
raw-trail raw-init-clss raw-learned-clss raw-backtrack-lvl raw-conflicting
λL (M, S). $(L \# M, S)$
$\lambda(M, S)$. $(tl\ M, S)$
λC (M, N, U, S). $(M, N, \{\#C\#\} + U, S)$
λC (M, N, U, S). $(M$, *removeAll-mset* C N, *removeAll-mset* C U, $S)$
$\lambda(k::nat)$ $(M, N, U, \text{-}, D)$. (M, N, U, k, D)
λD $(M, N, U, k, \text{-})$. (M, N, U, k, D)
λN. $([], N, \{\#\}, 0, None)$
⟨*proof*⟩

interpretation $conflict\text{-}driven\text{-}clause\text{-}learning_W$ *raw-trail raw-init-clss raw-learned-clss raw-backtrack-lvl raw-conflicting*
λL (M, S). $(L \# M, S)$
$\lambda(M, S)$. $(tl\ M, S)$
λC (M, N, U, S). $(M, N, \{\#C\#\} + U, S)$
λC (M, N, U, S). $(M$, *removeAll-mset* C N, *removeAll-mset* C U, $S)$
$\lambda(k::nat)$ $(M, N, U, \text{-}, D)$. (M, N, U, k, D)
λD $(M, N, U, k, \text{-})$. (M, N, U, k, D)
λN. $([], N, \{\#\}, 0, None)$
⟨*proof*⟩

declare *clauses-def*[*simp*]

lemma $cdcl_W$*-state-eq-equality*[*iff*]: *state-eq* S $T \longleftrightarrow S = T$
⟨*proof*⟩
declare *state-simp*[*simp del*]

lemma *reduce-trail-to-empty-trail*[*simp*]:
reduce-trail-to F $([], aa, ab, ac, b) = ([], aa, ab, ac, b)$
⟨*proof*⟩

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

lemma *reduce-trail-to*:
reduce-trail-to F S =
((*if length* (*raw-trail* S) $\geq$ *length* F
then drop (*length* (*raw-trail* S) $-$ *length* F) (*raw-trail* S)
else []), *raw-init-clss* S, *raw-learned-clss* S, *raw-backtrack-lvl* S, *raw-conflicting* S)
(**is** *?S* = -)
⟨*proof*⟩

3.1.4 CDCL Implementation

Definition of the rules

Types **lemma** *true-raw-init-clss-remdups*[*simp*]:
$I \models s$ $(mset \circ remdups)$ ‘ $N \longleftrightarrow I \models s$ $mset$ ‘ N
⟨*proof*⟩

lemma *satisfiable-mset-remdups*[*simp*]:

satisfiable ((*mset* ∘ *remdups*) ‘ *N*) ⟷ *satisfiable* (*mset* ‘ *N*)
⟨*proof*⟩

type-synonym $'v$ *cdcl*$_W$*-state-inv-st* = ($'v$, $'v$ *literal list*) *ann-lit list* ×
$'v$ *literal list list* × $'v$ *literal list list* × *nat* × $'v$ *literal list option*

We need some functions to convert between our abstract state $'v$ *cdcl*$_W$*-state* and the concrete state $'v$ *cdcl*$_W$*-state-inv-st*.

fun *convert* :: ($'a$, $'c$ *list*) *ann-lit* ⇒ ($'a$, $'c$ *multiset*) *ann-lit* **where**
convert (*Propagated L C*) = *Propagated L* (*mset C*) |
convert (*Decided K*) = *Decided K*

abbreviation *convertC* :: $'a$ *list option* ⇒ $'a$ *multiset option* **where**
convertC ≡ *map-option mset*

lemma *convert-Propagated*[*elim*!]:
convert z = *Propagated L C* ⟹ (∃ C'. *z* = *Propagated L* C' ∧ *C* = *mset* C')
⟨*proof*⟩

lemma *is-decided-convert*[*simp*]: *is-decided* (*convert x*) = *is-decided x*
⟨*proof*⟩

lemma *get-level-map-convert*[*simp*]:
get-level (*map convert M*) *x* = *get-level M x*
⟨*proof*⟩

lemma *get-maximum-level-map-convert*[*simp*]:
get-maximum-level (*map convert M*) *D* = *get-maximum-level M D*
⟨*proof*⟩

Conversion function

fun *toS* :: $'v$ *cdcl*$_W$*-state-inv-st* ⇒ $'v$ *cdcl*$_W$*-state* **where**
toS (*M*, *N*, *U*, *k*, *C*) = (*map convert M*, *mset* (*map mset N*), *mset* (*map mset U*), *k*, *convertC C*)

Definition an abstract type

typedef $'v$ *cdcl*$_W$*-state-inv* = {*S*::$'v$ *cdcl*$_W$*-state-inv-st*. *cdcl*$_W$*-all-struct-inv* (*toS S*)}
morphisms *rough-state-of state-of*
⟨*proof*⟩

instantiation *cdcl*$_W$*-state-inv* :: (*type*) *equal*
begin
definition *equal-cdcl*$_W$*-state-inv* :: $'v$ *cdcl*$_W$*-state-inv* ⇒ $'v$ *cdcl*$_W$*-state-inv* ⇒ *bool* **where**
equal-cdcl$_W$*-state-inv S* S' = (*rough-state-of S* = *rough-state-of* S')
instance
⟨*proof*⟩
end

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

lemma *atm-lit-of-convert*[*simp*]:
lit-of (*convert x*) = *lit-of x*
⟨*proof*⟩

lemma *undefined-lit-map-convert*[*iff*]:
undefined-lit (*map convert M*) *L* $\longleftrightarrow$ *undefined-lit M L*
⟨*proof*⟩

lemma *true-annot-map-convert*[*simp*]: *map convert M* $\models$*a N* $\longleftrightarrow$ *M* $\models$*a N*
⟨*proof*⟩

lemma *true-annots-map-convert*[*simp*]: *map convert M* $\models$*as N* $\longleftrightarrow$ *M* $\models$*as N*
⟨*proof*⟩

lemmas *propagateE*
lemma *find-first-unit-clause-some-is-propagate*:
assumes *H*: *find-first-unit-clause* (*N* @ *U*) *M* = *Some* (*L*, *C*)
shows *propagate* (*toS* (*M*, *N*, *U*, *k*, *None*)) (*toS* (*Propagated L C* # *M*, *N*, *U*, *k*, *None*))
⟨*proof*⟩

The Transitions

Propagate **definition** *do-propagate-step* **where**
do-propagate-step S =
(*case S of*
(*M*, *N*, *U*, *k*, *None*) ⇒
(*case find-first-unit-clause* (*N* @ *U*) *M of*
Some (*L*, *C*) ⇒ (*Propagated L C* # *M*, *N*, *U*, *k*, *None*)
| *None* ⇒ (*M*, *N*, *U*, *k*, *None*))
| *S* ⇒ *S*)

lemma *do-propgate-step*:
do-propagate-step S ≠ *S* ⟹ *propagate* (*toS S*) (*toS* (*do-propagate-step S*))
⟨*proof*⟩

lemma *do-propagate-step-option*[*simp*]:
raw-conflicting S ≠ *None* ⟹ *do-propagate-step S* = *S*
⟨*proof*⟩

lemma *do-propagate-step-no-step*:
assumes *dist*: ∀ *c*∈*set* (*raw-init-clss S* @ *raw-learned-clss S*). *distinct c* **and**
prop-step: *do-propagate-step S* = *S*
shows *no-step propagate* (*toS S*)
⟨*proof*⟩

Conflict **fun** *find-conflict* **where**
find-conflict M [] = *None* |
find-conflict M (*N* # *Ns*) = (*if* (∀ *c* ∈ *set N*. −*c* ∈ *lits-of-l M*) *then Some N else find-conflict M Ns*)

lemma *find-conflict-Some*:
find-conflict M Ns = *Some N* ⟹ *N* ∈ *set Ns* ∧ *M* $\models$*as CNot* (*mset N*)
⟨*proof*⟩

lemma *find-conflict-None*:
find-conflict M Ns = *None* $\longleftrightarrow$ (∀ *N* ∈ *set Ns*. ¬*M* $\models$*as CNot* (*mset N*))
⟨*proof*⟩

lemma *find-conflict-None-no-confl*:
find-conflict M (*N*@*U*) = *None* $\longleftrightarrow$ *no-step conflict* (*toS* (*M*, *N*, *U*, *k*, *None*))
⟨*proof*⟩

definition *do-conflict-step* **where**
do-conflict-step S =
  (*case S of*
    (*M*, *N*, *U*, *k*, *None*) ⇒
      (*case find-conflict M* (*N* @ *U*) *of*
        Some a ⇒ (*M*, *N*, *U*, *k*, *Some a*)
      | *None* ⇒ (*M*, *N*, *U*, *k*, *None*))
  | *S* ⇒ *S*)

lemma *do-conflict-step*:
  do-conflict-step S ≠ *S* ⟹ *conflict* (*toS S*) (*toS* (*do-conflict-step S*))
  ⟨*proof*⟩

lemma *do-conflict-step-no-step*:
  do-conflict-step S = *S* ⟹ *no-step conflict* (*toS S*)
  ⟨*proof*⟩

lemma *do-conflict-step-option*[*simp*]:
  raw-conflicting S ≠ *None* ⟹ *do-conflict-step S* = *S*
  ⟨*proof*⟩

lemma *do-conflict-step-raw-conflicting*[*dest*]:
  do-conflict-step S ≠ *S* ⟹ *raw-conflicting* (*do-conflict-step S*) ≠ *None*
  ⟨*proof*⟩

definition *do-cp-step* **where**
do-cp-step S =
  (*do-propagate-step o do-conflict-step*) *S*

lemma *cp-step-is-*$cdcl_W$*-cp*:
  assumes *H*: *do-cp-step S* ≠ *S*
  shows $cdcl_W$*-cp* (*toS S*) (*toS* (*do-cp-step S*))
⟨*proof*⟩

lemma *do-cp-step-eq-no-prop-no-confl*:
  do-cp-step S = *S* ⟹ *do-conflict-step S* = *S* ∧ *do-propagate-step S* = *S*
  ⟨*proof*⟩

lemma *no-*$cdcl_W$*-cp-iff-no-propagate-no-conflict*:
  no-step $cdcl_W$*-cp S* ⟷ *no-step propagate S* ∧ *no-step conflict S*
  ⟨*proof*⟩

lemma *do-cp-step-eq-no-step*:
  assumes *H*: *do-cp-step S* = *S* **and** ∀ *c* ∈ *set* (*raw-init-clss S* @ *raw-learned-clss S*). *distinct c*
  shows *no-step* $cdcl_W$*-cp* (*toS S*)
  ⟨*proof*⟩

lemma $cdcl_W$*-cp-*$cdcl_W$*-st*: $cdcl_W$*-cp S S'* ⟹ $cdcl_W{}^{**}$ *S S'*
  ⟨*proof*⟩

lemma $cdcl_W$*-all-struct-inv-rough-state*[*simp*]: $cdcl_W$*-all-struct-inv* (*toS* (*rough-state-of S*))
  ⟨*proof*⟩

lemma [*simp*]: $cdcl_W$*-all-struct-inv* (*toS S*) ⟹ *rough-state-of* (*state-of S*) = *S*
  ⟨*proof*⟩

lemma *rough-state-of-state-of-do-cp-step*[*simp*]:
rough-state-of (*state-of* (*do-cp-step* (*rough-state-of S*))) = *do-cp-step* (*rough-state-of S*)
⟨*proof*⟩

Skip **fun** *do-skip-step* :: $'v$ $cdcl_W$*-state-inv-st* ⇒ $'v$ $cdcl_W$*-state-inv-st* **where**
do-skip-step (*Propagated L C # Ls,N,U,k, Some D*) =
(*if* $-L \notin$ *set D* ∧ *D* ≠ []
then (*Ls, N, U, k, Some D*)
else (*Propagated L C #Ls, N, U, k, Some D*)) |
do-skip-step S = *S*

lemma *do-skip-step*:
do-skip-step S ≠ *S* ⟹ *skip* (*toS S*) (*toS* (*do-skip-step S*))
⟨*proof*⟩

lemma *do-skip-step-no*:
do-skip-step S = *S* ⟹ *no-step skip* (*toS S*)
⟨*proof*⟩

lemma *do-skip-step-raw-trail-is-None*[*iff*]:
do-skip-step S = (*a, b, c, d, None*) ⟷ *S* = (*a, b, c, d, None*)
⟨*proof*⟩

Resolve **fun** *maximum-level-code*:: $'a$ *literal list* ⇒ ($'a$, $'a$ *literal list*) *ann-lit list* ⇒ *nat*
where
maximum-level-code [] - = *0* |
maximum-level-code (*L # Ls*) *M* = *max* (*get-level M L*) (*maximum-level-code Ls M*)

lemma *maximum-level-code-eq-get-maximum-level*[*code, simp*]:
maximum-level-code D M = *get-maximum-level M* (*mset D*)
⟨*proof*⟩

fun *do-resolve-step* :: $'v$ $cdcl_W$*-state-inv-st* ⇒ $'v$ $cdcl_W$*-state-inv-st* **where**
do-resolve-step (*Propagated L C # Ls, N, U, k, Some D*) =
(*if* $-L \in$ *set D* ∧ *maximum-level-code* (*remove1* ($-L$) *D*) (*Propagated L C # Ls*) = *k*
then (*Ls, N, U, k, Some* (*remdups* (*remove1 L C* @ *remove1* ($-L$) *D*)))
else (*Propagated L C # Ls, N, U, k, Some D*)) |
do-resolve-step S = *S*

lemma *do-resolve-step*:
$cdcl_W$*-all-struct-inv* (*toS S*) ⟹ *do-resolve-step S* ≠ *S*
⟹ *resolve* (*toS S*) (*toS* (*do-resolve-step S*))
⟨*proof*⟩

lemma *do-resolve-step-no*:
do-resolve-step S = *S* ⟹ *no-step resolve* (*toS S*)
⟨*proof*⟩

lemma *rough-state-of-state-of-resolve*[*simp*]:
$cdcl_W$*-all-struct-inv* (*toS S*) ⟹ *rough-state-of* (*state-of* (*do-resolve-step S*)) = *do-resolve-step S*
⟨*proof*⟩

lemma *do-resolve-step-raw-trail-is-None*[*iff*]:
do-resolve-step S = (*a, b, c, d, None*) ⟷ *S* = (*a, b, c, d, None*)

$\langle proof \rangle$

Backjumping **lemma** *get-all-ann-decomposition-map-convert*:
$(get\text{-}all\text{-}ann\text{-}decomposition\ (map\ convert\ M)) =$
$map\ (\lambda(a,\ b).\ (map\ convert\ a,\ map\ convert\ b))\ (get\text{-}all\text{-}ann\text{-}decomposition\ M)$
$\langle proof \rangle$

lemma *do-backtrack-step*:
assumes
db: $do\text{-}backtrack\text{-}step\ S \neq S$ **and**
inv: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ (toS\ S)$
shows $backtrack\ (toS\ S)\ (toS\ (do\text{-}backtrack\text{-}step\ S))$
$\langle proof \rangle$

lemma *map-eq-list-length*:
$map\ f\ L = L' \Longrightarrow length\ L = length\ L'$
$\langle proof \rangle$

lemma *map-mmset-of-mlit-eq-cons*:
assumes $map\ convert\ M = a\ @\ c$
obtains $a'\ c'$ **where**
$M = a'\ @\ c'$ **and**
$a = map\ convert\ a'$ **and**
$c = map\ convert\ c'$
$\langle proof \rangle$

lemma *Decided-convert-iff*:
$Decided\ K = convert\ za \longleftrightarrow za = Decided\ K$
$\langle proof \rangle$

lemma *do-backtrack-step-no*:
assumes
db: $do\text{-}backtrack\text{-}step\ S = S$ **and**
inv: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ (toS\ S)$
shows $no\text{-}step\ backtrack\ (toS\ S)$
$\langle proof \rangle$

lemma *rough-state-of-state-of-backtrack*[*simp*]:
assumes *inv*: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ (toS\ S)$
shows $rough\text{-}state\text{-}of\ (state\text{-}of\ (do\text{-}backtrack\text{-}step\ S)) = do\text{-}backtrack\text{-}step\ S$
$\langle proof \rangle$

Decide **fun** *do-decide-step* **where**
$do\text{-}decide\text{-}step\ (M,\ N,\ U,\ k,\ None) =$
$(case\ find\text{-}first\text{-}unused\text{-}var\ N\ (lits\text{-}of\text{-}l\ M)\ of$
$None \Rightarrow (M,\ N,\ U,\ k,\ None)$
$|\ Some\ L \Rightarrow (Decided\ L\ \#\ M,\ N,\ U,\ k{+}1,\ None))\ |$
$do\text{-}decide\text{-}step\ S = S$

lemma *do-decide-step*:
$do\text{-}decide\text{-}step\ S \neq S \Longrightarrow decide\ (toS\ S)\ (toS\ (do\text{-}decide\text{-}step\ S))$
$\langle proof \rangle$

lemma *do-decide-step-no*:
$do\text{-}decide\text{-}step\ S = S \Longrightarrow no\text{-}step\ decide\ (toS\ S)$

$\langle proof \rangle$

lemma *rough-state-of-state-of-do-decide-step*[*simp*]:
$cdcl_W$*-all-struct-inv* (*toS S*) $\Longrightarrow$ *rough-state-of* (*state-of* (*do-decide-step S*)) = *do-decide-step S*
$\langle proof \rangle$

lemma *rough-state-of-state-of-do-skip-step*[*simp*]:
$cdcl_W$*-all-struct-inv* (*toS S*) $\Longrightarrow$ *rough-state-of* (*state-of* (*do-skip-step S*)) = *do-skip-step S*
$\langle proof \rangle$

Code generation

Type definition There are two invariants: one while applying conflict and propagate and one for the other rules

declare *rough-state-of-inverse*[*simp add*]
definition *Con* **where**
Con xs = *state-of* (*if* $cdcl_W$*-all-struct-inv* (*toS* (*fst xs*, *snd xs*)) *then xs*
else ([], [], [], *0*, *None*))

lemma [*code abstype*]:
Con (*rough-state-of S*) = *S*
$\langle proof \rangle$

definition *do-cp-step'* **where**
do-cp-step' S = *state-of* (*do-cp-step* (*rough-state-of S*))

typedef *'v* $cdcl_W$*-state-inv-from-init-state* =
{*S*:: *'v* $cdcl_W$*-state-inv-st*. $cdcl_W$*-all-struct-inv* (*toS S*)
∧ $cdcl_W$*-stgy*** (*S0-*$cdcl_W$ (*raw-init-clss* (*toS S*))) (*toS S*)}
morphisms *rough-state-from-init-state-of state-from-init-state-of*
$\langle proof \rangle$

instantiation $cdcl_W$*-state-inv-from-init-state* :: (*type*) *equal*
begin
definition *equal-*$cdcl_W$*-state-inv-from-init-state* :: *'v* $cdcl_W$*-state-inv-from-init-state* ⇒
'v $cdcl_W$*-state-inv-from-init-state* ⇒ *bool* **where**
equal-$cdcl_W$*-state-inv-from-init-state S S'* ⟷
(*rough-state-from-init-state-of S* = *rough-state-from-init-state-of S'*)
instance
$\langle proof \rangle$
end

definition *ConI* **where**
ConI S = *state-from-init-state-of* (*if* $cdcl_W$*-all-struct-inv* (*toS* (*fst S*, *snd S*))
∧ $cdcl_W$*-stgy*** (*S0-*$cdcl_W$ (*raw-init-clss* (*toS S*))) (*toS S*) *then S else* ([], [], [], *0*, *None*))

lemma [*code abstype*]:
ConI (*rough-state-from-init-state-of S*) = *S*
$\langle proof \rangle$

definition *id-of-I-to*:: *'v* $cdcl_W$*-state-inv-from-init-state* ⇒ *'v* $cdcl_W$*-state-inv* **where**
id-of-I-to S = *state-of* (*rough-state-from-init-state-of S*)

lemma [*code abstract*]:

rough-state-of (*id-of-I-to S*) = *rough-state-from-init-state-of S*
⟨*proof*⟩

Conflict and Propagate **function** *do-full1-cp-step* :: *'v* $cdcl_W$*-state-inv* ⇒ *'v* $cdcl_W$*-state-inv*
where
do-full1-cp-step S =
(*let S'* = *do-cp-step' S in*
if S = *S' then S else do-full1-cp-step S'*)
⟨*proof*⟩
termination
⟨*proof*⟩

lemma *do-full1-cp-step-fix-point-of-do-full1-cp-step*:
do-cp-step(*rough-state-of* (*do-full1-cp-step S*)) = (*rough-state-of* (*do-full1-cp-step S*))
⟨*proof*⟩

lemma *in-clauses-rough-state-of-is-distinct*:
c∈*set* (*raw-init-clss* (*rough-state-of S*) @ *raw-learned-clss* (*rough-state-of S*)) ⟹ *distinct c*
⟨*proof*⟩

lemma *do-full1-cp-step-full*:
full $cdcl_W$*-cp* (*toS* (*rough-state-of S*))
(*toS* (*rough-state-of* (*do-full1-cp-step S*)))
⟨*proof*⟩

lemma [*code abstract*]:
rough-state-of (*do-cp-step' S*) = *do-cp-step* (*rough-state-of S*)
⟨*proof*⟩

The other rules **fun** *do-other-step* **where**
do-other-step S =
(*let T* = *do-skip-step S in*
if T ≠ *S*
then T
else
(*let U* = *do-resolve-step T in*
if U ≠ *T*
then U else
(*let V* = *do-backtrack-step U in*
if V ≠ *U then V else do-decide-step V*)))

lemma *do-other-step*:
assumes *inv*: $cdcl_W$*-all-struct-inv* (*toS S*) **and**
st: *do-other-step S* ≠ *S*
shows $cdcl_W$*-o* (*toS S*) (*toS* (*do-other-step S*))
⟨*proof*⟩

lemma *do-other-step-no*:
assumes *inv*: $cdcl_W$*-all-struct-inv* (*toS S*) **and**
st: *do-other-step S* = *S*
shows *no-step* $cdcl_W$*-o* (*toS S*)
⟨*proof*⟩

lemma *rough-state-of-state-of-do-other-step*[*simp*]:
rough-state-of (*state-of* (*do-other-step* (*rough-state-of S*))) = *do-other-step* (*rough-state-of S*)

⟨*proof*⟩

definition *do-other-step′* **where**
do-other-step′ S =
  state-of (*do-other-step* (*rough-state-of S*))

lemma *rough-state-of-do-other-step′*[*code abstract*]:
 rough-state-of (*do-other-step′ S*) = *do-other-step* (*rough-state-of S*)
 ⟨*proof*⟩

definition *do-cdcl*$_W$*-stgy-step* **where**
do-cdcl$_W$*-stgy-step S* =
  (*let T* = *do-full1-cp-step S in*
    if T ≠ *S*
    then T
    else
      (*let U* = (*do-other-step′ T*) *in*
       (*do-full1-cp-step U*)))

definition *do-cdcl*$_W$*-stgy-step′* **where**
do-cdcl$_W$*-stgy-step′ S* = *state-from-init-state-of* (*rough-state-of* (*do-cdcl*$_W$*-stgy-step* (*id-of-I-to S*)))

lemma *toS-do-full1-cp-step-not-eq*: *do-full1-cp-step S* ≠ *S* ⟹
    toS (*rough-state-of S*) ≠ *toS* (*rough-state-of* (*do-full1-cp-step S*))
⟨*proof*⟩

do-full1-cp-step should not be unfolded anymore:

declare *do-full1-cp-step.simps*[*simp del*]

Correction of the transformation **lemma** *do-cdcl*$_W$*-stgy-step*:
  assumes *do-cdcl*$_W$*-stgy-step S* ≠ *S*
  shows *cdcl*$_W$*-stgy* (*toS* (*rough-state-of S*)) (*toS* (*rough-state-of* (*do-cdcl*$_W$*-stgy-step S*)))
⟨*proof*⟩

lemma *length-raw-trail-toS*[*simp*]:
  length (*raw-trail* (*toS S*)) = *length* (*raw-trail S*)
  ⟨*proof*⟩

lemma *raw-conflicting-noTrue-iff-toS*[*simp*]:
  raw-conflicting (*toS S*) ≠ *None* ⟷ *raw-conflicting S* ≠ *None*
  ⟨*proof*⟩

lemma *raw-trail-toS-neq-imp-raw-trail-neq*:
  raw-trail (*toS S*) ≠ *raw-trail* (*toS S′*) ⟹ *raw-trail S* ≠ *raw-trail S′*
  ⟨*proof*⟩

lemma *do-skip-step-raw-trail-changed-or-conflict*:
  assumes *d*: *do-other-step S* ≠ *S*
  and *inv*: *cdcl*$_W$*-all-struct-inv* (*toS S*)
  shows *raw-trail S* ≠ *raw-trail* (*do-other-step S*)
⟨*proof*⟩

lemma *do-full1-cp-step-induct*:
  (⋀*S*. (*S* ≠ *do-cp-step′ S* ⟹ *P* (*do-cp-step′ S*)) ⟹ *P S*) ⟹ *P a0*
  ⟨*proof*⟩

lemma *do-cp-step-neq-raw-trail-increase*:
$\exists$ *c. raw-trail* (*do-cp-step S*) = *c* @ *raw-trail S* $\wedge$($\forall$ *m* $\in$ *set c.* $\neg$ *is-decided m*)
⟨*proof*⟩

lemma *do-full1-cp-step-neq-raw-trail-increase*:
$\exists$ *c. raw-trail* (*rough-state-of* (*do-full1-cp-step S*)) = *c* @ *raw-trail* (*rough-state-of S*)
$\wedge$ ($\forall$ *m* $\in$ *set c.* $\neg$ *is-decided m*)
⟨*proof*⟩

lemma *do-cp-step-raw-conflicting*:
raw-conflicting (*rough-state-of S*) $\neq$ *None* $\Longrightarrow$ *do-cp-step′ S* = *S*
⟨*proof*⟩

lemma *do-full1-cp-step-raw-conflicting*:
raw-conflicting (*rough-state-of S*) $\neq$ *None* $\Longrightarrow$ *do-full1-cp-step S* = *S*
⟨*proof*⟩

lemma *do-decide-step-not-raw-conflicting-one-more-decide*:
assumes
raw-conflicting S = *None* **and**
do-decide-step S $\neq$ *S*
shows *Suc* (*length* (*filter is-decided* (*raw-trail S*)))
= *length* (*filter is-decided* (*raw-trail* (*do-decide-step S*)))
⟨*proof*⟩

lemma *do-decide-step-not-raw-conflicting-one-more-decide-bt*:
assumes *raw-conflicting S* $\neq$ *None* **and**
do-decide-step S $\neq$ *S*
shows *length* (*filter is-decided* (*raw-trail S*)) < *length* (*filter is-decided* (*raw-trail* (*do-decide-step S*)))
⟨*proof*⟩

lemma *count-decided-raw-trail-toS*:
count-decided (*raw-trail* (*toS S*)) = *count-decided* (*raw-trail S*)
⟨*proof*⟩

lemma *do-other-step-not-raw-conflicting-one-more-decide-bt*:
assumes
raw-conflicting (*rough-state-of S*) $\neq$ *None* **and**
raw-conflicting (*rough-state-of* (*do-other-step′ S*)) = *None* **and**
do-other-step′ S $\neq$ *S*
shows *count-decided* (*raw-trail* (*rough-state-of S*))
> *count-decided* (*raw-trail* (*rough-state-of* (*do-other-step′ S*)))
⟨*proof*⟩

lemma *do-other-step-not-raw-conflicting-one-more-decide*:
assumes *raw-conflicting* (*rough-state-of S*) = *None* **and**
do-other-step′ S $\neq$ *S*
shows *1* + *length* (*filter is-decided* (*raw-trail* (*rough-state-of S*)))
= *length* (*filter is-decided* (*raw-trail* (*rough-state-of* (*do-other-step′ S*))))
⟨*proof*⟩

lemma *rough-state-of-state-of-do-skip-step-rough-state-of*[*simp*]:
rough-state-of (*state-of* (*do-skip-step* (*rough-state-of S*))) = *do-skip-step* (*rough-state-of S*)
⟨*proof*⟩

lemma *raw-conflicting-do-resolve-step-iff*[*iff*]:

raw-conflicting (*do-resolve-step* S) = *None* $\longleftrightarrow$ *raw-conflicting* S = *None*
⟨*proof*⟩

lemma *raw-conflicting-do-skip-step-iff*[*iff*]:
raw-conflicting (*do-skip-step* S) = *None* $\longleftrightarrow$ *raw-conflicting* S = *None*
⟨*proof*⟩

lemma *raw-conflicting-do-decide-step-iff*[*iff*]:
raw-conflicting (*do-decide-step* S) = *None* $\longleftrightarrow$ *raw-conflicting* S = *None*
⟨*proof*⟩

lemma *raw-conflicting-do-backtrack-step-imp*[*simp*]:
do-backtrack-step $S \neq S \Longrightarrow$ *raw-conflicting* (*do-backtrack-step* S) = *None*
⟨*proof*⟩

lemma *do-skip-step-eq-iff-raw-trail-eq*:
do-skip-step $S = S \longleftrightarrow$ *raw-trail* (*do-skip-step* S) = *raw-trail* S
⟨*proof*⟩

lemma *do-decide-step-eq-iff-raw-trail-eq*:
do-decide-step $S = S \longleftrightarrow$ *raw-trail* (*do-decide-step* S) = *raw-trail* S
⟨*proof*⟩

lemma *do-backtrack-step-eq-iff-raw-trail-eq*:
assumes *no-dup* (*raw-trail* S)
shows *do-backtrack-step* $S = S \longleftrightarrow$ *raw-trail* (*do-backtrack-step* S) = *raw-trail* S
⟨*proof*⟩

lemma *do-resolve-step-eq-iff-raw-trail-eq*:
do-resolve-step $S = S \longleftrightarrow$ *raw-trail* (*do-resolve-step* S) = *raw-trail* S
⟨*proof*⟩

lemma *do-other-step-eq-iff-raw-trail-eq*:
assumes *no-dup* (*raw-trail* S)
shows *raw-trail* (*do-other-step* S) = *raw-trail* $S \longleftrightarrow$ *do-other-step* $S = S$
⟨*proof*⟩

lemma *do-full1-cp-step-do-other-step′-normal-form*[*dest!*]:
assumes H: *do-full1-cp-step* (*do-other-step′* S) = S
shows *do-other-step′* $S = S \wedge$ *do-full1-cp-step* $S = S$
⟨*proof*⟩

lemma *do-cdcl*$_W$*-stgy-step-no*:
assumes S: *do-cdcl*$_W$*-stgy-step* $S = S$
shows *no-step cdcl*$_W$*-stgy* (*toS* (*rough-state-of* S))
⟨*proof*⟩

lemma *toS-rough-state-of-state-of-rough-state-from-init-state-of*[*simp*]:
toS (*rough-state-of* (*state-of* (*rough-state-from-init-state-of* S)))
= *toS* (*rough-state-from-init-state-of* S)
⟨*proof*⟩

lemma *cdcl*$_W$*-cp-is-rtranclp-cdcl*$_W$: *cdcl*$_W$*-cp* S $T \Longrightarrow$ *cdcl*$_W$** S T
⟨*proof*⟩

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

lemma *cdcl$_W$-stgy-is-rtranclp-cdcl$_W$*:
$cdcl_W\text{-}stgy\ S\ T \Longrightarrow cdcl_W{}^{**}\ S\ T$
⟨*proof*⟩

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

lemma *clauses-toS-rough-state-of-do-cdcl$_W$-stgy-step*[*simp*]:
raw-init-clss (*toS* (*rough-state-of* (*do-cdcl$_W$-stgy-step* (*state-of* (*rough-state-from-init-state-of S*)))))
= *raw-init-clss* (*toS* (*rough-state-from-init-state-of S*)) (**is** - = *raw-init-clss* (*toS ?S*))
⟨*proof*⟩

lemma *rough-state-from-init-state-of-do-cdcl$_W$-stgy-step′*[*code abstract*]:
rough-state-from-init-state-of (*do-cdcl$_W$-stgy-step′ S*) =
rough-state-of (*do-cdcl$_W$-stgy-step* (*id-of-I-to S*))
⟨*proof*⟩

All rules together **function** *do-all-cdcl$_W$-stgy* **where**
do-all-cdcl$_W$-stgy S =
(*let T* = *do-cdcl$_W$-stgy-step′ S in*
if T = *S then S else do-all-cdcl$_W$-stgy T*)
⟨*proof*⟩
termination
⟨*proof*⟩

thm *do-all-cdcl$_W$-stgy.induct*
lemma *do-all-cdcl$_W$-stgy-induct*:
($\bigwedge S$. (*do-cdcl$_W$-stgy-step′ S* $\neq$ *S* $\Longrightarrow$ *P* (*do-cdcl$_W$-stgy-step′ S*)) $\Longrightarrow$ *P S*) $\Longrightarrow$ *P a0*
⟨*proof*⟩

lemma *no-step-cdcl$_W$-stgy-cdcl$_W$-all*:
fixes *S* :: *′a cdcl$_W$-state-inv-from-init-state*
shows *no-step cdcl$_W$-stgy* (*toS* (*rough-state-from-init-state-of* (*do-all-cdcl$_W$-stgy S*)))
⟨*proof*⟩

lemma *do-all-cdcl$_W$-stgy-is-rtranclp-cdcl$_W$-stgy*:
cdcl$_W$-stgy** (*toS* (*rough-state-from-init-state-of S*))
(*toS* (*rough-state-from-init-state-of* (*do-all-cdcl$_W$-stgy S*)))
⟨*proof*⟩

Final theorem:

lemma *DPLL-tot-correct*:
assumes
r: *rough-state-from-init-state-of* (*do-all-cdcl$_W$-stgy* (*state-from-init-state-of*
(([], *map remdups N*, [], *0*, *None*)))) = *S* **and**
S: (*M′*, *N′*, *U′*, *k*, *E*) = *toS S*
shows (*E* $\neq$ *Some* {#} $\wedge$ *satisfiable* (*set* (*map mset N*)))
$\vee$ (*E* = *Some* {#} $\wedge$ *unsatisfiable* (*set* (*map mset N*)))
⟨*proof*⟩

The Code The SML code is skipped in the documentation, but stays to ensure that some version of the exported code is working. The only difference between the generated code and the one used here is the export of the constructor ConI.

end

3.2 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: NOT's backjump assumes the existence of a clause that is suitable to backjump. This clause is obtained in W's CDCL by applying:

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

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

3.2.1 Inclusion of the states

context $\textit{conflict-driven-clause-learning}_W$
begin
declare $\textit{cdcl}_W\textit{.intros}[\textit{intro}]$ $\textit{cdcl}_W\textit{-bj.intros}[\textit{intro}]$ $\textit{cdcl}_W\textit{-o.intros}[\textit{intro}]$

lemma $\textit{backtrack-no-cdcl}_W\textit{-bj}$:
  assumes *cdcl*: $\textit{cdcl}_W\textit{-bj}\ T\ U$ **and** *inv*: $\textit{cdcl}_W\textit{-M-level-inv}\ V$
  shows $\neg \textit{backtrack}\ V\ T$
  $\langle \textit{proof} \rangle$

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

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

lemma $\textit{rtranclp-cdcl}_W\textit{-bj-skip-or-resolve-backtrack}$:
  assumes $\textit{cdcl}_W\textit{-bj}^{**}\ S\ U$ **and** *inv*: $\textit{cdcl}_W\textit{-M-level-inv}\ S$
  shows $\textit{skip-or-resolve}^{**}\ S\ U \lor (\exists\, T.\ \textit{skip-or-resolve}^{**}\ S\ T \land \textit{backtrack}\ T\ U)$
  $\langle \textit{proof} \rangle$

lemma $\textit{rtranclp-skip-or-resolve-rtranclp-cdcl}_W$:
  $\textit{skip-or-resolve}^{**}\ S\ T \Longrightarrow \textit{cdcl}_W{}^{**}\ S\ T$
  $\langle \textit{proof} \rangle$

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

definition inv_{NOT} :: *'st* ⇒ *bool* **where**
$inv_{NOT} \equiv \lambda S.$ *no-dup* (*trail S*)

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

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

3.2.2 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* ∧ *cdcl*$_W$*-M-level-inv a*}
⟨*proof*⟩

lemma *cdcl*$_W$*-bj-exists-normal-form*:
assumes *lev*: *cdcl*$_W$*-M-level-inv S*
shows ∃ *T*. *full cdcl*$_W$*-bj S T*
⟨*proof*⟩
lemma *rtranclp-skip-state-decomp*:
assumes *skip*** *S T* **and** *no-dup* (*trail S*)
shows
∃ *M*. *trail S* = *M* @ *trail T* ∧ (∀ *m*∈*set M*. ¬*is-decided m*)
init-clss S = *init-clss T*
learned-clss S = *learned-clss T*
backtrack-lvl S = *backtrack-lvl T*
conflicting S = *conflicting T*
⟨*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 theorem *rtranclp-skip-backtrack-backtrack*

lemma *rtranclp-skip-backtrack-backtrack-end*:

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

3.2.3 CDCL with Merging

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

lemma *rtranclp-*$cdcl_W$*-bj-rtranclp-*$cdcl_W$:
$cdcl_W\text{-}bj^{**}\ S\ T \Longrightarrow cdcl_W^{**}\ S\ T$
⟨*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 (λ*S 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
$cdcl_W\text{-}merge\text{-}restart^{**}\ S\ T$ **and**
conflicting S = None
shows *no-step* $cdcl_W$*-bj T*
⟨*proof*⟩

If *conflicting* $S \neq None$, we cannot say anything.

Remark that this theorem does not say anything about well-foundedness: even if you know that one relation is well-founded, it only states that the normal forms are shared.

lemma *conflicting-true-full-*$cdcl_W$*-iff-full-*$cdcl_W$*-merge*:
assumes *confl*: *conflicting S = None* **and** *lev*: $cdcl_W$*-M-level-inv S*
shows *full* $cdcl_W$ $S\ V \longleftrightarrow$ *full* $cdcl_W$*-merge-restart S V*
⟨*proof*⟩

lemma *init-state-true-full-*$cdcl_W$*-iff-full-*$cdcl_W$*-merge*:
shows *full* $cdcl_W$ (*init-state N*) $V \longleftrightarrow$ *full* $cdcl_W$*-merge-restart* (*init-state N*) *V*
⟨*proof*⟩

3.2.4 CDCL with Merge and Strategy

The intermediate step

inductive $cdcl_W\text{-}s'$ :: $'st \Rightarrow 'st \Rightarrow bool$ **where**
$conflict'$: *full1* $cdcl_W\text{-}cp\ S\ S' \Longrightarrow cdcl_W\text{-}s'\ S\ S'$ |
$decide'$: $decide\ S\ S' \Longrightarrow no\text{-}step\ cdcl_W\text{-}cp\ S \Longrightarrow full\ cdcl_W\text{-}cp\ S'\ S'' \Longrightarrow cdcl_W\text{-}s'\ S\ S''$ |
bj': $full1\ cdcl_W\text{-}bj\ S\ S' \Longrightarrow no\text{-}step\ cdcl_W\text{-}cp\ S \Longrightarrow full\ cdcl_W\text{-}cp\ S'\ S'' \Longrightarrow cdcl_W\text{-}s'\ S\ S''$

inductive-cases $cdcl_W\text{-}s'E$: $cdcl_W\text{-}s'\ S\ T$

lemma *rtranclp-*$cdcl_W$*-bj-full1-cdclp-*$cdcl_W$*-stgy*:
$cdcl_W\text{-}bj^{**}\ S\ S' \Longrightarrow full\ cdcl_W\text{-}cp\ S'\ S'' \Longrightarrow cdcl_W\text{-}stgy^{**}\ S\ S''$
⟨*proof*⟩

lemma $cdcl_W\text{-}s'$*-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 $cdcl_W$*-cp T U* **and**
$cdcl_W\text{-}bj^{**}\ T\ T'$ **and**
$cdcl_W$*-all-struct-inv T* **and**
no-step $cdcl_W\text{-}bj\ T'$
shows *full* $cdcl_W\text{-}cp\ T'\ U$
$\vee$ ($\exists\ U'\ U''.\ full\ cdcl_W\text{-}cp\ T'\ U'' \wedge full1\ cdcl_W\text{-}bj\ U\ U' \wedge full\ cdcl_W\text{-}cp\ U'\ U''$
$\wedge\ cdcl_W\text{-}s'^{**}\ U\ U''$)

$\langle proof \rangle$

lemma *$cdcl_W$-cp-$cdcl_W$-bj-bissimulation'*:
assumes
full $cdcl_W$-cp T U **and**
$cdcl_W$-bj** T T' **and**
$cdcl_W$-all-struct-inv T **and**
no-step $cdcl_W$-bj T'
shows *full $cdcl_W$-cp T' U*
$\vee$ ($\exists$ *U'*. *full1 $cdcl_W$-bj U U'* $\wedge$ ($\forall$ *U''*. *full $cdcl_W$-cp U' U''* $\longrightarrow$ *full $cdcl_W$-cp T' U''*
$\wedge$ *$cdcl_W$-s'** U U''*))
$\langle proof \rangle$

lemma *$cdcl_W$-stgy-$cdcl_W$-s'-connected*:
assumes *$cdcl_W$-stgy S U* **and** *$cdcl_W$-all-struct-inv S*
shows *$cdcl_W$-s' S U*
$\vee$ ($\exists$ *U'*. *full1 $cdcl_W$-bj U U'* $\wedge$ ($\forall$ *U''*. *full $cdcl_W$-cp U' U''* $\longrightarrow$ *$cdcl_W$-s' S U''*))
$\langle proof \rangle$

lemma *$cdcl_W$-stgy-$cdcl_W$-s'-connected'*:
assumes *$cdcl_W$-stgy S U* **and** *$cdcl_W$-all-struct-inv S*
shows *$cdcl_W$-s' S U*
$\vee$ ($\exists$ *U' U''*. *$cdcl_W$-s' S U''* $\wedge$ *full1 $cdcl_W$-bj U U'* $\wedge$ *full $cdcl_W$-cp U' U''*)
$\langle proof \rangle$

lemma *$cdcl_W$-stgy-$cdcl_W$-s'-no-step*:
assumes *$cdcl_W$-stgy S U* **and** *$cdcl_W$-all-struct-inv S* **and** *no-step $cdcl_W$-bj U*
shows *$cdcl_W$-s' S U*
$\langle proof \rangle$

lemma *rtranclp-$cdcl_W$-stgy-connected-to-rtranclp-$cdcl_W$-s'*:
assumes *$cdcl_W$-stgy** S U* **and** *inv*: *$cdcl_W$-M-level-inv S*
shows *$cdcl_W$-s'** S U* $\vee$ ($\exists$ *T*. *$cdcl_W$-s'** S T* $\wedge$ *$cdcl_W$-bj^{++} T U* $\wedge$ *conflicting U* $\neq$ *None*)
$\langle proof \rangle$

lemma *n-step-$cdcl_W$-stgy-iff-no-step-$cdcl_W$-cl-$cdcl_W$-o*:
assumes *inv*: *$cdcl_W$-all-struct-inv S*
shows *no-step $cdcl_W$-s' S* $\longleftrightarrow$ *no-step $cdcl_W$-cp S* $\wedge$ *no-step $cdcl_W$-o S* (**is** *?S' S* $\longleftrightarrow$ *?C S* $\wedge$ *?O S*)
$\langle proof \rangle$

lemma *$cdcl_W$-s'-tranclp-$cdcl_W$*:
$cdcl_W$-s' S S' $\Longrightarrow$ *$cdcl_W$$^{++}$ S S'*
$\langle proof \rangle$

lemma *tranclp-$cdcl_W$-s'-tranclp-$cdcl_W$*:
$cdcl_W$-s'$^{++}$ S S' $\Longrightarrow$ *$cdcl_W$$^{++}$ S S'*
$\langle proof \rangle$

lemma *rtranclp-$cdcl_W$-s'-rtranclp-$cdcl_W$*:
$cdcl_W$-s'** S S' $\Longrightarrow$ *$cdcl_W$** S S'*
$\langle proof \rangle$

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'*)
$\langle proof \rangle$

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* **for** S :: $'st$ **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

$\langle proof \rangle$

lemma *rtranclp-cdcl$_W$-merge-cp-is-rtranclp-cdcl$_W$-s'-without-decide*:
assumes
cdcl$_W$-merge-cp** S V
conflicting S = None
shows
$(cdcl_W\text{-}s'\text{-}without\text{-}decide^{**}\ S\ V)$
$\lor (\exists\, T.\ cdcl_W\text{-}s'\text{-}without\text{-}decide^{**}\ S\ T \land propagate^{++}\ T\ V)$
$\lor (\exists\, T\ U.\ cdcl_W\text{-}s'\text{-}without\text{-}decide^{**}\ S\ T \land full1\ cdcl_W\text{-}bj\ T\ U \land propagate^{**}\ U\ V)$
$\langle proof \rangle$

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
$(cdcl_W\text{-}merge\text{-}cp^{**}\ S\ V \land conflicting\ V = None)$
$\lor (cdcl_W\text{-}merge\text{-}cp^{**}\ S\ V \land conflicting\ V \neq None \land no\text{-}step\ cdcl_W\text{-}cp\ V \land no\text{-}step\ cdcl_W\text{-}bj\ V)$
$\lor (\exists\, T.\ cdcl_W\text{-}merge\text{-}cp^{**}\ S\ T \land conflict\ T\ V)$
$\langle proof \rangle$

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*
$\langle proof \rangle$

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*
$\langle proof \rangle$

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*
$\langle proof \rangle$

lemma *no-step-cdcl$_W$-merge-cp-no-step-cdcl$_W$-cp*:
$no\text{-}step\ cdcl_W\text{-}merge\text{-}cp\ S \Longrightarrow cdcl_W\text{-}M\text{-}level\text{-}inv\ S \Longrightarrow no\text{-}step\ cdcl_W\text{-}cp\ S$
$\langle proof \rangle$

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*
$\langle proof \rangle$

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* **for** S :: *$'$st* **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 $S = T \vee$ *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

3.3 Link between Weidenbach's and NOT's CDCL

3.3.1 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) $L \longleftrightarrow$ *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 $state_W \subseteq$ *dpll-state-ops*
$\lambda S.$ *convert-trail-from-W* (*trail* S)
clauses
$\lambda L\ S.$ *cons-trail* (*convert-ann-lit-from-NOT* L) S
$\lambda S.$ *tl-trail* S
$\lambda C\ S.$ *add-learned-cls* C S
$\lambda C\ S.$ *remove-cls* C S
⟨*proof*⟩

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

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

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

thm $cdcl_{NOT}$*-merge-bj-learn-proxy.axioms*
sublocale $conflict\text{-}driven\text{-}clause\text{-}learning_W \subseteq cdcl_{NOT}$*-merge-bj-learn-proxy*
$\lambda S.$ *convert-trail-from-W* (*trail* S)
clauses
$\lambda L\ S.$ *cons-trail* (*convert-ann-lit-from-NOT* L) S
$\lambda S.$ *tl-trail* S
$\lambda C\ S.$ *add-learned-cls* C S
$\lambda C\ S.$ *remove-cls* C S

λ- -. *True*
λ- $S.$ *conflicting* S = *None*
backjump-l-cond
inv_{NOT}

⟨*proof*⟩

sublocale *conflict-driven-clause-learning*$_W$ ⊆ *cdcl*$_{NOT}$*-merge-bj-learn-proxy2*
  λS. convert-trail-from-W (*trail S*)
  clauses
  λL S. cons-trail (*convert-ann-lit-from-NOT L*) *S*
  λS. tl-trail S
  λC S. add-learned-cls C S
  λC S. remove-cls C S
  λ- -. True
  λ- S. conflicting S = *None backjump-l-cond inv*$_{NOT}$
  ⟨*proof*⟩

sublocale *conflict-driven-clause-learning*$_W$ ⊆ *cdcl*$_{NOT}$*-merge-bj-learn*
  λS. convert-trail-from-W (*trail S*)
  clauses
  λL S. cons-trail (*convert-ann-lit-from-NOT L*) *S*
  λS. tl-trail S
  λC S. add-learned-cls C S
  λC 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-eq*$_{NOT}$ (**infix** $\sim_{NOT}$ *50*)

3.3.2 Additional Lemmas between NOT and W states

lemma *trail*$_W$*-eq-reduce-trail-to*$_{NOT}$*-eq*:
  trail S = *trail T* ⟹ *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*) ⟹
  trail (*reduce-trail-to*$_{NOT}$ *M* (*add-learned-cls D S*)) = *trail* (*reduce-trail-to*$_{NOT}$ *M S*)
 ⟨*proof*⟩

lemma *reduce-trail-to*$_{NOT}$*-reduce-trail-convert*:
  reduce-trail-to$_{NOT}$ *C S* = *reduce-trail-to* (*convert-trail-from-NOT C*) *S*
  ⟨*proof*⟩

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

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

lemma *skip-or-resolve-state-change*:
  assumes *skip-or-resolve*** *S T*

shows
$\exists M.\ trail\ S = M\ @\ trail\ T \wedge (\forall m \in set\ M.\ \neg is\text{-}decided\ m)$
clauses S = *clauses* T
backtrack-lvl S = *backtrack-lvl* T
⟨*proof*⟩

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

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

lemma $cdcl_W$*-merge-is-*$cdcl_{NOT}$*-merged-bj-learn*:
assumes
inv: $cdcl_W$*-all-struct-inv* S **and**
$cdcl_W$: $cdcl_W$*-merge* S T
shows $cdcl_{NOT}$*-merged-bj-learn* S T
$\vee$ (*no-step* $cdcl_W$*-merge* $T \wedge$ *conflicting* $T \neq$ *None*)
⟨*proof*⟩

abbreviation $cdcl_{NOT}$*-restart* **where**
$cdcl_{NOT}$*-restart* $\equiv$ *restart-ops.*$cdcl_{NOT}$*-raw-restart* $cdcl_{NOT}$ *restart*

lemma $cdcl_W$*-merge-restart-is-*$cdcl_{NOT}$*-merged-bj-learn-restart-no-step*:
assumes
inv: $cdcl_W$*-all-struct-inv* S **and**
$cdcl_W$:$cdcl_W$*-merge-restart* S T
shows $cdcl_{NOT}$*-restart*** S T $\vee$ (*no-step* $cdcl_W$*-merge* $T \wedge$ *conflicting* $T \neq$ *None*)
⟨*proof*⟩

abbreviation μ_{FW} :: $'st \Rightarrow nat$ **where**
$\mu_{FW}\ S \equiv$ (*if no-step* $cdcl_W$*-merge* S *then 0 else 1+*$\mu_{CDCL}{}'$*-merged* (*set-mset* (*init-clss* S)) S)

lemma $cdcl_W$*-merge-*μ_{FW}*-decreasing*:
assumes
inv: $cdcl_W$*-all-struct-inv* S **and**
fw: $cdcl_W$*-merge* S T
shows $\mu_{FW}\ T < \mu_{FW}\ S$
⟨*proof*⟩

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

sublocale *conflict-driven-clause-learning*$_W$*-termination*
⟨*proof*⟩

3.3.4 Correctness of $cdcl_W$-merge-stgy

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**
$cdcl_W$*-all-struct-inv* R
shows *full* $cdcl_W$*-stgy* R V $\longleftrightarrow$ *full* $cdcl_W$*-merge-stgy* R V
$\langle proof \rangle$

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))
$\langle proof \rangle$
end

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

3.4 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
for
trail :: *'st* $\Rightarrow$ (*'v*, *'v clause*) *ann-lits* **and**
init-clss :: *'st* $\Rightarrow$ *'v clauses* **and**
learned-clss :: *'st* $\Rightarrow$ *'v clauses* **and**
backtrack-lvl :: *'st* $\Rightarrow$ *nat* **and**
conflicting :: *'st* $\Rightarrow$ *'v clause option* **and**

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

init-state :: *'v clauses* $\Rightarrow$ *'st* +
fixes
add-init-cls :: *'v clause* $\Rightarrow$ *'st* $\Rightarrow$ *'st*
assumes
add-init-cls:
state st = (M, N, U, S') $\Longrightarrow$
state (*add-init-cls* C *st*) = (M, $\{\#C\#\}$ + N, U, S')
begin

lemma
trail-add-init-cls[*simp*]:
trail (*add-init-cls C st*) = *trail st* **and**
init-clss-add-init-cls[*simp*]:
init-clss (*add-init-cls C st*) = {#*C*#} + *init-clss st*
and
learned-clss-add-init-cls[*simp*]:
learned-clss (*add-init-cls C st*) = *learned-clss st* **and**
backtrack-lvl-add-init-cls[*simp*]:
backtrack-lvl (*add-init-cls C st*) = *backtrack-lvl st* **and**
conflicting-add-init-cls[*simp*]:
conflicting (*add-init-cls C st*) = *conflicting st*
⟨*proof*⟩

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* ⟷ *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
— 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**
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* ⇒ *'st* ⇒ *'st* **where**
add-new-clause-and-update C S =
(*if trail S* ⊨*as CNot C*
then update-conflicting (*Some C*) (*add-init-cls C*
(*cut-trail-wrt-clause C* (*trail S*) *S*))
else add-init-cls C S)

thm *cut-trail-wrt-clause.induct*
lemma *init-clss-cut-trail-wrt-clause*[*simp*]:
  init-clss (*cut-trail-wrt-clause* C M S) = *init-clss* S
  ⟨*proof*⟩

lemma *learned-clss-cut-trail-wrt-clause*[*simp*]:
  learned-clss (*cut-trail-wrt-clause* C M S) = *learned-clss* S
  ⟨*proof*⟩

lemma *conflicting-clss-cut-trail-wrt-clause*[*simp*]:
  conflicting (*cut-trail-wrt-clause* C M S) = *conflicting* S
  ⟨*proof*⟩

lemma *trail-cut-trail-wrt-clause*:
  $\exists M$. *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 ⊨*as* *CNot* C
  shows
    (*trail* ((*cut-trail-wrt-clause* C (*trail* T) T))) ⊨*as* *CNot* C
  ⟨*proof*⟩

lemma *cut-trail-wrt-clause-hd-trail-in-or-empty-trail*:
  (($\forall L \in\# C$. $-L \notin$ *lits-of-l* (*trail* T)) ∧ *trail* (*cut-trail-wrt-clause* C (*trail* T) T) = [])
    ∨ (−*lit-of* (*hd* (*trail* (*cut-trail-wrt-clause* C (*trail* T) T))) ∈# C
      ∧ *length* (*trail* (*cut-trail-wrt-clause* C (*trail* T) T)) ≥ *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* ⇒ *'st* ⇒ *bool* **for** S **where**
add-confl:
  trail S ⊨*asm* *init-clss* S ⟹ *distinct-mset* C ⟹ *conflicting* S = *None* ⟹
   trail S ⊨*as* *CNot* C ⟹
   full $cdcl_W$*-stgy*
     (*update-conflicting* (*Some* C)
       (*add-init-cls* C (*cut-trail-wrt-clause* C (*trail* S) S))) T ⟹
   incremental-$cdcl_W$ S T |
add-no-confl:
  trail S ⊨*asm* *init-clss* S ⟹ *distinct-mset* C ⟹ *conflicting* S = *None* ⟹
   ¬*trail* S ⊨*as* *CNot* C ⟹

full $cdcl_W$*-stgy* (*add-init-cls C S*) *T* $\Longrightarrow$
incremental-$cdcl_W$ *S T*

lemma $cdcl_W$*-all-struct-inv-add-new-clause-and-update-*$cdcl_W$*-all-struct-inv*:
assumes
inv-T: $cdcl_W$*-all-struct-inv T* **and**
tr-T-N[*simp*]: *trail T* $\models$*asm N* **and**
tr-C[*simp*]: *trail T* $\models$*as CNot C* **and**
[*simp*]: *distinct-mset C*
shows $cdcl_W$*-all-struct-inv* (*add-new-clause-and-update C T*) (**is** $cdcl_W$*-all-struct-inv ?T'*)
⟨*proof*⟩

lemma $cdcl_W$*-all-struct-inv-add-new-clause-and-update-*$cdcl_W$*-stgy-inv*:
assumes
inv-s: $cdcl_W$*-stgy-invariant T* **and**
inv: $cdcl_W$*-all-struct-inv T* **and**
tr-T-N[*simp*]: *trail T* $\models$*asm N* **and**
tr-C[*simp*]: *trail T* $\models$*as CNot C* **and**
[*simp*]: *distinct-mset C*
shows $cdcl_W$*-stgy-invariant* (*add-new-clause-and-update C T*)
(**is** $cdcl_W$*-stgy-invariant ?T'*)
⟨*proof*⟩

lemma *full-*$cdcl_W$*-stgy-inv-normal-form*:
assumes
full: *full* $cdcl_W$*-stgy S T* **and**
inv-s: $cdcl_W$*-stgy-invariant S* **and**
inv: $cdcl_W$*-all-struct-inv S*
shows *conflicting T* = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss S*))
∨ *conflicting T* = *None* ∧ *trail T* $\models$*asm init-clss S* ∧ *satisfiable* (*set-mset* (*init-clss S*))
⟨*proof*⟩

lemma *incremental-*$cdcl_W$*-inv*:
assumes
inc: *incremental-*$cdcl_W$ *S T* **and**
inv: $cdcl_W$*-all-struct-inv S* **and**
s-inv: $cdcl_W$*-stgy-invariant S*
shows
$cdcl_W$*-all-struct-inv T* **and**
$cdcl_W$*-stgy-invariant T*
⟨*proof*⟩

lemma *rtranclp-incremental-*$cdcl_W$*-inv*:
assumes
inc: *incremental-*$cdcl_W{}^{**}$ *S T* **and**
inv: $cdcl_W$*-all-struct-inv S* **and**
s-inv: $cdcl_W$*-stgy-invariant S*
shows
$cdcl_W$*-all-struct-inv T* **and**
$cdcl_W$*-stgy-invariant T*
⟨*proof*⟩

lemma *incremental-conclusive-state*:
assumes
inc: *incremental-*$cdcl_W$ *S T* **and**
inv: $cdcl_W$*-all-struct-inv S* **and**

s-inv: $cdcl_W$*-stgy-invariant S*
shows *conflicting T = Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss T*))
∨ *conflicting T = None* ∧ *trail T* ⊨*asm init-clss T* ∧ *satisfiable* (*set-mset* (*init-clss T*))
⟨*proof*⟩

lemma *tranclp-incremental-correct*:
assumes
inc: *incremental-*$cdcl_W{}^{++}$ *S T* **and**
inv: $cdcl_W$*-all-struct-inv S* **and**
s-inv: $cdcl_W$*-stgy-invariant S*
shows *conflicting T = Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss T*))
∨ *conflicting T = None* ∧ *trail T* ⊨*asm init-clss T* ∧ *satisfiable* (*set-mset* (*init-clss T*))
⟨*proof*⟩

end

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

3.4.1 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
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* +
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*$\hat{}\hat{}$(*card* (*set-mset* (*learned-clss* T)) − *card* (*set-mset* (*learned-clss* S)))) S T
  $\Longrightarrow$ *card* (*set-mset* (*learned-clss* T)) − *card* (*set-mset* (*learned-clss* S)) > *f n*
  $\Longrightarrow$ *restart* T U $\Longrightarrow$ *cdcl*$_W$*-merge-with-restart* (S, n) (U, *Suc n*) |
restart-full: *full1 cdcl*$_W$*-merge-stgy* S T $\Longrightarrow$ *cdcl*$_W$*-merge-with-restart* (S, n) (T, *Suc n*)

lemma *cdcl*$_W$*-merge-with-restart* S T $\Longrightarrow$ *cdcl*$_W$*-merge-restart*** (*fst* S) (*fst* T)
  ⟨*proof*⟩

lemma *cdcl*$_W$*-merge-with-restart-rtranclp-cdcl*$_W$:
  cdcl$_W$*-merge-with-restart* S T $\Longrightarrow$ *cdcl*$_W$** (*fst* S) (*fst* T)
  ⟨*proof*⟩

lemma *cdcl*$_W$*-merge-with-restart-increasing-number*:
  cdcl$_W$*-merge-with-restart* S T $\Longrightarrow$ *snd* T = *1* + *snd* S
  ⟨*proof*⟩

lemma *full1 cdcl*$_W$*-merge-stgy* S T $\Longrightarrow$ *cdcl*$_W$*-merge-with-restart* (S, n) (T, *Suc n*)
  ⟨*proof*⟩

lemma *cdcl*$_W$*-all-struct-inv-learned-clss-bound*:
  assumes *inv*: *cdcl*$_W$*-all-struct-inv* S
  shows *set-mset* (*learned-clss* S) ⊆ *simple-clss* (*atms-of-mm* (*init-clss* S))
⟨*proof*⟩

lemma *cdcl*$_W$*-merge-with-restart-init-clss*:
  cdcl$_W$*-merge-with-restart* S T $\Longrightarrow$ *cdcl*$_W$*-M-level-inv* (*fst* S) $\Longrightarrow$
  init-clss (*fst* S) = *init-clss* (*fst* T)
  ⟨*proof*⟩

lemma
  wf {(T, S). *cdcl*$_W$*-all-struct-inv* (*fst* S) ∧ *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*$\hat{}\hat{}$(*card* (*set-mset* (*learned-clss* T)) − *card* (*set-mset* (*learned-clss* S)))) S T $\Longrightarrow$
    card (*set-mset* (*learned-clss* T)) − *card* (*set-mset* (*learned-clss* S)) > *f n* $\Longrightarrow$
    restart T U $\Longrightarrow$
   cdcl$_W$*-with-restart* (S, n) (U, *Suc n*) |
restart-full: *full1 cdcl*$_W$*-stgy* S T $\Longrightarrow$ *cdcl*$_W$*-with-restart* (S, n) (T, *Suc n*)

lemma *cdcl*$_W$*-with-restart-rtranclp-cdcl*$_W$:
  cdcl$_W$*-with-restart* S T $\Longrightarrow$ *cdcl*$_W$** (*fst* S) (*fst* T)
  ⟨*proof*⟩

lemma *cdcl*$_W$*-with-restart-increasing-number*:

$cdcl_W$*-with-restart* S $T \Longrightarrow$ *snd* $T = 1 +$ *snd* S
⟨*proof*⟩

lemma *full1* $cdcl_W$*-stgy* S $T \Longrightarrow$ $cdcl_W$*-with-restart* (S, n) $(T, Suc\ n)$
⟨*proof*⟩

lemma $cdcl_W$*-with-restart-init-clss*:
$cdcl_W$*-with-restart* S $T \Longrightarrow$ $cdcl_W$*-M-level-inv* (*fst* S) $\Longrightarrow$ *init-clss* (*fst* S) = *init-clss* (*fst* T)
⟨*proof*⟩

lemma
wf $\{(T, S)$. $cdcl_W$*-all-struct-inv* (*fst* S) $\wedge$ $cdcl_W$*-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 $\exists k$::*nat*. $(2\ \hat{}\ (k - 1) \le i \wedge i < 2\ \hat{}\ k - 1) \vee i = 2\ \hat{}\ k - 1$
⟨*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 $(2::'a)^{k-1} \le i$ and $i \le (2::'a)^k - (1::'a)$

Then the sequence is then scaled by a constant unit run (called *ur* here), strictly positive.

function *luby-sequence-core* :: *nat* ⇒ *nat* **where**
luby-sequence-core *i* =
(*if* $\exists k.\ i = 2\hat{}k - 1$
then $2\hat{}((SOME\ k.\ i = 2\hat{}k - 1) - 1)$
else luby-sequence-core $(i - 2\hat{}((SOME\ k.\ 2\hat{}(k-1) \le i \wedge i < 2\hat{}k - 1) - 1) + 1)$)
⟨*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 ^ $(k - Suc\ 0) \leq i$ **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 ^ $(k - 1) \leq i$ **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 $n \equiv$ *ur* $*$ *luby-sequence-core* n

lemma *bounded-luby-sequence*: *unbounded luby-sequence*
  ⟨*proof*⟩

lemma *luby-sequence-core-0*: *luby-sequence-core* $0 = 1$
⟨*proof*⟩

lemma *luby-sequence-core* $n \geq 1$
⟨*proof*⟩
end

locale *luby-sequence-restart* =
  luby-sequence ur +
  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
  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*
begin

sublocale $cdcl_W$*-restart* - - - - - - - - - - - - *luby-sequence*
⟨*proof*⟩

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

3.4.2 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*: $(Ms, N) \neq (Ms', N')$
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*: $DPLL\text{-}ci\ Ms\ N = (Ms,\ N)$
and *inv*: $dpll_W\text{-}all\text{-}inv\ (toS\ Ms\ N)$
shows $conclusive\text{-}dpll_W\text{-}state\ (toS\ Ms\ N)$
⟨*proof*⟩

lemma *DPLL-step-obtains*:
obtains Ms' **where** $(Ms',\ N) = DPLL\text{-}step\ (Ms,\ N)$
⟨*proof*⟩

lemma *DPLL-ci-obtains*:
obtains Ms' **where** $(Ms',\ N) = DPLL\text{-}ci\ Ms\ N$
⟨*proof*⟩

lemma *DPLL-ci-no-more-step*:
assumes *step*: $DPLL\text{-}ci\ Ms\ N = (Ms',\ N')$
shows $DPLL\text{-}ci\ Ms'\ N' = (Ms',\ N')$
⟨*proof*⟩

lemma *DPLL-part-dpll$_W$-all-inv-final*:
fixes $M\ Ms'$:: $(int,\ unit)\ ann\text{-}lits$ **and**
N :: $int\ literal\ list\ list$
assumes *inv*: $dpll_W\text{-}all\text{-}inv\ (Ms,\ mset\ (map\ mset\ N))$
and *MsN*: $DPLL\text{-}part\ Ms\ N = (Ms',\ N)$
shows $conclusive\text{-}dpll_W\text{-}state\ (toS\ Ms'\ N) \land dpll_W{}^{**}\ (toS\ Ms\ N)\ (toS\ Ms'\ N)$
⟨*proof*⟩

Embedding the invariant into the type

Defining the type **typedef** $dpll_W\text{-}state$ =
$\{(M::(int,\ unit)\ ann\text{-}lits,\ N::int\ literal\ list\ list).$
$dpll_W\text{-}all\text{-}inv\ (toS\ M\ N)\}$
morphisms *rough-state-of state-of*
⟨*proof*⟩

lemma
$DPLL\text{-}part\text{-}dom\ ([],\ N)$
⟨*proof*⟩

Some type classes **instantiation** $dpll_W\text{-}state$ :: *equal*
begin
definition $equal\text{-}dpll_W\text{-}state :: dpll_W\text{-}state \Rightarrow dpll_W\text{-}state \Rightarrow bool$ **where**
$equal\text{-}dpll_W\text{-}state\ S\ S' = (rough\text{-}state\text{-}of\ S = rough\text{-}state\text{-}of\ S')$
instance
⟨*proof*⟩
end

DPLL **definition** $DPLL\text{-}step' :: dpll_W\text{-}state \Rightarrow dpll_W\text{-}state$ **where**
$DPLL\text{-}step'\ S = state\text{-}of\ (DPLL\text{-}step\ (rough\text{-}state\text{-}of\ S))$

declare *rough-state-of-inverse*[*simp*]

lemma *DPLL-step-dpll$_W$-conc-inv*:
$DPLL\text{-}step\ (rough\text{-}state\text{-}of\ S) \in \{(M,\ N).\ dpll_W\text{-}all\text{-}inv\ (toS\ M\ N)\}$

$\langle proof \rangle$

lemma *rough-state-of-DPLL-step'-DPLL-step*[*simp*]:
rough-state-of (*DPLL-step' S*) = *DPLL-step* (*rough-state-of S*)
$\langle proof \rangle$

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')
$\langle proof \rangle$
termination
$\langle proof \rangle$

lemma [*code*]:
DPLL-tot S =
(*let S' = DPLL-step' S in*
if S' = S then S else DPLL-tot S') $\langle proof \rangle$

lemma *DPLL-tot-DPLL-step-DPLL-tot*[*simp*]: *DPLL-tot* (*DPLL-step' S*) = *DPLL-tot S*
$\langle proof \rangle$

lemma *DOPLL-step'-DPLL-tot*[*simp*]:
DPLL-step' (*DPLL-tot S*) = *DPLL-tot S*
$\langle proof \rangle$

lemma *DPLL-tot-final-state*:
assumes *DPLL-tot S = S*
shows *conclusive-dpll$_W$-state* (*toS'* (*rough-state-of S*))
$\langle proof \rangle$

lemma *DPLL-tot-star*:
assumes *rough-state-of* (*DPLL-tot S*) = *S'*
shows *dpll$_W$*** (*toS'* (*rough-state-of S*)) (*toS' S'*)
$\langle proof \rangle$

lemma *rough-state-of-rough-state-of-Nil*[*simp*]:
rough-state-of (*state-of* ([], *N*)) = ([], *N*)
$\langle proof \rangle$

Theorem of correctness

lemma *DPLL-tot-correct*:
assumes *rough-state-of* (*DPLL-tot* (*state-of* (([], *N*)))) = (*M*, *N'*)
and (*M'*, *N''*) = *toS'* (*M*, *N'*)
shows *M'* ⊨*asm N''* ⟷ *satisfiable* (*set-mset N''*)
$\langle proof \rangle$

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* ($\forall A \in$ *set N*. ($\exists a\in$*set A*. $a \in$ *lits-of-l* (M)), M))

One version of the generated SML code is here, but not included in the generated document. The only differences are:

- export *'a literal* from the SML Module *Clausal-Logic*;
- export the constructor *Con* from *DPLL-W-Implementation*;
- export the *int* constructor from *Arith*.

  All these allows to test on the code on some examples.

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

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

3.4.3 Abstract Clause Representation

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

We assume the following:

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

locale *raw-cls* $=$
fixes
mset-cls :: *'cls* $\Rightarrow$ *'v clause*
begin
end

locale *raw-ccls-union* $=$
fixes
mset-cls :: *'cls* $\Rightarrow$ *'v clause* **and**
union-cls :: *'cls* $\Rightarrow$ *'cls* $\Rightarrow$ *'cls* **and**
remove-clit :: *'v literal* $\Rightarrow$ *'cls* $\Rightarrow$ *'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* #∪ *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 ∈# *mset-clss* C **and**
in-mset-clss-exists-preimage: b ∈# *mset-clss* $C \Longrightarrow \exists b'.$ *in-clss* b' $C \wedge$ *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 $C \vee$ *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* ∈# $\lambda L\ C.\ C$ + {#L#} *remove1-mset*
    ⟨*proof*⟩

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

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

begin

3.5 Weidenbach's CDCL with Abstract Clause Representation

We first instantiate the locale of Weidenbach's locale. Then we define another abstract state: the goal of this state is to be used for implementations. We add more assumptions on the function about the state. For example *cons-trail* is restricted to undefined literals.

3.5.1 Instantiation of the Multiset Version

type-synonym $'v$ $cdcl_W$*-mset* = ($'v$, $'v$ *clause*) *ann-lit list* ×
  $'v$ *clauses* ×
  $'v$ *clauses* ×
  nat × $'v$ *clause option*

We use definition, otherwise we could not use the simplification theorems we have already shown.

definition *trail* :: $'v$ $cdcl_W$*-mset* ⇒ ($'v$, $'v$ *clause*) *ann-lit list* **where**
trail ≡ λ(M, -). M

definition *init-clss* :: $'v$ $cdcl_W$*-mset* ⇒ $'v$ *clauses* **where**
init-clss ≡ λ(-, N, -). N

definition *learned-clss* :: $'v$ $cdcl_W$*-mset* ⇒ $'v$ *clauses* **where**
learned-clss ≡ λ(-, -, U, -). U

definition *backtrack-lvl* :: $'v$ $cdcl_W$*-mset* ⇒ *nat* **where**
backtrack-lvl ≡ λ(-, -, -, k, -). k

definition *conflicting* :: $'v$ $cdcl_W$*-mset* ⇒ $'v$ *clause option* **where**

conflicting $\equiv \lambda(\text{-}, \text{-}, \text{-}, \text{-}, C).\ C$

definition *cons-trail* :: ($'v$, $'v$ *clause*) *ann-lit* $\Rightarrow$ $'v$ *cdcl*$_W$*-mset* $\Rightarrow$ $'v$ *cdcl*$_W$*-mset* **where**
cons-trail $\equiv \lambda L\ (M, R).\ (L\ \#\ M, R)$

definition *tl-trail* **where**
tl-trail $\equiv \lambda(M, R).\ (tl\ M, R)$

definition *add-learned-cls* **where**
add-learned-cls $\equiv \lambda C\ (M, N, U, R).\ (M, N, \{\#C\#\} + U, R)$

definition *remove-cls* **where**
remove-cls $\equiv \lambda C\ (M, N, U, R).\ (M,$ *removeAll-mset* $C\ N$, *removeAll-mset* $C\ U, R)$

definition *update-backtrack-lvl* **where**
update-backtrack-lvl $\equiv \lambda k\ (M, N, U, \text{-}, D).\ (M, N, U, k, D)$

definition *update-conflicting* **where**
update-conflicting $\equiv \lambda D\ (M, N, U, k, \text{-}).\ (M, N, U, k, D)$

definition *init-state* **where**
init-state $\equiv \lambda N.\ ([], N, \{\#\}, 0, None)$

lemmas *cdcl*$_W$*-mset-state* = *trail-def cons-trail-def tl-trail-def add-learned-cls-def*
remove-cls-def update-backtrack-lvl-def update-conflicting-def init-clss-def learned-clss-def
backtrack-lvl-def conflicting-def init-state-def

interpretation *cdcl*$_W$*-mset*: *state*$_W$*-ops* **where**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
backtrack-lvl = *backtrack-lvl* **and**
conflicting = *conflicting* **and**

cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-backtrack-lvl = *update-backtrack-lvl* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
⟨*proof*⟩

interpretation *cdcl*$_W$*-mset*: *state*$_W$ **where**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
backtrack-lvl = *backtrack-lvl* **and**
conflicting = *conflicting* **and**

cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-backtrack-lvl = *update-backtrack-lvl* **and**
update-conflicting = *update-conflicting* **and**

init-state = *init-state*
⟨*proof*⟩

interpretation $cdcl_W$*-mset*: *conflict-driven-clause-learning*$_W$ **where**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
backtrack-lvl = *backtrack-lvl* **and**
conflicting = *conflicting* **and**

cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-backtrack-lvl = *update-backtrack-lvl* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
⟨*proof*⟩

lemma $cdcl_W$*-mset-state-eq-eq*: $cdcl_W$*-mset.state-eq* = (*op* =)
⟨*proof*⟩

notation $cdcl_W$*-mset.state-eq* (**infix** ∼*m* *49*)

3.5.2 Abstract Relation and Relation Theorems

This locales makes the lifting from the relation defined with multiset R and the version with an abstract state *R-abs*. We are lifting many different relations (each rule and the the strategy).

locale *relation-implied-relation-abs* =
fixes
$R :: 'v\ cdcl_W\text{-}mset \Rightarrow 'v\ cdcl_W\text{-}mset \Rightarrow bool$ **and**
$R\text{-}abs :: 'st \Rightarrow 'st \Rightarrow bool$ **and**
$state :: 'st \Rightarrow 'v\ cdcl_W\text{-}mset$ **and**
$inv :: 'v\ cdcl_W\text{-}mset \Rightarrow bool$
assumes
relation-compatible-state:
$inv\ (state\ S) \Longrightarrow R\text{-}abs\ S\ T \Longrightarrow R\ (state\ S)\ (state\ T)$ **and**
relation-compatible-abs:
$\bigwedge S\ S'\ T.\ inv\ S \Longrightarrow S \sim m\ state\ S' \Longrightarrow R\ S\ T \Longrightarrow \exists\ U.\ R\text{-}abs\ S'\ U \wedge T \sim m\ state\ U$ **and**
relation-invariant:
$\bigwedge S\ T.\ R\ S\ T \Longrightarrow inv\ S \Longrightarrow inv\ T$ **and**
relation-abs-right-compatible:
$\bigwedge S\ T\ U.\ inv\ (state\ S) \Longrightarrow R\text{-}abs\ S\ T \Longrightarrow state\ T \sim m\ state\ U \Longrightarrow R\text{-}abs\ S\ U$
begin

lemma *relation-compatible-eq*:
assumes
inv: $inv\ (state\ S)$ **and**
abs: $R\text{-}abs\ S\ T$ **and**
SS': $state\ S \sim m\ state\ S'$ **and**
TT': $state\ T \sim m\ state\ T'$
shows $R\text{-}abs\ S'\ T'$
⟨*proof*⟩

lemma *rtranclp-relation-invariant*:

R^{++} S $T \Longrightarrow$ *inv* $S \Longrightarrow$ *inv* T
⟨*proof*⟩

lemma *rtranclp-abs-rtranclp*:
R-abs** S $T \Longrightarrow$ *inv* (*state* S) $\Longrightarrow R^{**}$ (*state* S) (*state* T)
⟨*proof*⟩

lemma *tranclp-relation-tranclp-relation-abs-compatible*:
fixes S :: *'st*
assumes
R: R^{++} (*state* S) T **and**
inv: *inv* (*state* S)
shows $\exists$ U. *R-abs*$^{++}$ S U $\wedge$ T $\sim$*m* *state* U
⟨*proof*⟩

lemma *rtranclp-relation-rtranclp-relation-abs-compatible*:
fixes S :: *'st*
assumes
R: R^{**} (*state* S) T **and**
inv: *inv* (*state* S)
shows $\exists$ U. *R-abs*** S U $\wedge$ T $\sim$*m* *state* U
⟨*proof*⟩

lemma *no-step-iff*:
inv (*state* S) $\Longrightarrow$ *no-step* R (*state* S) $\longleftrightarrow$ *no-step* *R-abs* S
⟨*proof*⟩

lemma *tranclp-relation-compatible-eq-and-inv*:
assumes
inv: *inv* (*state* S) **and**
st: *R-abs*$^{++}$ S T **and**
SS': *state* S $\sim$*m* *state* S' **and**
TU: *state* T $\sim$*m* *state* U
shows *R-abs*$^{++}$ S' U $\wedge$ *inv* (*state* U)
⟨*proof*⟩

lemma
assumes
inv: *inv* (*state* S) **and**
st: *R-abs*$^{++}$ S T **and**
SS': *state* S $\sim$*m* *state* S' **and**
TU: *state* T $\sim$*m* *state* U
shows
tranclp-relation-compatible-eq: *R-abs*$^{++}$ S' U **and**
tranclp-relation-abs-invariant: *inv* (*state* U)
⟨*proof*⟩

lemma *tranclp-abs-tranclp*: *R-abs*$^{++}$ S $T \Longrightarrow$ *inv* (*state* S) $\Longrightarrow R^{++}$ (*state* S) (*state* T)
⟨*proof*⟩

lemma *full1-iff*:
assumes *inv*: *inv* (*state* S)
shows *full1* R (*state* S) (*state* T) $\longleftrightarrow$ *full1* *R-abs* S T (**is** *?R* $\longleftrightarrow$ *?R-abs*)
⟨*proof*⟩

lemma *full1-iff-compatible*:
assumes *inv*: $inv\ (state\ S)$ **and** SS': $S' \sim m\ state\ S$ **and** TT': $T' \sim m\ state\ T$
shows $full1\ R\ S'\ T' \longleftrightarrow full1\ R\text{-}abs\ S\ T$ (**is** $?R \longleftrightarrow ?R\text{-}abs$)
⟨*proof*⟩

lemma *full-if-full-abs*:
assumes $inv\ (state\ S)$ **and** $full\ R\text{-}abs\ S\ T$
shows $full\ R\ (state\ S)\ (state\ T)$
⟨*proof*⟩

The converse does *not* hold, since we cannot prove that $S = T$ given $state\ S = state\ S$.

lemma *full-abs-if-full*:
assumes $inv\ (state\ S)$ **and** $full\ R\ (state\ S)\ (state\ T)$
shows $full\ R\text{-}abs\ S\ T \lor (state\ S \sim m\ state\ T \land no\text{-}step\ R\ (state\ S))$
⟨*proof*⟩

lemma *full-exists-full-abs*:
assumes *inv*: $inv\ (state\ S)$ **and** *full*: $full\ R\ (state\ S)\ T$
obtains U **where** $full\ R\text{-}abs\ S\ U$ **and** $T \sim m\ state\ U$
⟨*proof*⟩

lemma *full1-exists-full1-abs*:
assumes *inv*: $inv\ (state\ S)$ **and** *full1*: $full1\ R\ (state\ S)\ T$
obtains U **where** $full1\ R\text{-}abs\ S\ U$ **and** $T \sim m\ state\ U$
⟨*proof*⟩

lemma *full1-right-compatible*:
assumes $inv\ (state\ S)$ **and**
full1: $full1\ R\text{-}abs\ S\ T$ **and** TV: $state\ T \sim m\ state\ V$
shows $full1\ R\text{-}abs\ S\ V$
⟨*proof*⟩

lemma *full-right-compatible*:
assumes *inv*: $inv\ (state\ S)$ **and**
full-ST: $full\ R\text{-}abs\ S\ T$ **and** TU: $state\ T \sim m\ state\ U$
shows $full\ R\text{-}abs\ S\ U \lor (S = T \land no\text{-}step\ R\text{-}abs\ S)$
⟨*proof*⟩

end

locale *relation-relation-abs* =
fixes
$R :: 'v\ cdcl_W\text{-}mset \Rightarrow 'v\ cdcl_W\text{-}mset \Rightarrow bool$ **and**
$R\text{-}abs :: 'st \Rightarrow 'st \Rightarrow bool$ **and**
$state :: 'st \Rightarrow 'v\ cdcl_W\text{-}mset$ **and**
$inv :: 'v\ cdcl_W\text{-}mset \Rightarrow bool$
assumes
relation-compatible-state:
$inv\ (state\ S) \Longrightarrow R\ (state\ S)\ (state\ T) \longleftrightarrow R\text{-}abs\ S\ T$ **and**
relation-compatible-abs:
$\bigwedge S\ S'\ T.\ inv\ S \Longrightarrow S \sim m\ state\ S' \Longrightarrow R\ S\ T \Longrightarrow \exists\, U.\ R\text{-}abs\ S'\ U \land T \sim m\ state\ U$ **and**
relation-invariant:
$\bigwedge S\ T.\ R\ S\ T \Longrightarrow inv\ S \Longrightarrow inv\ T$
begin

lemma *relation-compatible-eq*:

$inv\ (state\ S) \Longrightarrow R\text{-}abs\ S\ T \Longrightarrow state\ S \sim m\ state\ S' \Longrightarrow state\ T \sim m\ state\ T' \Longrightarrow R\text{-}abs\ S'\ T'$
⟨*proof*⟩

lemma *relation-right-compatible*:
$inv\ (state\ S) \Longrightarrow R\text{-}abs\ S\ T \Longrightarrow state\ T \sim m\ state\ U \Longrightarrow R\text{-}abs\ S\ U$
⟨*proof*⟩

sublocale *relation-implied-relation-abs*
⟨*proof*⟩

end

3.5.3 The State

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.

locale $abs\text{-}state_W\text{-}ops$ =
raw-clss mset-cls
mset-clss union-clss in-clss insert-clss remove-from-clss
+
raw-ccls-union mset-ccls union-ccls remove-clit
for
— Clause
mset-cls :: *'cls* ⇒ *'v clause* **and**

— Multiset of Clauses
mset-clss :: *'clss* ⇒ *'v clauses* **and**
union-clss :: *'clss* ⇒ *'clss* ⇒ *'clss* **and**
in-clss :: *'cls* ⇒ *'clss* ⇒ *bool* **and**
insert-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
remove-from-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**

mset-ccls :: *'ccls* ⇒ *'v clause* **and**
union-ccls :: *'ccls* ⇒ *'ccls* ⇒ *'ccls* **and**
remove-clit :: *'v literal* ⇒ *'ccls* ⇒ *'ccls*
+
fixes
ccls-of-cls :: *'cls* ⇒ *'ccls* **and**
cls-of-ccls :: *'ccls* ⇒ *'cls* **and**

conc-trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **and**
hd-raw-conc-trail :: *'st* ⇒ (*'v*, *'cls*) *ann-lit* **and**
raw-conc-init-clss :: *'st* ⇒ *'clss* **and**
raw-conc-learned-clss :: *'st* ⇒ *'clss* **and**
conc-backtrack-lvl :: *'st* ⇒ *nat* **and**
raw-conc-conflicting :: *'st* ⇒ *'ccls option* **and**

cons-conc-trail :: (*'v*, *'cls*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
tl-conc-trail :: *'st* ⇒ *'st* **and**
add-conc-confl-to-learned-cls :: *'st* ⇒ *'st* **and**
remove-cls :: *'cls* ⇒ *'st* ⇒ *'st* **and**
update-conc-backtrack-lvl :: *nat* ⇒ *'st* ⇒ *'st* **and**
mark-conflicting :: *'ccls* ⇒ *'st* ⇒ *'st* **and**

$reduce\text{-}conc\text{-}trail\text{-}to :: ('v, 'v\ clause)\ ann\text{-}lits \Rightarrow 'st \Rightarrow 'st$ **and**
$resolve\text{-}conflicting :: 'v\ literal \Rightarrow 'cls \Rightarrow 'st \Rightarrow 'st$ **and**

$conc\text{-}init\text{-}state :: 'clss \Rightarrow 'st$ **and**
$restart\text{-}state :: 'st \Rightarrow 'st$
assumes
mset-ccls-ccls-of-cls[*simp*]:
$mset\text{-}ccls\ (ccls\text{-}of\text{-}cls\ C) = mset\text{-}cls\ C$ **and**
mset-cls-cls-of-ccls[*simp*]:
$mset\text{-}cls\ (cls\text{-}of\text{-}ccls\ D) = mset\text{-}ccls\ D$ **and**
ex-mset-cls: $\exists\, a.\ mset\text{-}cls\ a = E$
begin
fun $mmset\text{-}of\text{-}mlit :: ('v, 'cls)\ ann\text{-}lit \Rightarrow ('v, 'v\ clause)\ ann\text{-}lit$
where
$mmset\text{-}of\text{-}mlit\ (Propagated\ L\ C) = Propagated\ L\ (mset\text{-}cls\ C)\ |$
$mmset\text{-}of\text{-}mlit\ (Decided\ L) = Decided\ L$

lemma *lit-of-mmset-of-mlit*[*simp*]:
$lit\text{-}of\ (mmset\text{-}of\text{-}mlit\ a) = lit\text{-}of\ a$
⟨*proof*⟩

lemma *lit-of-mmset-of-mlit-set-lit-of-l*[*simp*]:
$lit\text{-}of\ `\ mmset\text{-}of\text{-}mlit\ `\ set\ M' = lits\text{-}of\text{-}l\ M'$
⟨*proof*⟩

lemma *map-mmset-of-mlit-true-annots-true-cls*[*simp*]:
$map\ mmset\text{-}of\text{-}mlit\ M' \models as\ C \longleftrightarrow M' \models as\ C$
⟨*proof*⟩

abbreviation $conc\text{-}init\text{-}clss \equiv \lambda S.\ mset\text{-}clss\ (raw\text{-}conc\text{-}init\text{-}clss\ S)$
abbreviation $conc\text{-}learned\text{-}clss \equiv \lambda S.\ mset\text{-}clss\ (raw\text{-}conc\text{-}learned\text{-}clss\ S)$
abbreviation $conc\text{-}conflicting \equiv \lambda S.\ map\text{-}option\ mset\text{-}ccls\ (raw\text{-}conc\text{-}conflicting\ S)$

notation *in-clss* (**infix** !∈! *50*)
notation *union-clss* (**infix** ⊕ *50*)
notation *insert-clss* (**infix** !++! *50*)

notation *union-ccls* (**infix** !∪ *50*)

definition $raw\text{-}clauses :: 'st \Rightarrow 'clss$ **where**
$raw\text{-}clauses\ S = union\text{-}clss\ (raw\text{-}conc\text{-}init\text{-}clss\ S)\ (raw\text{-}conc\text{-}learned\text{-}clss\ S)$

abbreviation $conc\text{-}clauses :: 'st \Rightarrow 'v\ clauses$ **where**
$conc\text{-}clauses\ S \equiv mset\text{-}clss\ (raw\text{-}clauses\ S)$

definition $state :: 'st \Rightarrow 'v\ cdcl_W\text{-}mset$ **where**
$state = (\lambda S.\ (conc\text{-}trail\ S,\ conc\text{-}init\text{-}clss\ S,\ conc\text{-}learned\text{-}clss\ S,\ conc\text{-}backtrack\text{-}lvl\ S,$
$conc\text{-}conflicting\ S))$

end

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

Weidenbach state is a five-tuple composed of:

1. the trail is a list of decided literals;
2. the initial set of clauses (that is not changed during the whole calculus);
3. the learned clauses (clauses can be added or remove);
4. the 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 (*'ccls*, standing for conflicting clause) and one for the initial and learned clauses (*'cls*, 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-raw-conc-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 *abs-state*$_W$ =
abs-state$_W$*-ops*
— functions for clauses:
mset-cls
mset-clss union-clss in-clss insert-clss remove-from-clss

— functions for the conflicting clause:
mset-ccls union-ccls remove-clit

— Conversion between conflicting and non-conflicting
ccls-of-cls cls-of-ccls

— functions about the state:
— getter:
conc-trail hd-raw-conc-trail raw-conc-init-clss raw-conc-learned-clss conc-backtrack-lvl raw-conc-conflicting
— setter:
cons-conc-trail tl-conc-trail add-conc-confl-to-learned-cls remove-cls update-conc-backtrack-lvl mark-conflicting reduce-conc-trail-to resolve-conflicting

— Some specific states:
conc-init-state
restart-state
for
mset-cls :: *'cls* ⇒ *'v clause* **and**

mset-clss :: *'clss* ⇒ *'v clauses* **and**
union-clss :: *'clss* ⇒ *'clss* ⇒ *'clss* **and**
in-clss :: *'cls* ⇒ *'clss* ⇒ *bool* **and**
insert-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
remove-from-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**

mset-ccls :: *'ccls* ⇒ *'v clause* **and**
union-ccls :: *'ccls* ⇒ *'ccls* ⇒ *'ccls* **and**
remove-clit :: *'v literal* ⇒ *'ccls* ⇒ *'ccls* **and**

ccls-of-cls :: *'cls* ⇒ *'ccls* **and** *cls-of-ccls* :: *'ccls* ⇒ *'cls* **and**

conc-trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **and**
hd-raw-conc-trail :: *'st* ⇒ (*'v*, *'cls*) *ann-lit* **and**
raw-conc-init-clss :: *'st* ⇒ *'clss* **and**
raw-conc-learned-clss :: *'st* ⇒ *'clss* **and**
conc-backtrack-lvl :: *'st* ⇒ *nat* **and**
raw-conc-conflicting :: *'st* ⇒ *'ccls option* **and**

cons-conc-trail :: (*'v*, *'cls*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
tl-conc-trail :: *'st* ⇒ *'st* **and**
add-conc-confl-to-learned-cls :: *'st* ⇒ *'st* **and**
remove-cls :: *'cls* ⇒ *'st* ⇒ *'st* **and**
update-conc-backtrack-lvl :: *nat* ⇒ *'st* ⇒ *'st* **and**
mark-conflicting :: *'ccls* ⇒ *'st* ⇒ *'st* **and**
reduce-conc-trail-to :: (*'v*, *'v clause*) *ann-lits* ⇒ *'st* ⇒ *'st* **and**
resolve-conflicting :: *'v literal* ⇒ *'cls* ⇒ *'st* ⇒ *'st* **and**

conc-init-state :: *'clss* ⇒ *'st* **and**
restart-state :: *'st* ⇒ *'st* +

assumes

— Definition of *hd-raw-trail*:
hd-raw-conc-trail:
$conc\text{-}trail\ S \neq [] \Longrightarrow mmset\text{-}of\text{-}mlit\ (hd\text{-}raw\text{-}conc\text{-}trail\ S) = hd\ (conc\text{-}trail\ S)$ **and**

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

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

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

add-conc-confl-to-learned-cls:
$no\text{-}dup\ (conc\text{-}trail\ st) \Longrightarrow state\ st = (M, N, U, k, Some\ F) \Longrightarrow$
$state\ (add\text{-}conc\text{-}confl\text{-}to\text{-}learned\text{-}cls\ st) =$
$(M, N, \{\#F\#\} + U, k, None)$ **and**

update-conc-backtrack-lvl:
$\bigwedge S'.\ state\ st = (M, N, U, k, S') \Longrightarrow$
$state\ (update\text{-}conc\text{-}backtrack\text{-}lvl\ k'\ st) = (M, N, U, k', S')$ **and**

mark-conflicting:
$state\ st = (M, N, U, k, None) \Longrightarrow$
$state\ (mark\text{-}conflicting\ E\ st) = (M, N, U, k, Some\ (mset\text{-}ccls\ E))$ **and**

conc-conflicting-mark-conflicting[*simp*]:
$raw\text{-}conc\text{-}conflicting\ (mark\text{-}conflicting\ E\ st) = Some\ E$ **and**
resolve-conflicting:
$state\ st = (M, N, U, k, Some\ F) \Longrightarrow -L' \in\#\ F \Longrightarrow L' \in\#\ mset\text{-}cls\ D \Longrightarrow$
$state\ (resolve\text{-}conflicting\ L'\ D\ st) =$

$(M, N, U, k, Some\ (cdcl_W\text{-}mset.resolve\text{-}cls\ L'\ F\ (mset\text{-}cls\ D)))$ **and**

conc-init-state:
$state\ (conc\text{-}init\text{-}state\ Ns) = ([], mset\text{-}clss\ Ns, \{\#\}, 0, None)$ **and**

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

— Properties about *reduce-conc-trail-to*:
reduce-conc-trail-to[*simp*]:
$\bigwedge S'.\ conc\text{-}trail\ st = M2\ @\ M1 \Longrightarrow state\ st = (M, S') \Longrightarrow$
$state\ (reduce\text{-}conc\text{-}trail\text{-}to\ M1\ st) = (M1, S')$
begin

lemma
— Properties about the trail *conc-trail*:
conc-trail-cons-conc-trail[*simp*]:
$undefined\text{-}lit\ (conc\text{-}trail\ st)\ (lit\text{-}of\ L) \Longrightarrow$
$conc\text{-}trail\ (cons\text{-}conc\text{-}trail\ L\ st) = mmset\text{-}of\text{-}mlit\ L\ \#\ conc\text{-}trail\ st$ **and**
conc-trail-tl-conc-trail[*simp*]:
$conc\text{-}trail\ (tl\text{-}conc\text{-}trail\ st) = tl\ (conc\text{-}trail\ st)$ **and**
conc-trail-add-conc-confl-to-learned-cls[*simp*]:
$no\text{-}dup\ (conc\text{-}trail\ st) \Longrightarrow conc\text{-}conflicting\ st \neq None \Longrightarrow$
$conc\text{-}trail\ (add\text{-}conc\text{-}confl\text{-}to\text{-}learned\text{-}cls\ st) = conc\text{-}trail\ st$ **and**
conc-trail-remove-cls[*simp*]:
$conc\text{-}trail\ (remove\text{-}cls\ C\ st) = conc\text{-}trail\ st$ **and**
conc-trail-update-conc-backtrack-lvl[*simp*]:
$conc\text{-}trail\ (update\text{-}conc\text{-}backtrack\text{-}lvl\ k\ st) = conc\text{-}trail\ st$ **and**
conc-trail-mark-conflicting[*simp*]:
$raw\text{-}conc\text{-}conflicting\ st = None \Longrightarrow conc\text{-}trail\ (mark\text{-}conflicting\ E\ st) = conc\text{-}trail\ st$ **and**
conc-trail-resolve-conflicting[*simp*]:
$conc\text{-}conflicting\ st = Some\ F \Longrightarrow -L' \in\# F \Longrightarrow L' \in\# mset\text{-}cls\ D \Longrightarrow$
$conc\text{-}trail\ (resolve\text{-}conflicting\ L'\ D\ st) = conc\text{-}trail\ st$ **and**

— Properties about the initial clauses *conc-init-clss*:
conc-init-clss-cons-conc-trail[*simp*]:
$undefined\text{-}lit\ (conc\text{-}trail\ st)\ (lit\text{-}of\ L) \Longrightarrow$
$conc\text{-}init\text{-}clss\ (cons\text{-}conc\text{-}trail\ L\ st) = conc\text{-}init\text{-}clss\ st$
and
conc-init-clss-tl-conc-trail[*simp*]:
$conc\text{-}init\text{-}clss\ (tl\text{-}conc\text{-}trail\ st) = conc\text{-}init\text{-}clss\ st$ **and**
conc-init-clss-add-conc-confl-to-learned-cls[*simp*]:
$no\text{-}dup\ (conc\text{-}trail\ st) \Longrightarrow conc\text{-}conflicting\ st \neq None \Longrightarrow$
$conc\text{-}init\text{-}clss\ (add\text{-}conc\text{-}confl\text{-}to\text{-}learned\text{-}cls\ st) = conc\text{-}init\text{-}clss\ st$ **and**
conc-init-clss-remove-cls[*simp*]:
$conc\text{-}init\text{-}clss\ (remove\text{-}cls\ C\ st) = removeAll\text{-}mset\ (mset\text{-}cls\ C)\ (conc\text{-}init\text{-}clss\ st)$ **and**
conc-init-clss-update-conc-backtrack-lvl[*simp*]:
$conc\text{-}init\text{-}clss\ (update\text{-}conc\text{-}backtrack\text{-}lvl\ k\ st) = conc\text{-}init\text{-}clss\ st$ **and**
conc-init-clss-mark-conflicting[*simp*]:
$raw\text{-}conc\text{-}conflicting\ st = None \Longrightarrow$
$conc\text{-}init\text{-}clss\ (mark\text{-}conflicting\ E\ st) = conc\text{-}init\text{-}clss\ st$ **and**

conc-init-clss-resolve-conflicting[*simp*]:
$\quad$ *conc-conflicting st* = *Some F* $\Longrightarrow -L' \in\# F \Longrightarrow L' \in\#$ *mset-cls D* $\Longrightarrow$
$\quad\quad$ *conc-init-clss* (*resolve-conflicting L' D st*) = *conc-init-clss st* **and**

— Properties about the learned clauses *conc-learned-clss*:
conc-learned-clss-cons-conc-trail[*simp*]:
$\quad$ *undefined-lit* (*conc-trail st*) (*lit-of L*) $\Longrightarrow$
$\quad\quad$ *conc-learned-clss* (*cons-conc-trail L st*) = *conc-learned-clss st* **and**
conc-learned-clss-tl-conc-trail[*simp*]:
$\quad$ *conc-learned-clss* (*tl-conc-trail st*) = *conc-learned-clss st* **and**
conc-learned-clss-add-conc-confl-to-learned-cls[*simp*]:
$\quad$ *no-dup* (*conc-trail st*) $\Longrightarrow$ *conc-conflicting st* = *Some C'* $\Longrightarrow$
$\quad\quad$ *conc-learned-clss* (*add-conc-confl-to-learned-cls st*) = $\{\#C'\#\}$ + *conc-learned-clss st* **and**
conc-learned-clss-remove-cls[*simp*]:
$\quad$ *conc-learned-clss* (*remove-cls C st*) = *removeAll-mset* (*mset-cls C*) (*conc-learned-clss st*) **and**
conc-learned-clss-update-conc-backtrack-lvl[*simp*]:
$\quad$ *conc-learned-clss* (*update-conc-backtrack-lvl k st*) = *conc-learned-clss st* **and**
conc-learned-clss-mark-conflicting[*simp*]:
$\quad$ *raw-conc-conflicting st* = *None* $\Longrightarrow$
$\quad\quad$ *conc-learned-clss* (*mark-conflicting E st*) = *conc-learned-clss st* **and**
conc-learned-clss-clss-resolve-conflicting[*simp*]:
$\quad$ *conc-conflicting st* = *Some F* $\Longrightarrow -L' \in\# F \Longrightarrow L' \in\#$ *mset-cls D* $\Longrightarrow$
$\quad\quad$ *conc-learned-clss* (*resolve-conflicting L' D st*) = *conc-learned-clss st* **and**

— Properties about the backtracking level *conc-backtrack-lvl*:
conc-backtrack-lvl-cons-conc-trail[*simp*]:
$\quad$ *undefined-lit* (*conc-trail st*) (*lit-of L*) $\Longrightarrow$
$\quad\quad$ *conc-backtrack-lvl* (*cons-conc-trail L st*) = *conc-backtrack-lvl st* **and**
conc-backtrack-lvl-tl-conc-trail[*simp*]:
$\quad$ *conc-backtrack-lvl* (*tl-conc-trail st*) = *conc-backtrack-lvl st* **and**
conc-backtrack-lvl-add-conc-confl-to-learned-cls[*simp*]:
$\quad$ *no-dup* (*conc-trail st*) $\Longrightarrow$ *conc-conflicting st* $\neq$ *None* $\Longrightarrow$
$\quad\quad$ *conc-backtrack-lvl* (*add-conc-confl-to-learned-cls st*) = *conc-backtrack-lvl st* **and**
conc-backtrack-lvl-remove-cls[*simp*]:
$\quad$ *conc-backtrack-lvl* (*remove-cls C st*) = *conc-backtrack-lvl st* **and**
conc-backtrack-lvl-update-conc-backtrack-lvl[*simp*]:
$\quad$ *conc-backtrack-lvl* (*update-conc-backtrack-lvl k st*) = *k* **and**
conc-backtrack-lvl-mark-conflicting[*simp*]:
$\quad$ *raw-conc-conflicting st* = *None* $\Longrightarrow$
$\quad\quad$ *conc-backtrack-lvl* (*mark-conflicting E st*) = *conc-backtrack-lvl st* **and**
conc-backtrack-lvl-clss-clss-resolve-conflicting[*simp*]:
$\quad$ *conc-conflicting st* = *Some F* $\Longrightarrow -L' \in\# F \Longrightarrow L' \in\#$ *mset-cls D* $\Longrightarrow$
$\quad\quad$ *conc-backtrack-lvl* (*resolve-conflicting L' D st*) = *conc-backtrack-lvl st* **and**

— Properties about the conflicting clause *conc-conflicting*:
conc-conflicting-cons-conc-trail[*simp*]:
$\quad$ *undefined-lit* (*conc-trail st*) (*lit-of L*) $\Longrightarrow$
$\quad\quad$ *conc-conflicting* (*cons-conc-trail L st*) = *conc-conflicting st* **and**
conc-conflicting-tl-conc-trail[*simp*]:
$\quad$ *conc-conflicting* (*tl-conc-trail st*) = *conc-conflicting st* **and**
conc-conflicting-add-conc-confl-to-learned-cls[*simp*]:
$\quad$ *no-dup* (*conc-trail st*) $\Longrightarrow$ *conc-conflicting st* = *Some C'* $\Longrightarrow$
$\quad\quad$ *conc-conflicting* (*add-conc-confl-to-learned-cls st*) = *None*
$\quad$ **and**
raw-conc-conflicting-add-conc-confl-to-learned-cls[*simp*]:
$\quad$ *no-dup* (*conc-trail st*) $\Longrightarrow$ *conc-conflicting st* = *Some C'* $\Longrightarrow$

$raw\text{-}conc\text{-}conflicting\ (add\text{-}conc\text{-}confl\text{-}to\text{-}learned\text{-}cls\ st) = None$ **and**
$conc\text{-}conflicting\text{-}remove\text{-}cls[simp]$:
$conc\text{-}conflicting\ (remove\text{-}cls\ C\ st) = conc\text{-}conflicting\ st$ **and**
$conc\text{-}conflicting\text{-}update\text{-}conc\text{-}backtrack\text{-}lvl[simp]$:
$conc\text{-}conflicting\ (update\text{-}conc\text{-}backtrack\text{-}lvl\ k\ st) = conc\text{-}conflicting\ st$ **and**
$conc\text{-}conflicting\text{-}clss\text{-}clss\text{-}resolve\text{-}conflicting[simp]$:
$conc\text{-}conflicting\ st = Some\ F \Longrightarrow -L' \in\#\ F \Longrightarrow L' \in\#\ mset\text{-}cls\ D \Longrightarrow$
$conc\text{-}conflicting\ (resolve\text{-}conflicting\ L'\ D\ st) =$
$Some\ (cdcl_W\text{-}mset.resolve\text{-}cls\ L'\ F\ (mset\text{-}cls\ D))$ **and**

— Properties about the initial state *conc-init-state*:
$conc\text{-}init\text{-}state\text{-}conc\text{-}trail[simp]$: $conc\text{-}trail\ (conc\text{-}init\text{-}state\ Ns) = []$ **and**
$conc\text{-}init\text{-}state\text{-}clss[simp]$: $conc\text{-}init\text{-}clss\ (conc\text{-}init\text{-}state\ Ns) = mset\text{-}clss\ Ns$ **and**
$conc\text{-}init\text{-}state\text{-}conc\text{-}learned\text{-}clss[simp]$: $conc\text{-}learned\text{-}clss\ (conc\text{-}init\text{-}state\ Ns) = \{\#\}$ **and**
$conc\text{-}init\text{-}state\text{-}conc\text{-}backtrack\text{-}lvl[simp]$: $conc\text{-}backtrack\text{-}lvl\ (conc\text{-}init\text{-}state\ Ns) = 0$ **and**
$conc\text{-}init\text{-}state\text{-}conc\text{-}conflicting[simp]$: $conc\text{-}conflicting\ (conc\text{-}init\text{-}state\ Ns) = None$ **and**

— Properties about *reduce-conc-trail-to*:
$trail\text{-}reduce\text{-}conc\text{-}trail\text{-}to[simp]$:
$conc\text{-}trail\ st = M2\ @\ M1 \Longrightarrow conc\text{-}trail\ (reduce\text{-}conc\text{-}trail\text{-}to\ M1\ st) = M1$ **and**
$conc\text{-}init\text{-}clss\text{-}reduce\text{-}conc\text{-}trail\text{-}to[simp]$:
$conc\text{-}trail\ st = M2\ @\ M1 \Longrightarrow$
$conc\text{-}init\text{-}clss\ (reduce\text{-}conc\text{-}trail\text{-}to\ M1\ st) = conc\text{-}init\text{-}clss\ st$ **and**
$conc\text{-}learned\text{-}clss\text{-}reduce\text{-}conc\text{-}trail\text{-}to[simp]$:
$conc\text{-}trail\ st = M2\ @\ M1 \Longrightarrow$
$conc\text{-}learned\text{-}clss\ (reduce\text{-}conc\text{-}trail\text{-}to\ M1\ st) = conc\text{-}learned\text{-}clss\ st$ **and**
$conc\text{-}backtrack\text{-}lvl\text{-}reduce\text{-}conc\text{-}trail\text{-}to[simp]$:
$conc\text{-}trail\ st = M2\ @\ M1 \Longrightarrow$
$conc\text{-}backtrack\text{-}lvl\ (reduce\text{-}conc\text{-}trail\text{-}to\ M1\ st) = conc\text{-}backtrack\text{-}lvl\ st$ **and**
$conc\text{-}conflicting\text{-}reduce\text{-}conc\text{-}trail\text{-}to[simp]$:
$conc\text{-}trail\ st = M2\ @\ M1 \Longrightarrow$
$conc\text{-}conflicting\ (reduce\text{-}conc\text{-}trail\text{-}to\ M1\ st) = conc\text{-}conflicting\ st$
⟨*proof*⟩

lemma
shows
$clauses\text{-}cons\text{-}conc\text{-}trail[simp]$:
$undefined\text{-}lit\ (conc\text{-}trail\ S)\ (lit\text{-}of\ L) \Longrightarrow$
$conc\text{-}clauses\ (cons\text{-}conc\text{-}trail\ L\ S) = conc\text{-}clauses\ S$ **and**

$clss\text{-}tl\text{-}conc\text{-}trail[simp]$: $conc\text{-}clauses\ (tl\text{-}conc\text{-}trail\ S) = conc\text{-}clauses\ S$ **and**
$clauses\text{-}update\text{-}conc\text{-}backtrack\text{-}lvl[simp]$:
$conc\text{-}clauses\ (update\text{-}conc\text{-}backtrack\text{-}lvl\ k\ S) = conc\text{-}clauses\ S$ **and**
$clauses\text{-}mark\text{-}conflicting[simp]$:
$raw\text{-}conc\text{-}conflicting\ S = None \Longrightarrow$
$conc\text{-}clauses\ (mark\text{-}conflicting\ D\ S) = conc\text{-}clauses\ S$ **and**
$clauses\text{-}remove\text{-}cls[simp]$:
$conc\text{-}clauses\ (remove\text{-}cls\ C\ S) = removeAll\text{-}mset\ (mset\text{-}cls\ C)\ (conc\text{-}clauses\ S)$ **and**
$clauses\text{-}add\text{-}conc\text{-}confl\text{-}to\text{-}learned\text{-}cls[simp]$:
$no\text{-}dup\ (conc\text{-}trail\ S) \Longrightarrow conc\text{-}conflicting\ S = Some\ C' \Longrightarrow$
$conc\text{-}clauses\ (add\text{-}conc\text{-}confl\text{-}to\text{-}learned\text{-}cls\ S) = \{\#C'\#\} + conc\text{-}clauses\ S$ **and**
$clauses\text{-}restart[simp]$: $conc\text{-}clauses\ (restart\text{-}state\ S) \subseteq\#\ conc\text{-}clauses\ S$ **and**
$clauses\text{-}conc\text{-}init\text{-}state[simp]$: $\bigwedge N.\ conc\text{-}clauses\ (conc\text{-}init\text{-}state\ N) = mset\text{-}clss\ N$
⟨*proof*⟩

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

abbreviation *state-eq* :: $'st \Rightarrow 'st \Rightarrow bool$ (**infix** $\sim$ *36*) **where**
$S \sim T \equiv$ *state* $S \sim m$ *state* T

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

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

lemma *clauses-reduce-conc-trail-to*[*simp*]:
  conc-trail $S = M2$ @ $M1 \Longrightarrow$ *conc-clauses* (*reduce-conc-trail-to* $M1$ S) $=$ *conc-clauses* S
  ⟨*proof*⟩

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

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

lemma *raw-conc-conflicting-update-backtracl-lvl*[*simp*]:
  raw-conc-conflicting (*update-conc-backtrack-lvl* k S) $=$ *None* $\longleftrightarrow$ *raw-conc-conflicting* $S =$ *None*
  ⟨*proof*⟩

end — end of $state_W$ locale

3.5.4 CDCL Rules

locale *abs-conflict-driven-clause-learning*$_W$ =
abs-state$_W$
— functions for clauses:
mset-cls
mset-clss union-clss in-clss insert-clss remove-from-clss

— functions for the conflicting clause:
mset-ccls union-ccls remove-clit

— conversion
ccls-of-cls cls-of-ccls

— functions for the state:
— access functions:
conc-trail hd-raw-conc-trail raw-conc-init-clss raw-conc-learned-clss conc-backtrack-lvl
raw-conc-conflicting
— changing state:
cons-conc-trail tl-conc-trail add-conc-confl-to-learned-cls remove-cls update-conc-backtrack-lvl
mark-conflicting reduce-conc-trail-to resolve-conflicting

— get state:
conc-init-state
restart-state
for
mset-cls :: $'cls \Rightarrow 'v$ *clause* **and**

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 :: $'cls \Rightarrow 'clss \Rightarrow 'clss$ **and**

mset-ccls :: $'ccls \Rightarrow 'v$ *clause* **and**
union-ccls :: $'ccls \Rightarrow 'ccls \Rightarrow 'ccls$ **and**
remove-clit :: $'v$ *literal* $\Rightarrow 'ccls \Rightarrow 'ccls$ **and**

ccls-of-cls :: $'cls \Rightarrow 'ccls$ **and**
cls-of-ccls :: $'ccls \Rightarrow 'cls$ **and**

conc-trail :: $'st \Rightarrow ('v, 'v$ *clause*$)$ *ann-lits* **and**
hd-raw-conc-trail :: $'st \Rightarrow ('v, 'cls)$ *ann-lit* **and**
raw-conc-init-clss :: $'st \Rightarrow 'clss$ **and**
raw-conc-learned-clss :: $'st \Rightarrow 'clss$ **and**
conc-backtrack-lvl :: $'st \Rightarrow$ *nat* **and**
raw-conc-conflicting :: $'st \Rightarrow 'ccls$ *option* **and**

cons-conc-trail :: $('v, 'cls)$ *ann-lit* $\Rightarrow 'st \Rightarrow 'st$ **and**
tl-conc-trail :: $'st \Rightarrow 'st$ **and**
add-conc-confl-to-learned-cls :: $'st \Rightarrow 'st$ **and**
remove-cls :: $'cls \Rightarrow 'st \Rightarrow 'st$ **and**
update-conc-backtrack-lvl :: *nat* $\Rightarrow 'st \Rightarrow 'st$ **and**
mark-conflicting :: $'ccls \Rightarrow 'st \Rightarrow 'st$ **and**

reduce-conc-trail-to :: ($'v$, $'v$ *clause*) *ann-lits* ⇒ $'st$ ⇒ $'st$ **and**
resolve-conflicting :: $'v$ *literal* ⇒ $'cls$ ⇒ $'st$ ⇒ $'st$ **and**

conc-init-state :: $'clss$ ⇒ $'st$ **and**
restart-state :: $'st$ ⇒ $'st$
begin

lemma *clauses-state-conc-clauses*[*simp*]: *cdcl*$_W$*-mset.clauses* (*state S*) = *conc-clauses S*
⟨*proof*⟩

lemma *conflicting-None-iff-raw-conc-conflicting*[*simp*]:
conflicting (*state S*) = *None* ⟷ *raw-conc-conflicting S* = *None*
⟨*proof*⟩

lemma *trail-state-add-conc-confl-to-learned-cls*:
no-dup (*conc-trail S*) ⟹ *conc-conflicting S* ≠ *None* ⟹
trail (*state* (*add-conc-confl-to-learned-cls S*)) = *trail* (*state S*)
⟨*proof*⟩

lemma *trail-state-update-backtrack-lvl*:
trail (*state* (*update-conc-backtrack-lvl i S*)) = *trail* (*state S*)
⟨*proof*⟩

lemma *trail-state-update-conflicting*:
raw-conc-conflicting S = *None* ⟹ *trail* (*state* (*mark-conflicting i S*)) = *trail* (*state S*)
⟨*proof*⟩

lemma *trail-state-conc-trail*[*simp*]:
trail (*state S*) = *conc-trail S*
⟨*proof*⟩

lemma *init-clss-state-conc-init-clss*[*simp*]:
init-clss (*state S*) = *conc-init-clss S*
⟨*proof*⟩

lemma *learned-clss-state-conc-learned-clss*[*simp*]:
learned-clss (*state S*) = *conc-learned-clss S*
⟨*proof*⟩

lemma *tl-trail-state-tl-con-trail*[*simp*]:
tl-trail (*state S*) = *state* (*tl-conc-trail S*)
⟨*proof*⟩

lemma *add-learned-cls-state-add-conc-confl-to-learned-cls*[*simp*]:
assumes *no-dup* (*conc-trail S*) **and** *raw-conc-conflicting S* = *Some D*
shows *update-conflicting None* (*add-learned-cls* (*mset-ccls D*) (*state S*)) =
state (*add-conc-confl-to-learned-cls S*)
⟨*proof*⟩

lemma *state-cons-cons-trail-cons-trail*[*simp*]:
undefined-lit (*trail* (*state S*)) (*lit-of L*) ⟹
cons-trail (*mmset-of-mlit L*) (*state S*) = *state* (*cons-conc-trail L S*)
⟨*proof*⟩

lemma *state-cons-cons-trail-cons-trail-propagated*[*simp*]:
undefined-lit (*trail* (*state S*)) *K* ⟹

cons-trail (*Propagated* K (*mset-cls* C)) (*state* S) = *state* (*cons-conc-trail* (*Propagated* K C) S)
⟨*proof*⟩

lemma *state-cons-cons-trail-cons-trail-propagated-ccls*[*simp*]:
undefined-lit (*trail* (*state* S)) $K \Longrightarrow$
cons-trail (*Propagated* K (*mset-ccls* C)) (*state* S) =
state (*cons-conc-trail* (*Propagated* K (*cls-of-ccls* C)) S)
⟨*proof*⟩

lemma *state-cons-cons-trail-cons-trail-decided*[*simp*]:
undefined-lit (*trail* (*state* S)) $K \Longrightarrow$
cons-trail (*Decided* K) (*state* S) = *state* (*cons-conc-trail* (*Decided* K) S)
⟨*proof*⟩

lemma *state-mark-conflicting-update-conflicting*[*simp*]:
assumes *raw-conc-conflicting* S = *None*
shows
update-conflicting (*Some* (*mset-ccls* D)) (*state* S) = *state* (*mark-conflicting* D S)
update-conflicting (*Some* (*mset-cls* D')) (*state* S) =
state (*mark-conflicting* ((*ccls-of-cls* D')) S)
⟨*proof*⟩

lemma *update-backtrack-lvl-state*[*simp*]:
update-backtrack-lvl i (*state* S) = *state* (*update-conc-backtrack-lvl* i S)
⟨*proof*⟩

lemma *conc-conflicting-conflicting*[*simp*]:
conflicting (*state* S) = *conc-conflicting* S
⟨*proof*⟩

lemma *update-conflicting-resolve-state-mark-conflicting*[*simp*]:
raw-conc-conflicting S = *Some* $D' \Longrightarrow -L \in\#$ *mset-ccls* $D' \Longrightarrow L \in\#$ *mset-cls* $E' \Longrightarrow$
update-conflicting (*Some* (*remove1-mset* ($-$ L) (*mset-ccls* D') $\#\cup$ *remove1-mset* L (*mset-cls* E')))
(*state* (*tl-conc-trail* S)) =
state (*resolve-conflicting* L E' (*tl-conc-trail* S))
⟨*proof*⟩

lemma *add-learned-update-backtrack-update-conflicting*[*simp*]:
no-dup (*conc-trail* S) $\Longrightarrow$ *raw-conc-conflicting* S = *Some* $D' \Longrightarrow$ *add-learned-cls* (*mset-ccls* D')
(*update-backtrack-lvl* i
(*update-conflicting* *None*
(*state* S))) =
state (*add-conc-confl-to-learned-cls* (*update-conc-backtrack-lvl* i S))
⟨*proof*⟩

lemma *conc-backtrack-lvl-backtrack-lvl*[*simp*]:
backtrack-lvl (*state* S) = *conc-backtrack-lvl* S
⟨*proof*⟩

lemma *state-state*:
$cdcl_W$*-mset.state* (*state* S) = (*trail* (*state* S), *init-clss* (*state* S), *learned-clss* (*state* S),
backtrack-lvl (*state* S), *conflicting* (*state* S))
⟨*proof*⟩

lemma *state-reduce-conc-trail-to-reduce-conc-trail-to*[*simp*]:
assumes [*simp*]: *conc-trail* S = *M2* @ *M1*

shows $cdcl_W$*-mset.reduce-trail-to M1* (*state S*) = *state* (*reduce-conc-trail-to M1 S*) (**is** *?RS* = *?SR*)
⟨*proof*⟩

lemma *state-conc-init-state*: *state* (*conc-init-state N*) = *init-state* (*mset-clss N*)
⟨*proof*⟩

More robust version of *in-mset-clss-exists-preimage*:

lemma *in-clauses-preimage*:
assumes *b*: *b* ∈# $cdcl_W$*-mset.clauses* (*state C*)
shows ∃ *b'*. *b'* !∈! *raw-clauses C* ∧ *mset-cls b'* = *b*
⟨*proof*⟩

lemma *state-reduce-conc-trail-to-reduce-conc-trail-to-decomp*[*simp*]:
assumes (*P* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*conc-trail S*))
shows $cdcl_W$*-mset.reduce-trail-to M1* (*state S*) = *state* (*reduce-conc-trail-to M1 S*)
⟨*proof*⟩

inductive *propagate-abs* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
propagate-abs-rule: *conc-conflicting S* = *None* ⟹
E !∈! *raw-clauses S* ⟹
L ∈# *mset-cls E* ⟹
conc-trail S ⊨*as CNot* (*mset-cls E* − {#*L*#}) ⟹
undefined-lit (*conc-trail S*) *L* ⟹
T ∼ *cons-conc-trail* (*Propagated L E*) *S* ⟹
propagate-abs S T

inductive-cases *propagate-absE*: *propagate-abs S T*

lemma *propagate-propagate-abs*:
$cdcl_W$*-mset.propagate* (*state S*) (*state T*) ⟷ *propagate-abs S T* (**is** *?mset* ⟷ *?abs*)
⟨*proof*⟩

lemma *propagate-compatible-abs*:
assumes *SS'*: *S* ∼*m state S'* **and** *abs*: $cdcl_W$*-mset.propagate S T*
obtains *U* **where** *propagate-abs S' U* **and** *T* ∼*m state U*
⟨*proof*⟩

interpretation *propagate-abs*: *relation-relation-abs* $cdcl_W$*-mset.propagate propagate-abs state*
λ-. *True*
⟨*proof*⟩

inductive *conflict-abs* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
conflict-abs-rule:
conc-conflicting S = *None* ⟹
D !∈! *raw-clauses S* ⟹
conc-trail S ⊨*as CNot* (*mset-cls D*) ⟹
T ∼ *mark-conflicting* (*ccls-of-cls D*) *S* ⟹
conflict-abs S T

inductive-cases *conflict-absE*: *conflict-abs S T*

lemma *conflict-conflict-abs*:
$cdcl_W$*-mset.conflict* (*state S*) (*state T*) ⟷ *conflict-abs S T* (**is** *?mset* ⟷ *?abs*)
⟨*proof*⟩

lemma *conflict-compatible-abs*:

assumes *SS'*: $S \sim m$ *state* S' **and** *conflict*: $cdcl_W$*-mset.conflict* S T
obtains U **where** *conflict-abs* S' U **and** $T \sim m$ *state* U
⟨*proof*⟩

interpretation *conflict-abs*: *relation-relation-abs* $cdcl_W$*-mset.conflict conflict-abs state*
λ-. *True*
⟨*proof*⟩

inductive *backtrack-abs* :: *'st* ⇒ *'st* ⇒ *bool* **for** S :: *'st* **where**
backtrack-abs-rule:
raw-conc-conflicting S = *Some* D ⟹
L ∈# *mset-ccls* D ⟹
(*Decided* K # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*conc-trail* S)) ⟹
get-level (*conc-trail* S) L = *conc-backtrack-lvl* S ⟹
get-level (*conc-trail* S) L = *get-maximum-level* (*conc-trail* S) (*mset-ccls* D) ⟹
get-maximum-level (*conc-trail* S) (*mset-ccls* D − {#L#}) ≡ i ⟹
get-level (*conc-trail* S) K = i + *1* ⟹
T ∼ *cons-conc-trail* (*Propagated* L (*cls-of-ccls* D))
(*reduce-conc-trail-to M1*
(*add-conc-confl-to-learned-cls*
(*update-conc-backtrack-lvl* i S))) ⟹
backtrack-abs S T

inductive-cases *backtrack-absE*: *backtrack-abs* S T

lemma *backtrack-backtrack-abs*:
assumes *inv*: $cdcl_W$*-mset.*$cdcl_W$*-all-struct-inv* (*state* S)
shows $cdcl_W$*-mset.backtrack* (*state* S) (*state* T) ⟷ *backtrack-abs* S T (**is** *?conc* ⟷ *?abs*)
⟨*proof*⟩

lemma *backtrack-exists-backtrack-abs-step*:
assumes *bt*: $cdcl_W$*-mset.backtrack* S T **and** *inv*: $cdcl_W$*-mset.*$cdcl_W$*-all-struct-inv* S **and**
SS': $S \sim m$ *state* S'
obtains U **where** *backtrack-abs* S' U **and** $T \sim m$ *state* U
⟨*proof*⟩

interpretation *backtrack-abs*: *relation-relation-abs* $cdcl_W$*-mset.backtrack backtrack-abs state*
$cdcl_W$*-mset.*$cdcl_W$*-all-struct-inv*
⟨*proof*⟩

inductive *decide-abs* :: *'st* ⇒ *'st* ⇒ *bool* **for** S :: *'st* **where**
decide-abs-rule:
conc-conflicting S = *None* ⟹
undefined-lit (*conc-trail* S) L ⟹
atm-of L ∈ *atms-of-mm* (*conc-init-clss* S) ⟹
T ∼ *cons-conc-trail* (*Decided* L) (*incr-lvl* S) ⟹
decide-abs S T

inductive-cases *decide-absE*: *decide-abs* S T

lemma *decide-decide-abs*:
$cdcl_W$*-mset.decide* (*state* S) (*state* T) ⟷ *decide-abs* S T
⟨*proof*⟩

interpretation *decide-abs*: *relation-relation-abs* $cdcl_W$*-mset.decide decide-abs state*
λ-. *True*

⟨*proof*⟩

inductive *skip-abs* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
skip-abs-rule:
conc-trail S = *Propagated L C'* # *M* ⟹
raw-conc-conflicting S = *Some E* ⟹
−*L* ∉# *mset-ccls E* ⟹
mset-ccls E ≠ {#} ⟹
T ∼ *tl-conc-trail S* ⟹
skip-abs S T

inductive-cases *skip-absE*: *skip-abs S T*

lemma *skip-skip-abs*:
$cdcl_W$*-mset.skip* (*state S*) (*state T*) ⟷ *skip-abs S T* (**is** *?conc* ⟷ *?abs*)
⟨*proof*⟩

lemma *skip-exists-skip-abs*:
assumes *skip*: $cdcl_W$*-mset.skip S T* **and** *SS'*: *S* ∼*m state S'*
obtains *U* **where** *skip-abs S' U* **and** *T* ∼*m state U*
⟨*proof*⟩

interpretation *skip-abs*: *relation-relation-abs* $cdcl_W$*-mset.skip skip-abs state*
λ-. True
⟨*proof*⟩

inductive *resolve-abs* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
resolve-abs-rule: *conc-trail S* ≠ [] ⟹
hd-raw-conc-trail S = *Propagated L E* ⟹
L ∈# *mset-cls E* ⟹
raw-conc-conflicting S = *Some D'* ⟹
−*L* ∈# *mset-ccls D'* ⟹
get-maximum-level (*conc-trail S*) (*mset-ccls* (*remove-clit* (−*L*) *D'*)) = *conc-backtrack-lvl S* ⟹
T ∼ *resolve-conflicting L E* (*tl-conc-trail S*) ⟹
resolve-abs S T

inductive-cases *resolve-absE*: *resolve-abs S T*

lemma *resolve-resolve-abs*:
$cdcl_W$*-mset.resolve* (*state S*) (*state T*) ⟷ *resolve-abs S T* (**is** *?conc* ⟷ *?abs*)
⟨*proof*⟩

lemma *resolve-exists-resolve-abs*:
assumes
res: $cdcl_W$*-mset.resolve S T* **and**
SS': *S* ∼*m state S'*
obtains *U* **where** *resolve-abs S' U* **and** *T* ∼*m state U*
⟨*proof*⟩

interpretation *resolve-abs*: *relation-relation-abs* $cdcl_W$*-mset.resolve resolve-abs state*
λ-. True
⟨*proof*⟩

inductive *restart* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
restart: *conc-conflicting S* = *None* ⟹
¬*conc-trail S* ⊨*asm conc-clauses S* ⟹

$T \sim$ *restart-state* $S \Longrightarrow$
restart S T

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

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

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

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

inductive *cdcl$_W$-abs-rf* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **for** S :: *'st* **where**
restart: *restart-abs* S $T \Longrightarrow$ *cdcl$_W$-abs-rf* S T |
forget: *forget-abs* S $T \Longrightarrow$ *cdcl$_W$-abs-rf* S T

inductive *cdcl$_W$-abs-bj* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
skip: *skip-abs* S $S' \Longrightarrow$ *cdcl$_W$-abs-bj* S S' |
resolve: *resolve-abs* S $S' \Longrightarrow$ *cdcl$_W$-abs-bj* S S' |
backtrack: *backtrack-abs* S $S' \Longrightarrow$ *cdcl$_W$-abs-bj* S S'

inductive-cases *cdcl$_W$-abs-bjE*: *cdcl$_W$-abs-bj* S T

lemma *cdcl$_W$-abs-bj-cdcl$_W$-abs-bj*:
cdcl$_W$-mset.cdcl$_W$-all-struct-inv (*state* S) $\Longrightarrow$
cdcl$_W$-mset.cdcl$_W$-bj (*state* S) (*state* T) $\longleftrightarrow$ *cdcl$_W$-abs-bj* S T
⟨*proof*⟩

interpretation *cdcl$_W$-abs-bj*: *relation-relation-abs cdcl$_W$-mset.cdcl$_W$-bj cdcl$_W$-abs-bj state cdcl$_W$-mset.cdcl$_W$-all-struct-inv*
⟨*proof*⟩

inductive *cdcl$_W$-abs-o* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **for** S :: *'st* **where**
decide: *decide-abs* S $S' \Longrightarrow$ *cdcl$_W$-abs-o* S S' |
bj: *cdcl$_W$-abs-bj* S $S' \Longrightarrow$ *cdcl$_W$-abs-o* S S'

inductive *cdcl$_W$-abs* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **for** S :: *'st* **where**
propagate: *propagate-abs* S $S' \Longrightarrow$ *cdcl$_W$-abs* S S' |
conflict: *conflict-abs* S $S' \Longrightarrow$ *cdcl$_W$-abs* S S' |
other: *cdcl$_W$-abs-o* S $S' \Longrightarrow$ *cdcl$_W$-abs* S S'|
rf: *cdcl$_W$-abs-rf* S $S' \Longrightarrow$ *cdcl$_W$-abs* S S'

3.5.5 Higher level strategy

The rules described previously do not lead to a conclusive state. We have add a strategy and show the inclusion in the multiset version.

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

lemma *cdcl$_W$-merge-cp-cdcl$_W$-abs-merge-cp*:
assumes
cp: *cdcl$_W$-merge-abs-cp S T* **and**
inv: *cdcl$_W$-mset.cdcl$_W$-all-struct-inv* (*state S*)
shows *cdcl$_W$-mset.cdcl$_W$-merge-cp* (*state S*) (*state T*)
⟨*proof*⟩

lemma *cdcl$_W$-merge-cp-abs-exists-cdcl$_W$-merge-cp*:
assumes
cp: *cdcl$_W$-mset.cdcl$_W$-merge-cp* (*state S*) *T* **and**
inv: *cdcl$_W$-mset.cdcl$_W$-all-struct-inv* (*state S*)
obtains *U* **where** *cdcl$_W$-merge-abs-cp S U* **and** *T* ∼*m state U*
⟨*proof*⟩

lemma *no-step-cdcl$_W$-merge-cp-no-step-cdcl$_W$-abs-merge-cp*:
assumes
inv: *cdcl$_W$-mset.cdcl$_W$-all-struct-inv* (*state S*)
shows *no-step cdcl$_W$-merge-abs-cp S* ⟷ *no-step cdcl$_W$-mset.cdcl$_W$-merge-cp* (*state S*)
(**is** *?abs* ⟷ *?conc*)
⟨*proof*⟩

lemma *cdcl$_W$-merge-abs-cp-right-compatible*:
cdcl$_W$-merge-abs-cp S V ⟹ *cdcl$_W$-mset.cdcl$_W$-all-struct-inv* (*state S*) ⟹
V ∼ *W* ⟹ *cdcl$_W$-merge-abs-cp S W*
⟨*proof*⟩

interpretation *cdcl$_W$-merge-abs-cp*: *relation-implied-relation-abs*
cdcl$_W$-mset.cdcl$_W$-merge-cp cdcl$_W$-merge-abs-cp state cdcl$_W$-mset.cdcl$_W$-all-struct-inv
⟨*proof*⟩

inductive *cdcl$_W$-merge-abs-stgy* **for** *S* :: *'st* **where**
fw-s-cp: *full1 cdcl$_W$-merge-abs-cp S T* ⟹ *cdcl$_W$-merge-abs-stgy S T* |
fw-s-decide: *decide-abs S T* ⟹ *no-step cdcl$_W$-merge-abs-cp S* ⟹ *full cdcl$_W$-merge-abs-cp T U*
⟹ *cdcl$_W$-merge-abs-stgy S U*

lemma *cdcl$_W$-cp-cdcl$_W$-abs-cp*:
assumes *stgy*: *cdcl$_W$-merge-abs-stgy S T* **and**
inv: *cdcl$_W$-mset.cdcl$_W$-all-struct-inv* (*state S*)
shows *cdcl$_W$-mset.cdcl$_W$-merge-stgy* (*state S*) (*state T*)
⟨*proof*⟩

lemma *cdcl$_W$-merge-abs-stgy-exists-cdcl$_W$-merge-stgy*:
assumes
inv: *cdcl$_W$-mset.cdcl$_W$-all-struct-inv S* **and**
SS': *S* ∼*m state S'* **and**
st: *cdcl$_W$-mset.cdcl$_W$-merge-stgy S T*
shows ∃ *U*. *cdcl$_W$-merge-abs-stgy S' U* ∧ *T* ∼*m state U*
⟨*proof*⟩

lemma *cdcl$_W$-merge-abs-stgy-right-compatible*:
assumes
inv: *cdcl$_W$-mset.cdcl$_W$-all-struct-inv* (*state S*) **and**
st: *cdcl$_W$-merge-abs-stgy S T* **and**
TU: *T* ∼ *V*

shows *cdcl*$_W$*-merge-abs-stgy S V*
⟨*proof*⟩

interpretation *cdcl*$_W$*-merge-abs-stgy*: *relation-implied-relation-abs*
cdcl$_W$*-mset.cdcl*$_W$*-merge-stgy cdcl*$_W$*-merge-abs-stgy state cdcl*$_W$*-mset.cdcl*$_W$*-all-struct-inv*
⟨*proof*⟩

lemma *cdcl*$_W$*-merge-abs-stgy-final-State-conclusive*:
fixes *T* :: *'st*
assumes
full: *full cdcl*$_W$*-merge-abs-stgy* (*conc-init-state N*) *T* **and**
n-d: *distinct-mset-mset* (*mset-clss N*)
shows (*conc-conflicting T* = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*mset-clss N*)))
∨ (*conc-conflicting T* = *None* ∧ *conc-trail T* ⊨*asm mset-clss N*
∧ *satisfiable* (*set-mset* (*mset-clss N*)))
⟨*proof*⟩

end

end

3.6 2-Watched-Literal

theory *CDCL-Two-Watched-Literals*
imports *CDCL-W-Abstract-State*
begin

First we define here the core of the two-watched literal data structure:

1. A clause is composed of (at most) two watched literals.
2. It is sufficient to find the candidates for propagation and conflict from the clauses such that the new literal is watched.

While this it the principle behind the two-watched literals, an implementation have to remember the candidates that have been found so far while updating the data structure.

We will directly on the two-watched literals data structure with lists: it could be also seen as a state over some abstract clause representation we would later refine as lists. However, as we need a way to select element from a clause, working on lists is better.

3.6.1 Essence of 2-WL

Data structure and Access Functions

Only the 2-watched literals have to be verified here: the backtrack level and the trail that appear in the state are not related to the 2-watched algoritm.

datatype *'v twl-clause* =
TWL-Clause (*watched*: *'v literal list*) (*unwatched*: *'v literal list*)

datatype *'v twl-state* =
TWL-State (*raw-trail*: (*'v*, *'v twl-clause*) *ann-lits*)
(*raw-init-clss*: *'v twl-clause list*)
(*raw-learned-clss*: *'v twl-clause list*) (*backtrack-lvl*: *nat*)

($raw\text{-}conflicting$: $'v$ $literal$ $list$ $option$)

fun $mmset\text{-}of\text{-}mlit$ :: $('v, 'v\ twl\text{-}clause)\ ann\text{-}lit \Rightarrow ('v, 'v\ clause)\ ann\text{-}lit$
where
$mmset\text{-}of\text{-}mlit\ (Propagated\ L\ C) = Propagated\ L\ (mset\ (watched\ C\ @\ unwatched\ C))\ |$
$mmset\text{-}of\text{-}mlit\ (Decided\ L) = Decided\ L$

lemma $lit\text{-}of\text{-}mmset\text{-}of\text{-}mlit[simp]$: $lit\text{-}of\ (mmset\text{-}of\text{-}mlit\ x) = lit\text{-}of\ x$
⟨proof⟩

lemma $lits\text{-}of\text{-}mmset\text{-}of\text{-}mlit[simp]$: $lits\text{-}of\ (mmset\text{-}of\text{-}mlit\ `\ S) = lits\text{-}of\ S$
⟨proof⟩

abbreviation $trail$ **where**
$trail\ S \equiv map\ mmset\text{-}of\text{-}mlit\ (raw\text{-}trail\ S)$

abbreviation $clauses\text{-}of\text{-}l$ **where**
$clauses\text{-}of\text{-}l \equiv \lambda L.\ mset\ (map\ mset\ L)$

definition $raw\text{-}clause$ :: $'v\ twl\text{-}clause \Rightarrow 'v\ literal\ list$ **where**
$raw\text{-}clause\ C \equiv watched\ C\ @\ unwatched\ C$

definition $clause$ :: $'v\ twl\text{-}clause \Rightarrow 'v\ clause$ **where**
$clause\ C \equiv mset\ (raw\text{-}clause\ C)$

lemma $clause\text{-}def\text{-}lambda$:
$clause = (\lambda C.\ mset\ (raw\text{-}clause\ C))$
⟨proof⟩

abbreviation $raw\text{-}clss$ :: $'v\ twl\text{-}state \Rightarrow 'v\ clauses$ **where**
$raw\text{-}clss\ S \equiv mset\ (map\ clause\ (raw\text{-}init\text{-}clss\ S\ @\ raw\text{-}learned\text{-}clss\ S))$

abbreviation $raw\text{-}clss\text{-}l$ :: $'a\ twl\text{-}clause\ list \Rightarrow 'a\ literal\ multiset\ multiset$ **where**
$raw\text{-}clss\text{-}l\ C \equiv mset\ (map\ clause\ C)$

interpretation $raw\text{-}cls\ clause$ ⟨proof⟩

lemma $mset\text{-}map\text{-}clause\text{-}remove1\text{-}cond$:
$mset\ (map\ (\lambda x.\ mset\ (unwatched\ x) + mset\ (watched\ x))$
$(remove1\text{-}cond\ (\lambda D.\ clause\ D = clause\ a)\ Cs)) =$
$remove1\text{-}mset\ (clause\ a)\ (mset\ (map\ clause\ Cs))$
⟨proof⟩

interpretation $raw\text{-}clss$
$clause$
$raw\text{-}clss\text{-}l\ op\ @$
$\lambda L\ C.\ L \in set\ C\ op\ \#\ \lambda C.\ remove1\text{-}cond\ (\lambda D.\ clause\ D = clause\ C)$
⟨proof⟩

lemma $ex\text{-}mset\text{-}unwatched\text{-}watched$:
$\exists\, a.\ mset\ (unwatched\ a) + mset\ (watched\ a) = E$
⟨proof⟩

interpretation twl: $abs\text{-}state_W\text{-}ops$
$clause$
$raw\text{-}clss\text{-}l\ op\ @$

$\lambda L\ C.\ L \in set\ C\ op\ \#\ \lambda C.\ remove1\text{-}cond\ (\lambda D.\ clause\ D = clause\ C)$

$mset\ \lambda xs\ ys.\ case\text{-}prod\ append\ (fold\ (\lambda x\ (ys,\ zs).\ (remove1\ x\ ys,\ x\ \#\ zs))\ xs\ (ys,\ []))$
remove1

$raw\text{-}clause\ \lambda C.\ TWL\text{-}Clause\ []\ C$
$trail\ \lambda S.\ hd\ (raw\text{-}trail\ S)$
raw-init-clss raw-learned-clss backtrack-lvl raw-conflicting
rewrites
$twl.mmset\text{-}of\text{-}mlit = mmset\text{-}of\text{-}mlit$
$\langle proof \rangle$

declare *CDCL-Two-Watched-Literals.twl.mset-ccls-ccls-of-cls*[*simp del*]

definition
$candidates\text{-}propagate :: 'v\ twl\text{-}state \Rightarrow ('v\ literal \times 'v\ twl\text{-}clause)\ set$
where
$candidates\text{-}propagate\ S =$
$\{(L,\ C) \mid L\ C.$
$C \in set\ (twl.raw\text{-}clauses\ S)\ \wedge$
$set\ (watched\ C) - (uminus\ `\ lits\text{-}of\text{-}l\ (trail\ S)) = \{L\}\ \wedge$
$undefined\text{-}lit\ (raw\text{-}trail\ S)\ L\}$

definition $candidates\text{-}conflict :: 'v\ twl\text{-}state \Rightarrow 'v\ twl\text{-}clause\ set$ **where**
$candidates\text{-}conflict\ S =$
$\{C.\ C \in set\ (twl.raw\text{-}clauses\ S)\ \wedge$
$set\ (watched\ C) \subseteq uminus\ `\ lits\text{-}of\text{-}l\ (raw\text{-}trail\ S)\}$

primrec (*nonexhaustive*) $index :: 'a\ list \Rightarrow 'a \Rightarrow nat$ **where**
$index\ (a\ \#\ l)\ c = (if\ a = c\ then\ 0\ else\ 1{+}index\ l\ c)$

lemma *index-nth*:
$a \in set\ l \Longrightarrow l\ !\ (index\ l\ a) = a$
$\langle proof \rangle$

Invariants

The structural invariants states that there are at most two watched elements, that the watched literals are distinct, and that there are 2 watched literals if there are at least than two different literals in the full clauses.

primrec $struct\text{-}wf\text{-}twl\text{-}cls :: 'v\ twl\text{-}clause \Rightarrow bool$ **where**
$struct\text{-}wf\text{-}twl\text{-}cls\ (TWL\text{-}Clause\ W\ UW) \longleftrightarrow$
$distinct\ W \wedge length\ W \le 2 \wedge (length\ W < 2 \longrightarrow set\ UW \subseteq set\ W)$

We need the following property about updates: if there is a literal L with $-\ L$ in the trail, and L is not watched, then it stays unwatched; i.e., while updating with *rewatch*, L does not get swapped with a watched literal L' such that $-\ L'$ is in the trail. This corresponds to the laziness of the data structure.

Remark that M is a trail: literals at the end were the first to be added to the trail.

primrec $watched\text{-}only\text{-}lazy\text{-}updates :: ('v,\ 'mark)\ ann\text{-}lits \Rightarrow$
$'v\ twl\text{-}clause \Rightarrow bool$
where
$watched\text{-}only\text{-}lazy\text{-}updates\ M\ (TWL\text{-}Clause\ W\ UW) \longleftrightarrow$
$(\forall L' \in set\ W.\ \forall L \in set\ UW.$

$-L' \in$ *lits-of-l M* $\longrightarrow -L \in$ *lits-of-l M* $\longrightarrow L \notin$ *set W* $\longrightarrow$
index (*map lit-of M*) $(-L') \leq$ *index* (*map lit-of M*) $(-L)$)

If the negation of a watched literal is included in the trail, then the negation of every unwatched literals is also included in the trail. Otherwise, the data-structure has to be updated.

primrec *watched-wf-twl-cls* :: ('*a*, '*b*) *ann-lits* ⇒ '*a twl-clause* ⇒
bool **where**
watched-wf-twl-cls M (*TWL-Clause W UW*) ⟷
(∀ *L* ∈ *set W*. $-L \in$ *lits-of-l M* ⟶ (∀ *L'* ∈ *set UW*. *L'* ∉ *set W* ⟶ $-L' \in$ *lits-of-l M*))

Here are the invariant strictly related to the 2-WL data structure.

primrec *wf-twl-cls* :: ('*v*, '*mark*) *ann-lits* ⇒ '*v twl-clause* ⇒ *bool* **where**
wf-twl-cls M (*TWL-Clause W UW*) ⟷
struct-wf-twl-cls (*TWL-Clause W UW*) ∧ *watched-wf-twl-cls M* (*TWL-Clause W UW*) ∧
watched-only-lazy-updates M (*TWL-Clause W UW*)

lemma *wf-twl-cls-annotation-independant*:
assumes *M*: *map lit-of M* = *map lit-of M'*
shows *wf-twl-cls M* (*TWL-Clause W UW*) ⟷ *wf-twl-cls M'* (*TWL-Clause W UW*)
⟨*proof*⟩

lemma *wf-twl-cls-wf-twl-cls-tl*:
assumes *wf*: *wf-twl-cls M C* **and** *n-d*: *no-dup M*
shows *wf-twl-cls* (*tl M*) *C*
⟨*proof*⟩

lemma *wf-twl-cls-append*:
assumes
n-d: *no-dup* (*M'* @ *M*) **and**
wf: *wf-twl-cls* (*M'* @ *M*) *C*
shows *wf-twl-cls M C*
⟨*proof*⟩

definition *wf-twl-state* :: '*v twl-state* ⇒ *bool* **where**
wf-twl-state S ⟷
(∀ *C* ∈ *set* (*twl.raw-clauses S*). *wf-twl-cls* (*raw-trail S*) *C*) ∧ *no-dup* (*raw-trail S*)

lemma *wf-candidates-propagate-sound*:
assumes *wf*: *wf-twl-state S* **and**
cand: (*L*, *C*) ∈ *candidates-propagate S*
shows *raw-trail S* ⊨*as CNot* (*mset* (*removeAll L* (*raw-clause C*))) ∧ *undefined-lit* (*raw-trail S*) *L*
(**is** *?Not* ∧ *?undef*)
⟨*proof*⟩

lemma *wf-candidates-propagate-complete*:
assumes *wf*: *wf-twl-state S* **and**
c-mem: *C* ∈ *set* (*twl.raw-clauses S*) **and**
l-mem: *L* ∈ *set* (*raw-clause C*) **and**
unsat: *trail S* ⊨*as CNot* (*mset-set* (*set* (*raw-clause C*) − {*L*})) **and**
undef: *undefined-lit* (*raw-trail S*) *L*
shows (*L*, *C*) ∈ *candidates-propagate S*
⟨*proof*⟩

lemma *wf-candidates-conflict-sound*:
assumes *wf*: *wf-twl-state S* **and**

cand: $C \in$ *candidates-conflict S*
shows *trail S* $\models$*as CNot* (*clause C*) $\wedge$ $C \in$ *set* (*twl.raw-clauses S*)
⟨*proof*⟩

lemma *wf-candidates-conflict-complete*:
assumes *wf*: *wf-twl-state S* **and**
c-mem: $C \in$ *set* (*twl.raw-clauses S*) **and**
unsat: *trail S* $\models$*as CNot* (*clause C*)
shows $C \in$ *candidates-conflict S*
⟨*proof*⟩

typedef *'v wf-twl* = {*S*::*'v twl-state. wf-twl-state S*}
morphisms *rough-state-of-twl twl-of-rough-state*
⟨*proof*⟩

lemma [*code abstype*]:
twl-of-rough-state (*rough-state-of-twl S*) = *S*
⟨*proof*⟩

lemma *wf-twl-state-rough-state-of-twl*[*simp*]: *wf-twl-state* (*rough-state-of-twl S*)
⟨*proof*⟩

abbreviation *candidates-conflict-twl* :: *'v wf-twl* $\Rightarrow$ *'v twl-clause set* **where**
candidates-conflict-twl S $\equiv$ *candidates-conflict* (*rough-state-of-twl S*)

abbreviation *candidates-propagate-twl* :: *'v wf-twl* $\Rightarrow$ (*'v literal* $\times$ *'v twl-clause*) *set* **where**
candidates-propagate-twl S $\equiv$ *candidates-propagate* (*rough-state-of-twl S*)

abbreviation *raw-trail-twl* :: *'a wf-twl* $\Rightarrow$ (*'a*, *'a twl-clause*) *ann-lits* **where**
raw-trail-twl S $\equiv$ *raw-trail* (*rough-state-of-twl S*)

abbreviation *trail-twl* :: *'a wf-twl* $\Rightarrow$ (*'a*, *'a literal multiset*) *ann-lits* **where**
trail-twl S $\equiv$ *trail* (*rough-state-of-twl S*)

abbreviation *raw-clauses-twl* :: *'a wf-twl* $\Rightarrow$ *'a twl-clause list* **where**
raw-clauses-twl S $\equiv$ *twl.raw-clauses* (*rough-state-of-twl S*)

abbreviation *raw-init-clss-twl* :: *'a wf-twl* $\Rightarrow$ *'a twl-clause list* **where**
raw-init-clss-twl S $\equiv$ *raw-init-clss* (*rough-state-of-twl S*)

abbreviation *raw-learned-clss-twl* :: *'a wf-twl* $\Rightarrow$ *'a twl-clause list* **where**
raw-learned-clss-twl S $\equiv$ *raw-learned-clss* (*rough-state-of-twl S*)

abbreviation *backtrack-lvl-twl* **where**
backtrack-lvl-twl S $\equiv$ *backtrack-lvl* (*rough-state-of-twl S*)

abbreviation *raw-conflicting-twl* **where**
raw-conflicting-twl S $\equiv$ *raw-conflicting* (*rough-state-of-twl S*)

lemma *wf-candidates-twl-conflict-complete*:
assumes
c-mem: $C \in$ *set* (*raw-clauses-twl S*) **and**
unsat: *trail-twl S* $\models$*as CNot* (*clause C*)
shows $C \in$ *candidates-conflict-twl S*
⟨*proof*⟩

abbreviation *update-backtrack-lvl* **where**
update-backtrack-lvl k S ≡
TWL-State (*raw-trail S*) (*raw-init-clss S*) (*raw-learned-clss S*) *k* (*raw-conflicting S*)

abbreviation *update-conflicting* **where**
update-conflicting C S ≡
TWL-State (*raw-trail S*) (*raw-init-clss S*) (*raw-learned-clss S*) (*backtrack-lvl S*) *C*

Abstract 2-WL

definition *tl-trail* **where**
tl-trail S =
TWL-State (*tl* (*raw-trail S*)) (*raw-init-clss S*) (*raw-learned-clss S*) (*backtrack-lvl S*)
(*raw-conflicting S*)

locale *abstract-twl* =
fixes
watch :: *'v twl-state* ⇒ *'v literal list* ⇒ *'v twl-clause* **and**
rewatch :: *'v literal* ⇒ *'v twl-state* ⇒
'v twl-clause ⇒ *'v twl-clause* **and**
restart-learned :: *'v twl-state* ⇒ *'v twl-clause list*
assumes
clause-watch: *no-dup* (*raw-trail S*) ⟹ *clause* (*watch S C*) = *mset C* **and**
wf-watch: *no-dup* (*raw-trail S*) ⟹ *wf-twl-cls* (*raw-trail S*) (*watch S C*) **and**
clause-rewatch: *clause* (*rewatch L' S C'*) = *clause C'* **and**
wf-rewatch:
no-dup (*raw-trail S*) ⟹ *undefined-lit* (*raw-trail S*) (*lit-of L*) ⟹
wf-twl-cls (*raw-trail S*) *C'* ⟹
wf-twl-cls (*L* # *raw-trail S*) (*rewatch* (*lit-of L*) *S C'*)
and
restart-learned: *mset* (*restart-learned S*) ⊆# *mset* (*raw-learned-clss S*) — We need *mset* and not *set* to take care of duplicates.
begin

definition
cons-trail :: (*'v*, *'v twl-clause*) *ann-lit* ⇒ *'v twl-state* ⇒ *'v twl-state*
where
cons-trail L S =
TWL-State (*L* # *raw-trail S*) (*map* (*rewatch* (*lit-of L*) *S*) (*raw-init-clss S*))
(*map* (*rewatch* (*lit-of L*) *S*) (*raw-learned-clss S*)) (*backtrack-lvl S*) (*raw-conflicting S*)

definition
add-init-cls :: *'v literal list* ⇒ *'v twl-state* ⇒ *'v twl-state*
where
add-init-cls C S =
TWL-State (*raw-trail S*) (*watch S C* # *raw-init-clss S*) (*raw-learned-clss S*) (*backtrack-lvl S*)
(*raw-conflicting S*)

definition
add-learned-cls :: *'v literal list* ⇒ *'v twl-state* ⇒ *'v twl-state*
where
add-learned-cls C S =
TWL-State (*raw-trail S*) (*raw-init-clss S*) (*watch S C* # *raw-learned-clss S*) (*backtrack-lvl S*)
(*raw-conflicting S*)

definition

remove-cls :: *'v literal list* ⇒ *'v twl-state* ⇒ *'v twl-state*
where
remove-cls C S =
TWL-State (*raw-trail S*)
(*removeAll-cond* (*λD. clause D* = *mset C*) (*raw-init-clss S*))
(*removeAll-cond* (*λD. clause D* = *mset C*) (*raw-learned-clss S*))
(*backtrack-lvl S*)
(*raw-conflicting S*)

definition *init-state* :: *'v literal list list* ⇒ *'v twl-state* **where**
init-state N = *fold add-init-cls N* (*TWL-State* [] [] [] *0 None*)

lemma *unchanged-fold-add-init-cls*:
raw-trail (*fold add-init-cls Cs* (*TWL-State M N U k C*)) = *M*
raw-learned-clss (*fold add-init-cls Cs* (*TWL-State M N U k C*)) = *U*
backtrack-lvl (*fold add-init-cls Cs* (*TWL-State M N U k C*)) = *k*
raw-conflicting (*fold add-init-cls Cs* (*TWL-State M N U k C*)) = *C*
⟨*proof*⟩

lemma *unchanged-init-state*[*simp*]:
raw-trail (*init-state N*) = []
raw-learned-clss (*init-state N*) = []
backtrack-lvl (*init-state N*) = *0*
raw-conflicting (*init-state N*) = *None*
⟨*proof*⟩

lemma *clauses-init-fold-add-init*:
no-dup M ⟹
twl.conc-init-clss (*fold add-init-cls Cs* (*TWL-State M N U k C*)) =
clauses-of-l Cs + *raw-clss-l N*
⟨*proof*⟩

lemma *init-clss-init-state*[*simp*]: *twl.conc-init-clss* (*init-state N*) = *clauses-of-l N*
⟨*proof*⟩

definition *restart'* **where**
restart' S = *TWL-State* [] (*raw-init-clss S*) (*restart-learned S*) *0 None*

end

Instanciation of the previous locale

definition *watch-nat* :: *'v twl-state* ⇒ *'v literal list* ⇒ *'v twl-clause* **where**
watch-nat S C =
(*let*
C' = *remdups C*;
neg-not-assigned = *filter* (*λL.* −*L* ∉ *lits-of-l* (*raw-trail S*)) *C'*;
neg-assigned-sorted-by-trail = *filter* (*λL. L* ∈ *set C*) (*map* (*λL.* −*lit-of L*) (*raw-trail S*));
W = *take 2* (*neg-not-assigned* @ *neg-assigned-sorted-by-trail*);
UW = *foldr remove1 W C*
in TWL-Clause W UW)

lemma *list-cases2*:
fixes *l* :: *'a list*
assumes
l = [] ⟹ *P* **and**

$\bigwedge x.\ l = [x] \Longrightarrow P$ **and**
$\bigwedge x\ y\ xs.\ l = x \# y \# xs \Longrightarrow P$
shows P
⟨*proof*⟩

lemma *filter-in-list-prop-verifiedD*:
assumes $[L\leftarrow P\ .\ Q\ L] = l$
shows $\forall x \in set\ l.\ x \in set\ P \wedge Q\ x$
⟨*proof*⟩

lemma *no-dup-filter-diff*:
assumes *n-d*: $no\text{-}dup\ M$ **and** *H*: $[L\leftarrow map\ (\lambda L.\ -\ lit\text{-}of\ L)\ M.\ L \in set\ C] = l$
shows *distinct l*
⟨*proof*⟩

lemma *watch-nat-lists-disjointD*:
assumes
l: $[L\leftarrow remdups\ C.\ -\ L \notin lits\text{-}of\text{-}l\ (raw\text{-}trail\ S)] = l$ **and**
l′: $[L\leftarrow map\ (\lambda L.\ -\ lit\text{-}of\ L)\ (raw\text{-}trail\ S)\ .\ L \in set\ C] = l'$
shows $\forall x \in set\ l.\ \forall y \in set\ l'.\ x \neq y$
⟨*proof*⟩

lemma *watch-nat-list-cases-witness*[*consumes 2*, *case-names Nil-Nil Nil-single Nil-other single-Nil single-other other*]:
fixes
$C :: 'v\ literal\ list$ **and**
$S :: 'v\ twl\text{-}state$
defines
$xs \equiv [L\leftarrow remdups\ C.\ -\ L \notin lits\text{-}of\text{-}l\ (raw\text{-}trail\ S)]$ **and**
$ys \equiv [L\leftarrow map\ (\lambda L.\ -\ lit\text{-}of\ L)\ (raw\text{-}trail\ S)\ .\ L \in set\ C]$
assumes
n-d: $no\text{-}dup\ (raw\text{-}trail\ S)$ **and**
Nil-Nil: $xs = [] \Longrightarrow ys = [] \Longrightarrow P$ **and**
Nil-single:
$\bigwedge a.\ xs = [] \Longrightarrow ys = [a] \Longrightarrow a \in set\ C \Longrightarrow P$ **and**
Nil-other: $\bigwedge a\ b\ ys'.\ xs = [] \Longrightarrow ys = a \# b \# ys' \Longrightarrow a \neq b \Longrightarrow P$ **and**
single-Nil: $\bigwedge a.\ xs = [a] \Longrightarrow ys = [] \Longrightarrow P$ **and**
single-other: $\bigwedge a\ b\ ys'.\ xs = [a] \Longrightarrow ys = b \# ys' \Longrightarrow a \neq b \Longrightarrow P$ **and**
other: $\bigwedge a\ b\ xs'.\ xs = a \# b \# xs' \Longrightarrow a \neq b \Longrightarrow P$
shows P
⟨*proof*⟩

lemma *watch-nat-list-cases* [*consumes 1*, *case-names Nil-Nil Nil-single Nil-other single-Nil single-other other*]:
fixes
$C :: 'v\ literal\ list$ **and**
$S :: 'v\ twl\text{-}state$
defines
$xs \equiv [L\leftarrow remdups\ C\ .\ -\ L \notin lits\text{-}of\text{-}l\ (raw\text{-}trail\ S)]$ **and**
$ys \equiv [L\leftarrow map\ (\lambda L.\ -\ lit\text{-}of\ L)\ (raw\text{-}trail\ S)\ .\ L \in set\ C]$
assumes
n-d: $no\text{-}dup\ (raw\text{-}trail\ S)$ **and**
Nil-Nil: $xs = [] \Longrightarrow ys = [] \Longrightarrow P$ **and**
Nil-single:
$\bigwedge a.\ xs = [] \Longrightarrow ys = [a] \Longrightarrow a \in set\ C \Longrightarrow P$ **and**
Nil-other: $\bigwedge a\ b\ ys'.\ xs = [] \Longrightarrow ys = a \# b \# ys' \Longrightarrow a \neq b \Longrightarrow P$ **and**

single-Nil: $\bigwedge a.\ xs = [a] \Longrightarrow ys = [] \Longrightarrow P$ **and**
single-other: $\bigwedge a\ b\ ys'.\ xs = [a] \Longrightarrow ys = b \ \#\ ys' \Longrightarrow a \neq b \Longrightarrow P$ **and**
other: $\bigwedge a\ b\ xs'.\ xs = a \ \#\ b \ \#\ xs' \Longrightarrow a \neq b \Longrightarrow P$
shows P
⟨*proof*⟩

lemma *watch-nat-lists-set-union-witness*:
fixes
C :: *'v literal list* **and**
S :: *'v twl-state*
defines
$xs \equiv [L \leftarrow remdups\ C.\ -\ L \notin lits\text{-}of\text{-}l\ (raw\text{-}trail\ S)]$ **and**
$ys \equiv [L \leftarrow map\ (\lambda L.\ -\ lit\text{-}of\ L)\ (raw\text{-}trail\ S)\ .\ L \in set\ C]$
assumes *n-d*: *no-dup* (*raw-trail* S)
shows *set* C = *set xs* $\cup$ *set ys*
⟨*proof*⟩

lemma *mset-intersection-inclusion*: $A + (B - A) = B \longleftrightarrow A \subseteq\# B$
⟨*proof*⟩

lemma *clause-watch-nat*:
assumes *no-dup* (*raw-trail* S)
shows *clause* (*watch-nat* S C) = *mset* C
⟨*proof*⟩

lemma *index-uminus-index-map-uminus*:
$-a \in set\ L \Longrightarrow index\ L\ (-a) = index\ (map\ uminus\ L)\ (a{::}'a\ literal)$
⟨*proof*⟩

lemma *index-filter*:
$a \in set\ L \Longrightarrow b \in set\ L \Longrightarrow P\ a \Longrightarrow P\ b \Longrightarrow$
$index\ L\ a \leq index\ L\ b \longleftrightarrow index\ (filter\ P\ L)\ a \leq index\ (filter\ P\ L)\ b$
⟨*proof*⟩

lemma *foldr-remove1-W-Nil*[*simp*]: *foldr remove1 W* [] = []
⟨*proof*⟩

lemma *image-lit-of-mmset-of-mlit*[*simp*]:
lit-of ‘ *mmset-of-mlit* ‘ A = *lit-of* ‘ A
⟨*proof*⟩

lemma *distinct-filter-eq*:
assumes *distinct xs*
shows $[L \leftarrow xs.\ L = a] = (if\ a \in set\ xs\ then\ [a]\ else\ [])$
⟨*proof*⟩

lemma *no-dup-distinct-map-uminus-lit-of*:
$no\text{-}dup\ xs \Longrightarrow distinct\ (map\ (\lambda L.\ -\ lit\text{-}of\ L)\ xs)$
⟨*proof*⟩

lemma *wf-watch-witness*:
fixes C :: *'v literal list* **and**
S :: *'v twl-state*
defines
ass: $neg\text{-}not\text{-}assigned \equiv filter\ (\lambda L.\ -L \notin lits\text{-}of\text{-}l\ (raw\text{-}trail\ S))\ (remdups\ C)$ **and**
tr: $neg\text{-}assigned\text{-}sorted\text{-}by\text{-}trail \equiv filter\ (\lambda L.\ L \in set\ C)\ (map\ (\lambda L.\ -lit\text{-}of\ L)\ (raw\text{-}trail\ S))$

defines
W: $W \equiv$ *take 2* (*neg-not-assigned* @ *neg-assigned-sorted-by-trail*)
assumes
n-d[*simp*]: *no-dup* (*raw-trail* S)
shows *wf-twl-cls* (*raw-trail* S) (*TWL-Clause* W (*foldr remove1* W C))
⟨*proof*⟩

lemma *wf-watch-nat*: *no-dup* (*raw-trail* S) $\Longrightarrow$ *wf-twl-cls* (*raw-trail* S) (*watch-nat* S C)
⟨*proof*⟩

definition
rewatch-nat ::
$'v$ *literal* $\Rightarrow$ $'v$ *twl-state* $\Rightarrow$ $'v$ *twl-clause* $\Rightarrow$ $'v$ *twl-clause*
where
rewatch-nat L S C =
(*if* $-$ $L \in$ *set* (*watched* C) *then*
case filter ($\lambda L'$. $L' \notin$ *set* (*watched* C) $\wedge$ $-$ $L' \notin$ *insert* L (*lits-of-l* (*trail* S)))
(*unwatched* C) *of*
[] $\Rightarrow$ C
| L' # - $\Rightarrow$
TWL-Clause (L' # *remove1* ($-L$) (*watched* C)) ($-L$ # *remove1* L' (*unwatched* C))
else
C)

lemma *clause-rewatch-nat*:
fixes UW :: $'v$ *literal list* **and**
S :: $'v$ *twl-state* **and**
L :: $'v$ *literal* **and** C :: $'v$ *twl-clause*
shows *clause* (*rewatch-nat* L S C) = *clause* C
⟨*proof*⟩

lemma *filter-sorted-list-of-multiset-Nil*:
[$x \leftarrow$ *sorted-list-of-multiset* M. p x] = [] $\longleftrightarrow$ ($\forall x \in\# M$. $\neg$ p x)
⟨*proof*⟩

lemma *filter-sorted-list-of-multiset-ConsD*:
[$x \leftarrow$ *sorted-list-of-multiset* M. p x] = x # *xs* $\Longrightarrow$ p x
⟨*proof*⟩

lemma *mset-minus-single-eq-mempty*:
a $-$ {#b#} = {#} $\longleftrightarrow$ a = {#b#} $\vee$ a = {#}
⟨*proof*⟩

lemma *size-mset-le-2-cases*:
assumes *size* $W \leq 2$
shows W = {#} $\vee$ ($\exists$ a. W = {#a#}) $\vee$ ($\exists$ a b. W = {#a,b#})
⟨*proof*⟩

lemma *filter-sorted-list-of-multiset-eqD*:
assumes [$x \leftarrow$ *sorted-list-of-multiset* A. p x] = x # *xs* (**is** *?comp* = -)
shows $x \in\# A$
⟨*proof*⟩

lemma *clause-rewatch-witness'*:
assumes
wf: *wf-twl-cls* (*raw-trail* S) C **and**

undef: *undefined-lit* (*raw-trail S*) (*lit-of L*)
shows *wf-twl-cls* (*L* # *raw-trail S*) (*rewatch-nat* (*lit-of L*) *S C*)
⟨*proof*⟩

interpretation *twl*: *abstract-twl watch-nat rewatch-nat raw-learned-clss*
⟨*proof*⟩

interpretation *twl2*: *abstract-twl watch-nat rewatch-nat* λ-. []
⟨*proof*⟩

end

3.6.2 Two Watched-Literals with invariant

theory *CDCL-Two-Watched-Literals-Invariant*
imports *CDCL-Two-Watched-Literals DPLL-CDCL-W-Implementation*
begin

Interpretation for *conflict-driven-clause-learning*$_W$.*cdcl*$_W$

We define here the 2-WL with the invariant of well-foundedness and show the role of the candidates by defining an equivalent CDCL procedure using the candidates given by the datastructure.

context *abstract-twl*
begin

Direct Interpretation **lemma** *mset-map-removeAll-cond*:
mset (*map clause*
(*removeAll-cond* (λ*D*. *clause D* = *clause C*) *N*))
= *mset* (*removeAll* (*clause C*) (*map clause N*))
⟨*proof*⟩

lemma *mset-raw-init-clss-init-state*:
mset (*map clause* (*raw-init-clss* (*init-state* (*map raw-clause N*))))
= *mset* (*map clause N*)
⟨*proof*⟩

fun *reduce-trail-to* **where**
reduce-trail-to M1 S =
(*case S of*
(*TWL-State M N U k C*) ⇒ *TWL-State* (*drop* (*length M* − *length M1*) *M*) *N U k C*)

abbreviation *resolve-conflicting* **where**
resolve-conflicting L D S ≡
update-conflicting
(*Some* (*union-mset-list* (*remove1* (−*L*) (*the* (*raw-conflicting S*))) (*remove1 L* (*raw-clause D*))))
S

interpretation *rough-cdcl*: *abs-state*$_W$-*ops*
clause
raw-clss-l op @
λ*L C*. *L* ∈ *set C op* # λ*C*. *remove1-cond* (λ*D*. *clause D* = *clause C*)

mset λ*xs ys. case-prod append* (*fold* (λ*x* (*ys*, *zs*). (*remove1 x ys*, *x* # *zs*)) *xs* (*ys*, []))
remove1

raw-clause λC. *TWL-Clause* [] *C*
trail λS. *hd* (*raw-trail S*)
raw-init-clss raw-learned-clss backtrack-lvl raw-conflicting
cons-trail tl-trail λS. *update-conflicting None* (*add-learned-cls* (*the* (*raw-conflicting S*)) *S*)
λC. *remove-cls* (*raw-clause C*)
update-backtrack-lvl
λC. *update-conflicting* (*Some C*) *reduce-trail-to resolve-conflicting*
λN. *init-state* (*map raw-clause N*) *restart'*
rewrites
rough-cdcl.mmset-of-mlit = *mmset-of-mlit*
⟨*proof*⟩

interpretation *rough-cdcl*: $\textit{abs-state}_W$
clause
raw-clss-l op @
λL *C*. *L* $\in$ *set C op* # λC. *remove1-cond* (λD. *clause D* = *clause C*)

mset λ*xs ys. case-prod append* (*fold* (λ*x* (*ys*, *zs*). (*remove1 x ys*, *x* # *zs*)) *xs* (*ys*, []))
remove1

raw-clause λC. *TWL-Clause* [] *C*
trail λS. *hd* (*raw-trail S*)
raw-init-clss raw-learned-clss backtrack-lvl raw-conflicting
cons-trail tl-trail λS. *update-conflicting None* (*add-learned-cls* (*the* (*raw-conflicting S*)) *S*)
λC. *remove-cls* (*raw-clause C*)
update-backtrack-lvl
λC. *update-conflicting* (*Some C*) *reduce-trail-to resolve-conflicting*
λN. *init-state* (*map raw-clause N*) *restart'*
⟨*proof*⟩

interpretation *rough-cdcl*: $\textit{abs-conflict-driven-clause-learning}_W$
clause
raw-clss-l op @
λL *C*. *L* $\in$ *set C op* # λC. *remove1-cond* (λD. *clause D* = *clause C*)

mset λ*xs ys. case-prod append* (*fold* (λ*x* (*ys*, *zs*). (*remove1 x ys*, *x* # *zs*)) *xs* (*ys*, []))
remove1

raw-clause λC. *TWL-Clause* [] *C*
trail λS. *hd* (*raw-trail S*)
raw-init-clss raw-learned-clss backtrack-lvl raw-conflicting
cons-trail tl-trail λS. *update-conflicting None* (*add-learned-cls* (*the* (*raw-conflicting S*)) *S*)
λC. *remove-cls* (*raw-clause C*)
update-backtrack-lvl
λC. *update-conflicting* (*Some C*) *reduce-trail-to resolve-conflicting*
λN. *init-state* (*map raw-clause N*) *restart'*
⟨*proof*⟩

declare *local.rough-cdcl.mset-ccls-ccls-of-cls*[*simp del*]

Opaque Type with Invariant **declare** *rough-cdcl.state-simp*[*simp del*]

definition *cons-trail-twl* :: ('*v*, '*v twl-clause*) *ann-lit* $\Rightarrow$ '*v wf-twl* $\Rightarrow$ '*v wf-twl*

where
cons-trail-twl L S ≡ *twl-of-rough-state* (*cons-trail L* (*rough-state-of-twl S*))

lemma *wf-twl-state-cons-trail*:
assumes
undef: *undefined-lit* (*raw-trail S*) (*lit-of L*) **and**
wf: *wf-twl-state S*
shows *wf-twl-state* (*cons-trail L S*)
⟨*proof*⟩

lemma *rough-state-of-twl-cons-trail*:
undefined-lit (*raw-trail-twl S*) (*lit-of L*) ⟹
rough-state-of-twl (*cons-trail-twl L S*) = *cons-trail L* (*rough-state-of-twl S*)
⟨*proof*⟩

abbreviation *add-init-cls-twl* **where**
add-init-cls-twl C S ≡ *twl-of-rough-state* (*add-init-cls C* (*rough-state-of-twl S*))

lemma *wf-twl-add-init-cls*: *wf-twl-state S* ⟹ *wf-twl-state* (*add-init-cls L S*)
⟨*proof*⟩

lemma *rough-state-of-twl-add-init-cls*:
rough-state-of-twl (*add-init-cls-twl L S*) = *add-init-cls L* (*rough-state-of-twl S*)
⟨*proof*⟩

abbreviation *add-learned-cls-twl* **where**
add-learned-cls-twl C S ≡ *twl-of-rough-state* (*add-learned-cls C* (*rough-state-of-twl S*))

lemma *wf-twl-add-learned-cls*: *wf-twl-state S* ⟹ *wf-twl-state* (*add-learned-cls L S*)
⟨*proof*⟩

lemma *rough-state-of-twl-add-learned-cls*:
rough-state-of-twl (*add-learned-cls-twl L S*) = *add-learned-cls L* (*rough-state-of-twl S*)
⟨*proof*⟩

abbreviation *remove-cls-twl* **where**
remove-cls-twl C S ≡ *twl-of-rough-state* (*remove-cls C* (*rough-state-of-twl S*))

lemma *set-removeAll-condD*: *x* ∈ *set* (*removeAll-cond f xs*) ⟹ *x* ∈ *set xs*
⟨*proof*⟩

lemma *wf-twl-remove-cls*: *wf-twl-state S* ⟹ *wf-twl-state* (*remove-cls L S*)
⟨*proof*⟩

lemma *rough-state-of-twl-remove-cls*:
rough-state-of-twl (*remove-cls-twl L S*) = *remove-cls L* (*rough-state-of-twl S*)
⟨*proof*⟩

abbreviation *init-state-twl* **where**
init-state-twl N ≡ *twl-of-rough-state* (*init-state N*)

lemma *wf-twl-state-wf-twl-state-fold-add-init-cls*:
assumes *wf-twl-state S*
shows *wf-twl-state* (*fold add-init-cls N S*)
⟨*proof*⟩

lemma *wf-twl-state-epsilon-state*[*simp*]:
wf-twl-state (*TWL-State* [] [] [] *0 None*)
⟨*proof*⟩

lemma *wf-twl-init-state*: *wf-twl-state* (*init-state N*)
⟨*proof*⟩

lemma *rough-state-of-twl-init-state*:
rough-state-of-twl (*init-state-twl N*) = *init-state N*
⟨*proof*⟩

abbreviation *tl-trail-twl* **where**
tl-trail-twl S ≡ *twl-of-rough-state* (*tl-trail* (*rough-state-of-twl S*))

lemma *wf-twl-state-tl-trail*: *wf-twl-state S* ⟹ *wf-twl-state* (*tl-trail S*)
⟨*proof*⟩

lemma *rough-state-of-twl-tl-trail*:
rough-state-of-twl (*tl-trail-twl S*) = *tl-trail* (*rough-state-of-twl S*)
⟨*proof*⟩

abbreviation *update-backtrack-lvl-twl* **where**
update-backtrack-lvl-twl k S ≡ *twl-of-rough-state* (*update-backtrack-lvl k* (*rough-state-of-twl S*))

lemma *wf-twl-state-update-backtrack-lvl*:
wf-twl-state S ⟹ *wf-twl-state* (*update-backtrack-lvl k S*)
⟨*proof*⟩

lemma *rough-state-of-twl-update-backtrack-lvl*:
rough-state-of-twl (*update-backtrack-lvl-twl k S*) = *update-backtrack-lvl k*
(*rough-state-of-twl S*)
⟨*proof*⟩

abbreviation *update-conflicting-twl* **where**
update-conflicting-twl k S ≡ *twl-of-rough-state* (*update-conflicting k* (*rough-state-of-twl S*))

lemma *wf-twl-state-update-conflicting*:
wf-twl-state S ⟹ *wf-twl-state* (*update-conflicting k S*)
⟨*proof*⟩

lemma *rough-state-of-twl-update-add-learned-cls*:
rough-state-of-twl (*update-conflicting-twl None* (*add-learned-cls-twl C S*)) =
update-conflicting None (*add-learned-cls C* (*rough-state-of-twl S*))
(**is** *rough-state-of-twl ?upd* = *update-conflicting None ?le*)
⟨*proof*⟩

abbreviation *reduce-trail-to-twl* **where**
reduce-trail-to-twl M1 S ≡ *twl-of-rough-state* (*reduce-trail-to M1* (*rough-state-of-twl S*))

abbreviation *resolve-conflicting-twl* **where**
resolve-conflicting-twl L D S ≡ *twl-of-rough-state* (*resolve-conflicting L D* (*rough-state-of-twl S*))

lemma *rough-state-of-twl-update-conflicting*:
rough-state-of-twl (*update-conflicting-twl k S*) = *update-conflicting k*
(*rough-state-of-twl S*)
⟨*proof*⟩

abbreviation *raw-clauses-twl* **where**
raw-clauses-twl S ≡ *twl.raw-clauses* (*rough-state-of-twl S*)

abbreviation *restart-twl* **where**
restart-twl S ≡ *twl-of-rough-state* (*restart′* (*rough-state-of-twl S*))
lemma *mset-union-mset-setD*:
  mset A ⊆# *mset B* ⟹ *set A* ⊆ *set B*
  ⟨*proof*⟩

lemma *wf-wf-restart′*: *wf-twl-state S* ⟹ *wf-twl-state* (*restart′ S*)
  ⟨*proof*⟩

lemma *rough-state-of-twl-restart-twl*:
  rough-state-of-twl (*restart-twl S*) = *restart′* (*rough-state-of-twl S*)
  ⟨*proof*⟩

lemma *undefined-lit-trail-twl-raw-trail*[*iff*]:
  undefined-lit (*trail-twl S*) *L* ⟷ *undefined-lit* (*raw-trail-twl S*) *L*
  ⟨*proof*⟩

lemma *wf-twl-reduce-trail-to*:
  assumes *trail S* = *M2* @ *M1* **and** *wf*: *wf-twl-state S*
  shows *wf-twl-state* (*reduce-trail-to M1 S*)
⟨*proof*⟩

lemma *trail-twl-twl-rough-state-reduce-trail-to*:
  assumes *trail-twl st* = *M2* @ *M1*
  shows *trail-twl* (*twl-of-rough-state* (*reduce-trail-to M1* (*rough-state-of-twl st*))) = *M1*
⟨*proof*⟩

lemma *twl-of-rough-state-reduce-trail-to*:
  assumes *trail-twl st* = *M2* @ *M1* **and**
    S: *rough-cdcl.state* (*rough-state-of-twl st*) = (*M*, *S*)
  shows
    rough-cdcl.state
      (*rough-state-of-twl* (*twl-of-rough-state* (*reduce-trail-to M1* (*rough-state-of-twl st*)))) =
      (*M1*, *S*) (**is** *?st*) **and**
    raw-init-clss-twl (*twl-of-rough-state* (*reduce-trail-to M1* (*rough-state-of-twl st*)))
      = *raw-init-clss-twl st* (**is** *?A*) **and**
    raw-learned-clss-twl (*twl-of-rough-state* (*reduce-trail-to M1* (*rough-state-of-twl st*)))
      = *raw-learned-clss-twl st* (**is** *?B*) **and**
    backtrack-lvl-twl (*twl-of-rough-state* (*reduce-trail-to M1* (*rough-state-of-twl st*)))
      = *backtrack-lvl-twl st* (**is** *?C*) **and**
    rough-cdcl.conc-conflicting (*rough-state-of-twl* (*twl-of-rough-state*
        (*reduce-trail-to M1* (*rough-state-of-twl st*))))
      = *rough-cdcl.conc-conflicting* (*rough-state-of-twl st*) (**is** *?D*)
⟨*proof*⟩

lemma *add-learned-cls-rough-state-of-twl-simp*:
  assumes *raw-conflicting-twl st* = *Some z*
  shows
    trail (*add-learned-cls z* (*rough-state-of-twl st*)) = *trail-twl st*
    rough-cdcl.conc-init-clss (*add-learned-cls z* (*rough-state-of-twl st*)) =
      rough-cdcl.conc-init-clss (*rough-state-of-twl st*)

rough-cdcl.conc-learned-clss (*local.add-learned-cls z* (*rough-state-of-twl st*)) =
{#*mset z*#} + *rough-cdcl.conc-learned-clss* (*rough-state-of-twl st*)
backtrack-lvl (*add-learned-cls z* (*rough-state-of-twl st*)) = *backtrack-lvl-twl st*
⟨*proof*⟩

sublocale *wf-twl*: *abs-state*$_W$*-ops*
clause
raw-clss-l op @
λ*L C*. *L* ∈ *set C op* # λ*C*. *remove1-cond* (λ*D*. *clause D* = *clause C*)

mset λ*xs ys*. *case-prod append* (*fold* (λ*x* (*ys*, *zs*). (*remove1 x ys*, *x* # *zs*)) *xs* (*ys*, []))
remove1

λ*C*. *raw-clause C* λ*C*. *TWL-Clause* [] *C*
trail-twl λ*S*. *hd* (*raw-trail-twl S*)
raw-init-clss-twl
raw-learned-clss-twl
backtrack-lvl-twl
raw-conflicting-twl
cons-trail-twl
tl-trail-twl
λ*S*. *update-conflicting-twl None* (*add-learned-cls-twl* (*the* (*raw-conflicting-twl S*)) *S*)
λ*C*. *remove-cls-twl* (*raw-clause C*)
update-backtrack-lvl-twl
λ*C*. *update-conflicting-twl* (*Some C*)
reduce-trail-to-twl
resolve-conflicting-twl
λ*N*. *init-state-twl* (*map raw-clause N*)
restart-twl
⟨*proof*⟩

sublocale *wf-twl*: *abs-state*$_W$
clause
raw-clss-l op @
λ*L C*. *L* ∈ *set C op* # λ*C*. *remove1-cond* (λ*D*. *clause D* = *clause C*)

mset λ*xs ys*. *case-prod append* (*fold* (λ*x* (*ys*, *zs*). (*remove1 x ys*, *x* # *zs*)) *xs* (*ys*, []))
remove1

λ*C*. *raw-clause C* λ*C*. *TWL-Clause* [] *C*
trail-twl λ*S*. *hd* (*raw-trail-twl S*)
raw-init-clss-twl
raw-learned-clss-twl
backtrack-lvl-twl
raw-conflicting-twl
cons-trail-twl
tl-trail-twl
λ*S*. *update-conflicting-twl None* (*add-learned-cls-twl* (*the* (*raw-conflicting-twl S*)) *S*)
λ*C*. *remove-cls-twl* (*raw-clause C*)
update-backtrack-lvl-twl
λ*C*. *update-conflicting-twl* (*Some C*)
reduce-trail-to-twl
resolve-conflicting-twl
λ*N*. *init-state-twl* (*map raw-clause N*)
restart-twl
⟨*proof*⟩

sublocale *wf-twl*: $abs\text{-}conflict\text{-}driven\text{-}clause\text{-}learning_W$
clause
raw-clss-l op @
$\lambda L\ C.\ L \in set\ C$ *op* # $\lambda C.$ *remove1-cond* $(\lambda D.\ clause\ D = clause\ C)$

mset $\lambda xs\ ys.$ *case-prod append* (*fold* $(\lambda x\ (ys, zs).$ (*remove1 x ys*, *x* # *zs*)) *xs* (*ys*, []))
remove1

$\lambda C.$ *raw-clause C* $\lambda C.$ *TWL-Clause* [] *C*
trail-twl $\lambda S.$ *hd* (*raw-trail-twl S*)
raw-init-clss-twl
raw-learned-clss-twl
backtrack-lvl-twl
raw-conflicting-twl
cons-trail-twl
tl-trail-twl
$\lambda S.$ *update-conflicting-twl None* (*add-learned-cls-twl* (*the* (*raw-conflicting-twl S*)) *S*)
$\lambda C.$ *remove-cls-twl* (*raw-clause C*)
update-backtrack-lvl-twl
$\lambda C.$ *update-conflicting-twl* (*Some C*)
reduce-trail-to-twl
resolve-conflicting-twl
$\lambda N.$ *init-state-twl* (*map raw-clause N*)
restart-twl
⟨*proof*⟩

declare *local.rough-cdcl.mset-ccls-ccls-of-cls*[*simp del*]
abbreviation *state-eq-twl* (**infix** $\sim TWL$ *51*) **where**
state-eq-twl $S\ S' \equiv$ *rough-cdcl.state-eq* (*rough-state-of-twl S*) (*rough-state-of-twl* S')
notation *wf-twl.state-eq* (**infix** $\sim$ *51*)

To avoid ambiguities:

no-notation *state-eq-twl* (**infix** $\sim$ *51*)

Alternative Definition of CDCL using the candidates of 2-WL

inductive *propagate-twl*
:: $'v$ *wf-twl* $\Rightarrow$ $'v$ *wf-twl* $\Rightarrow$ *bool* **where**
propagate-twl-rule: $(L, C) \in$ *candidates-propagate-twl S* $\Longrightarrow$
$S' \sim$ *cons-trail-twl* (*Propagated L C*) $S \Longrightarrow$
raw-conflicting-twl $S = None \Longrightarrow$
propagate-twl $S\ S'$

inductive-cases *propagate-twlE*: *propagate-twl S T*
lemma *propagate-twl-iff-propagate*:
assumes *inv*: $cdcl_W\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv$ (*wf-twl.state S*)
shows *wf-twl.propagate-abs S T* $\longleftrightarrow$ *propagate-twl S T* (**is** $?P \longleftrightarrow ?T$)
⟨*proof*⟩

no-notation *twl.state-eq-twl* (**infix** $\sim TWL$ *51*)

inductive *conflict-twl* **where**
conflict-twl-rule:
$C \in$ *candidates-conflict-twl S* $\Longrightarrow$
$S' \sim$ *update-conflicting-twl* (*Some* (*raw-clause C*)) $S \Longrightarrow$
raw-conflicting-twl $S = None \Longrightarrow$

conflict-twl S S'

inductive-cases *conflict-twlE*: *conflict-twl* S T

lemma *conflict-twl-iff-conflict*:
shows *wf-twl.conflict-abs* S T $\longleftrightarrow$ *conflict-twl* S T (**is** *?C* $\longleftrightarrow$ *?T*)
$\langle$*proof*$\rangle$

We have shown that we we can use *conflict-twl* and *propagate-twl* in a CDCL calculus.

end

end