Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

December 6, 2019

Contents

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

0.1 Weidenbach's DPLL

0.1.1 Rules

type-synonym *'a dpll$_W$-ann-lit* = (*'a*, *unit*) *ann-lit*
type-synonym *'a dpll$_W$-ann-lits* = (*'a*, *unit*) *ann-lits*
type-synonym *'v dpll$_W$-state* = *'v dpll$_W$-ann-lits* × *'v clauses*

abbreviation *trail* :: *'v dpll$_W$-state* ⇒ *'v dpll$_W$-ann-lits* **where**
trail ≡ *fst*
abbreviation *clauses* :: *'v dpll$_W$-state* ⇒ *'v clauses* **where**
clauses ≡ *snd*

inductive *dpll$_W$* :: *'v dpll$_W$-state* ⇒ *'v dpll$_W$-state* ⇒ *bool* **where**
propagate: *add-mset L C* ∈# *clauses S* ⟹ *trail S* ⊨*as CNot C* ⟹ *undefined-lit* (*trail S*) *L*
⟹ *dpll$_W$ S* (*Propagated L* () # *trail S*, *clauses S*) |
decided: *undefined-lit* (*trail S*) *L* ⟹ *atm-of L* ∈ *atms-of-mm* (*clauses S*)
⟹ *dpll$_W$ S* (*Decided L* # *trail S*, *clauses S*) |
backtrack: *backtrack-split* (*trail S*) = (*M'*, *L* # *M*) ⟹ *is-decided L* ⟹ *D* ∈# *clauses S*
⟹ *trail S* ⊨*as CNot D* ⟹ *dpll$_W$ S* (*Propagated* (− (*lit-of L*)) () # *M*, *clauses S*)

0.1.2 Invariants

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

lemma *dpll$_W$-consistent-interp-inv*:
assumes *dpll$_W$ S S'*
and *consistent-interp* (*lits-of-l* (*trail S*))
and *no-dup* (*trail S*)

shows *consistent-interp* (*lits-of-l* (*trail* S'))
⟨*proof*⟩

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

lemma *atms-of-ms-lit-of-atms-of*: *atms-of-ms* (*unmark* ‘ c) = *atm-of* ‘ *lit-of* ‘ c
⟨*proof*⟩

theorem 2.8.3 page 86 of Weidenbach's book

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

theorem 2.8.4 page 86 of Weidenbach's book

theorem *dpll$_W$-propagate-is-conclusion-of-decided*:
assumes *dpll$_W$* S S'
and *all-decomposition-implies-m* (*clauses* S) (*get-all-ann-decomposition* (*trail* S))
and *atm-of* ‘ *lits-of-l* (*trail* S) ⊆ *atms-of-mm* (*clauses* S)
shows *set-mset* (*clauses* S') ∪ {{#*lit-of* L#} |L. *is-decided* L ∧ L ∈ *set* (*trail* S')}
⊨*ps* *unmark* ‘ ⋃(*set* ‘ *snd* ‘ *set* (*get-all-ann-decomposition* (*trail* S')))
⟨*proof*⟩

theorem 2.8.5 page 86 of Weidenbach's book

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

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 $dpll_W\text{-}all\text{-}inv\ S \equiv$
(*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 $dpll_W\text{-}all\text{-}inv\text{-}dest[dest]$:
assumes $dpll_W\text{-}all\text{-}inv\ S$
shows *all-decomposition-implies-m* (*clauses S*) (*get-all-ann-decomposition* (*trail S*))
and *atm-of* ‘ *lits-of-l* (*trail S*) ⊆ *atms-of-mm* (*clauses S*)
and *consistent-interp* (*lits-of-l* (*trail S*)) ∧ *no-dup* (*trail S*)
⟨*proof*⟩

lemma $rtranclp\text{-}dpll_W\text{-}all\text{-}inv$:
assumes *rtranclp* $dpll_W$ *S S′*
and $dpll_W\text{-}all\text{-}inv\ S$
shows $dpll_W\text{-}all\text{-}inv\ S'$
⟨*proof*⟩

lemma $dpll_W\text{-}all\text{-}inv$:
assumes $dpll_W$ *S S′*
and $dpll_W\text{-}all\text{-}inv\ S$
shows $dpll_W\text{-}all\text{-}inv\ S'$
⟨*proof*⟩

lemma $rtranclp\text{-}dpll_W\text{-}inv\text{-}starting\text{-}from\text{-}0$:
assumes *rtranclp* $dpll_W$ *S S′*
and *inv*: *trail S* = []
shows $dpll_W\text{-}all\text{-}inv\ S'$
⟨*proof*⟩

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

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

theorem 2.8.7 page 87 of Weidenbach's book

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

theorem 2.8.6 page 86 of Weidenbach's book

lemma $dpll_W\text{-}sound$:
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*⟩

0.1.3 Termination

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

lemma *length-*$dpll_W$*-mes*:
assumes *length* $M \leq n$
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 *Cons-lexn-iff*:
shows ‹(x # *xs*, y # *ys*) ∈ *lexn* $R\ n \longleftrightarrow$ (*length* (x # *xs*) = n ∧ *length* (y # *ys*) = n ∧
((*x*,*y*) ∈ R ∨ ($x = y$ ∧ (*xs*, *ys*) ∈ *lexn* R (n − *1*))))›
⟨*proof*⟩
declare *append-same-lexn*[*simp*] *prepend-same-lexn*[*simp*] *Cons-lexn-iff*[*simp*]
declare *lexn.simps*(*2*)[*simp del*]

lemma $dpll_W$*-card-decrease*:
assumes
dpll: $dpll_W\ S\ S'$ **and**
[*simp*]: *length* (*trail* S') ≤ *card vars* **and**
length (*trail* S) ≤ *card vars*
shows
($dpll_W$*-mes* (*trail* S') (*card vars*), $dpll_W$*-mes* (*trail* S) (*card vars*)) ∈ *lexn less-than* (*card vars*)
⟨*proof*⟩

theorem 2.8.8 page 87 of Weidenbach's book

lemma $dpll_W$*-card-decrease′*:
assumes *dpll*: $dpll_W\ S\ S'$
and *atm-incl*: *atm-of* ‘ *lits-of-l* (*trail* S) ⊆ *atms-of-mm* (*clauses* S)
and *no-dup*: *no-dup* (*trail* S)
shows ($dpll_W$*-mes* (*trail* S') (*card* (*atms-of-mm* (*clauses* S'))),
$dpll_W$*-mes* (*trail* S) (*card* (*atms-of-mm* (*clauses* S)))) ∈ *lex less-than*
⟨*proof*⟩

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

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

0.1.4 Final States

Proposition 2.8.1: final states are the normal forms of *dpll$_W$*

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

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

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

Level of literals and clauses

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

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

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

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

lemma *get-level-Neg-Pos*: ‹*get-level M* (*Neg L*) = *get-level M* (*Pos L*)›
⟨*proof*⟩

lemma *count-decided-0-iff*:
‹*count-decided M* = *0* ⟷ (∀ *L* ∈ *set M*. ¬*is-decided L*)›
⟨*proof*⟩

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

⟨*count-decided* (*M* @ *M'*) = *count-decided M* + *count-decided M'*⟩
⟨*proof*⟩

lemma *atm-of-notin-get-level-eq-0*[*simp*]:
assumes *undefined-lit M L*
shows *get-level M L* = *0*
⟨*proof*⟩

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

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

lemma *get-level-skip*[*simp*]:
assumes *undefined-lit M L*
shows *get-level* (*M* @ *M'*) *L* = *get-level M' L*
⟨*proof*⟩

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

lemma *get-level-skip-end*[*simp*]:
assumes *defined-lit M L*
shows *get-level* (*M* @ *M'*) *L* = *get-level M L* + *count-decided M'*
⟨*proof*⟩

lemma *get-level-skip-beginning*[*simp*]:
assumes *atm-of L'* ≠ *atm-of* (*lit-of K*)
shows *get-level* (*K* # *M*) *L'* = *get-level M L'*
⟨*proof*⟩

lemma *get-level-take-beginning*[*simp*]:
assumes *atm-of L'* = *atm-of* (*lit-of K*)
shows *get-level* (*K* # *M*) *L'* = *count-decided* (*K* # *M*)
⟨*proof*⟩

lemma *get-level-cons-if*:
⟨*get-level* (*K* # *M*) *L'* =
(*if atm-of L'* = *atm-of* (*lit-of K*) *then count-decided* (*K* # *M*) *else get-level M L'*)⟩
⟨*proof*⟩

lemma *get-level-skip-beginning-not-decided*[*simp*]:
assumes *undefined-lit S L*
and ∀ *s*∈*set S*. ¬*is-decided s*
shows *get-level* (*M* @ *S*) *L* = *get-level M L*
⟨*proof*⟩

lemma *get-level-skip-all-not-decided*[*simp*]:
fixes *M*
assumes ∀ *m*∈*set M*. ¬ *is-decided m*
shows *get-level M L* = *0*
⟨*proof*⟩

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

definition *get-maximum-level* :: ('*a*, '*b*) *ann-lits* ⇒ '*a clause* ⇒ *nat*

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

lemma *get-maximum-level-ge-get-level*:
L ∈# D ⟹ get-maximum-level M D ≥ get-level M L
⟨*proof*⟩

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

lemma *get-maximum-level-exists-lit-of-max-level*:
D ≠ {#} ⟹ ∃ L∈# 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-add-mset*:
get-maximum-level M (add-mset L D) = max (get-level M L) (get-maximum-level M D)
⟨*proof*⟩

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

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

lemma *get-maximum-level-single*:
‹*get-maximum-level M {#x#} = get-level M x*›
⟨*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-cong*:
assumes ‹*∀ L ∈# D. get-level M L = get-level M′ L*›
shows ‹*get-maximum-level M D = get-maximum-level M′ D*›
⟨*proof*⟩

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

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

lemma *get-maximum-level-skip-beginning*:
assumes *DH*: *∀ x ∈# D. undefined-lit c x*
shows *get-maximum-level (c @ H) D = get-maximum-level H D*

⟨*proof*⟩

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

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

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

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

lemma *count-decided-ge-get-maximum-level*:
count-decided M ≥ *get-maximum-level M D*
⟨*proof*⟩

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

lemma *get-maximum-level-mono*:
‹*D* ⊆# *D*′ ⟹ *get-maximum-level M D* ≤ *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*⟩

lemma *get-all-mark-of-propagated-tl-proped*:
‹*M* ≠ [] ⟹ *is-proped* (*hd M*) ⟹ *get-all-mark-of-propagated* (*tl M*) = *tl* (*get-all-mark-of-propagated M*)›
⟨*proof*⟩

Properties about the levels

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

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

lemma *le-count-decided-decomp*:

assumes ‹*no-dup M*›
shows ‹*i* < *count-decided M* ⟷ (∃ *c K c'*. *M* = *c* @ *Decided K* # *c'* ∧ *get-level M K* = *Suc i*)›
(**is** *?A* ⟷ *?B*)
⟨*proof*⟩

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

lemma
fixes *K*
defines ‹*M* ≡ *replicate 3* (*Decided K*)›
defines ‹*i* ≡ *1*›
assumes ‹*i* < *count-decided M* ⟷ (∃ *c K c'*. *M* = *c* @ *Decided K* # *c'* ∧ *get-level M K* = *Suc i*)›
shows *False*
⟨*proof*⟩

lemma *Suc-count-decided-gt-get-level*:
‹*get-level M L* < *Suc* (*count-decided M*)›
⟨*proof*⟩

lemma *get-level-neq-Suc-count-decided*[*simp*]:
‹*get-level M L* ≠ *Suc* (*count-decided M*)›
⟨*proof*⟩

lemma *length-get-all-ann-decomposition*: ‹*length* (*get-all-ann-decomposition M*) = *1*+*count-decided M*›
⟨*proof*⟩

lemma *get-maximum-level-remove-non-max-lvl*:
‹*get-level M a* < *get-maximum-level M D* ⟹
get-maximum-level M (*remove1-mset a D*) = *get-maximum-level M D*›
⟨*proof*⟩

lemma *exists-lit-max-level-in-negate-ann-lits*:
‹*negate-ann-lits M* ≠ {#} ⟹ ∃ *L*∈#*negate-ann-lits M*. *get-level M L* = *count-decided M*›
⟨*proof*⟩

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

Chapter 1

Weidenbach's CDCL

The organisation of the development is the following:

- `CDCL_W.thy` contains the specification of the rules: the rules and the strategy are defined, and we proof the correctness of CDCL.
- `CDCL_W_Termination.thy` contains the proof of termination, based on the book.
- `CDCL_W_Merge.thy` contains a variant of the calculus: some rules of the raw calculus are always applied together (like the rules analysing the conflict and then backtracking). This is useful for the refinement from NOT.
- `CDCL_WNOT.thy` proves the inclusion of Weidenbach's version of CDCL in NOT's version. We use here the version defined in `CDCL_W_Merge.thy`. We need this, because NOT's backjump corresponds to multiple applications of three rules in Weidenbach's calculus. We show also the termination of the calculus without strategy. There are two different refinement: on from NOT's to Weidenbach's CDCL and another to W's CDCL with strategy.

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

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

1.1 Weidenbach's CDCL with Multisets

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

1.1.1 The State

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

locale *state$_W$-ops* =
  fixes
    state :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* × *'v clauses* × *'v clauses* × *'v clause option* ×
      'b **and**
    trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **and**
    init-clss :: *'st* ⇒ *'v clauses* **and**
    learned-clss :: *'st* ⇒ *'v clauses* **and**
    conflicting :: *'st* ⇒ *'v clause option* **and**

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

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

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

abbreviation *hd-trail* :: *'st* ⇒ (*'v*, *'v clause*) *ann-lit* **where**
hd-trail S ≡ *hd* (*trail S*)

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

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

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

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

end

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

Weidenbach state is a five-tuple composed of:

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

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

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

The representation of the clauses annotating literals in the trail is slightly different: being able to convert it to *'v clause* is enough (needed for function *hd-trail* below).

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

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

      — Some specific states:
    init-state
  for
    state-eq :: *'st* ⇒ *'st* ⇒ *bool* (**infix** ∼ *50*) **and**
    state :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* × *'v clauses* × *'v clauses* × *'v clause option* ×
      'b **and**
    trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **and**
    init-clss :: *'st* ⇒ *'v clauses* **and**
    learned-clss :: *'st* ⇒ *'v clauses* **and**
    conflicting :: *'st* ⇒ *'v clause option* **and**

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

    init-state :: *'v clauses* ⇒ *'st* +
  assumes
    state-eq-ref[*simp*, *intro*]: ‹*S* ∼ *S*› **and**
    state-eq-sym: ‹*S* ∼ *T* ⟷ *T* ∼ *S*› **and**
    state-eq-trans: ‹*S* ∼ *T* ⟹ *T* ∼ *U'* ⟹ *S* ∼ *U'*› **and**
    state-eq-state: ‹*S* ∼ *T* ⟹ *state S* = *state T*› **and**

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

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

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

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

    update-conflicting:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

init-state-learned-clss[*simp*]: *learned-clss* (*init-state N*) = {#} **and**
init-state-conflicting[*simp*]: *conflicting* (*init-state N*) = *None*
⟨*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-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*⟩

lemma *state-eq-trans'*: ‹*S* ∼ *S'* ⟹ *T* ∼ *S'* ⟹ *T* ∼ *S*›
⟨*proof*⟩

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

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

lemma
shows
state-eq-trail[*state-simp*]: *S* ∼ *T* ⟹ *trail S* = *trail T* **and**
state-eq-init-clss[*state-simp*]: *S* ∼ *T* ⟹ *init-clss S* = *init-clss T* **and**
state-eq-learned-clss[*state-simp*]: *S* ∼ *T* ⟹ *learned-clss S* = *learned-clss T* **and**
state-eq-conflicting[*state-simp*]: *S* ∼ *T* ⟹ *conflicting S* = *conflicting T* **and**
state-eq-clauses[*state-simp*]: *S* ∼ *T* ⟹ *clauses S* = *clauses T* **and**
state-eq-undefined-lit[*state-simp*]: *S* ∼ *T* ⟹ *undefined-lit* (*trail S*) *L* = *undefined-lit* (*trail T*) *L* **and**
state-eq-backtrack-lvl[*state-simp*]: *S* ∼ *T* ⟹ *backtrack-lvl S* = *backtrack-lvl T*
⟨*proof*⟩

lemma *state-eq-conflicting-None*:
S ∼ *T* ⟹ *conflicting T* = *None* ⟹ *conflicting S* = *None*
⟨*proof*⟩

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

declare *state-simp*[*simp*]

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

⟨*proof*⟩

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

lemma *reduce-trail-to-induct*:
assumes
⟨$\bigwedge F\ S.\ length\ (trail\ S) = length\ F \Longrightarrow P\ F\ S$⟩ **and**
⟨$\bigwedge F\ S.\ trail\ S = [] \Longrightarrow P\ F\ S$⟩ **and**
⟨$\bigwedge F\ S.\ length\ (trail\ S) \neq length\ F \Longrightarrow trail\ S \neq [] \Longrightarrow P\ F\ (tl\text{-}trail\ S) \Longrightarrow P\ F\ S$⟩
shows
⟨$P\ F\ S$⟩
⟨*proof*⟩

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

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

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

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

⟨*proof*⟩

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

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

lemma *reduce-trail-to-trail-tl-trail-decomp*[*simp*]:
trail S = *F'* @ *Decided K* # *F* ⟹ *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-length*:
length M = *length M'* ⟹ *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*) ≥ *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 *reduce-trail-to-state-eq*:
‹*S* ∼ *S'* ⟹ *length M* = *length M'* ⟹ *reduce-trail-to M S* ∼ *reduce-trail-to M' S'*›
⟨*proof*⟩

lemma *conflicting-cons-trail-conflicting*[*iff*]:
conflicting (*cons-trail L S*) = *None* ⟷ *conflicting S* = *None*
⟨*proof*⟩

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

lemma *reduce-trail-to-compow-tl-trail-le*:
assumes ‹*length M* < *length* (*trail M'*)›

shows ‹*reduce-trail-to M M'* = (*tl-trail*^^(*length* (*trail M'*) − *length M*)) *M'*›
⟨*proof*⟩

lemma *reduce-trail-to-compow-tl-trail-eq*:
‹*length M* = *length* (*trail M'*) ⟹ *reduce-trail-to M M'* = (*tl-trail*^^(*length* (*trail M'*) − *length M*)) *M'*›
⟨*proof*⟩

lemma *reduce-trail-to-compow-tl-trail*:
‹*length M* ≤ *length* (*trail M'*) ⟹ *reduce-trail-to M M'* = (*tl-trail*^^(*length* (*trail M'*) − *length M*)) *M'*›
⟨*proof*⟩

lemma *tl-trail-reduce-trail-to-cons*:
‹*length* (*L* # *M*) < *length* (*trail M'*) ⟹ *tl-trail* (*reduce-trail-to* (*L* # *M*) *M'*) = *reduce-trail-to M M'*›
⟨*proof*⟩

lemma *compow-tl-trail-add-learned-cls-swap*:
‹(*tl-trail* ^^ *n*) (*add-learned-cls D S*) ∼ *add-learned-cls D* ((*tl-trail* ^^ *n*) *S*)›
⟨*proof*⟩

lemma *reduce-trail-to-add-learned-cls-state-eq*:
‹*length M* ≤ *length* (*trail S*) ⟹
reduce-trail-to M (*add-learned-cls D S*) ∼ *add-learned-cls D* (*reduce-trail-to M S*)›
⟨*proof*⟩

lemma *compow-tl-trail-update-conflicting-swap*:
‹(*tl-trail* ^^ *n*) (*update-conflicting D S*) ∼ *update-conflicting D* ((*tl-trail* ^^ *n*) *S*)›
⟨*proof*⟩

lemma *reduce-trail-to-update-conflicting-state-eq*:
‹*length M* ≤ *length* (*trail S*) ⟹
reduce-trail-to M (*update-conflicting D S*) ∼ *update-conflicting D* (*reduce-trail-to M S*)›
⟨*proof*⟩

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

lemma *additional-info-reduce-trail-to*[*simp*]:
‹*additional-info* (*reduce-trail-to F S*) = *additional-info S*›
⟨*proof*⟩

lemma *reduce-trail-to*:
state (*reduce-trail-to F S*) =

((*if length* (*trail S*) ≥ *length F*
then drop (*length* (*trail S*) − *length F*) (*trail S*)
else []), *init-clss S*, *learned-clss S*, *conflicting S*, *additional-info S*)
⟨*proof*⟩

end — end of $state_W$ locale

1.1.2 CDCL Rules

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

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

— get state:
init-state
for
state-eq :: *'st* ⇒ *'st* ⇒ *bool* (**infix** ∼ *50*) **and**
state :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* × *'v clauses* × *'v clauses* × *'v clause option* ×
'b **and**
trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **and**
init-clss :: *'st* ⇒ *'v clauses* **and**
learned-clss :: *'st* ⇒ *'v clauses* **and**
conflicting :: *'st* ⇒ *'v clause option* **and**

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

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

inductive *propagate* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
propagate-rule: *conflicting S* = *None* ⟹
E ∈# *clauses S* ⟹
L ∈# *E* ⟹
trail S ⊨*as CNot* (*E* − {#*L*#}) ⟹
undefined-lit (*trail S*) *L* ⟹
T ∼ *cons-trail* (*Propagated L E*) *S* ⟹
propagate S T

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

inductive *conflict* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
conflict-rule:
conflicting S = *None* ⟹

$D \in\# \mathit{clauses}\ S \Longrightarrow$
$\mathit{trail}\ S \models \mathit{as}\ \mathit{CNot}\ D \Longrightarrow$
$T \sim \mathit{update\text{-}conflicting}\ (\mathit{Some}\ D)\ S \Longrightarrow$
$\mathit{conflict}\ S\ T$

inductive-cases *conflictE*: $\mathit{conflict}\ S\ T$

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

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

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

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

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

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

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

backtrackg S T

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

inductive *decide* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
decide-rule:
conflicting S = None ⟹
undefined-lit (*trail S*) *L* ⟹
atm-of L ∈ *atms-of-mm* (*init-clss S*) ⟹
T ∼ *cons-trail* (*Decided L*) *S* ⟹
decide S T

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

inductive *skip* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
skip-rule:
trail S = Propagated L C' # M ⟹
conflicting S = Some E ⟹
−L ∉# *E* ⟹
E ≠ {#} ⟹
T ∼ *tl-trail S* ⟹
skip S T

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

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

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

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

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

inductive *restart* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
restart: *state S* = (*M*, *N*, *U*, *None*, *S'*) ⟹
U' ⊆# *U* ⟹
state T = ([], *N*, *U'*, *None*, *S'*) ⟹
restart S T

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

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

inductive *forget* :: *'st* ⇒ *'st* ⇒ *bool* **where**
forget-rule:
conflicting S = None ⟹

$C \in\# \mathit{learned\text{-}clss}\ S \Longrightarrow$
$\neg(\mathit{trail}\ S) \models \mathit{asm}\ \mathit{clauses}\ S \Longrightarrow$
$C \notin \mathit{set}\ (\mathit{get\text{-}all\text{-}mark\text{-}of\text{-}propagated}\ (\mathit{trail}\ S)) \Longrightarrow$
$C \notin\# \mathit{init\text{-}clss}\ S \Longrightarrow$
$\mathit{removeAll\text{-}mset}\ C\ (\mathit{clauses}\ S) \models \mathit{pm}\ C \Longrightarrow$
$T \sim \mathit{remove\text{-}cls}\ C\ S \Longrightarrow$
$\mathit{forget}\ S\ T$

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

inductive $\mathit{cdcl}_W\text{-}\mathit{rf} :: 'st \Rightarrow 'st \Rightarrow \mathit{bool}$ **for** $S :: 'st$ **where**
restart: $\mathit{restart}\ S\ T \Longrightarrow \mathit{cdcl}_W\text{-}\mathit{rf}\ S\ T \mid$
forget: $\mathit{forget}\ S\ T \Longrightarrow \mathit{cdcl}_W\text{-}\mathit{rf}\ S\ T$

inductive $\mathit{cdcl}_W\text{-}\mathit{bj} :: 'st \Rightarrow 'st \Rightarrow \mathit{bool}$ **where**
skip: $\mathit{skip}\ S\ S' \Longrightarrow \mathit{cdcl}_W\text{-}\mathit{bj}\ S\ S' \mid$
resolve: $\mathit{resolve}\ S\ S' \Longrightarrow \mathit{cdcl}_W\text{-}\mathit{bj}\ S\ S' \mid$
backtrack: $\mathit{backtrack}\ S\ S' \Longrightarrow \mathit{cdcl}_W\text{-}\mathit{bj}\ S\ S'$

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

inductive $\mathit{cdcl}_W\text{-}\mathit{o} :: 'st \Rightarrow 'st \Rightarrow \mathit{bool}$ **for** $S :: 'st$ **where**
decide: $\mathit{decide}\ S\ S' \Longrightarrow \mathit{cdcl}_W\text{-}\mathit{o}\ S\ S' \mid$
bj: $\mathit{cdcl}_W\text{-}\mathit{bj}\ S\ S' \Longrightarrow \mathit{cdcl}_W\text{-}\mathit{o}\ S\ S'$

inductive $\mathit{cdcl}_W\text{-}\mathit{restart} :: 'st \Rightarrow 'st \Rightarrow \mathit{bool}$ **for** $S :: 'st$ **where**
propagate: $\mathit{propagate}\ S\ S' \Longrightarrow \mathit{cdcl}_W\text{-}\mathit{restart}\ S\ S' \mid$
conflict: $\mathit{conflict}\ S\ S' \Longrightarrow \mathit{cdcl}_W\text{-}\mathit{restart}\ S\ S' \mid$
other: $\mathit{cdcl}_W\text{-}\mathit{o}\ S\ S' \Longrightarrow \mathit{cdcl}_W\text{-}\mathit{restart}\ S\ S' \mid$
rf: $\mathit{cdcl}_W\text{-}\mathit{rf}\ S\ S' \Longrightarrow \mathit{cdcl}_W\text{-}\mathit{restart}\ S\ S'$

lemma $\mathit{rtranclp\text{-}propagate\text{-}is\text{-}rtranclp\text{-}cdcl}_W\text{-}\mathit{restart}$:
$\mathit{propagate}^{**}\ S\ S' \Longrightarrow \mathit{cdcl}_W\text{-}\mathit{restart}^{**}\ S\ S'$
⟨*proof*⟩

inductive $\mathit{cdcl}_W :: 'st \Rightarrow 'st \Rightarrow \mathit{bool}$ **for** $S :: 'st$ **where**
W-propagate: $\mathit{propagate}\ S\ S' \Longrightarrow \mathit{cdcl}_W\ S\ S' \mid$
W-conflict: $\mathit{conflict}\ S\ S' \Longrightarrow \mathit{cdcl}_W\ S\ S' \mid$
W-other: $\mathit{cdcl}_W\text{-}\mathit{o}\ S\ S' \Longrightarrow \mathit{cdcl}_W\ S\ S'$

lemma $\mathit{cdcl}_W\text{-}\mathit{cdcl}_W\text{-}\mathit{restart}$:
$\mathit{cdcl}_W\ S\ T \Longrightarrow \mathit{cdcl}_W\text{-}\mathit{restart}\ S\ T$
⟨*proof*⟩

lemma $\mathit{rtranclp\text{-}cdcl}_W\text{-}\mathit{cdcl}_W\text{-}\mathit{restart}$:
$\langle \mathit{cdcl}_W{}^{**}\ S\ T \Longrightarrow \mathit{cdcl}_W\text{-}\mathit{restart}^{**}\ S\ T \rangle$
⟨*proof*⟩

lemma $\mathit{cdcl}_W\text{-}\mathit{restart\text{-}all\text{-}rules\text{-}induct}$[*consumes 1*, *case-names propagate conflict forget restart decide skip resolve backtrack*]:
fixes $S :: 'st$
assumes
$\mathit{cdcl}_W\text{-}\mathit{restart}$: $\mathit{cdcl}_W\text{-}\mathit{restart}\ S\ S'$ **and**
propagate: $\bigwedge T.\ \mathit{propagate}\ S\ T \Longrightarrow P\ S\ T$ **and**
conflict: $\bigwedge T.\ \mathit{conflict}\ S\ T \Longrightarrow P\ S\ T$ **and**
forget: $\bigwedge T.\ \mathit{forget}\ S\ T \Longrightarrow P\ S\ T$ **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*$_W$*-restart-all-induct*[*consumes 1*, *case-names propagate conflict forget restart decide skip resolve backtrack*]:
fixes $S :: 'st$
assumes
cdcl$_W$*-restart*: *cdcl*$_W$*-restart* $S\ S'$ **and**
propagateH: $\bigwedge C\ L\ T.$ *conflicting* $S = None \Longrightarrow$
$C \in\#$ *clauses* $S \Longrightarrow$
$L \in\#\ C \Longrightarrow$
trail $S \models as$ *CNot* (*remove1-mset* $L\ C$) $\Longrightarrow$
undefined-lit (*trail* S) $L \Longrightarrow$
$T \sim$ *cons-trail* (*Propagated* $L\ C$) $S \Longrightarrow$
$P\ S\ T$ **and**
conflictH: $\bigwedge D\ T.$ *conflicting* $S = None \Longrightarrow$
$D \in\#$ *clauses* $S \Longrightarrow$
trail $S \models as$ *CNot* $D \Longrightarrow$
$T \sim$ *update-conflicting* (*Some* D) $S \Longrightarrow$
$P\ S\ T$ **and**
forgetH: $\bigwedge C\ T.$ *conflicting* $S = None \Longrightarrow$
$C \in\#$ *learned-clss* $S \Longrightarrow$
$\neg$(*trail* S) $\models asm$ *clauses* $S \Longrightarrow$
$C \notin$ *set* (*get-all-mark-of-propagated* (*trail* S)) $\Longrightarrow$
$C \notin\#$ *init-clss* $S \Longrightarrow$
removeAll-mset C (*clauses* S) $\models pm\ C \Longrightarrow$
$T \sim$ *remove-cls* $C\ S \Longrightarrow$
$P\ S\ T$ **and**
restartH: $\bigwedge T\ U.$ *conflicting* $S = None \Longrightarrow$
state T = ([], *init-clss* S, U, *None*, *additional-info* S) $\Longrightarrow$
$U \subseteq\#$ *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) $S \Longrightarrow$
$P\ S\ T$ **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*) ⟹
P S T **and**
backtrackH: ⋀*L D K i M1 M2 T D′*.
conflicting S = *Some* (*add-mset L D*) ⟹
(*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*)) ⟹
get-level (*trail S*) *L* = *backtrack-lvl S* ⟹
get-level (*trail S*) *L* = *get-maximum-level* (*trail S*) (*add-mset L D′*) ⟹
get-maximum-level (*trail S*) *D′* ≡ *i* ⟹
get-level (*trail S*) *K* = *i*+*1* ⟹
D′ ⊆# *D* ⟹
clauses S ⊨*pm add-mset L D′* ⟹
T ∼ *cons-trail* (*Propagated L* (*add-mset L D′*))
(*reduce-trail-to M1*
(*add-learned-cls* (*add-mset L D′*)
(*update-conflicting None S*))) ⟹
P S T
shows *P S S′*
⟨*proof*⟩

lemma *cdcl*$_W$*-o-induct*[*consumes 1*, *case-names decide skip resolve backtrack*]:
fixes *S* :: *′st*
assumes *cdcl*$_W$*-restart*: *cdcl*$_W$*-o S T* **and**
decideH: ⋀*L T*. *conflicting S* = *None* ⟹ *undefined-lit* (*trail S*) *L*
⟹ *atm-of L* ∈ *atms-of-mm* (*init-clss S*)
⟹ *T* ∼ *cons-trail* (*Decided L*) *S*
⟹ *P S T* **and**
skipH: ⋀*L C′ M E T*.
trail S = *Propagated L C′* # *M* ⟹
conflicting S = *Some E* ⟹
−*L* ∉# *E* ⟹ *E* ≠ {#} ⟹
T ∼ *tl-trail S* ⟹
P S T **and**
resolveH: ⋀*L E M D T*.
trail S = *Propagated L E* # *M* ⟹
L ∈# *E* ⟹
hd-trail S = *Propagated L E* ⟹
conflicting S = *Some D* ⟹
−*L* ∈# *D* ⟹
get-maximum-level (*trail S*) ((*remove1-mset* (−*L*) *D*)) = *backtrack-lvl S* ⟹
T ∼ *update-conflicting*
(*Some* (*resolve-cls L D E*)) (*tl-trail S*) ⟹
P S T **and**
backtrackH: ⋀*L D K i M1 M2 T D′*.
conflicting S = *Some* (*add-mset L D*) ⟹
(*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*)) ⟹
get-level (*trail S*) *L* = *backtrack-lvl S* ⟹
get-level (*trail S*) *L* = *get-maximum-level* (*trail S*) (*add-mset L D′*) ⟹
get-maximum-level (*trail S*) *D′* ≡ *i* ⟹
get-level (*trail S*) *K* = *i*+*1* ⟹
D′ ⊆# *D* ⟹
clauses S ⊨*pm add-mset L D′* ⟹
T ∼ *cons-trail* (*Propagated L* (*add-mset L D′*))
(*reduce-trail-to M1*
(*add-learned-cls* (*add-mset L D′*)
(*update-conflicting None S*))) ⟹
P S T

shows $P\ S\ T$
⟨*proof*⟩

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*⟩

lemma *backtrack-backtrackg*:
‹*backtrack* $S\ T \Longrightarrow$ *backtrackg* $S\ T$›
⟨*proof*⟩

lemma *simple-backtrack-backtrackg*:
‹*simple-backtrack* $S\ T \Longrightarrow$ *backtrackg* $S\ T$›
⟨*proof*⟩

1.1.3 Structural Invariants

Properties of the trail

We here establish that:

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

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

lemma *backtrack-lit-skiped*:
assumes
L: *get-level* (*trail S*) *L* = *backtrack-lvl S* **and**
M1: (*Decided K # M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*)) **and**
no-dup: *no-dup* (*trail S*) **and**
lev-K: *get-level* (*trail S*) *K* = *i* + *1*

shows *undefined-lit M1 L*
⟨*proof*⟩

lemma *cdcl$_W$-restart-distinctinv-1*:
assumes
cdcl$_W$-restart S S′ **and**
n-d: *no-dup* (*trail S*)
shows *no-dup* (*trail S′*)
⟨*proof*⟩

Item 1 page 94 of Weidenbach's book

lemma *cdcl$_W$-restart-consistent-inv-2*:
assumes
cdcl$_W$-restart S S′ **and**
no-dup (*trail S*)
shows *consistent-interp* (*lits-of-l* (*trail S′*))
⟨*proof*⟩

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*)

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$_W$-restart-consistent-inv*:
fixes *S S′* :: *'st*
assumes
cdcl$_W$-restart S S′ **and**
cdcl$_W$-M-level-inv S
shows *cdcl$_W$-M-level-inv S′*
⟨*proof*⟩

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

lemma *tranclp-cdcl$_W$-restart-consistent-inv*:
assumes
cdcl$_W$-restart^{++} S S′ **and**
cdcl$_W$-M-level-inv S
shows *cdcl$_W$-M-level-inv S′*
⟨*proof*⟩

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

lemma *backtrack-ex-decomp*:
  assumes
    M-l: *no-dup* (*trail S*) **and**
    i-S: $i <$ *backtrack-lvl S*
  shows $\exists K\ M1\ M2.$ (*Decided K # M1*, *M2*) $\in$ *set* (*get-all-ann-decomposition* (*trail S*)) $\wedge$
    get-level (*trail S*) *K* = *Suc i*
⟨*proof*⟩

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 ($\sim$)

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

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

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

lemma *decide-state-eq-compatible*:
  assumes
    dec: *decide S T* **and**
    SS': $S \sim S'$ **and**
    TT': $T \sim T'$
  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$_W$-restart-state-eq-compatible*:
  assumes
    cdcl$_W$-restart S T **and** ¬*restart S T* **and**
    $S \sim S'$
    $T \sim T'$
  shows *cdcl$_W$-restart S' T'*
  ⟨*proof*⟩

lemma *cdcl$_W$-bj-state-eq-compatible*:
  assumes
    cdcl$_W$-bj S T
    $T \sim T'$
  shows *cdcl$_W$-bj S T'*
  ⟨*proof*⟩

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

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

Conservation of some Properties

lemma *cdcl$_W$-o-no-more-init-clss*:
  assumes
    cdcl$_W$-o S S' **and**
    inv: *cdcl$_W$-M-level-inv S*

shows *init-clss S = init-clss S′*
⟨*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$_W$-restart-init-clss*:
assumes
cdcl$_W$-restart S T
shows *init-clss S = init-clss T*
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-restart-init-clss*:
cdcl$_W$-restart** *S T* ⟹ *init-clss S = init-clss T*
⟨*proof*⟩

lemma *tranclp-cdcl$_W$-restart-init-clss*:
cdcl$_W$-restart^{++} *S T* ⟹ *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 belong to the clauses. We could also restrict it to entailment by the clauses, to allow forgetting this clauses.

definition (**in** *state$_W$-ops*) *reasons-in-clauses* :: ‹*'st* ⇒ *bool*› **where**
‹*reasons-in-clauses* (*S* :: *'st*) ⟷
(*set* (*get-all-mark-of-propagated* (*trail S*)) ⊆ *set-mset* (*clauses S*))›

definition (**in** *state$_W$-ops*) *cdcl$_W$-learned-clause* :: ‹*'st* ⇒ *bool*› **where**
cdcl$_W$-learned-clause (*S* :: *'st*) ⟷
((∀ *T*. *conflicting S = Some T* ⟶ *clauses S* ⊨*pm T*)
∧ *reasons-in-clauses S*)

lemma *cdcl$_W$-learned-clause-alt-def*:
‹*cdcl$_W$-learned-clause* (*S* :: *'st*) ⟷
((∀ *T*. *conflicting S = Some T* ⟶ *clauses S* ⊨*pm T*)
∧ *set* (*get-all-mark-of-propagated* (*trail S*)) ⊆ *set-mset* (*clauses S*))›

⟨*proof*⟩

lemma *reasons-in-clauses-init-state*[*simp*]: ‹*reasons-in-clauses* (*init-state N*)›
⟨*proof*⟩

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

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

Item 4 page 95 of Weidenbach's book

lemma *cdcl*$_W$*-restart-learned-clss*:
assumes
cdcl$_W$*-restart S S′* **and**
learned: *cdcl*$_W$*-learned-clause S* **and**
lev-inv: *cdcl*$_W$*-M-level-inv S*
shows *cdcl*$_W$*-learned-clause S′*
⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-restart-learned-clss*:
assumes
cdcl$_W$*-restart*** *S S′* **and**
cdcl$_W$*-M-level-inv S*
cdcl$_W$*-learned-clause S*
shows *cdcl*$_W$*-learned-clause S′*
⟨*proof*⟩

lemma *cdcl*$_W$*-restart-reasons-in-clauses*:
assumes
cdcl$_W$*-restart S S′* **and**
learned: *reasons-in-clauses S*
shows *reasons-in-clauses S′*
⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-restart-reasons-in-clauses*:
assumes
cdcl$_W$*-restart*** *S S′* **and**
learned: *reasons-in-clauses S*
shows *reasons-in-clauses S′*
⟨*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′* ⟷
(∀ *T*. *conflicting S′* = *Some T* ⟶ *atms-of T* ⊆ *atms-of-mm* (*init-clss S′*))
∧ (∀ *L mark*. *Propagated L mark* ∈ *set* (*trail S′*) ⟶ *atms-of mark* ⊆ *atms-of-mm* (*init-clss S′*))
∧ *atms-of-mm* (*learned-clss S′*) ⊆ *atms-of-mm* (*init-clss S′*)
∧ *atm-of* ‘ (*lits-of-l* (*trail S′*)) ⊆ *atms-of-mm* (*init-clss S′*)

lemma *no-strange-atm-decomp*:
assumes *no-strange-atm S*

shows *conflicting S = Some T ⟹ atms-of T ⊆ atms-of-mm (init-clss S)*
and *(∀ L mark. Propagated L mark ∈ set (trail S) ⟶ atms-of mark ⊆ atms-of-mm (init-clss S))*
and *atms-of-mm (learned-clss S) ⊆ atms-of-mm (init-clss S)*
and *atm-of ‘ (lits-of-l (trail S)) ⊆ atms-of-mm (init-clss S)*
⟨*proof*⟩

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

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

lemma *atms-of-ms-learned-clss-restart-state-in-atms-of-ms-learned-clssI*:
atms-of-mm (learned-clss S) ⊆ atms-of-mm (init-clss S) ⟹
x ∈ atms-of-mm (learned-clss T) ⟹
learned-clss T ⊆# learned-clss S ⟹
x ∈ atms-of-mm (init-clss S)
⟨*proof*⟩

lemma (**in** −) *atms-of-subset-mset-mono*: ‹*D′ ⊆# D ⟹ atms-of D′ ⊆ atms-of D*›
⟨*proof*⟩

lemma *cdcl$_W$-restart-no-strange-atm-explicit*:
assumes
cdcl$_W$-restart S S′ **and**
lev: *cdcl$_W$-M-level-inv S* **and**
conf: *∀ T. conflicting S = Some T ⟶ atms-of T ⊆ atms-of-mm (init-clss S)* **and**
decided: *∀ L mark. Propagated L mark ∈ set (trail S)*
⟶ atms-of mark ⊆ atms-of-mm (init-clss S) **and**
learned: *atms-of-mm (learned-clss S) ⊆ atms-of-mm (init-clss S)* **and**
trail: *atm-of ‘ (lits-of-l (trail S)) ⊆ atms-of-mm (init-clss S)*
shows
(∀ T. conflicting S′ = Some T ⟶ atms-of T ⊆ atms-of-mm (init-clss S′)) ∧
(∀ L mark. Propagated L mark ∈ set (trail S′) ⟶ atms-of mark ⊆ atms-of-mm (init-clss S′)) ∧
atms-of-mm (learned-clss S′) ⊆ atms-of-mm (init-clss S′) ∧
atm-of ‘ (lits-of-l (trail S′)) ⊆ atms-of-mm (init-clss S′)
*(***is** *?C S′ ∧ ?M S′ ∧ ?U S′ ∧ ?V S′)*
⟨*proof*⟩

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

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

No Duplicates all Around

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

definition *distinct-cdcl$_W$-state* (*S* ::'*st*)
⟷ ((∀ *T*. *conflicting S* = *Some T* ⟶ *distinct-mset T*)
∧ *distinct-mset-mset* (*learned-clss S*)
∧ *distinct-mset-mset* (*init-clss S*)
∧ (∀ *L mark*. (*Propagated L mark* ∈ *set* (*trail S*) ⟶ *distinct-mset mark*)))

lemma *distinct-cdcl$_W$-state-decomp*:
assumes *distinct-cdcl$_W$-state S*
shows
∀ *T*. *conflicting S* = *Some T* ⟶ *distinct-mset T* **and**
distinct-mset-mset (*learned-clss S*) **and**
distinct-mset-mset (*init-clss S*) **and**
∀ *L mark*. (*Propagated L mark* ∈ *set* (*trail S*) ⟶ *distinct-mset mark*)
⟨*proof*⟩

lemma *distinct-cdcl$_W$-state-decomp-2*:
assumes *distinct-cdcl$_W$-state S* **and** *conflicting S* = *Some T*
shows *distinct-mset T*
⟨*proof*⟩

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

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

lemma *rtanclp-distinct-cdcl$_W$-state-inv*:
assumes
*cdcl$_W$-restart*** *S S'* **and**
cdcl$_W$-M-level-inv S **and**
distinct-cdcl$_W$-state S
shows *distinct-cdcl$_W$-state S'*
⟨*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* :: '*st* ⇒ *bool* **where**

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

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

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*⟩

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

Item 5 page 95 of Weidenbach's book

lemma *cdcl$_W$-restart-propagate-is-conclusion*:
assumes
cdcl$_W$-restart S S' **and**

inv: $cdcl_W$*-M-level-inv* S **and**
decomp: *all-decomposition-implies-m* (*clauses* S) (*get-all-ann-decomposition* (*trail* S)) **and**
learned: $cdcl_W$*-learned-clause* S **and**
confl: $\forall\, T.$ *conflicting* S = *Some* $T \longrightarrow$ *trail* $S \models as$ *CNot* T **and**
alien: *no-strange-atm* S
shows *all-decomposition-implies-m* (*clauses* S') (*get-all-ann-decomposition* (*trail* S'))
⟨*proof*⟩

lemma $cdcl_W$*-restart-propagate-is-false*:
assumes
$cdcl_W$*-restart* S S' **and**
lev: $cdcl_W$*-M-level-inv* S **and**
learned: $cdcl_W$*-learned-clause* S **and**
decomp: *all-decomposition-implies-m* (*clauses* S) (*get-all-ann-decomposition* (*trail* S)) **and**
confl: $\forall\, T.$ *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
$cdcl_W$*-restart* 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#}) ∧ $L \in\#$ *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#}) ∧ $L \in\#$ *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-*$cdcl_W$*-restart*[*simp*]:
$cdcl_W$*-conflicting* (*init-state* N)
⟨*proof*⟩

definition $cdcl_W$*-learned-clauses-entailed-by-init* **where**
‹$cdcl_W$*-learned-clauses-entailed-by-init* $S \longleftrightarrow$ *init-clss* $S \models psm$ *learned-clss* S›

lemma $cdcl_W$*-learned-clauses-entailed-init*[*simp*]:
‹$cdcl_W$*-learned-clauses-entailed-by-init* (*init-state* N)›
⟨*proof*⟩

lemma $cdcl_W$*-learned-clauses-entailed*:

assumes
cdcl$_W$-restart: *cdcl$_W$-restart S S′* **and**
2: *cdcl$_W$-learned-clause S* **and**
9: ‹*cdcl$_W$-learned-clauses-entailed-by-init S*›
shows ‹*cdcl$_W$-learned-clauses-entailed-by-init S′*›
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-learned-clauses-entailed*:
assumes
cdcl$_W$-restart: *cdcl$_W$-restart** S S′* **and**
2: *cdcl$_W$-learned-clause S* **and**
4: *cdcl$_W$-M-level-inv S* **and**
9: ‹*cdcl$_W$-learned-clauses-entailed-by-init S*›
shows ‹*cdcl$_W$-learned-clauses-entailed-by-init S′*›
⟨*proof*⟩

Putting all the Invariants Together

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

lemma *rtranclp-cdcl$_W$-restart-all-inv*:
assumes
cdcl$_W$-restart: *rtranclp cdcl$_W$-restart S S′* **and**
1: *all-decomposition-implies-m* (*clauses S*) (*get-all-ann-decomposition* (*trail S*)) **and**
2: *cdcl$_W$-learned-clause S* **and**
4: *cdcl$_W$-M-level-inv S* **and**
5: *no-strange-atm S* **and**
7: *distinct-cdcl$_W$-state S* **and**
8: *cdcl$_W$-conflicting S*
shows
all-decomposition-implies-m (*clauses S′*) (*get-all-ann-decomposition* (*trail S′*)) **and**
cdcl$_W$-learned-clause S′ **and**
cdcl$_W$-M-level-inv S′ **and**
no-strange-atm S′ **and**
distinct-cdcl$_W$-state S′ **and**
cdcl$_W$-conflicting S′
⟨*proof*⟩

lemma *all-invariant-S0-cdcl$_W$-restart*:
assumes *distinct-mset-mset N*

shows
all-decomposition-implies-m (*init-clss* (*init-state N*))
(*get-all-ann-decomposition* (*trail* (*init-state N*))) **and**
cdcl$_W$*-learned-clause* (*init-state N*) **and**
∀ *T*. *conflicting* (*init-state N*) = *Some T* ⟶ (*trail* (*init-state N*))⊨*as CNot T* **and**
no-strange-atm (*init-state N*) **and**
consistent-interp (*lits-of-l* (*trail* (*init-state N*))) **and**
∀ *L mark a b*. *a* @ *Propagated L mark* # *b* = *trail* (*init-state N*) ⟶
(*b* ⊨*as CNot* (*mark* − {#*L*#}) ∧ *L* ∈# *mark*) **and**
distinct-cdcl$_W$*-state* (*init-state N*)
⟨*proof*⟩

Item 6 page 95 of Weidenbach's book

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

Item 5 page 95 of Weidenbach's book

We have actually a much stronger theorem, namely *all-decomposition-implies-propagated-lits-are-implied*, that show that the only choices we made are decided in the formula

lemma
assumes *all-decomposition-implies-m N* (*get-all-ann-decomposition M*)
and ∀ *m* ∈ *set M*. ¬*is-decided m*
shows *set-mset N* ⊨*ps unmark-l M*
⟨*proof*⟩

Item 7 page 95 of Weidenbach's book (part 1)

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

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

lemma *conflict-with-false-implies-terminated*:
assumes *cdcl*$_W$*-restart S S′* **and** *conflicting S* = *Some* {#}
shows *False*
⟨*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$_W$-restart S S′ **and**
lev: *cdcl$_W$-M-level-inv S* **and**
conflicting: *cdcl$_W$-conflicting S* **and**
no-tauto: ∀ *s* ∈# *learned-clss S*. ¬*tautology s*
shows ∀ *s* ∈# *learned-clss S′*. ¬*tautology s*
⟨*proof*⟩

definition *final-cdcl$_W$-restart-state* (*S* :: *'st*)
⟷ (*trail S* ⊨*asm init-clss S*
∨ ((∀ *L* ∈ *set* (*trail S*). ¬*is-decided L*) ∧
(∃ *C* ∈# *init-clss S*. *trail S* ⊨*as CNot C*)))

definition *termination-cdcl$_W$-restart-state* (*S* :: *'st*)
⟷ (*trail S* ⊨*asm init-clss S*
∨ ((∀ *L* ∈ *atms-of-mm* (*init-clss S*). *L* ∈ *atm-of* ‘ *lits-of-l* (*trail S*))
∧ (∃ *C* ∈# *init-clss S*. *trail S* ⊨*as CNot C*)))

1.1.4 CDCL Strong Completeness

lemma *cdcl$_W$-restart-can-do-step*:
assumes
consistent-interp (*set M*) **and**
distinct M **and**
atm-of ‘ (*set M*) ⊆ *atms-of-mm N*
shows ∃ *S*. *rtranclp cdcl$_W$-restart* (*init-state N*) *S*
∧ *state-butlast S* = (*map* (λ*L*. *Decided L*) *M*, *N*, {#}, *None*)
⟨*proof*⟩

theorem 2.9.11 page 98 of Weidenbach's book

lemma *cdcl$_W$-restart-strong-completeness*:
assumes
MN: *set M* ⊨*sm N* **and**
cons: *consistent-interp* (*set M*) **and**
dist: *distinct M* **and**
atm: *atm-of* ‘ (*set M*) ⊆ *atms-of-mm N*
obtains *S* **where**
state-butlast S = (*map* (λ*L*. *Decided L*) *M*, *N*, {#}, *None*) **and**
rtranclp cdcl$_W$-restart (*init-state N*) *S* **and**
final-cdcl$_W$-restart-state S
⟨*proof*⟩

1.1.5 Higher level strategy

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

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

Definition

lemma *tranclp-conflict*:
tranclp conflict S S′ ⟹ conflict S S′
⟨*proof*⟩

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

lemma *tranclp-conflict-iff*:
full1 conflict S S′ ⟷ conflict S S′
⟨*proof*⟩

lemma *no-conflict-after-conflict*:
conflict S T ⟹ ¬conflict T U
⟨*proof*⟩

lemma *no-propagate-after-conflict*:
conflict S T ⟹ ¬propagate T U
⟨*proof*⟩

inductive $cdcl_W$*-stgy* :: *'st ⇒ 'st ⇒ bool* **for** *S* :: *'st* **where**
conflict′: *conflict S S′ ⟹* $cdcl_W$*-stgy S S′* |
propagate′: *propagate S S′ ⟹* $cdcl_W$*-stgy S S′* |
other′: *no-step conflict S ⟹ no-step propagate S ⟹* $cdcl_W$*-o S S′ ⟹* $cdcl_W$*-stgy S S′*

lemma $cdcl_W$*-stgy-*$cdcl_W$: $cdcl_W$*-stgy S T ⟹* $cdcl_W$ *S T*
⟨*proof*⟩

lemma $cdcl_W$*-stgy-induct*[*consumes 1*, *case-names conflict propagate decide skip resolve backtrack*]:
assumes
‹$cdcl_W$*-stgy S T*› **and**
‹⋀*T. conflict S T ⟹ P T*› **and**
‹⋀*T. propagate S T ⟹ P T*› **and**
‹⋀*T. no-step conflict S ⟹ no-step propagate S ⟹ decide S T ⟹ P T*› **and**
‹⋀*T. no-step conflict S ⟹ no-step propagate S ⟹ skip S T ⟹ P T*› **and**
‹⋀*T. no-step conflict S ⟹ no-step propagate S ⟹ resolve S T ⟹ P T*› **and**
‹⋀*T. no-step conflict S ⟹ no-step propagate S ⟹ backtrack S T ⟹ P T*›
shows
‹*P T*›
⟨*proof*⟩

lemma $cdcl_W$*-stgy-cases*[*consumes 1*, *case-names conflict propagate decide skip resolve backtrack*]:
assumes
‹$cdcl_W$*-stgy S T*› **and**
‹*conflict S T ⟹ P*› **and**
‹*propagate S T ⟹ P*› **and**
‹*no-step conflict S ⟹ no-step propagate S ⟹ decide S T ⟹ P*› **and**
‹*no-step conflict S ⟹ no-step propagate S ⟹ skip S T ⟹ P*› **and**
‹*no-step conflict S ⟹ no-step propagate S ⟹ resolve S T ⟹ P*› **and**
‹*no-step conflict S ⟹ no-step propagate S ⟹ backtrack S T ⟹ P*›
shows
‹*P*›
⟨*proof*⟩

Invariants

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

lemma *rtranclp-cdcl$_W$-stgy-no-more-init-clss*:
assumes *cdcl$_W$-stgy** S S′*
shows *init-clss S = init-clss S′*
⟨*proof*⟩

Literal of highest level in conflicting clauses

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

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

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

Literal of highest level in decided literals

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

lemma *backtrack$_W$-rule*:
assumes
confl: ‹*conflicting S = Some (add-mset L D)*› **and**
decomp: ‹(*Decided K # M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*))› **and**
lev-L: ‹*get-level* (*trail S*) *L* = *backtrack-lvl S*› **and**
max-lev: ‹*get-level* (*trail S*) *L* = *get-maximum-level* (*trail S*) (*add-mset L D*)› **and**
max-D: ‹*get-maximum-level* (*trail S*) *D* ≡ *i*› **and**
lev-K: ‹*get-level* (*trail S*) *K* = *i* + *1*› **and**
T: ‹*T* ∼ *cons-trail* (*Propagated L* (*add-mset L D*))
(*reduce-trail-to M1*
(*add-learned-cls* (*add-mset L D*)
(*update-conflicting None S*)))› **and**
lev-inv: *cdcl$_W$-M-level-inv S* **and**
conf: ‹*cdcl$_W$-conflicting S*› **and**
learned: ‹*cdcl$_W$-learned-clause S*›
shows ‹*backtrack S T*›

⟨*proof*⟩

lemma *backtrack-no-decomp*:
assumes
S: *conflicting S* = *Some* (*add-mset L E*) **and**
L: *get-level* (*trail S*) *L* = *backtrack-lvl S* **and**
D: *get-maximum-level* (*trail S*) *E* < *backtrack-lvl S* **and**
bt: *backtrack-lvl S* = *get-maximum-level* (*trail S*) (*add-mset L E*) **and**
lev-inv: $cdcl_W$*-M-level-inv S* **and**
conf: ‹$cdcl_W$*-conflicting S*› **and**
learned: ‹$cdcl_W$*-learned-clause S*›
shows ∃ *S'*. $cdcl_W$*-o S S'* ∃ *S'*. *backtrack S S'*
⟨*proof*⟩

lemma *no-analyse-backtrack-Ex-simple-backtrack*:
assumes
bt: ‹*backtrack S T*› **and**
lev-inv: $cdcl_W$*-M-level-inv S* **and**
conf: ‹$cdcl_W$*-conflicting S*› **and**
learned: ‹$cdcl_W$*-learned-clause S*› **and**
no-dup: ‹*distinct-cdcl*$_W$*-state S*› **and**
ns-s: ‹*no-step skip S*› **and**
ns-r: ‹*no-step resolve S*›
shows ‹*Ex*(*simple-backtrack S*)›
⟨*proof*⟩

lemma *trail-begins-with-decided-conflicting-exists-backtrack*:
assumes
confl-k: ‹*conflict-is-false-with-level S*› **and**
conf: ‹$cdcl_W$*-conflicting S*› **and**
level-inv: ‹$cdcl_W$*-M-level-inv S*› **and**
no-dup: ‹*distinct-cdcl*$_W$*-state S*› **and**
learned: ‹$cdcl_W$*-learned-clause S*› **and**
alien: ‹*no-strange-atm S*› **and**
tr-ne: ‹*trail S* ≠ []› **and**
L': ‹*hd-trail S* = *Decided L'*› **and**
nempty: ‹*C* ≠ {#}› **and**
confl: ‹*conflicting S* = *Some C*›
shows ‹*Ex* (*backtrack S*)› **and** ‹*no-step skip S*› **and** ‹*no-step resolve S*›
⟨*proof*⟩

lemma *conflicting-no-false-can-do-step*:
assumes
confl: ‹*conflicting S* = *Some C*› **and**
nempty: ‹*C* ≠ {#}› **and**
confl-k: ‹*conflict-is-false-with-level S*› **and**
conf: ‹$cdcl_W$*-conflicting S*› **and**
level-inv: ‹$cdcl_W$*-M-level-inv S*› **and**
no-dup: ‹*distinct-cdcl*$_W$*-state S*› **and**
learned: ‹$cdcl_W$*-learned-clause S*› **and**
alien: ‹*no-strange-atm S*› **and**
termi: ‹*no-step cdcl*$_W$*-stgy S*›
shows *False*
⟨*proof*⟩

lemma $cdcl_W$*-stgy-final-state-conclusive2*:

assumes
termi: *no-step cdcl$_W$-stgy S* **and**
decomp: *all-decomposition-implies-m* (*clauses S*) (*get-all-ann-decomposition* (*trail S*)) **and**
learned: *cdcl$_W$-learned-clause S* **and**
level-inv: *cdcl$_W$-M-level-inv S* **and**
alien: *no-strange-atm S* **and**
no-dup: *distinct-cdcl$_W$-state S* **and**
confl: *cdcl$_W$-conflicting S* **and**
confl-k: *conflict-is-false-with-level S*
shows (*conflicting S* = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*clauses S*)))
∨ (*conflicting S* = *None* ∧ *trail S* ⊨*as set-mset* (*clauses S*))
⟨*proof*⟩

lemma *cdcl$_W$-stgy-final-state-conclusive*:
assumes
termi: *no-step cdcl$_W$-stgy S* **and**
decomp: *all-decomposition-implies-m* (*clauses S*) (*get-all-ann-decomposition* (*trail S*)) **and**
learned: *cdcl$_W$-learned-clause S* **and**
level-inv: *cdcl$_W$-M-level-inv S* **and**
alien: *no-strange-atm S* **and**
no-dup: *distinct-cdcl$_W$-state S* **and**
confl: *cdcl$_W$-conflicting S* **and**
confl-k: *conflict-is-false-with-level S* **and**
learned-entailed: ‹*cdcl$_W$-learned-clauses-entailed-by-init S*›
shows (*conflicting S* = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss S*)))
∨ (*conflicting S* = *None* ∧ *trail S* ⊨*as set-mset* (*init-clss S*))
⟨*proof*⟩

lemma *cdcl$_W$-stgy-tranclp-cdcl$_W$-restart*:
cdcl$_W$-stgy S S′ ⟹ *cdcl$_W$-restart*$^{++}$ *S S′*
⟨*proof*⟩

lemma *tranclp-cdcl$_W$-stgy-tranclp-cdcl$_W$-restart*:
cdcl$_W$-stgy^{++} *S S′* ⟹ *cdcl$_W$-restart*$^{++}$ *S S′*
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-stgy-rtranclp-cdcl$_W$-restart*:
cdcl$_W$-stgy** *S S′* ⟹ *cdcl$_W$-restart*** *S S′*
⟨*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 *propagate-high-levelE*:
assumes *propagate S T*
obtains *M′ N′ U L C* **where**

state-butlast S = (*M′*, *N′*, *U*, *None*) **and**
state-butlast T = (*Propagated L* (*C* + {#*L*#}) # *M′*, *N′*, *U*, *None*) **and**
C + {#*L*#} ∈# *local.clauses S* **and**
M′ ⊨*as CNot C* **and**
undefined-lit (*trail S*) *L*
⟨*proof*⟩

lemma *cdcl*$_W$*-propagate-conflict-completeness*:
assumes
MN: *set M* ⊨*s set-mset N* **and**
cons: *consistent-interp* (*set M*) **and**
tot: *total-over-m* (*set M*) (*set-mset N*) **and**
lits-of-l (*trail S*) ⊆ *set M* **and**
init-clss S = *N* **and**
propagate** *S S′* **and**
learned-clss S = {#}
shows *length* (*trail S*) ≤ *length* (*trail S′*) ∧ *lits-of-l* (*trail S′*) ⊆ *set M*
⟨*proof*⟩

lemma
assumes *propagate*** *S X*
shows
rtranclp-propagate-init-clss: *init-clss X* = *init-clss S* **and**
rtranclp-propagate-learned-clss: *learned-clss X* = *learned-clss S*
⟨*proof*⟩

lemma *cdcl*$_W$*-stgy-strong-completeness-n*:
assumes
MN: *set M* ⊨*s set-mset N* **and**
cons: *consistent-interp* (*set M*) **and**
tot: *total-over-m* (*set M*) (*set-mset N*) **and**
atm-incl: *atm-of* ‘ (*set M*) ⊆ *atms-of-mm N* **and**
distM: *distinct M* **and**
length: *n* ≤ *length M*
shows
∃ *M′ S*. *length M′* ≥ *n* ∧
lits-of-l M′ ⊆ *set M* ∧
no-dup M′ ∧
state-butlast S = (*M′*, *N*, {#}, *None*) ∧
cdcl$_W$*-stgy*** (*init-state N*) *S*
⟨*proof*⟩

lemma *cdcl*$_W$*-stgy-strong-completeness′*:
assumes
MN: *set M* ⊨*s set-mset N* **and**
cons: *consistent-interp* (*set M*) **and**
tot: *total-over-m* (*set M*) (*set-mset N*) **and**
atm-incl: *atm-of* ‘ (*set M*) ⊆ *atms-of-mm N* **and**
distM: *distinct M*
shows
∃ *M′ S*. *lits-of-l M′* = *set M* ∧
state-butlast S = (*M′*, *N*, {#}, *None*) ∧
cdcl$_W$*-stgy*** (*init-state N*) *S*
⟨*proof*⟩

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

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

No conflict with only variables of level less than backtrack level

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

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

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

lemma *cdcl$_W$-o-no-smaller-confl-inv*:
  fixes *S S'* :: '*st*
  assumes
    cdcl$_W$-o S S' **and**
    n-s: *no-step conflict S* **and**
    lev: *cdcl$_W$-M-level-inv S* **and**
    max-lev: *conflict-is-false-with-level S* **and**
    smaller: *no-smaller-confl S*
  shows *no-smaller-confl S'*
  ⟨*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$_W$-stgy-no-smaller-confl*:
  assumes *cdcl$_W$-stgy S S'*
  and *n-l*: *no-smaller-confl S*
  and *conflict-is-false-with-level S*
  and *cdcl$_W$-M-level-inv S*
  shows *no-smaller-confl S'*
  ⟨*proof*⟩

lemma *conflict-conflict-is-false-with-level*:
assumes
conflict: *conflict S T* **and**
smaller: *no-smaller-confl S* **and**
M-lev: *cdcl$_W$-M-level-inv S*
shows *conflict-is-false-with-level T*
⟨*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**
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**
dist: *distinct-cdcl$_W$-state S* **and**
conflicting: *cdcl$_W$-conflicting S* **and**
decomp: *all-decomposition-implies-m* (*clauses S*) (*get-all-ann-decomposition* (*trail S*)) **and**
learned: *cdcl$_W$-learned-clause S* **and**
alien: *no-strange-atm S*
shows *no-smaller-confl S′* ∧ *conflict-is-false-with-level S′*
⟨*proof*⟩

Final States are Conclusive

theorem 2.9.9 page 97 of Weidenbach's book

lemma *full-cdcl$_W$-stgy-final-state-conclusive*:
fixes *S′* :: *'st*
assumes *full*: *full cdcl$_W$-stgy* (*init-state N*) *S′*
and *no-d*: *distinct-mset-mset N*
shows (*conflicting S′* = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss S′*)))
∨ (*conflicting S′* = *None* ∧ *trail S′* ⊨*asm init-clss S′*)
⟨*proof*⟩

lemma *cdcl$_W$-o-fst-empty-conflicting-false*:
assumes
cdcl$_W$-o S S′ **and**
trail S = [] **and**
conflicting S ≠ *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*⟩

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*⟩

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

declare $conflict\text{-}or\text{-}propagate\text{-}def[simp]$

lemma $conflict\text{-}or\text{-}propagate\text{-}intros$:
$conflict$ S $T \Longrightarrow$ $conflict\text{-}or\text{-}propagate$ S T
$propagate$ S $T \Longrightarrow$ $conflict\text{-}or\text{-}propagate$ S T
⟨*proof*⟩

theorem 2.9.9 page 97 of Weidenbach's book

lemma full-$cdcl_W$-stgy-final-state-conclusive-from-init-state:
fixes $S' :: 'st$
assumes *full*: $full$ $cdcl_W$-stgy $(init\text{-}state$ $N)$ 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*⟩

1.1.6 Structural Invariant

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

The invariant contains all the structural invariants that holds,

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

lemma $cdcl_W$-all-struct-inv-inv:
assumes $cdcl_W$-restart S S' **and** $cdcl_W$-all-struct-inv S
shows $cdcl_W$-all-struct-inv S'

⟨*proof*⟩

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

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

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

lemma *beginning-not-decided-invert*:
assumes *A*: *M* @ *A* = *M'* @ *Decided K* # *H* **and**
nm: ∀ *m*∈*set M*. ¬*is-decided m*
shows ∃ *M*. *A* = *M* @ *Decided K* # *H*
⟨*proof*⟩

1.1.7 Strategy-Specific Invariant

definition *cdcl$_W$-stgy-invariant* **where**
cdcl$_W$-stgy-invariant S ⟷
conflict-is-false-with-level S
∧ *no-smaller-confl S*

lemma *cdcl$_W$-stgy-cdcl$_W$-stgy-invariant*:
assumes
cdcl$_W$-restart: *cdcl$_W$-stgy S T* **and**
inv-s: *cdcl$_W$-stgy-invariant S* **and**
inv: *cdcl$_W$-all-struct-inv S*
shows
cdcl$_W$-stgy-invariant T
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-stgy-cdcl$_W$-stgy-invariant*:
assumes
cdcl$_W$-restart: *cdcl$_W$-stgy*** *S T* **and**
inv-s: *cdcl$_W$-stgy-invariant S* **and**
inv: *cdcl$_W$-all-struct-inv S*
shows
cdcl$_W$-stgy-invariant T
⟨*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* **and**
learned-entailed: ‹*cdcl$_W$-learned-clauses-entailed-by-init S*›
shows *conflicting T* = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss S*))
∨ *conflicting T* = *None* ∧ *trail T* ⊨*asm init-clss S* ∧ *satisfiable* (*set-mset* (*init-clss S*))
⟨*proof*⟩

lemma *full-cdcl$_W$-stgy-inv-normal-form2*:
assumes
full: *full cdcl$_W$-stgy S T* **and**
inv-s: *cdcl$_W$-stgy-invariant S* **and**
inv: *cdcl$_W$-all-struct-inv S*
shows *conflicting T = Some {#} ∧ unsatisfiable (set-mset (clauses T))*
∨ conflicting T = None ∧ satisfiable (set-mset (clauses T))
⟨*proof*⟩

1.1.8 Additional Invariant: No Smaller Propagation

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

lemma *propagated-cons-eq-append-decide-cons*:
Propagated L E # Ms = M' @ Decided K # M ⟷
M' ≠ [] ∧ Ms = tl M' @ Decided K # M ∧ hd M' = Propagated L E
⟨*proof*⟩

lemma *in-get-all-mark-of-propagated-in-trail*:
‹*C ∈ set (get-all-mark-of-propagated M) ⟷ (∃ L. Propagated L C ∈ set M)*›
⟨*proof*⟩

lemma *no-smaller-propa-tl*:
assumes
‹*no-smaller-propa S*› **and**
‹*trail S ≠ []*› **and**
‹*¬is-decided(hd-trail S)*› **and**
‹*trail U = tl (trail S)*› **and**
‹*clauses U = clauses S*›
shows
‹*no-smaller-propa U*›
⟨*proof*⟩

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

lemma *decide-no-smaller-step*:
assumes *dec*: ‹*decide S T*› **and** *smaller-propa*: ‹*no-smaller-propa S*› **and**
n-s: ‹*no-step propagate S*›
shows ‹*no-smaller-propa T*›
⟨*proof*⟩

lemma *no-smaller-propa-reduce-trail-to*:
‹*no-smaller-propa S ⟹ no-smaller-propa (reduce-trail-to M1 S)*›
⟨*proof*⟩

lemma *backtrackg-no-smaller-propa*:
assumes *o*: ‹*backtrackg S T*› **and** *smaller-propa*: ‹*no-smaller-propa S*› **and**
n-d: ‹*no-dup (trail S)*› **and**
n-s: ‹*no-step propagate S*› **and**
tr-CNot: ‹*trail S ⊨as CNot (the (conflicting S))*›

shows ‹*no-smaller-propa T*›
⟨*proof*⟩

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

lemma *cdcl$_W$-stgy-no-smaller-propa*:
assumes
cdcl: ‹*cdcl$_W$-stgy S T*› **and**
smaller-propa: ‹*no-smaller-propa S*› **and**
inv: ‹*cdcl$_W$-all-struct-inv S*›
shows ‹*no-smaller-propa T*›
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-stgy-no-smaller-propa*:
assumes
cdcl: ‹*cdcl$_W$-stgy** S T*› **and**
smaller-propa: ‹*no-smaller-propa S*› **and**
inv: ‹*cdcl$_W$-all-struct-inv S*›
shows ‹*no-smaller-propa T*›
⟨*proof*⟩

lemma *hd-trail-level-ge-1-length-gt-1*:
fixes *S* :: *'st*
defines *M*[*symmetric*, *simp*]: ‹*M* ≡ *trail S*›
defines *L*[*symmetric*, *simp*]: ‹*L* ≡ *hd M*›
assumes
smaller: ‹*no-smaller-propa S*› **and**
struct: ‹*cdcl$_W$-all-struct-inv S*› **and**
dec: ‹*count-decided M* ≥ *1*› **and**
proped: ‹*is-proped L*›
shows ‹*size* (*mark-of L*) > *1*›
⟨*proof*⟩

1.1.9 More Invariants: Conflict is False if no decision

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

definition *conflict-non-zero-unless-level-0* :: ‹*'st* ⇒ *bool*› **where**
‹*conflict-non-zero-unless-level-0 S* ⟷
(*conflicting S* = *Some* {#} ⟶ *count-decided* (*trail S*) = *0*)›

definition *no-false-clause*:: ‹*'st* ⇒ *bool*› **where**
‹*no-false-clause S* ⟷ (∀ *C* ∈# *clauses S*. *C* ≠ {#})›

lemma *cdcl$_W$-restart-no-false-clause*:
assumes
‹*cdcl$_W$-restart S T*›
‹*no-false-clause S*›
shows ‹*no-false-clause T*›
⟨*proof*⟩

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

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

⟨*no-false-clause S*⟩ **and**
⟨*conflict-non-zero-unless-level-0 S*⟩
shows ⟨*conflict-non-zero-unless-level-0 T*⟩
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-restart-no-false-clause*:
assumes
⟨*cdcl$_W$-restart** S T*⟩
⟨*no-false-clause S*⟩
shows ⟨*no-false-clause T*⟩
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-restart-conflict-non-zero-unless-level-0*:
assumes
⟨*cdcl$_W$-restart** S T*⟩
⟨*no-false-clause S*⟩ **and**
⟨*conflict-non-zero-unless-level-0 S*⟩
shows ⟨*conflict-non-zero-unless-level-0 T*⟩
⟨*proof*⟩

definition *propagated-clauses-clauses* :: *'st* ⇒ *bool* **where**
⟨*propagated-clauses-clauses S* ≡ ∀ *L K. Propagated L K* ∈ *set* (*trail S*) ⟶ *K* ∈# *clauses S*⟩

lemma *propagate-single-literal-clause-get-level-is-0*:
assumes
smaller: ⟨*no-smaller-propa S*⟩ **and**
propa-tr: ⟨*Propagated L* {#*L*#} ∈ *set* (*trail S*)⟩ **and**
n-d: ⟨*no-dup* (*trail S*)⟩ **and**
propa: ⟨*propagated-clauses-clauses S*⟩
shows ⟨*get-level* (*trail S*) *L* = *0*⟩
⟨*proof*⟩

Conflict Minimisation

Remove Literals of Level 0 **lemma** *conflict-minimisation-level-0*:
fixes *S* :: *'st*
defines *D*[*simp*]: ⟨*D* ≡ *the* (*conflicting S*)⟩
defines [*simp*]: ⟨*M* ≡ *trail S*⟩
defines ⟨*D'* ≡ *filter-mset* (λ*L. get-level M L* > *0*) *D*⟩
assumes
ns-s: ⟨*no-step skip S*⟩ **and**
ns-r: ⟨*no-step resolve S*⟩ **and**
inv-s: *cdcl$_W$-stgy-invariant S* **and**
inv: *cdcl$_W$-all-struct-inv S* **and**
conf: ⟨*conflicting S* ≠ *None*⟩ ⟨*conflicting S* ≠ *Some* {#}⟩ **and**
M-nempty: ⟨*M* ~= []⟩
shows
clauses S ⊨*pm D'* **and**
⟨− *lit-of* (*hd M*) ∈# *D'*⟩
⟨*proof*⟩

lemma *literals-of-level0-entailed*:
assumes
struct-invs: ⟨*cdcl$_W$-all-struct-inv S*⟩ **and**
in-trail: ⟨*L* ∈ *lits-of-l* (*trail S*)⟩ **and**
lev: ⟨*get-level* (*trail S*) *L* = *0*⟩

shows
‹*clauses S* ⊨*pm* {#*L*#}›
⟨*proof*⟩

1.1.10 Some higher level use on the invariants

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

lemma *conflict-conflict-is-false-with-level-all-inv*:
‹*conflict S T* ⟹
no-smaller-confl S ⟹
cdcl$_W$*-all-struct-inv S* ⟹
conflict-is-false-with-level T›
⟨*proof*⟩

lemma *cdcl*$_W$*-stgy-ex-lit-of-max-level-all-inv*:
assumes
cdcl$_W$*-stgy S S′* **and**
n-l: *no-smaller-confl S* **and**
conflict-is-false-with-level S **and**
cdcl$_W$*-all-struct-inv S*
shows *conflict-is-false-with-level S′*
⟨*proof*⟩

lemma *cdcl*$_W$*-o-conflict-is-false-with-level-inv-all-inv*:
assumes
‹*cdcl*$_W$*-o S T*›
‹*cdcl*$_W$*-all-struct-inv S*›
‹*conflict-is-false-with-level S*›
shows ‹*conflict-is-false-with-level T*›
⟨*proof*⟩

lemma *no-step-cdcl*$_W$*-total*:
assumes
‹*no-step cdcl*$_W$ *S*›
‹*conflicting S* = *None*›
‹*no-strange-atm S*›
shows ‹*total-over-m* (*lits-of-l* (*trail S*)) (*set-mset* (*clauses S*))›
⟨*proof*⟩

lemma *cdcl*$_W$*-Ex-cdcl*$_W$*-stgy*:
assumes
‹*cdcl*$_W$ *S T*›
shows ‹*Ex*(*cdcl*$_W$*-stgy S*)›
⟨*proof*⟩

lemma *no-step-skip-hd-in-conflicting*:
assumes
inv-s: ‹*cdcl*$_W$*-stgy-invariant S*› **and**
inv: ‹*cdcl*$_W$*-all-struct-inv S*› **and**
ns: ‹*no-step skip S*› **and**
confl: ‹*conflicting S* ≠ *None*› ‹*conflicting S* ≠ *Some* {#}›

shows ‹$-$*lit-of* (*hd* (*trail S*)) ∈# *the* (*conflicting S*)›
⟨*proof*⟩

lemma
fixes *S*
assumes
nss: ‹*no-step skip S*› **and**
nsr: ‹*no-step resolve S*› **and**
invs: ‹*cdcl*$_W$*-all-struct-inv S*› **and**
stgy: ‹*cdcl*$_W$*-stgy-invariant S*› **and**
confl: ‹*conflicting S* ≠ *None*› **and**
confl': ‹*conflicting S* ≠ *Some* {#}›
shows *no-skip-no-resolve-single-highest-level*:
‹*the* (*conflicting S*) =
add-mset (−(*lit-of* (*hd* (*trail S*)))) {#*L* ∈# *the* (*conflicting S*).
get-level (*trail S*) *L* < *local.backtrack-lvl S*#}› (**is** *?A*) **and**
no-skip-no-resolve-level-lvl-nonzero:
‹0 < *backtrack-lvl S*› (**is** *?B*) **and**
no-skip-no-resolve-level-get-maximum-lvl-le:
‹*get-maximum-level* (*trail S*) (*remove1-mset* (−(*lit-of* (*hd* (*trail S*)))) (*the* (*conflicting S*)))
< *backtrack-lvl S*› (**is** *?C*)
⟨*proof*⟩

end

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

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

1.1.11 Termination

No Relearning of a clause

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

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

lemma *cdcl*$_W$*-stgy-no-relearned-clause*:
assumes
cdcl: ‹*backtrack S T*› **and**
inv: ‹*cdcl*$_W$*-all-struct-inv S*› **and**
smaller: ‹*no-smaller-propa S*›
shows
‹*mark-of* (*hd-trail T*) ∉# *clauses S*›
⟨*proof*⟩

lemma *cdcl*$_W$*-stgy-no-relearned-larger-clause*:
assumes
cdcl: ‹*backtrack S T*› **and**
inv: ‹*cdcl*$_W$*-all-struct-inv S*› **and**
smaller: ‹*no-smaller-propa S*› **and**

smaller-conf: ‹*no-smaller-confl S*› **and**
E-subset: ‹*E* ⊂# *mark-of* (*hd-trail T*)›
shows ‹*E* ∉# *clauses S*›
⟨*proof*⟩

lemma *cdcl*$_W$*-stgy-no-relearned-highest-subres-clause*:
assumes
cdcl: ‹*backtrack S T*› **and**
inv: ‹*cdcl*$_W$*-all-struct-inv S*› **and**
smaller: ‹*no-smaller-propa S*› **and**
smaller-conf: ‹*no-smaller-confl S*› **and**
E-subset: ‹*mark-of* (*hd-trail T*) = *add-mset* (*lit-of* (*hd-trail T*)) *E*›
shows ‹*add-mset* (− *lit-of* (*hd-trail T*)) *E* ∉# *clauses S*›
⟨*proof*⟩

lemma *cdcl*$_W$*-stgy-distinct-mset*:
assumes
cdcl: ‹*cdcl*$_W$*-stgy S T*› **and**
inv: *cdcl*$_W$*-all-struct-inv S* **and**
smaller: ‹*no-smaller-propa S*› **and**
dist: ‹*distinct-mset* (*clauses S*)›
shows
‹*distinct-mset* (*clauses T*)›
⟨*proof*⟩

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

lemma *cdcl*$_W$*-stgy-learned-distinct-mset*:
assumes
cdcl: ‹*cdcl*$_W$*-stgy S T*› **and**
inv: *cdcl*$_W$*-all-struct-inv S* **and**
smaller: ‹*no-smaller-propa S*› **and**
dist: ‹*distinct-mset* (*learned-clss S* + *remdups-mset* (*init-clss S*))›
shows
‹*distinct-mset* (*learned-clss T* + *remdups-mset* (*init-clss T*))›
⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-stgy-distinct-mset-clauses*:
assumes
st: *cdcl*$_W$*-stgy*** *R S* **and**
invR: *cdcl*$_W$*-all-struct-inv R* **and**
dist: *distinct-mset* (*clauses R*) **and**
no-smaller: ‹*no-smaller-propa R*›
shows *distinct-mset* (*clauses S*)
⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-stgy-distinct-mset-learned-clauses*:
assumes
st: *cdcl*$_W$*-stgy*** *R S* **and**
invR: *cdcl*$_W$*-all-struct-inv R* **and**
dist: *distinct-mset* (*learned-clss R* + *remdups-mset* (*init-clss R*)) **and**
no-smaller: ‹*no-smaller-propa R*›
shows *distinct-mset* (*learned-clss S* + *remdups-mset* (*init-clss S*))
⟨*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*⟩

lemma *cdcl$_W$-stgy-learned-distinct-mset-new*:
  assumes
    cdcl: ‹*cdcl$_W$-stgy S T*› **and**
    inv: *cdcl$_W$-all-struct-inv S* **and**
    smaller: ‹*no-smaller-propa S*› **and**
    dist: ‹*distinct-mset* (*learned-clss S* − *A*)›
  shows ‹*distinct-mset* (*learned-clss T* − *A*)›
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-stgy-distinct-mset-clauses-new-abs*:
  assumes
    st: *cdcl$_W$-stgy** R S* **and**
    invR: *cdcl$_W$-all-struct-inv R* **and**
    no-smaller: ‹*no-smaller-propa R*› **and**
    ‹*distinct-mset* (*learned-clss R* − *A*)›
  shows *distinct-mset* (*learned-clss S* − *A*)
  ⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-stgy-distinct-mset-clauses-new*:
  assumes
    st: *cdcl$_W$-stgy** R S* **and**
    invR: *cdcl$_W$-all-struct-inv R* **and**
    no-smaller: ‹*no-smaller-propa R*›
  shows *distinct-mset* (*learned-clss S* − *learned-clss R*)
  ⟨*proof*⟩

Decrease of a Measure

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

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

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

⟨*proof*⟩

lemma *learned-clss-less-upper-bound*:
fixes $S :: 'st$
assumes
distinct-cdcl$_W$-state S **and**
$\forall s \in\# \textit{learned-clss}\ S.\ \neg \textit{tautology}\ s$
shows $\textit{card}(\textit{set-mset}\ (\textit{learned-clss}\ S)) \leq 3 \mathbin{\widehat{}} \textit{card}\ (\textit{atms-of-mm}\ (\textit{learned-clss}\ S))$
⟨*proof*⟩

lemma *cdcl$_W$-restart-measure-decreasing*:
fixes $S :: 'st$
assumes
cdcl$_W$-restart $S\ S'$ **and**
no-restart:
$\neg(\textit{learned-clss}\ S \subseteq\# \textit{learned-clss}\ S' \wedge [] = \textit{trail}\ S' \wedge \textit{conflicting}\ S' = \textit{None})$
and
no-forget: $\textit{learned-clss}\ S \subseteq\# \textit{learned-clss}\ S'$ **and**
no-relearn: $\bigwedge S'.\ \textit{backtrack}\ S\ S' \Longrightarrow \textit{mark-of}\ (\textit{hd-trail}\ S') \notin\# \textit{learned-clss}\ S$
and
alien: *no-strange-atm* S **and**
M-level: *cdcl$_W$-M-level-inv* S **and**
no-taut: $\forall s \in\# \textit{learned-clss}\ S.\ \neg \textit{tautology}\ s$ **and**
no-dup: *distinct-cdcl$_W$-state* S **and**
confl: *cdcl$_W$-conflicting* S
shows $(\textit{cdcl}_W\textit{-restart-measure}\ S',\ \textit{cdcl}_W\textit{-restart-measure}\ S) \in \textit{lexn less-than}\ 3$
⟨*proof*⟩

lemma *cdcl$_W$-stgy-step-decreasing*:
fixes $S\ T :: 'st$
assumes
cdcl: ‹*cdcl$_W$-stgy* $S\ T$› **and**
struct-inv: ‹*cdcl$_W$-all-struct-inv* S› **and**
smaller: ‹*no-smaller-propa* S›
shows $(\textit{cdcl}_W\textit{-restart-measure}\ T,\ \textit{cdcl}_W\textit{-restart-measure}\ S) \in \textit{lexn less-than}\ 3$
⟨*proof*⟩

lemma *empty-trail-no-smaller-propa*: ‹$\textit{trail}\ R = [] \Longrightarrow \textit{no-smaller-propa}\ R$›
⟨*proof*⟩

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

lemma *tranclp-cdcl$_W$-stgy-decreasing*:
fixes $R\ S\ T :: 'st$
assumes *cdcl$_W$-stgy*$^{++}$ $R\ S$ **and**
tr: *trail* $R = []$ **and**
cdcl$_W$-all-struct-inv R
shows $(\textit{cdcl}_W\textit{-restart-measure}\ S,\ \textit{cdcl}_W\textit{-restart-measure}\ R) \in \textit{lexn less-than}\ 3$
⟨*proof*⟩

lemma *tranclp-cdcl$_W$-stgy-S0-decreasing*:
fixes $R\ S\ T :: 'st$
assumes
pl: *cdcl$_W$-stgy*$^{++}$ $(\textit{init-state}\ N)\ S$ **and**
no-dup: *distinct-mset-mset* N

shows (*cdcl$_W$-restart-measure S*, *cdcl$_W$-restart-measure* (*init-state N*)) ∈ *lexn less-than 3*
⟨*proof*⟩

lemma *wf-tranclp-cdcl$_W$-stgy*:
wf {(*S*::'*st*, *init-state N*)| *S N*. *distinct-mset-mset N* ∧ *cdcl$_W$-stgy*$^{++}$ (*init-state N*) *S*}
⟨*proof*⟩

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

lemma *no-conflict-after-decide*:
assumes
dec: ‹*decide S T*› **and**
inv: ‹*cdcl$_W$-all-struct-inv T*› **and**
smaller: ‹*no-smaller-propa T*› **and**
smaller-confl: ‹*no-smaller-confl T*›
shows ‹¬*conflict T U*›
⟨*proof*⟩

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

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

lemma *comp-list-weight-propa-trail-append*[*simp*]:
‹*comp-list-weight-propa-trail b* (*M* @ *M'*) =
comp-list-weight-propa-trail (*b* − *length M'*) *M* @ *list-weight-propa-trail M'*›
⟨*proof*⟩

lemma *comp-list-weight-propa-trail-append-single*[*simp*]:
‹*comp-list-weight-propa-trail b* (*M* @ [*K*]) =
comp-list-weight-propa-trail (*b* − *1*) *M* @ [*is-proped K*]›
⟨*proof*⟩

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

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

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

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

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

⟨*proof*⟩

lemma *of-list-weight-replicate-True*[*simp*]: ‹*of-list-weight* (*replicate n True*) = 2^n − 1›
⟨*proof*⟩

lemma *of-list-weight-le*: ‹*of-list-weight xs* ≤ 2^(*length xs*) − 1›
⟨*proof*⟩

lemma *of-list-weight-lt*: ‹*of-list-weight xs* < 2^(*length xs*)›
⟨*proof*⟩

lemma [*simp*]: ‹*of-list-weight* (*comp-list-weight-propa-trail n* []) = 0›
⟨*proof*⟩

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

lemma *length-comp-list-weight-propa-trail*[*simp*]: ‹*length* (*comp-list-weight-propa-trail a M*) = *max* (*length M*) *a*›
⟨*proof*⟩

lemma (**in** −)*pow2-times-n*:
‹*Suc a* ≤ *n* ⟹ 2 * 2^(*n* − *Suc a*) = (2::*nat*)^ (*n* − *a*)›
‹*Suc a* ≤ *n* ⟹ 2^(*n* − *Suc a*) * 2 = (2::*nat*)^ (*n* − *a*)›
‹*Suc a* ≤ *n* ⟹ 2^(*n* − *Suc a*) * *b* * 2 = (2::*nat*)^ (*n* − *a*) * *b*›
‹*Suc a* ≤ *n* ⟹ 2^(*n* − *Suc a*) * (*b* * 2) = (2::*nat*)^ (*n* − *a*) * *b*›
‹*Suc a* ≤ *n* ⟹ 2^(*n* − *Suc a*) * (2 * *b*) = (2::*nat*)^ (*n* − *a*) * *b*›
‹*Suc a* ≤ *n* ⟹ 2 * *b* * 2^(*n* − *Suc a*) = (2::*nat*)^ (*n* − *a*) * *b*›
‹*Suc a* ≤ *n* ⟹ 2 * (*b* * 2^(*n* − *Suc a*)) = (2::*nat*)^ (*n* − *a*) * *b*›
⟨*proof*⟩

lemma *decide-propa-weight*:
‹*decide S T* ⟹ *n* ≥ *length* (*trail T*) ⟹ *propa-weight n* (*trail S*) ≤ *propa-weight n* (*trail T*)›
⟨*proof*⟩

lemma *propagate-propa-weight*:
‹*propagate S T* ⟹ *n* ≥ *length* (*trail T*) ⟹ *propa-weight n* (*trail S*) < *propa-weight n* (*trail T*)›
⟨*proof*⟩

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

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

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

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

Then the simple part:

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

Comments on the proof:

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

  Comments on the bound:

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

lemma *cdcl-pow2-n-learned-clauses*:
  assumes
    cdcl: ‹$cdcl_W{}^{**}\ S\ T$› **and**
    confl: ‹*conflicting* $S = None$› **and**
    inv: ‹$cdcl_W$*-all-struct-inv* S›
  shows ‹*size* (*learned-clss* T) $\leq$ *size* (*learned-clss* S) + $2\ \widehat{}\ (card\ (atms\text{-}of\text{-}mm\ (init\text{-}clss\ S)))$›
    (**is** ‹- $\leq$ - + *?b*›)
⟨*proof*⟩

Application of the previous theorem to an initial state:

corollary *cdcl-pow2-n-learned-clauses2*:
  assumes
    cdcl: ‹$cdcl_W{}^{**}$ (*init-state* N) T› **and**
    inv: ‹$cdcl_W$*-all-struct-inv* (*init-state* N)›
  shows ‹*size* (*learned-clss* T) $\leq$ $2\ \widehat{}\ (card\ (atms\text{-}of\text{-}mm\ N))$›
  ⟨*proof*⟩

end

end

1.2 Merging backjump rules

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

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

1. $conflict\text{-}driven\text{-}clause\text{-}learning_W.conflict$ to find the conflict

2. the conflict is analysed by repetitive application of *conflict-driven-clause-learning$_W$.resolve* and *conflict-driven-clause-learning$_W$.skip*,

3. finally *conflict-driven-clause-learning$_W$.backtrack* is used to backtrack.

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

1.2.1 Inclusion of the States

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

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

lemma *backtrack-no-cdcl$_W$-bj*:
  assumes *cdcl*: *cdcl$_W$-bj T U*
  shows ¬*backtrack V T*
  ⟨*proof*⟩

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

inductive *skip-or-resolve* :: *'st* ⇒ *'st* ⇒ *bool* **where**
s-or-r-skip[*intro*]: *skip S T* ⟹ *skip-or-resolve S T* |
s-or-r-resolve[*intro*]: *resolve S T* ⟹ *skip-or-resolve S T*

lemma *rtranclp-cdcl$_W$-bj-skip-or-resolve-backtrack*:
  assumes *cdcl$_W$-bj*** *S U*
  shows *skip-or-resolve*** *S U* ∨ (∃ *T*. *skip-or-resolve*** *S T* ∧ *backtrack T U*)
  ⟨*proof*⟩

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

definition *backjump-l-cond* :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool* **where**
backjump-l-cond ≡ λ*C C' L S T*. *True*

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

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

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

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

1.2.2 More lemmas about Conflict, Propagate and Backjumping

Termination

lemma *cdcl$_W$-bj-measure*:
assumes *cdcl$_W$-bj S T*
shows *length (trail S) + (if conflicting S = None then 0 else 1)*
> length (trail T) + (if conflicting T = None then 0 else 1)
⟨*proof*⟩

lemma *wf-cdcl$_W$-bj*:
wf {(b,a). cdcl$_W$-bj a b}
⟨*proof*⟩

lemma *cdcl$_W$-bj-exists-normal-form*:
shows ∃ *T. full cdcl$_W$-bj S T*
⟨*proof*⟩

lemma *rtranclp-skip-state-decomp*:
assumes *skip** S T*
shows
∃ *M. trail S = M @ trail T* ∧ (∀ *m*∈*set M*. ¬*is-decided m*)
init-clss S = init-clss T
learned-clss S = learned-clss T
backtrack-lvl S = backtrack-lvl T
conflicting S = conflicting T
⟨*proof*⟩

Analysing is confluent

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

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

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: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$
shows $backtrack\ T\ W$
⟨*proof*⟩

lemma $cdcl_W\text{-}bj\text{-}decomp\text{-}resolve\text{-}skip\text{-}and\text{-}bj$:
assumes $cdcl_W\text{-}bj^{**}\ S\ T$
shows ($skip\text{-}or\text{-}resolve^{**}\ S\ T$
$\vee\ (\exists\ U.\ skip\text{-}or\text{-}resolve^{**}\ S\ U \wedge backtrack\ U\ T)$)
⟨*proof*⟩

1.2.3 CDCL with Merging

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

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

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

lemma $cdcl_W\text{-}merge\text{-}restart\text{-}conflicting\text{-}true\text{-}or\text{-}no\text{-}step$:
assumes $cdcl_W\text{-}merge\text{-}restart\ S\ T$
shows $conflicting\ T = None \vee no\text{-}step\ cdcl_W\text{-}restart\ T$
⟨*proof*⟩

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

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

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

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

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

lemma *cdcl$_W$-all-struct-inv-tranclp-cdcl$_W$-merge-tranclp-cdcl$_W$-merge-cdcl$_W$-all-struct-inv*:
assumes
inv: *cdcl$_W$-all-struct-inv b*
cdcl$_W$-merge^{++} b a
shows $(\lambda S\ T.\ cdcl_W\text{-}all\text{-}struct\text{-}inv\ S \wedge cdcl_W\text{-}merge\ S\ T)^{++}\ b\ a$
⟨*proof*⟩

lemma *backtrack-is-full1-cdcl$_W$-bj*:
assumes *bt*: *backtrack S T*
shows *full1 cdcl$_W$-bj S T*
⟨*proof*⟩

lemma *rtrancl-cdcl$_W$-conflicting-true-cdcl$_W$-merge-restart*:
assumes *cdcl$_W$-restart** S V* **and** *inv*: *cdcl$_W$-M-level-inv S* **and** *conflicting S = None*
shows (*cdcl$_W$-merge-restart** S V* ∧ *conflicting V = None*)
∨ (∃ *T U*. *cdcl$_W$-merge-restart** S T* ∧ *conflicting V* ≠ *None* ∧ *conflict T U* ∧ *cdcl$_W$-bj** U V*)
⟨*proof*⟩

lemma *no-step-cdcl$_W$-restart-no-step-cdcl$_W$-merge-restart*:
no-step cdcl$_W$-restart S ⟹ *no-step cdcl$_W$-merge-restart S*
⟨*proof*⟩

lemma *no-step-cdcl$_W$-merge-restart-no-step-cdcl$_W$-restart*:
assumes
conflicting S = None **and**
cdcl$_W$-M-level-inv S **and**
no-step cdcl$_W$-merge-restart S
shows *no-step cdcl$_W$-restart S*
⟨*proof*⟩

lemma *cdcl$_W$-merge-restart-no-step-cdcl$_W$-bj*:
assumes
cdcl$_W$-merge-restart S T
shows *no-step cdcl$_W$-bj T*
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-merge-restart-no-step-cdcl$_W$-bj*:
assumes
cdcl$_W$-merge-restart** S T **and**
conflicting S = None
shows *no-step cdcl$_W$-bj T*
⟨*proof*⟩

If *conflicting S* ≠ *None*, we cannot say anything.

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

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

lemma *init-state-true-full-cdcl$_W$-restart-iff-full-cdcl$_W$-merge*:
shows *full cdcl$_W$-restart* (*init-state N*) *V* ⟷ *full cdcl$_W$-merge-restart* (*init-state N*) *V*
⟨*proof*⟩

1.2.4 CDCL with Merge and Strategy

The intermediate step

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

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

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

lemma *cdcl$_W$-s'-is-rtranclp-cdcl$_W$-stgy*:
$cdcl_W\text{-}s'\ S\ T \Longrightarrow cdcl_W\text{-}stgy^{**}\ S\ T$
⟨*proof*⟩

lemma *cdcl$_W$-stgy-cdcl$_W$-s'-no-step*:
assumes $cdcl_W\text{-}stgy\ S\ U$ **and** $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$ **and** $no\text{-}step\ cdcl_W\text{-}bj\ U$
shows $cdcl_W\text{-}s'\ S\ U$
⟨*proof*⟩

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

lemma *n-step-cdcl$_W$-stgy-iff-no-step-cdcl$_W$-restart-cl-cdcl$_W$-o*:
assumes *inv*: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$
shows $no\text{-}step\ cdcl_W\text{-}s'\ S \longleftrightarrow no\text{-}step\ cdcl_W\text{-}stgy\ S$ (**is** $?S'\ S \longleftrightarrow ?C\ S$)
⟨*proof*⟩

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

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

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

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

end

end

Chapter 2

NOT's CDCL and DPLL

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

The organisation of the development is the following:

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

2.1 Measure

This measure show the termination of the core of CDCL: each step improves the number of literals we know for sure.
This measure can also be seen as the increasing lexicographic order: it is an order on bounded sequences, when each element is bounded. The proof involves a measure like the one defined here (the same?).

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

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

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

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

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

lemma *sum-of-powers*: $0 \leq k \Longrightarrow (k - 1) * (\sum i{=}0..{<}n.\ k\widehat{}i) = k\widehat{}n - (1{::}nat)$
⟨*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 $\mu_C\ s\ b\ M < b\widehat{}s$
⟨*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 $\mu_C\ s\ b\ M < b\ \widehat{}\ s$
⟨*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* $\{(ys, xs).\ length\ xs < s \wedge length\ ys < s \wedge$

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

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

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

lemma *acyclic-*νNOT: *acyclic* $(\nu NOT\ s\ base)$
⟨*proof*⟩

lemma *wf-*νNOT: *wf* $(\nu NOT\ s\ base)$
⟨*proof*⟩

end
theory *CDCL-NOT*
imports
Weidenbach-Book-Base.WB-List-More
Weidenbach-Book-Base.Wellfounded-More
Entailment-Definition.Partial-Annotated-Herbrand-Interpretation
CDCL-WNOT-Measure
begin

2.2 NOT's CDCL

2.2.1 Auxiliary Lemmas and Measure

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

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

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

2.2.2 Initial Definitions

The State

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

locale *dpll-state-ops* =
fixes
trail :: ‹$'st \Rightarrow ('v,\ unit)\ ann\text{-}lits$› **and**
clauses$_{NOT}$ :: ‹$'st \Rightarrow 'v\ clauses$› **and**
prepend-trail :: ‹$('v,\ unit)\ ann\text{-}lit \Rightarrow 'st \Rightarrow 'st$› **and**
tl-trail :: ‹$'st \Rightarrow 'st$› **and**
add-cls$_{NOT}$ :: ‹$'v\ clause \Rightarrow 'st \Rightarrow 'st$› **and**
remove-cls$_{NOT}$ :: ‹$'v\ clause \Rightarrow 'st \Rightarrow 'st$›

begin
abbreviation $state_{NOT} :: \langle 'st \Rightarrow ('v, unit)\ ann\text{-}lit\ list \times 'v\ clauses\rangle$ **where**
$\langle state_{NOT}\ S \equiv (trail\ S, clauses_{NOT}\ S)\rangle$
end

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

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

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

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

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

declare $reduce\text{-}trail\text{-}to_{NOT}.simps[simp\ del]$

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

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

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

lemma *reduce-trail-to$_{NOT}$-length*:
‹*length M* = *length M'* ⟹ *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 $state\text{-}eq_{NOT}$ :: ⟨'*st* ⇒ '*st* ⇒ *bool*⟩ (**infix** ∼ 50) **where**
⟨*S* ∼ *T* ⟷ *trail S* = *trail T* ∧ $clauses_{NOT}$ *S* = $clauses_{NOT}$ *T*⟩

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

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

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

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

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

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

end — End on locale *dpll-state*.

Definition of the Transitions

Each possible is in its own locale.

locale *propagate-ops* =
dpll-state trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
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
propagate-conds :: ⟨('*v*, *unit*) *ann-lit* ⇒ '*st* ⇒ '*st* ⇒ *bool*⟩
begin
inductive $propagate_{NOT}$ :: ⟨'*st* ⇒ '*st* ⇒ *bool*⟩ **where**
$propagate_{NOT}$[*intro*]: ⟨*add-mset L C* ∈# $clauses_{NOT}$ *S* ⟹ *trail S* ⊨*as CNot C*

$\Longrightarrow$ *undefined-lit* (*trail S*) *L*
$\Longrightarrow$ *propagate-conds* (*Propagated L* ()) *S T*
$\Longrightarrow$ *T* $\sim$ *prepend-trail* (*Propagated L* ()) *S*
$\Longrightarrow$ *propagate*$_{NOT}$ *S T*⟩
inductive-cases *propagate*$_{NOT}$*E*[*elim*]: ⟨*propagate*$_{NOT}$ *S T*⟩

end

locale *decide-ops* =
dpll-state trail clauses$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$
for
trail :: ⟨*'st* ⇒ (*'v*, *unit*) *ann-lits*⟩ **and**
clauses$_{NOT}$ :: ⟨*'st* ⇒ *'v clauses*⟩ **and**
prepend-trail :: ⟨(*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *'st*⟩ **and**
tl-trail :: ⟨*'st* ⇒*'st*⟩ **and**
add-cls$_{NOT}$ :: ⟨*'v clause* ⇒ *'st* ⇒ *'st*⟩ **and**
remove-cls$_{NOT}$ :: ⟨*'v clause* ⇒ *'st* ⇒ *'st*⟩ +
fixes
decide-conds :: ⟨*'st* ⇒ *'st* ⇒ *bool*⟩
begin
inductive *decide*$_{NOT}$ :: ⟨*'st* ⇒ *'st* ⇒ *bool*⟩ **where**
decide$_{NOT}$[*intro*]:
⟨*undefined-lit* (*trail S*) *L* $\Longrightarrow$
atm-of L $\in$ *atms-of-mm* (*clauses*$_{NOT}$ *S*) $\Longrightarrow$
T $\sim$ *prepend-trail* (*Decided L*) *S* $\Longrightarrow$
decide-conds S T $\Longrightarrow$
decide$_{NOT}$ *S T*⟩

inductive-cases *decide*$_{NOT}$*E*[*elim*]: ⟨*decide*$_{NOT}$ *S S'*⟩
end

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

inductive *backjump* **where**
⟨*trail S* = *F'* @ *Decided K* # *F*
$\Longrightarrow$ *T* $\sim$ *prepend-trail* (*Propagated L* ()) (*reduce-trail-to*$_{NOT}$ *F S*)
$\Longrightarrow$ *C* $\in\#$ *clauses*$_{NOT}$ *S*
$\Longrightarrow$ *trail S* $\models$*as CNot C*
$\Longrightarrow$ *undefined-lit F L*
$\Longrightarrow$ *atm-of L* $\in$ *atms-of-mm* (*clauses*$_{NOT}$ *S*) $\cup$ *atm-of* ‘ (*lits-of-l* (*trail S*))
$\Longrightarrow$ *clauses*$_{NOT}$ *S* $\models$*pm add-mset L C'*
$\Longrightarrow$ *F* $\models$*as CNot C'*
$\Longrightarrow$ *backjump-conds C C' L S T*
$\Longrightarrow$ *backjump S T*⟩
inductive-cases *backjumpE*: ⟨*backjump S T*⟩

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

end

2.2.3 DPLL with Backjumping

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

We cannot add a like condition *atms-of C′ ⊆ atms-of-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-of L ∈ atm-of ‘ lits-of-l (F′ @ Decided K # F)* is important, otherwise you are not sure that you can backtrack.

Definition

We define dpll with backjumping:

inductive *dpll-bj* :: ‹*'st ⇒ 'st ⇒ bool*› **for** *S* :: *'st* **where**
bj-decide$_{NOT}$: ‹*decide$_{NOT}$ S S′* ⟹ *dpll-bj S S′*› |
bj-propagate$_{NOT}$: ‹*propagate$_{NOT}$ S S′* ⟹ *dpll-bj S S′*› |
bj-backjump: ‹*backjump S S′* ⟹ *dpll-bj S S′*›

lemmas *dpll-bj-induct* = *dpll-bj.induct*[*split-format*(*complete*)]
thm *dpll-bj-induct*[*OF dpll-with-backjumping-ops-axioms*]
lemma *dpll-bj-all-induct*[*consumes 2*, *case-names decide*$_{NOT}$ *propagate*$_{NOT}$ *backjump*]:
fixes *S T* :: ‹*'st*›
assumes
‹*dpll-bj S T*› **and**
‹*inv S*›
‹$\bigwedge$*L T. undefined-lit* (*trail S*) *L* $\Longrightarrow$ *atm-of L* $\in$ *atms-of-mm* (*clauses*$_{NOT}$ *S*)
$\Longrightarrow$ *T* $\sim$ *prepend-trail* (*Decided L*) *S*
$\Longrightarrow$ *P S T*› **and**
‹$\bigwedge$*C L T. add-mset L C* $\in\#$ *clauses*$_{NOT}$ *S* $\Longrightarrow$ *trail S* $\models as$ *CNot C* $\Longrightarrow$ *undefined-lit* (*trail S*) *L*
$\Longrightarrow$ *T* $\sim$ *prepend-trail* (*Propagated L* ()) *S*
$\Longrightarrow$ *P S T*› **and**
‹$\bigwedge$*C F' K F L C' T. C* $\in\#$ *clauses*$_{NOT}$ *S* $\Longrightarrow$ *F'* @ *Decided K* # *F* $\models as$ *CNot C*
$\Longrightarrow$ *trail S* = *F'* @ *Decided K* # *F*
$\Longrightarrow$ *undefined-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* $\models pm$ *add-mset L C'*
$\Longrightarrow$ *F* $\models as$ *CNot C'*
$\Longrightarrow$ *T* $\sim$ *prepend-trail* (*Propagated L* ()) (*reduce-trail-to*$_{NOT}$ *F S*)
$\Longrightarrow$ *P S T*›
shows ‹*P S T*›
⟨*proof*⟩

Basic properties

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

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

Valuations **lemma** *dpll-bj-sat-iff*:
assumes ‹*dpll-bj S T*› **and** ‹*inv S*›
shows ‹*I* $\models sm$ *clauses*$_{NOT}$ *S* $\longleftrightarrow$ *I* $\models sm$ *clauses*$_{NOT}$ *T*›
⟨*proof*⟩

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

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

shows ‹*atm-of* ‘ (*lits-of-l* (*trail T*)) ⊆ *atms-of-mm* (*clauses*$_{NOT}$ *S*)›
⟨*proof*⟩

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

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

Termination

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

lemma *take-length-get-all-ann-decomposition-decided-sandwich*:
‹*take* (*length* (*get-all-ann-decomposition F*))
(*map* (*f o snd*) (*rev* (*get-all-ann-decomposition* (*F′* @ *Decided K* # *F*))))
=
map (*f o snd*) (*rev* (*get-all-ann-decomposition F*))
›
⟨*proof*⟩

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

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

Well-foundedness The bounds are the following:

- *1* + *card* (*atms-of-ms A*): *card* (*atms-of-ms A*) is an upper bound on the length of the list. As *get-all-ann-decomposition* appends an possibly empty couple at the end, adding one is needed.

- *2* + *card* (*atms-of-ms A*): *card* (*atms-of-ms A*) is an upper bound on the number of elements, where adding one is necessary for the same reason as for the bound on the list, and one is needed to have a strict bound.

abbreviation *unassigned-lit* :: ‹*'b clause set* ⇒ *'a list* ⇒ *nat*› **where**
‹*unassigned-lit N M* ≡ *card* (*atms-of-ms N*) − *length M*›

lemma *dpll-bj-trail-mes-increasing-prop*:
fixes *M* :: ‹(*'v*, *unit*) *ann-lits* › **and** *N* :: ‹*'v clauses*›
assumes
‹*dpll-bj S T*› **and**
‹*inv S*› **and**
NA: ‹*atms-of-mm* (*clauses*$_{NOT}$ *S*) ⊆ *atms-of-ms A*› **and**
MA: ‹*atm-of* ‘ *lits-of-l* (*trail S*) ⊆ *atms-of-ms A*› **and**
n-d: ‹*no-dup* (*trail S*)› **and**
finite: ‹*finite A*›
shows ‹*μ*$_C$ (*1*+*card* (*atms-of-ms A*)) (*2*+*card* (*atms-of-ms A*)) (*trail-weight T*)
> *μ*$_C$ (*1*+*card* (*atms-of-ms A*)) (*2*+*card* (*atms-of-ms A*)) (*trail-weight S*)›
⟨*proof*⟩

lemma *dpll-bj-trail-mes-decreasing-prop*:
assumes *dpll*: ‹*dpll-bj S T*› **and** *inv*: ‹*inv S*› **and**
N-A: ‹*atms-of-mm* (*clauses*$_{NOT}$ *S*) ⊆ *atms-of-ms A*› **and**
M-A: ‹*atm-of* ‘ *lits-of-l* (*trail S*) ⊆ *atms-of-ms A*› **and**
nd: ‹*no-dup* (*trail S*)› **and**
fin-A: ‹*finite A*›
shows ‹(*2*+*card* (*atms-of-ms A*)) ^ (*1*+*card* (*atms-of-ms A*))
− *μ*$_C$ (*1*+*card* (*atms-of-ms A*)) (*2*+*card* (*atms-of-ms A*)) (*trail-weight T*)
< (*2*+*card* (*atms-of-ms A*)) ^ (*1*+*card* (*atms-of-ms A*))
− *μ*$_C$ (*1*+*card* (*atms-of-ms A*)) (*2*+*card* (*atms-of-ms A*)) (*trail-weight S*)›
⟨*proof*⟩

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

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

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

lemma *dpll-bj-trail-mes-decreasing-less-than*:
assumes *dpll*: ‹*dpll-bj S T*› **and** *inv*: ‹*inv S*› **and**
N-A: ‹*atms-of-mm* (*clauses*$_{NOT}$ *S*) ⊆ *atms-of-ms A*› **and**
M-A: ‹*atm-of* ‘ *lits-of-l* (*trail S*) ⊆ *atms-of-ms A*› **and**
nd: ‹*no-dup* (*trail S*)› **and**
fin-A: ‹*finite A*›
shows ‹(*DPLL-mes*$_W$ (*atms-of-ms A*) (*trail T*), *DPLL-mes*$_W$ (*atms-of-ms A*) (*trail S*)) ∈
lexn less-than (*card* ((*atms-of-ms A*)))›
⟨*proof*⟩

lemma

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

Normal Forms

We prove that given a normal form of DPLL, with some structural invariants, then either N is satisfiable and the built valuation M is a model; or N is unsatisfiable.
Idea of the proof: We have to prove tat *satisfiable N*, ¬ *M* ⊨*as N* and there is no remaining step is incompatible.

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

   The assumption are saying that we have a finite upper bound A for the literals, that we cannot do any step ∀ *S*′. ¬ *dpll-bj S S*′

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

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

locale *dpll-with-backjumping* =
dpll-with-backjumping-ops trail clauses$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$ *inv*
decide-conds backjump-conds propagate-conds
for
trail :: ‹′*st* ⇒ (′*v*, *unit*) *ann-lits*› **and**
clauses$_{NOT}$ :: ‹′*st* ⇒ ′*v clauses*› **and**
prepend-trail :: ‹(′*v*, *unit*) *ann-lit* ⇒ ′*st* ⇒ ′*st*› **and**
tl-trail :: ‹′*st* ⇒′*st*› **and**
add-cls$_{NOT}$ :: ‹′*v clause* ⇒ ′*st* ⇒ ′*st*› **and**
remove-cls$_{NOT}$ :: ‹′*v clause* ⇒ ′*st* ⇒ ′*st*› **and**
inv :: ‹′*st* ⇒ *bool*› **and**

decide-conds :: ‹'*st* ⇒ '*st* ⇒ *bool*› **and**
backjump-conds :: ‹'*v clause* ⇒ '*v clause* ⇒ '*v literal* ⇒ '*st* ⇒ '*st* ⇒ *bool*› **and**
propagate-conds :: ‹('*v*, *unit*) *ann-lit* ⇒ '*st* ⇒ '*st* ⇒ *bool*›
+
assumes *dpll-bj-inv*: ‹⋀*S T*. *dpll-bj S T* ⟹ *inv S* ⟹ *inv T*›
begin

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

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

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

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

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

lemma *rtranclp-dpll-bj-inv-incl-dpll-bj-inv-trancl*:
‹{(*T*, *S*). *dpll-bj*$^{++}$ *S T*

$\wedge$ *atms-of-mm* (*clauses*$_{NOT}$ *S*) $\subseteq$ *atms-of-ms A* $\wedge$ *atm-of* ‘ *lits-of-l* (*trail S*) $\subseteq$ *atms-of-ms A*
$\wedge$ *no-dup* (*trail S*) $\wedge$ *inv S*}
$\subseteq$ {(*T*, *S*). *dpll-bj S T* $\wedge$ *atms-of-mm* (*clauses*$_{NOT}$ *S*) $\subseteq$ *atms-of-ms A*
$\wedge$ *atm-of* ‘ *lits-of-l* (*trail S*) $\subseteq$ *atms-of-ms A* $\wedge$ *no-dup* (*trail S*) $\wedge$ *inv S*}$^+$›
(**is** ‹*?A* $\subseteq$ *?B*$^+$›)
⟨*proof*⟩

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

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

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

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

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

lemma *tranclp-dpll-bj-trail-mes-decreasing-prop*:
assumes *dpll*: ‹*dpll-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**
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 the locale *dpll-with-backjumping*.

2.2.4 CDCL

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

Learn and Forget

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

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

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

lemma *learn-μ_C-stable*:
  assumes ⟨*learn S T*⟩ **and** ⟨*no-dup* (*trail S*)⟩
  shows ⟨μ_C *A B* (*trail-weight S*) = μ_C *A B* (*trail-weight T*)⟩
  ⟨*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-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
    forget-conds :: ⟨*'v clause* ⇒ *'st* ⇒ *bool*⟩
begin

inductive $forget_{NOT}$ :: ⟨'*st* ⇒ '*st* ⇒ *bool*⟩ **where**
$forget_{NOT}$:
⟨*removeAll-mset C*(*clauses*$_{NOT}$ *S*) ⊨*pm C* ⟹
forget-conds C S ⟹
C ∈# *clauses*$_{NOT}$ *S* ⟹
T ∼ *remove-cls*$_{NOT}$ *C S* ⟹

$forget_{NOT}$ *S T*⟩
inductive-cases $forget_{NOT}E$: ⟨$forget_{NOT}$ *S T*⟩

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

locale $learn\text{-}and\text{-}forget_{NOT}$ =
learn-ops trail clauses$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$ *learn-conds* +
forget-ops trail clauses$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$ *forget-conds*
for
trail :: ⟨'*st* ⇒ ('*v*, *unit*) *ann-lits*⟩ **and**
clauses$_{NOT}$ :: ⟨'*st* ⇒ '*v clauses*⟩ **and**
prepend-trail :: ⟨('*v*, *unit*) *ann-lit* ⇒ '*st* ⇒ '*st*⟩ **and**
tl-trail :: ⟨'*st* ⇒'*st*⟩ **and**
add-cls$_{NOT}$ :: ⟨'*v clause* ⇒ '*st* ⇒ '*st*⟩ **and**
remove-cls$_{NOT}$ :: ⟨'*v clause* ⇒ '*st* ⇒ '*st*⟩ **and**
learn-conds forget-conds :: ⟨'*v clause* ⇒ '*st* ⇒ *bool*⟩
begin
inductive $learn\text{-}and\text{-}forget_{NOT}$ :: ⟨'*st* ⇒ '*st* ⇒ *bool*⟩
where
lf-learn: ⟨*learn S T* ⟹ $learn\text{-}and\text{-}forget_{NOT}$ *S T*⟩ |
lf-forget: ⟨$forget_{NOT}$ *S T* ⟹ $learn\text{-}and\text{-}forget_{NOT}$ *S T*⟩
end

Definition of CDCL

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

inductive $cdcl_{NOT}$ :: ⟨'*st* ⇒ '*st* ⇒ *bool*⟩ **for** *S* :: '*st* **where**

c-dpll-bj: ⟨*dpll-bj S S′ ⟹ cdcl$_{NOT}$ S S′*⟩ |
c-learn: ⟨*learn S S′ ⟹ cdcl$_{NOT}$ S S′*⟩ |
c-forget$_{NOT}$: ⟨*forget$_{NOT}$ S S′ ⟹ cdcl$_{NOT}$ S S′*⟩

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

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

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

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

lemma *cdcl$_{NOT}$-atms-of-ms-clauses-decreasing*:
assumes ⟨*cdcl$_{NOT}$ S T*⟩**and** ⟨*inv S*⟩
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 ⟨*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**
⟨*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**
‹*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*›
shows ‹*I*⊨*sextm clauses*$_{NOT}$ *S* ⟷ *I*⊨*sextm clauses*$_{NOT}$ *T*›
⟨*proof*⟩

end — End of the locale *conflict-driven-clause-learning-ops*.

CDCL with invariant

locale *conflict-driven-clause-learning* =
conflict-driven-clause-learning-ops +
assumes *cdcl*$_{NOT}$*-inv*: ‹⋀*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**
atms-clauses-S: ‹*atms-of-mm* (*clauses*$_{NOT}$ *S*) ⊆ *A*› **and**
atms-trail-S: ‹*atm-of* ‘(*lits-of-l* (*trail S*)) ⊆ *A*›
shows ‹*atm-of* ‘ (*lits-of-l* (*trail T*)) ⊆ *A* ∧ *atms-of-mm* (*clauses*$_{NOT}$ *T*) ⊆ *A*›
⟨*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*›
shows ‹*I*⊨*sextm clauses*$_{NOT}$ *S* ⟷ *I*⊨*sextm clauses*$_{NOT}$ *T*›
⟨*proof*⟩

definition *cdcl$_{NOT}$-NOT-all-inv* **where**
‹*cdcl$_{NOT}$-NOT-all-inv A S* ⟷ (*finite A* ∧ *inv S* ∧ *atms-of-mm* (*clauses$_{NOT}$ S*) ⊆ *atms-of-ms A*
∧ *atm-of* ‘ *lits-of-l* (*trail S*) ⊆ *atms-of-ms A* ∧ *no-dup* (*trail S*))›

lemma *cdcl$_{NOT}$-NOT-all-inv*:
assumes ‹*cdcl$_{NOT}$** S T*› **and** ‹*cdcl$_{NOT}$-NOT-all-inv A S*›
shows ‹*cdcl$_{NOT}$-NOT-all-inv A T*›
⟨*proof*⟩

abbreviation *learn-or-forget* **where**
‹*learn-or-forget S T* ≡ *learn S T* ∨ *forget$_{NOT}$ S T*›

lemma *rtranclp-learn-or-forget-cdcl$_{NOT}$*:
‹*learn-or-forget** S T* ⟹ *cdcl$_{NOT}$** S T*›
⟨*proof*⟩

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

lemma *infinite-cdcl$_{NOT}$-exists-learn-and-forget-infinite-chain*:
assumes
‹⋀*i*. *cdcl$_{NOT}$* (*f i*) (*f*(*Suc i*))› **and**
inv: ‹*cdcl$_{NOT}$-NOT-all-inv A* (*f 0*)›
shows ‹∃*j*. ∀*i*≥*j*. *learn-or-forget* (*f i*) (*f* (*Suc i*))›
⟨*proof*⟩

lemma *wf-cdcl$_{NOT}$-no-learn-and-forget-infinite-chain*:
assumes
no-infinite-lf: ‹⋀*f j*. ¬ (∀ *i*≥*j*. *learn-or-forget* (*f i*) (*f* (*Suc i*)))›
shows ‹*wf* {(*T*, *S*). *cdcl$_{NOT}$ S T* ∧ *cdcl$_{NOT}$-NOT-all-inv A S*}›
(**is** ‹*wf* {(*T*, *S*). *cdcl$_{NOT}$ S T* ∧ *?inv S*}›)
⟨*proof*⟩

lemma *inv-and-tranclp-cdcl-$_{NOT}$-tranclp-cdcl$_{NOT}$-and-inv*:
‹*cdcl$_{NOT}$*$^{++}$ *S T* ∧ *cdcl$_{NOT}$-NOT-all-inv A S* ⟷ (λ*S T*. *cdcl$_{NOT}$ S T* ∧ *cdcl$_{NOT}$-NOT-all-inv A S*)$^{++}$ *S T*›
(**is** ‹*?A* ∧ *?I* ⟷ *?B*›)
⟨*proof*⟩

lemma *wf-tranclp-cdcl$_{NOT}$-no-learn-and-forget-infinite-chain*:
assumes
no-infinite-lf: ‹⋀*f j*. ¬ (∀ *i*≥*j*. *learn-or-forget* (*f i*) (*f* (*Suc i*)))›
shows ‹*wf* {(*T*, *S*). *cdcl$_{NOT}$*$^{++}$ *S T* ∧ *cdcl$_{NOT}$-NOT-all-inv A S*}›
⟨*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 the locale *conflict-driven-clause-learning*.

Termination

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

Restricting learn and forget

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

lemma *cdcl$_{NOT}$-learn-all-induct*[*consumes 1*, *case-names dpll-bj learn forget$_{NOT}$*]:

fixes $S\ T :: \langle 'st \rangle$
assumes $\langle cdcl_{NOT}\ S\ T \rangle$ **and**
dpll: $\langle \bigwedge T.\ dpll\text{-}bj\ S\ T \Longrightarrow P\ S\ T \rangle$ **and**
learning:
$\langle \bigwedge C\ F\ K\ F'\ C'\ L\ T.\ clauses_{NOT}\ S \models pm\ C \Longrightarrow$
$atms\text{-}of\ C \subseteq atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \cup atm\text{-}of\ `\ (lits\text{-}of\text{-}l\ (trail\ S)) \Longrightarrow$
$distinct\text{-}mset\ C \Longrightarrow$
$\neg\ tautology\ C \Longrightarrow$
$learn\text{-}restrictions\ C\ S \Longrightarrow$
$trail\ S = F' @ Decided\ K \#\ F \Longrightarrow$
$C = add\text{-}mset\ L\ C' \Longrightarrow$
$F \models as\ CNot\ C' \Longrightarrow$
$add\text{-}mset\ L\ C' \notin\# clauses_{NOT}\ S \Longrightarrow$
$T \sim add\text{-}cls_{NOT}\ C\ S \Longrightarrow$
$P\ S\ T \rangle$ **and**
forgetting: $\langle \bigwedge C\ T.\ removeAll\text{-}mset\ C\ (clauses_{NOT}\ S) \models pm\ C \Longrightarrow$
$C \in\# clauses_{NOT}\ S \Longrightarrow$
$\neg(\exists F'\ F\ K\ L.\ trail\ S = F' @ Decided\ K \#\ F \wedge F \models as\ CNot\ (C - \{\#L\#\})) \Longrightarrow$
$T \sim remove\text{-}cls_{NOT}\ C\ S \Longrightarrow$
$forget\text{-}restrictions\ C\ S \Longrightarrow$
$P\ S\ T \rangle$
shows $\langle P\ S\ T \rangle$
⟨*proof*⟩

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

lemma *learn-always-simple-clauses*:
assumes
learn: $\langle learn\ S\ T \rangle$ **and**
n-d: $\langle no\text{-}dup\ (trail\ S) \rangle$
shows $\langle set\text{-}mset\ (clauses_{NOT}\ T - clauses_{NOT}\ S)$
$\subseteq simple\text{-}clss\ (atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \cup atm\text{-}of\ `\ lits\text{-}of\text{-}l\ (trail\ S)) \rangle$
⟨*proof*⟩

definition $\langle conflicting\text{-}bj\text{-}clss\ S \equiv$
$\{C+\{\#L\#\}\ |C\ L.\ C+\{\#L\#\} \in\# clauses_{NOT}\ S \wedge distinct\text{-}mset\ (C+\{\#L\#\})$
$\wedge \neg tautology\ (C+\{\#L\#\})$
$\wedge (\exists F'\ K\ F.\ trail\ S = F' @ Decided\ K \#\ F \wedge F \models as\ CNot\ C)\} \rangle$

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

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

lemma *conflicting-bj-clss-add-cls*$_{NOT}$*-state-eq*:
assumes
T: $\langle T \sim add\text{-}cls_{NOT}\ C'\ S \rangle$ **and**
n-d: $\langle no\text{-}dup\ (trail\ S) \rangle$
shows $\langle conflicting\text{-}bj\text{-}clss\ T$
$= conflicting\text{-}bj\text{-}clss\ S$
$\cup (if\ \exists C\ L.\ C' = add\text{-}mset\ L\ C \wedge distinct\text{-}mset\ (add\text{-}mset\ L\ C) \wedge \neg tautology\ (add\text{-}mset\ L\ C)$

$\wedge$ ($\exists$ F' K d F. *trail* S = F' @ *Decided* K # F $\wedge$ F $\models$*as* *CNot* C)
then {C'} *else* {})›
⟨*proof*⟩

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

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

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

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

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

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

lemma *do-not-forget-before-backtrack-rule-clause-learned-clause-untouched*:
assumes ‹*forget*$_{NOT}$ S T›
shows ‹*conflicting-bj-clss* S = *conflicting-bj-clss* T›
⟨*proof*⟩

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

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

lemma *set-insert-neq*:
‹A ≠ *insert* a A $\longleftrightarrow$ a $\notin$ A›
⟨*proof*⟩

lemma *learn-*μ_L*-decrease*:
assumes *learnST*: ‹*learn* S T› **and** *n-d*: ‹*no-dup* (*trail* S)› **and**
A: ‹*atms-of-mm* (*clauses*$_{NOT}$ S) $\cup$ *atm-of* ' *lits-of-l* (*trail* S) $\subseteq$ A› **and**
fin-A: ‹*finite* A›
shows ‹(μ_L (*card* A) T, μ_L (*card* A) S) $\in$ *less-than* <*lex*> *less-than*›
⟨*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*) $\subseteq$ *atms-of-ms A* and in the clauses *atms-of-mm* (*clauses*$_{NOT}$ *S*) $\subseteq$ *atms-of-ms A*. This can the the set of all the literals in the starting set of clauses.
- *no-dup* (*trail S*): no duplicate in the trail. This is invariant along the path.

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

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

definition μ_C' :: ‹'*v clause set* ⇒ '*st* ⇒ *nat*› **where**
‹μ_C' *A T* ≡ μ_C (*1*+*card* (*atms-of-ms A*)) (*2*+*card* (*atms-of-ms A*)) (*trail-weight T*)›

definition μ_{CDCL}' :: ‹'*v clause set* ⇒ '*st* ⇒ *nat*› **where**
‹μ_{CDCL}' *A T* ≡
((*2*+*card* (*atms-of-ms A*)) ^ (*1*+*card* (*atms-of-ms A*)) − μ_C' *A T*) ∗ (*1*+ *3*^*card* (*atms-of-ms A*)) ∗ *2*
+ *conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) *T* ∗ *2*
+ *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 ‹μ_{CDCL}' *A T* < μ_{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*))

$\leq$ *card* $\{C.\ C \in\#\ clauses_{NOT}\ S \wedge (tautology\ C \vee \neg distinct\text{-}mset\ C)\} + 3\ \widehat{}\ (card\ A)$⟩
(**is** ⟨*card ?T* $\leq$ *card ?S* + -⟩)
⟨*proof*⟩

definition μ_{CDCL}'-*bound* :: ⟨'*v clause set* ⇒ '*st* ⇒ *nat*⟩ **where**
⟨μ_{CDCL}'-*bound A S* =
((*2* + *card* (*atms-of-ms A*)) $\widehat{}$ (*1* + *card* (*atms-of-ms A*))) ∗ (*1* + *3* $\widehat{}$ *card* (*atms-of-ms A*)) ∗ *2*
+ *2*∗*3* $\widehat{}$ (*card* (*atms-of-ms A*))
+ *card* $\{C.\ C \in\#\ clauses_{NOT}\ S \wedge (tautology\ C \vee \neg distinct\text{-}mset\ C)\}$ + *3* $\widehat{}$ (*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*) ⊆ *atms-of-ms A*⟩ **and**
⟨*atm-of* '(*lits-of-l* (*trail S*)) ⊆ *atms-of-ms A*⟩ **and**
n-d: ⟨*no-dup* (*trail S*)⟩ **and**
finite[*simp*]: ⟨*finite* (*atms-of-ms A*)⟩
shows ⟨μ_{CDCL}'-*bound A T* ≤ μ_{CDCL}'-*bound A S*⟩
⟨*proof*⟩

end — End of the locale *conflict-driven-clause-learning-learning-before-backjump-only-distinct-learnt*.

2.2.5 CDCL with Restarts

Definition

locale *restart-ops* =

fixes
$cdcl_{NOT}$ :: ⟨'st ⇒ 'st ⇒ bool⟩ **and**
restart :: ⟨'st ⇒ 'st ⇒ bool⟩
begin
inductive $cdcl_{NOT}$-*raw-restart* :: ⟨'st ⇒ 'st ⇒ bool⟩ **where**
⟨$cdcl_{NOT}$ *S T* ⟹ $cdcl_{NOT}$-*raw-restart S T*⟩ |
⟨*restart S T* ⟹ $cdcl_{NOT}$-*raw-restart S T*⟩

end

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

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

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

Increasing restarts

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

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

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

locale $cdcl_{NOT}$-*increasing-restarts-ops* =
  restart-ops $cdcl_{NOT}$ *restart* **for**
    restart :: ‹$'st \Rightarrow 'st \Rightarrow bool$› **and**
    $cdcl_{NOT}$ :: ‹$'st \Rightarrow 'st \Rightarrow bool$› +
  fixes
    f :: ‹*nat* ⇒ *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: ‹$\bigwedge A\ S\ T.\ cdcl_{NOT}\text{-}inv\ S \Longrightarrow bound\text{-}inv\ A\ S \Longrightarrow cdcl_{NOT}\ S\ T \Longrightarrow bound\text{-}inv\ A\ T$› **and**
    $cdcl_{NOT}$-*measure*: ‹$\bigwedge A\ S\ T.\ cdcl_{NOT}\text{-}inv\ S \Longrightarrow bound\text{-}inv\ A\ S \Longrightarrow cdcl_{NOT}\ S\ T \Longrightarrow \mu\ A\ T < \mu\ A\ S$› **and**
    measure-bound2: ‹$\bigwedge A\ T\ U.\ cdcl_{NOT}\text{-}inv\ T \Longrightarrow bound\text{-}inv\ A\ T \Longrightarrow cdcl_{NOT}{}^{**}\ T\ U \Longrightarrow \mu\ A\ U \leq \mu\text{-}bound\ A\ T$› **and**
    measure-bound4: ‹$\bigwedge A\ T\ U.\ cdcl_{NOT}\text{-}inv\ T \Longrightarrow bound\text{-}inv\ A\ T \Longrightarrow cdcl_{NOT}{}^{**}\ T\ U \Longrightarrow \mu\text{-}bound\ A\ U \leq \mu\text{-}bound\ A\ T$› **and**
    $cdcl_{NOT}$-*restart-inv*: ‹$\bigwedge A\ U\ V.\ cdcl_{NOT}\text{-}inv\ U \Longrightarrow restart\ U\ V \Longrightarrow bound\text{-}inv\ A\ U \Longrightarrow bound\text{-}inv\ A\ V$›
    and
    exists-bound: ‹$\bigwedge R\ S.\ cdcl_{NOT}\text{-}inv\ R \Longrightarrow restart\ R\ S \Longrightarrow \exists A.\ bound\text{-}inv\ A\ S$› **and**
    $cdcl_{NOT}$-*inv*: ‹$\bigwedge S\ T.\ cdcl_{NOT}\text{-}inv\ S \Longrightarrow cdcl_{NOT}\ S\ T \Longrightarrow cdcl_{NOT}\text{-}inv\ T$› **and**
    $cdcl_{NOT}$-*inv-restart*: ‹$\bigwedge S\ T.\ cdcl_{NOT}\text{-}inv\ S \Longrightarrow restart\ S\ T \Longrightarrow cdcl_{NOT}\text{-}inv\ T$›
begin

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

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

lemma *rtranclp*-$cdcl_{NOT}$-$cdcl_{NOT}$-*inv*:
  assumes
    ‹$cdcl_{NOT}{}^{**}\ S\ T$› **and**
    ‹$cdcl_{NOT}\text{-}inv\ S$›
  shows ‹$cdcl_{NOT}\text{-}inv\ T$›
  ⟨*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 ‹μ A $T < \mu$ A $S - n$›
⟨*proof*⟩

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

lemma *rtranclp-*$cdcl_{NOT}$*-measure*:
assumes
‹$cdcl_{NOT}^{**}$ S T› **and**
‹*bound-inv* A S› **and**
‹$cdcl_{NOT}$*-inv* S›
shows ‹μ A $T \leq \mu$ 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
$\Longrightarrow$ $cdcl_{NOT}$*-restart* (S, n) (U, *Suc* n)› |
restart-full: ‹*full1* $cdcl_{NOT}$ S $T \Longrightarrow$ $cdcl_{NOT}$*-restart* (S, n) (T, *Suc* n)›

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

lemma $cdcl_{NOT}$*-restart-*$cdcl_{NOT}$*-raw-restart*:
‹$cdcl_{NOT}$*-restart* S $T \Longrightarrow$ $cdcl_{NOT}$*-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 ⟹ 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-cls$_{NOT}$ remove-cls$_{NOT}$
for
trail :: ‹*'st ⇒ ('v, unit) ann-lits*› **and**
clauses$_{NOT}$ :: ‹*'st ⇒ 'v clauses*› **and**
prepend-trail :: ‹*('v, unit) ann-lit ⇒ 'st ⇒ 'st*› **and**
tl-trail :: ‹*'st ⇒'st*› **and**
add-cls$_{NOT}$ :: ‹*'v clause ⇒ 'st ⇒ 'st*› **and**
remove-cls$_{NOT}$ :: ‹*'v clause ⇒ 'st ⇒ 'st*› **and**
f :: ‹*nat ⇒ nat*› **and**
restart :: ‹*'st ⇒ 'st ⇒ bool*› **and**
bound-inv :: ‹*'bound ⇒ 'st ⇒ bool*› **and**
μ :: ‹*'bound ⇒ 'st ⇒ nat*› **and**
cdcl$_{NOT}$ :: ‹*'st ⇒ 'st ⇒ bool*› **and**
cdcl$_{NOT}$-inv :: ‹*'st ⇒ bool*› **and**
μ-bound :: ‹*'bound ⇒ 'st ⇒ nat*› +
assumes
measure-bound: ‹⋀*A T V n. cdcl$_{NOT}$-inv T ⟹ bound-inv A T*
⟹ cdcl$_{NOT}$-restart (T, n) (V, Suc n) ⟹ μ A V ≤ μ-bound A T› **and**
cdcl$_{NOT}$-raw-restart-μ-bound:
‹*cdcl$_{NOT}$-restart (T, a) (V, b) ⟹ cdcl$_{NOT}$-inv T ⟹ bound-inv A T*
⟹ μ-bound A V ≤ μ-bound A T›
begin

lemma *rtranclp-cdcl$_{NOT}$-raw-restart-μ-bound*:
‹*cdcl$_{NOT}$-restart** (T, a) (V, b) ⟹ cdcl$_{NOT}$-inv T ⟹ bound-inv A T*
⟹ μ-bound A V ≤ μ-bound A T›
⟨*proof*⟩

lemma *cdcl$_{NOT}$-raw-restart-measure-bound*:
⟨*cdcl$_{NOT}$-restart* (*T*, *a*) (*V*, *b*) ⟹ *cdcl$_{NOT}$-inv T* ⟹ *bound-inv A T*
⟹ *μ A V* ≤ *μ-bound A T*⟩
⟨*proof*⟩

lemma *rtranclp-cdcl$_{NOT}$-raw-restart-measure-bound*:
⟨*cdcl$_{NOT}$-restart*** (*T*, *a*) (*V*, *b*) ⟹ *cdcl$_{NOT}$-inv T* ⟹ *bound-inv A T*
⟹ *μ A V* ≤ *μ-bound A T*⟩
⟨*proof*⟩

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

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

lemma *rtranclp-cdcl$_{NOT}$-with-inv-inv-rtranclp-cdcl$_{NOT}$*:
assumes
inv: ⟨*cdcl$_{NOT}$-inv S*⟩ **and**
binv: ⟨*bound-inv A S*⟩
shows ⟨(λ*S T*. *cdcl$_{NOT}$ S T* ∧ *cdcl$_{NOT}$-inv S* ∧ *bound-inv A S*)** *S T* ⟷ *cdcl$_{NOT}$*** *S T*⟩
(**is** ⟨*?A*** *S T* ⟷ *?B*** *S T*⟩)
⟨*proof*⟩

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

end

2.2.6 Merging backjump and learning

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

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

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

inductive *backjump-l* **where**
backjump-l: ⟨*trail* S = F' @ *Decided* K # F
⟹ $T \sim$ *prepend-trail* (*Propagated* L ()) (*reduce-trail-to*$_{NOT}$ F (*add-cls*$_{NOT}$ C'' S))
⟹ $C \in\#$ *clauses*$_{NOT}$ S
⟹ *trail* $S \models as$ *CNot* C
⟹ *undefined-lit* F L
⟹ *atm-of* $L \in$ *atms-of-mm* (*clauses*$_{NOT}$ S) ∪ *atm-of* ‘ (*lits-of-l* (*trail* S))
⟹ *clauses*$_{NOT}$ $S \models pm$ *add-mset* L C'
⟹ C'' = *add-mset* L C'
⟹ $F \models as$ *CNot* C'
⟹ *backjump-l-cond* C C' L S T
⟹ *backjump-l* S T⟩

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

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

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

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

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

begin

abbreviation *backjump-conds* :: ‹*'v clause ⇒ 'v clause ⇒ 'v literal ⇒ 'st ⇒ 'st ⇒ bool*›
  where
‹*backjump-conds* ≡ λ*C C' L' S T*. *distinct-mset C'* ∧ *L'* ∉# *C'* ∧ ¬*tautology* (*add-mset L' C'*)›

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

end

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

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

lemma *tautology-single-add*:
  ‹*tautology* (*L* + {#*a*#}) ⟷ *tautology L* ∨ −*a* ∈# *L*›
  ⟨*proof*⟩

lemma *backjump-l-implies-exists-backjump*:
  assumes *bj*: ‹*backjump-l S T*› **and** ‹*inv S*› **and** *n-d*: ‹*no-dup* (*trail S*)›
  shows ‹∃ *U*. *backjump S U*›
⟨*proof*⟩

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

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

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

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

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

lemma *cdcl$_{NOT}$-merged-bj-learn-is-tranclp-cdcl$_{NOT}$*:
  ‹*cdcl$_{NOT}$-merged-bj-learn S T* ⟹ *inv S* ⟹ *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* ⟹ *inv S* ⟹ *cdcl$_{NOT}$*** *S T* ∧ *inv T*›
⟨*proof*⟩

lemma *rtranclp-cdcl$_{NOT}$-merged-bj-learn-is-rtranclp-cdcl$_{NOT}$*:
  ‹*cdcl$_{NOT}$-merged-bj-learn*** *S T* ⟹ *inv S* ⟹ *cdcl$_{NOT}$*** *S T*›
  ⟨*proof*⟩

lemma *rtranclp-cdcl$_{NOT}$-merged-bj-learn-inv*:
  ‹*cdcl$_{NOT}$-merged-bj-learn*** *S T* ⟹ *inv S* ⟹ *inv T*›
  ⟨*proof*⟩

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

definition μ_C' :: ‹'*v clause set* ⇒ '*st* ⇒ *nat*› **where**
‹μ_C' *A T* ≡ μ_C (*1*+*card* (*atms-of-ms A*)) (*2*+*card* (*atms-of-ms A*)) (*trail-weight T*)›

definition μ_{CDCL}'*-merged* :: ‹*'v clause set* ⇒ *'st* ⇒ *nat*› **where**
‹μ_{CDCL}'*-merged A T* ≡
((*2*+*card* (*atms-of-ms A*)) ^ (*1*+*card* (*atms-of-ms A*)) − μ_C' *A T*) ∗ *2* + *card* (*set-mset* (*clauses*$_{NOT}$ *T*))›

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

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

lemma *in-atms-neg-defined*: ‹*x* ∈ *atms-of C'* ⟹ *F* ⊨*as CNot C'* ⟹ *x* ∈ *atm-of* ‘ *lits-of-l F*›
⟨*proof*⟩

lemma *cdcl*$_{NOT}$*-merged-bj-learn-atms-of-ms-clauses-decreasing*:
assumes ‹*cdcl*$_{NOT}$*-merged-bj-learn S T*›**and** ‹*inv S*›
shows ‹*atms-of-mm* (*clauses*$_{NOT}$ *T*) ⊆ *atms-of-mm* (*clauses*$_{NOT}$ *S*) ∪ *atm-of* ‘ (*lits-of-l* (*trail S*))›
⟨*proof*⟩

lemma *cdcl*$_{NOT}$*-merged-bj-learn-atms-in-trail-in-set*:
assumes
‹*cdcl*$_{NOT}$*-merged-bj-learn S T*› **and** ‹*inv S*› **and**
‹*atms-of-mm* (*clauses*$_{NOT}$ *S*) ⊆ *A*› **and**
‹*atm-of* ‘ (*lits-of-l* (*trail S*)) ⊆ *A*›
shows ‹*atm-of* ‘ (*lits-of-l* (*trail T*)) ⊆ *A*›
⟨*proof*⟩

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

lemma *cdcl*$_{NOT}$*-merged-bj-learn-trail-clauses-bound*:
assumes
cdcl: ‹*cdcl*$_{NOT}$*-merged-bj-learn S T*› **and**
inv: ‹*inv S*› **and**
atms-clauses-S: ‹*atms-of-mm* (*clauses*$_{NOT}$ *S*) ⊆ *A*› **and**

atms-trail-S: ‹*atm-of* ‘(*lits-of-l* (*trail S*)) ⊆ *A*›
shows ‹*atm-of* ‘ (*lits-of-l* (*trail T*)) ⊆ *A* ∧ *atms-of-mm* (*clauses*$_{NOT}$ *T*) ⊆ *A*›
⟨*proof*⟩

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

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

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

lemma *rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-all-decomposition-implies*:
assumes ‹*cdcl*$_{NOT}$*-merged-bj-learn*** *S T*› **and** *inv*: ‹*inv S*›
‹*all-decomposition-implies-m* (*clauses*$_{NOT}$ *S*) (*get-all-ann-decomposition* (*trail S*))›
shows
‹*all-decomposition-implies-m* (*clauses*$_{NOT}$ *T*) (*get-all-ann-decomposition* (*trail T*))›
⟨*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

2.2.7 Instantiations

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

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

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

sublocale *cdcl$_{NOT}$-increasing-restarts-ops* ‹$\lambda S\ T.\ T \sim$ *reduce-trail-to$_{NOT}$* ([]::'*a list*) *S*› *cdcl$_{NOT}$ f*
‹$\lambda A\ S.$ *atms-of-mm* (*clauses$_{NOT}$ S*) $\subseteq$ *atms-of-ms A* $\wedge$ *atm-of* ‘ *lits-of-l* (*trail S*) $\subseteq$ *atms-of-ms A* $\wedge$ *finite A*›
μ_{CDCL}' ‹$\lambda S.$ *inv S* $\wedge$ *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*) $\subseteq$ *atms-of-ms A*›
‹*atm-of* ‘ *lits-of-l* (*trail T*) $\subseteq$ *atms-of-ms A*›
‹*finite A*›
shows ‹*μ_{CDCL}' A V* $\leq$ *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*) $\subseteq$ *atms-of-ms A*›
‹*atm-of* ‘ *lits-of-l* (*trail T*) $\subseteq$ *atms-of-ms A*›
‹*finite A*›
shows ‹*μ_{CDCL}'-bound A V* $\leq$ *μ_{CDCL}'-bound A T*›
⟨*proof*⟩

sublocale *cdcl$_{NOT}$-increasing-restarts* - - - - - -
f
‹$\lambda S\ T.\ T \sim$ *reduce-trail-to$_{NOT}$* ([]::'*a list*) *S*›
‹$\lambda A\ S.$ *atms-of-mm* (*clauses$_{NOT}$ S*) $\subseteq$ *atms-of-ms A*
$\wedge$ *atm-of* ‘ *lits-of-l* (*trail S*) $\subseteq$ *atms-of-ms A* $\wedge$ *finite A*›
μ_{CDCL}' cdcl$_{NOT}$
‹$\lambda S.$ *inv S* $\wedge$ *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* ⊨*sextm clauses*$_{NOT}$ (*fst S*) ⟷ *I* ⊨*sextm clauses*$_{NOT}$ (*fst T*)›
⟨*proof*⟩

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

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

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

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

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

definition *not-simplified-cls* :: $\langle 'b\ clause\ multiset \Rightarrow 'b\ clauses \rangle$
where
$\langle not\text{-}simplified\text{-}cls\ A \equiv \{\#C \in\# A.\ C \notin simple\text{-}clss\ (atms\text{-}of\text{-}mm\ A)\#\} \rangle$

lemma *not-simplified-cls-tautology-distinct-mset*:
$\langle not\text{-}simplified\text{-}cls\ A = \{\#C \in\# A.\ tautology\ C \vee \neg distinct\text{-}mset\ C\#\} \rangle$
⟨*proof*⟩

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

lemma $cdcl_{NOT}$*-merged-bj-learn-clauses-bound*:
assumes
$\langle cdcl_{NOT}\text{-}merged\text{-}bj\text{-}learn\ S\ T \rangle$ **and**
inv: $\langle inv\ S \rangle$ **and**
atms-clss: $\langle atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \subseteq atms\text{-}of\text{-}ms\ A \rangle$ **and**
atms-trail: $\langle atm\text{-}of\ `(lits\text{-}of\text{-}l\ (trail\ S)) \subseteq atms\text{-}of\text{-}ms\ A \rangle$ **and**
fin-A[*simp*]: $\langle finite\ A \rangle$
shows $\langle set\text{-}mset\ (clauses_{NOT}\ T) \subseteq set\text{-}mset\ (not\text{-}simplified\text{-}cls\ (clauses_{NOT}\ S))$
$\cup\ simple\text{-}clss\ (atms\text{-}of\text{-}ms\ A) \rangle$
⟨*proof*⟩

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

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

lemma *rtranclp-*$cdcl_{NOT}$*-merged-bj-learn-clauses-bound*:
assumes
$\langle cdcl_{NOT}\text{-}merged\text{-}bj\text{-}learn^{**}\ S\ T \rangle$ **and**
$\langle inv\ S \rangle$ **and**
$\langle atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \subseteq atms\text{-}of\text{-}ms\ A \rangle$ **and**
$\langle atm\text{-}of\ `(lits\text{-}of\text{-}l\ (trail\ S)) \subseteq atms\text{-}of\text{-}ms\ A \rangle$ **and**
finite[*simp*]: $\langle finite\ A \rangle$
shows $\langle set\text{-}mset\ (clauses_{NOT}\ T) \subseteq set\text{-}mset\ (not\text{-}simplified\text{-}cls\ (clauses_{NOT}\ S))$
$\cup\ simple\text{-}clss\ (atms\text{-}of\text{-}ms\ A) \rangle$
⟨*proof*⟩

abbreviation μ_{CDCL}'*-bound* **where**
$\langle \mu_{CDCL}'\text{-}bound\ A\ T \equiv ((2{+}card\ (atms\text{-}of\text{-}ms\ A))\ \widehat{}\ (1{+}card\ (atms\text{-}of\text{-}ms\ A))) * 2$
$+\ card\ (set\text{-}mset\ (not\text{-}simplified\text{-}cls(clauses_{NOT}\ T))) \rangle$

$+ 3 \hat{} card\ (atms\text{-}of\text{-}ms\ A)\rangle$

lemma *rtranclp-cdcl$_{NOT}$-merged-bj-learn-clauses-bound-card*:
assumes
$\langle cdcl_{NOT}\text{-}merged\text{-}bj\text{-}learn^{**}\ S\ T\rangle$ **and**
$\langle inv\ S\rangle$ **and**
$\langle atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \subseteq atms\text{-}of\text{-}ms\ A\rangle$ **and**
$\langle atm\text{-}of\ `(lits\text{-}of\text{-}l\ (trail\ S)) \subseteq atms\text{-}of\text{-}ms\ A\rangle$ **and**
finite: $\langle finite\ A\rangle$
shows $\langle \mu_{CDCL}'\text{-}merged\ A\ T \leq \mu_{CDCL}'\text{-}bound\ A\ S\rangle$
⟨*proof*⟩

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

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

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

sublocale *cdcl$_{NOT}$-increasing-restarts - - - - - - f*
$\langle \lambda S\ T.\ T \sim reduce\text{-}trail\text{-}to_{NOT}\ ([]::'a\ list)\ S\rangle$
$\langle \lambda A\ S.\ atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \subseteq atms\text{-}of\text{-}ms\ A$
$\wedge\ atm\text{-}of\ `\ lits\text{-}of\text{-}l\ (trail\ S) \subseteq atms\text{-}of\text{-}ms\ A \wedge finite\ A\rangle$
μ_{CDCL}'-*merged cdcl$_{NOT}$-merged-bj-learn*
$\langle \lambda S.\ inv\ S \wedge no\text{-}dup\ (trail\ S)\rangle$
$\langle \lambda A\ T.\ ((2{+}card\ (atms\text{-}of\text{-}ms\ A))\ \hat{}\ (1{+}card\ (atms\text{-}of\text{-}ms\ A))) * 2$
$+\ card\ (set\text{-}mset\ (not\text{-}simplified\text{-}cls(clauses_{NOT}\ T)))$
$+ 3 \hat{} card\ (atms\text{-}of\text{-}ms\ A)\rangle$
⟨*proof*⟩

lemma *true-clss-ext-decrease-right-insert*: $\langle I \models sext\ insert\ C\ (set\text{-}mset\ M) \Longrightarrow I \models sextm\ M\rangle$
⟨*proof*⟩

lemma *true-clss-ext-decrease-add-implied*:
assumes ‹$M \models pm\ C$›
shows ‹$I \models sext\ insert\ C\ (set\text{-}mset\ M) \longleftrightarrow I \models sextm\ M$›
⟨*proof*⟩

lemma *cdcl$_{NOT}$-merged-bj-learn-bj-sat-ext-iff*:
assumes ‹*cdcl$_{NOT}$-merged-bj-learn S T*› **and** *inv*: ‹*inv S*›
shows ‹$I \models sextm\ clauses_{NOT}\ S \longleftrightarrow I \models sextm\ clauses_{NOT}\ T$›
⟨*proof*⟩

lemma *rtranclp-cdcl$_{NOT}$-merged-bj-learn-bj-sat-ext-iff*:
assumes ‹*cdcl$_{NOT}$-merged-bj-learn** S T*›**and** ‹*inv S*›
shows ‹$I \models sextm\ clauses_{NOT}\ S \longleftrightarrow I \models sextm\ clauses_{NOT}\ T$›
⟨*proof*⟩

lemma *cdcl$_{NOT}$-restart-eq-sat-iff*:
assumes
‹*cdcl$_{NOT}$-restart S T*› **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-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*)) ⊆ *atms-of-ms A*› **and**

atms-trail: ‹*atm-of* ‘ *lits-of-l* (*trail* (*fst S*)) ⊆ *atms-of-ms A*› **and**
fin: ‹*finite A*›
shows ‹*unsatisfiable* (*set-mset* (*clauses*$_{NOT}$ (*fst S*)))
∨ *lits-of-l* (*trail* (*fst T*)) ⊨*sextm* *clauses*$_{NOT}$ (*fst S*) ∧
satisfiable (*set-mset* (*clauses*$_{NOT}$ (*fst S*)))›
⟨*proof*⟩

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

end
theory *CDCL-WNOT*
imports *CDCL-NOT CDCL-W-Merge*
begin

2.3 Link between Weidenbach's and NOT's CDCL

2.3.1 Inclusion of the states

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

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

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

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

lemma *is-decided-convert-trail-from-W*[*simp*]:
‹*is-decided* (*convert-ann-lit-from-W L*) = *is-decided L*›
⟨*proof*⟩

lemma *count-decided-conver-Trail-from-W*[*simp*]:
‹*count-decided* (*convert-trail-from-W M*) = *count-decided M*›
⟨*proof*⟩

The values *0* and {#} are dummy values.

consts *dummy-cls* :: *'cls*
fun *convert-ann-lit-from-NOT*
:: (*'v*, *'mark*) *ann-lit* ⇒ (*'v*, *'cls*) *ann-lit* **where**
convert-ann-lit-from-NOT (*Propagated L* -) = *Propagated L dummy-cls* |
convert-ann-lit-from-NOT (*Decided L*) = *Decided L*

abbreviation *convert-trail-from-NOT* **where**
convert-trail-from-NOT ≡ *map convert-ann-lit-from-NOT*

lemma *undefined-lit-convert-trail-from-NOT*[*simp*]:
undefined-lit (*convert-trail-from-NOT F*) *L* ⟷ *undefined-lit F L*
⟨*proof*⟩

lemma *lits-of-l-convert-trail-from-NOT*:
lits-of-l (*convert-trail-from-NOT F*) = *lits-of-l F*
⟨*proof*⟩

lemma *convert-trail-from-W-from-NOT*[*simp*]:
convert-trail-from-W (*convert-trail-from-NOT M*) = *M*
⟨*proof*⟩

lemma *convert-trail-from-W-convert-lit-from-NOT*[*simp*]:
convert-ann-lit-from-W (*convert-ann-lit-from-NOT L*) = *L*
⟨*proof*⟩

abbreviation $trail_{NOT}$ **where**
$trail_{NOT}$ *S* ≡ *convert-trail-from-W* (*fst S*)

lemma *undefined-lit-convert-trail-from-W*[*iff*]:
undefined-lit (*convert-trail-from-W M*) *L* ⟷ *undefined-lit M L*
⟨*proof*⟩

lemma *lit-of-convert-ann-lit-from-NOT*[*iff*]:
lit-of (*convert-ann-lit-from-NOT L*) = *lit-of L*
⟨*proof*⟩

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

sublocale $state_W \subseteq$ *dpll-state* **where**
$trail = \lambda S.\ convert\text{-}trail\text{-}from\text{-}W\ (trail\ S)$ **and**
$clauses_{NOT} = clauses$ **and**
$prepend\text{-}trail = \lambda L\ S.\ cons\text{-}trail\ (convert\text{-}ann\text{-}lit\text{-}from\text{-}NOT\ L)\ S$ **and**
$tl\text{-}trail = \lambda S.\ tl\text{-}trail\ S$ **and**
$add\text{-}cls_{NOT} = \lambda C\ S.\ add\text{-}learned\text{-}cls\ C\ S$ **and**
$remove\text{-}cls_{NOT} = \lambda C\ S.\ remove\text{-}cls\ C\ S$
⟨*proof*⟩

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

2.3.2 Inclusion of Weidendenbch's CDCL without Strategy

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

sublocale $conflict\text{-}driven\text{-}clause\text{-}learning_W \subseteq cdcl_{NOT}\text{-}merge\text{-}bj\text{-}learn\text{-}proxy$ **where**
$trail = \lambda S.\ convert\text{-}trail\text{-}from\text{-}W\ (trail\ S)$ **and**
$clauses_{NOT} = clauses$ **and**
$prepend\text{-}trail = \lambda L\ S.\ cons\text{-}trail\ (convert\text{-}ann\text{-}lit\text{-}from\text{-}NOT\ L)\ S$ **and**
$tl\text{-}trail = \lambda S.\ tl\text{-}trail\ S$ **and**
$add\text{-}cls_{NOT} = \lambda C\ S.\ add\text{-}learned\text{-}cls\ C\ S$ **and**
$remove\text{-}cls_{NOT} = \lambda C\ S.\ remove\text{-}cls\ C\ S$ **and**
$decide\text{-}conds = \lambda\text{- -}.\ True$ **and**
$propagate\text{-}conds = \lambda\text{- - -}.\ True$ **and**
$forget\text{-}conds = \lambda\text{-}\ S.\ conflicting\ S = None$ **and**
$backjump\text{-}l\text{-}cond = backjump\text{-}l\text{-}cond$ **and**
$inv = inv_{NOT}$
⟨*proof*⟩

sublocale $conflict\text{-}driven\text{-}clause\text{-}learning_W \subseteq cdcl_{NOT}\text{-}merge\text{-}bj\text{-}learn$ **where**
$trail = \lambda S.\ convert\text{-}trail\text{-}from\text{-}W\ (trail\ S)$ **and**
$clauses_{NOT} = clauses$ **and**
$prepend\text{-}trail = \lambda L\ S.\ cons\text{-}trail\ (convert\text{-}ann\text{-}lit\text{-}from\text{-}NOT\ L)\ S$ **and**
$tl\text{-}trail = \lambda S.\ tl\text{-}trail\ S$ **and**
$add\text{-}cls_{NOT} = \lambda C\ S.\ add\text{-}learned\text{-}cls\ C\ S$ **and**
$remove\text{-}cls_{NOT} = \lambda C\ S.\ remove\text{-}cls\ C\ S$ **and**
$decide\text{-}conds = \lambda\text{- -}.\ True$ **and**
$propagate\text{-}conds = \lambda\text{- - -}.\ True$ **and**
$forget\text{-}conds = \lambda\text{-}\ S.\ conflicting\ S = None$ **and**
$backjump\text{-}l\text{-}cond = backjump\text{-}l\text{-}cond$ **and**

$inv = inv_{NOT}$
⟨*proof*⟩

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

Notations are lost while proving locale inclusion:

notation *state-eq*$_{NOT}$ (**infix** $\sim_{NOT}$ *50*)

2.3.3 Additional Lemmas between NOT and W states

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

lemma *trail-reduce-trail-to*$_{NOT}$*-add-learned-cls*:
$no\text{-}dup\ (trail\ S) \Longrightarrow$
$trail\ (reduce\text{-}trail\text{-}to_{NOT}\ M\ (add\text{-}learned\text{-}cls\ D\ S)) = trail\ (reduce\text{-}trail\text{-}to_{NOT}\ M\ S)$
⟨*proof*⟩

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

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

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

lemma *skip-or-resolve-state-change*:
assumes $skip\text{-}or\text{-}resolve^{**}\ S\ T$
shows
$\exists M.\ trail\ S = M\ @@\ trail\ T \wedge (\forall m \in set\ M.\ \neg is\text{-}decided\ m)$
$clauses\ S = clauses\ T$
$backtrack\text{-}lvl\ S = backtrack\text{-}lvl\ T$
$init\text{-}clss\ S = init\text{-}clss\ T$
$learned\text{-}clss\ S = learned\text{-}clss\ T$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

lemma $cdcl_W$-*conflicting-true-*$cdcl_W$-*merge-restart*:
  assumes $cdcl_W$ S V **and** *confl*: *conflicting* S = *None*
  shows ($cdcl_W$-*merge* S $V \wedge$ *conflicting* V = *None*) $\vee$ (*conflicting* $V \neq$ *None* $\wedge$ *conflict* S V)
  ⟨*proof*⟩

lemma *trancl-*$cdcl_W$-*conflicting-true-*$cdcl_W$-*merge-restart*:
  assumes $cdcl_W{}^{++}$ S V **and** *inv*: $cdcl_W$-*M-level-inv* S **and** *conflicting* S = *None*
  shows ($cdcl_W$-*merge*$^{++}$ S $V \wedge$ *conflicting* V = *None*)
    $\vee$ ($\exists$ T U. $cdcl_W$-*merge*** S $T \wedge$ *conflicting* $V \neq$ *None* $\wedge$ *conflict* T $U \wedge$ $cdcl_W$-*bj*** U V)
  ⟨*proof*⟩

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

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

end

2.3.5 Inclusion of Weidendenbch's CDCL with Strategy

context *conflict-driven-clause-learning*$_W$
begin
abbreviation *propagate-conds* **where**
propagate-conds $\equiv$ λ-. *propagate*

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

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

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

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

interpretation *cdcl*$_W$*-with-strategy*: *cdcl*$_{NOT}$*-merge-bj-learn-proxy* **where**
trail = λS. *convert-trail-from-W* (*trail* S) **and**
clauses$_{NOT}$ = *clauses* **and**
prepend-trail = λL S. *cons-trail* (*convert-ann-lit-from-NOT* L) S **and**
tl-trail = λS. *tl-trail* S **and**
add-cls$_{NOT}$ = λC S. *add-learned-cls* C S **and**
remove-cls$_{NOT}$ = λC S. *remove-cls* C S **and**
decide-conds = *decide-conds* **and**
propagate-conds = *propagate-conds* **and**
forget-conds = λ- -. *False* **and**
backjump-l-cond = *backjump-l-conds-stgy* **and**
inv = *inv*$_{NOT}$*-stgy*
⟨*proof*⟩

lemma *cdcl*$_W$*-with-strategy-cdcl*$_{NOT}$*-merged-bj-learn-conflict*:
assumes
cdcl$_W$*-with-strategy.cdcl*$_{NOT}$*-merged-bj-learn* S T
conflicting S = *None*
shows
conflicting T = *None*
⟨*proof*⟩

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

⟨*proof*⟩

lemma *cdcl$_W$-with-strategy-cdcl$_{NOT}$-merged-bj-learn-cdcl$_W$-stgy*:
assumes
cdcl$_W$-with-strategy.cdcl$_{NOT}$-merged-bj-learn S V
shows
cdcl$_W$-stgy** S V
⟨*proof*⟩

lemma *rtranclp-transition-function*:
‹R^{**} *a b* $\Longrightarrow$ $\exists f\ j.\ (\forall i{<}j.\ R\ (f\ i)\ (f\ (Suc\ i))) \wedge f\ 0 = a \wedge f\ j = b$›
⟨*proof*⟩

lemma *cdcl$_W$-bj-cdcl$_W$-stgy*: ‹*cdcl$_W$-bj S T* $\Longrightarrow$ *cdcl$_W$-stgy S T*›
⟨*proof*⟩

lemma *cdcl$_W$-restart-propagated-clauses-clauses*:
‹*cdcl$_W$-restart S T* $\Longrightarrow$ *propagated-clauses-clauses S* $\Longrightarrow$ *propagated-clauses-clauses T*›
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-restart-propagated-clauses-clauses*:
‹*cdcl$_W$-restart** S T* $\Longrightarrow$ *propagated-clauses-clauses S* $\Longrightarrow$ *propagated-clauses-clauses T*›
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-stgy-propagated-clauses-clauses*:
‹*cdcl$_W$-stgy** S T* $\Longrightarrow$ *propagated-clauses-clauses S* $\Longrightarrow$ *propagated-clauses-clauses T*›
⟨*proof*⟩

lemma *conflicting-clause-bt-lvl-gt-0-backjump*:
assumes
inv: ‹*inv$_{NOT}$-stgy S*› **and**
C: ‹*C* $\in\#$ *clauses S*› **and**
tr-C: ‹*trail S* $\models as$ *CNot C*› **and**
bt: ‹*backtrack-lvl S* > 0›
shows ‹$\exists\ T\ U\ V.$ *conflict S T* $\wedge$ *full skip-or-resolve T U* $\wedge$ *backtrack U V*›
⟨*proof*⟩

lemma *conflict-full-skip-or-resolve-backtrack-backjump-l*:
assumes
conf: ‹*conflict S T*› **and**
full: ‹*full skip-or-resolve T U*› **and**
bt: ‹*backtrack U V*› **and**
inv: ‹*cdcl$_W$-all-struct-inv S*›
shows ‹*cdcl$_W$-with-strategy.backjump-l S V*›
⟨*proof*⟩

lemma *is-decided-o-convert-ann-lit-from-W*[*simp*]:
‹*is-decided o convert-ann-lit-from-W* $=$ *is-decided*›
⟨*proof*⟩

lemma *cdcl$_W$-with-strategy-propagate$_{NOT}$-propagate-iff*[*iff*]:
‹*cdcl$_W$-with-strategy.propagate$_{NOT}$ S T* $\longleftrightarrow$ *propagate S T*› (**is** *?NOT* $\longleftrightarrow$ *?W*)
⟨*proof*⟩

interpretation *cdcl$_W$-with-strategy*: *cdcl$_{NOT}$-merge-bj-learn* **where**

trail = λS. *convert-trail-from-W* (*trail S*) **and**
clauses$_{NOT}$ = *clauses* **and**
prepend-trail = $\lambda L\ S$. *cons-trail* (*convert-ann-lit-from-NOT L*) *S* **and**
tl-trail = λS. *tl-trail S* **and**
add-cls$_{NOT}$ = $\lambda C\ S$. *add-learned-cls C S* **and**
remove-cls$_{NOT}$ = $\lambda C\ S$. *remove-cls C S* **and**
decide-conds = *decide-conds* **and**
propagate-conds = *propagate-conds* **and**
forget-conds = λ- -. *False* **and**
backjump-l-cond = *backjump-l-conds-stgy* **and**
inv = *inv*$_{NOT}$*-stgy*
⟨*proof*⟩

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

end

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

context *conflict-driven-clause-learning*$_W$
begin
lemma *rtranclp-cdcl*$_W$*-merge-stgy-distinct-mset-clauses*:
assumes
invR: *cdcl*$_W$*-all-struct-inv R* **and**
st: *cdcl*$_W$*-s'*** *R S* **and**
smaller: ‹*no-smaller-propa R*› **and**
dist: *distinct-mset* (*clauses R*)
shows *distinct-mset* (*clauses S*)
⟨*proof*⟩
end

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

Chapter 3

Extensions on Weidenbach's CDCL

We here extend our calculus.

3.1 Restarts

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

This is an unrestricted version.

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

lemma *cdcl*$_W$*-stgy-cdcl*$_W$*-restart*: ‹*cdcl*$_W$*-stgy* $S\ S' \Longrightarrow$ *cdcl*$_W$*-restart* $S\ S'$›
⟨*proof*⟩

lemma *cdcl*$_W$*-restart-stgy-cdcl*$_W$*-restart*:
‹*cdcl*$_W$*-restart-stgy* $S\ T \Longrightarrow$ *cdcl*$_W$*-restart* (*fst* S) (*fst* T)›
⟨*proof*⟩

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

lemma *cdcl*$_W$*-stgy-cdcl*$_W$*-restart-stgy*:
‹*cdcl*$_W$*-stgy* $S\ T \Longrightarrow$ *cdcl*$_W$*-restart-stgy* (S, n) (T, n)›
⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-stgy-cdcl*$_W$*-restart-stgy*:
‹*cdcl*$_W$*-stgy*** $S\ T \Longrightarrow$ *cdcl*$_W$*-restart-stgy*** (S, n) (T, n)›
⟨*proof*⟩

lemma *cdcl*$_W$*-restart-dcl*$_W$*-all-struct-inv*:
‹*cdcl*$_W$*-restart-stgy* $S\ T \Longrightarrow$ *cdcl*$_W$*-all-struct-inv* (*fst* S) $\Longrightarrow$ *cdcl*$_W$*-all-struct-inv* (*fst* T)›
⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-restart-dcl*$_W$*-all-struct-inv*:
‹*cdcl*$_W$*-restart-stgy*** $S\ T \Longrightarrow$ *cdcl*$_W$*-all-struct-inv* (*fst* S) $\Longrightarrow$ *cdcl*$_W$*-all-struct-inv* (*fst* T)›
⟨*proof*⟩

lemma *restart-cdcl*$_W$*-stgy-invariant*:

⟨*restart S T* ⟹ *cdcl$_W$-stgy-invariant T*⟩
⟨*proof*⟩

lemma *cdcl$_W$-restart-dcl$_W$-stgy-invariant*:
⟨*cdcl$_W$-restart-stgy S T* ⟹ *cdcl$_W$-all-struct-inv* (*fst S*) ⟹ *cdcl$_W$-stgy-invariant* (*fst S*) ⟹
cdcl$_W$-stgy-invariant (*fst T*)⟩
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-restart-dcl$_W$-stgy-invariant*:
⟨*cdcl$_W$-restart-stgy** S T* ⟹ *cdcl$_W$-all-struct-inv* (*fst S*) ⟹ *cdcl$_W$-stgy-invariant* (*fst S*) ⟹
cdcl$_W$-stgy-invariant (*fst T*)⟩
⟨*proof*⟩

end

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

— get state:
init-state
for
state-eq :: ⟨*'st* ⇒ *'st* ⇒ *bool*⟩ (**infix** ∼ *50*) **and**
state :: ⟨*'st* ⇒ (*'v*, *'v clause*) *ann-lits* × *'v clauses* × *'v clauses* × *'v clause option* ×
'b⟩ **and**
trail :: ⟨*'st* ⇒ (*'v*, *'v clause*) *ann-lits*⟩ **and**
init-clss :: ⟨*'st* ⇒ *'v clauses*⟩ **and**
learned-clss :: ⟨*'st* ⇒ *'v clauses*⟩ **and**
conflicting :: ⟨*'st* ⇒ *'v clause option*⟩ **and**

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

init-state :: ⟨*'v clauses* ⇒ *'st*⟩ +
fixes
f :: ⟨*nat* ⇒ *nat*⟩

locale *cdcl$_W$-restart-restart* =
cdcl$_W$-restart-restart-ops +
assumes
f: ⟨*unbounded f*⟩

The condition of the differences of cardinality has to be strict. Otherwise, you could be in a strange state, where nothing remains to do, but a restart is done. See the proof of well-foundedness. The same applies for the *cdcl$_W$-stgy$^{+\downarrow}$ S T*: With a *cdcl$_W$-stgy$^{\downarrow}$ S T*, this rules could be applied one after the other, doing nothing each time.

context $cdcl_W$-*restart-restart-ops*
begin
inductive $cdcl_W$-*merge-with-restart* **where**
restart-step:
‹($cdcl_W$-*stgy*$^{\frown\frown}$(*card* (*set-mset* (*learned-clss* *T*)) − *card* (*set-mset* (*learned-clss* *S*)))) *S* *T*
⟹ *card* (*set-mset* (*learned-clss* *T*)) − *card* (*set-mset* (*learned-clss* *S*)) > *f* *n*
⟹ *restart* *T* *U* ⟹ $cdcl_W$-*merge-with-restart* (*S*, *n*) (*U*, *Suc* *n*)› |
restart-full: ‹*full1* $cdcl_W$-*stgy* *S* *T* ⟹ $cdcl_W$-*merge-with-restart* (*S*, *n*) (*T*, *Suc* *n*)›

lemma $cdcl_W$-*merge-with-restart-rtranclp-cdcl*$_W$-*restart*:
‹$cdcl_W$-*merge-with-restart* *S* *T* ⟹ $cdcl_W$-*restart*** (*fst* *S*) (*fst* *T*)›
⟨*proof*⟩

lemma $cdcl_W$-*merge-with-restart-increasing-number*:
‹$cdcl_W$-*merge-with-restart* *S* *T* ⟹ *snd* *T* = 1 + *snd* *S*›
⟨*proof*⟩

lemma ‹*full1* $cdcl_W$-*stgy* *S* *T* ⟹ $cdcl_W$-*merge-with-restart* (*S*, *n*) (*T*, *Suc* *n*)›
⟨*proof*⟩

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

lemma $cdcl_W$-*merge-with-restart-init-clss*:
‹$cdcl_W$-*merge-with-restart* *S* *T* ⟹ $cdcl_W$-*M-level-inv* (*fst* *S*) ⟹
init-clss (*fst* *S*) = *init-clss* (*fst* *T*)›
⟨*proof*⟩

lemma (**in** $cdcl_W$-*restart-restart*)
‹*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: ‹*no-smaller-propa* (*fst* *R*)›
shows ‹*distinct-mset* (*clauses* (*fst* *S*))›
⟨*proof*⟩

inductive $cdcl_W$-*restart-with-restart* **where**
restart-step:
‹$cdcl_W$-*stgy*** *S* *T* ⟹
card (*set-mset* (*learned-clss* *T*)) − *card* (*set-mset* (*learned-clss* *S*)) > *f* *n* ⟹
restart *T* *U* ⟹
$cdcl_W$-*restart-with-restart* (*S*, *n*) (*U*, *Suc* *n*)› |
restart-full: ‹*full1* $cdcl_W$-*stgy* *S* *T* ⟹ $cdcl_W$-*restart-with-restart* (*S*, *n*) (*T*, *Suc* *n*)›

lemma $cdcl_W$-*restart-with-restart-rtranclp-cdcl*$_W$-*restart*:
‹$cdcl_W$-*restart-with-restart* *S* *T* ⟹ $cdcl_W$-*restart*** (*fst* *S*) (*fst* *T*)›
⟨*proof*⟩

lemma $cdcl_W$-*restart-with-restart-increasing-number*:
‹$cdcl_W$-*restart-with-restart* *S* *T* ⟹ *snd* *T* = 1 + *snd* *S*›

⟨*proof*⟩

lemma ‹*full1 cdcl$_W$-stgy S T* ⟹ *cdcl$_W$-restart-with-restart* (*S*, *n*) (*T*, *Suc n*)›
⟨*proof*⟩

lemma *cdcl$_W$-restart-with-restart-init-clss*:
‹*cdcl$_W$-restart-with-restart S T* ⟹ *cdcl$_W$-M-level-inv* (*fst S*) ⟹
init-clss (*fst S*) = *init-clss* (*fst T*)›
⟨*proof*⟩

theorem (**in** *cdcl$_W$-restart-restart*)
‹*wf* {(*T*, *S*). *cdcl$_W$-all-struct-inv* (*fst S*) ∧ *cdcl$_W$-restart-with-restart S T*}›
⟨*proof*⟩

lemma *cdcl$_W$-restart-with-restart-distinct-mset-clauses*:
assumes *invR*: ‹*cdcl$_W$-all-struct-inv* (*fst R*)› **and**
st: ‹*cdcl$_W$-restart-with-restart R S*› **and**
dist: ‹*distinct-mset* (*clauses* (*fst R*))› **and**
R: ‹*no-smaller-propa* (*fst R*)›
shows ‹*distinct-mset* (*clauses* (*fst S*))›
⟨*proof*⟩

end

locale *luby-sequence* =
fixes *ur* :: *nat*
assumes ‹*ur* > *0*›
begin

lemma *exists-luby-decomp*:
fixes *i* ::*nat*
shows ‹∃*k*::*nat*. (*2* ^ (*k* − *1*) ≤ *i* ∧ *i* < *2* ^ *k* − *1*) ∨ *i* = *2* ^ *k* − *1*›
⟨*proof*⟩

Luby sequences are defined by:

- $2^k - 1$, if $i = (2::'a)^k - (1::'a)$
- *luby-sequence-core* $(i - 2^{k-1} + 1)$, if $(2::'a)^{k-1} \leq i$ and $i \leq (2::'a)^k - (1::'a)$

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

function *luby-sequence-core* :: ‹*nat* ⇒ *nat*› **where**
‹*luby-sequence-core i* =
(*if* ∃ *k*. *i* = *2*^*k* − *1*
then *2*^((*SOME k*. *i* = *2*^*k* − *1*) − *1*)
else luby-sequence-core (*i* − *2*^((*SOME k*. *2*^(*k*−*1*)≤ *i* ∧ *i* < *2*^*k* − *1*) − *1*) + *1*))›
⟨*proof*⟩
termination
⟨*proof*⟩

declare *luby-sequence-core.simps*[*simp del*]

lemma *two-pover-n-eq-two-power-n'-eq*:
assumes *H*: ‹(*2*::*nat*) ^ (*k*::*nat*) − *1* = *2* ^ *k'* − *1*›
shows ‹*k'* = *k*›

⟨*proof*⟩

lemma *luby-sequence-core-two-power-minus-one*:
  ‹*luby-sequence-core* (*2^k − 1*) = *2^(k−1)*› (**is** ‹*?L* = *?K*›)
⟨*proof*⟩

lemma *different-luby-decomposition-false*:
  assumes
    H: ‹*2 ^ (k − Suc 0)* ≤ *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)* ≤ *i*› **and**
    i-k: ‹*i* < *2^k − 1*›
  shows ‹*luby-sequence-core i* = *luby-sequence-core* (*i − 2 ^ (k − 1) + 1*)›
⟨*proof*⟩

lemma *unbounded-luby-sequence-core*: ‹*unbounded luby-sequence-core*›
  ⟨*proof*⟩

abbreviation *luby-sequence* :: ‹*nat* ⇒ *nat*› **where**
‹*luby-sequence n* ≡ *ur* ∗ *luby-sequence-core n*›

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

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

lemma ‹*luby-sequence-core n* ≥ *1*›
⟨*proof*⟩
end

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

      — get state:
    init-state
  for
    ur :: *nat* **and**
    state-eq :: ‹*'st* ⇒ *'st* ⇒ *bool*› (**infix** ∼ *50*) **and**
    state :: ‹*'st* ⇒ (*'v*, *'v clause*) *ann-lits* × *'v clauses* × *'v clauses* × *'v clause option* ×
      'b› **and**
    trail :: ‹*'st* ⇒ (*'v*, *'v clause*) *ann-lits*› **and**

hd-trail :: ‹*'st* ⇒ (*'v*, *'v clause*) *ann-lit*› **and**
init-clss :: ‹*'st* ⇒ *'v clauses*› **and**
learned-clss :: ‹*'st* ⇒ *'v clauses*› **and**
conflicting :: ‹*'st* ⇒ *'v clause option*› **and**

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

init-state :: ‹*'v clauses* ⇒ *'st*›
begin

sublocale $cdcl_W$*-restart-restart* **where**
f = *luby-sequence*
⟨*proof*⟩

end

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

3.2 Incremental SAT solving

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

— Some specific states:
init-state
for
state-eq :: *'st* ⇒ *'st* ⇒ *bool* (**infix** ∼ *50*) **and**
state :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* × *'v clauses* × *'v clauses* × *'v clause option* × *'b* **and**
trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **and**
init-clss :: *'st* ⇒ *'v clauses* **and**
learned-clss :: *'st* ⇒ *'v clauses* **and**
conflicting :: *'st* ⇒ *'v clause option* **and**

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

init-state :: *'v clauses* ⇒ *'st* +
fixes
add-init-cls :: *'v clause* ⇒ *'st* ⇒ *'st*
assumes
add-init-cls:
state st = (*M*, *N*, *U*, *S'*) ⟹
state (*add-init-cls C st*) = (*M*, {#*C*#} + *N*, *U*, *S'*)

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

— Some specific states:
init-state
add-init-cls
for
state-eq :: *'st* ⇒ *'st* ⇒ *bool* (**infix** ∼ *50*) **and**
state :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* × *'v clauses* × *'v clauses* × *'v clause option* ×
'b **and**
trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **and**
init-clss :: *'st* ⇒ *'v clauses* **and**
learned-clss :: *'st* ⇒ *'v clauses* **and**
conflicting :: *'st* ⇒ *'v clause option* **and**

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

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

locale *state*$_W$*-adding-init-clause* =
state$_W$*-adding-init-clause-ops*
state-eq
state
— functions about the state:
— getter:
trail init-clss learned-clss conflicting
— setter:
cons-trail tl-trail add-learned-cls remove-cls update-conflicting

— Some specific states:
init-state add-init-cls
for
state-eq :: *'st* ⇒ *'st* ⇒ *bool* (**infix** ∼ *50*) **and**

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

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

init-state :: *'v clauses* ⇒ *'st* **and**
add-init-cls :: *'v clause* ⇒ *'st* ⇒ *'st*
begin

sublocale *state*$_W$
⟨*proof*⟩

lemma
trail-add-init-cls[*simp*]:
trail (*add-init-cls C st*) = *trail st* **and**
init-clss-add-init-cls[*simp*]:
init-clss (*add-init-cls C st*) = {#*C*#} + *init-clss st*
and
learned-clss-add-init-cls[*simp*]:
learned-clss (*add-init-cls C st*) = *learned-clss st* **and**
conflicting-add-init-cls[*simp*]:
conflicting (*add-init-cls C st*) = *conflicting st*
⟨*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*
state-eq
state
— functions for the state:
— access functions:
trail init-clss learned-clss conflicting
— changing state:
cons-trail tl-trail add-learned-cls remove-cls update-conflicting

— get state:

init-state
— Adding a clause:
add-init-cls
for
state-eq :: *'st* ⇒ *'st* ⇒ *bool* (**infix** ∼ *50*) **and**
state :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* × *'v clauses* × *'v clauses* × *'v clause option* ×
'b **and**
trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **and**
init-clss :: *'st* ⇒ *'v clauses* **and**
learned-clss :: *'st* ⇒ *'v clauses* **and**
conflicting :: *'st* ⇒ *'v clause option* **and**

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

init-state :: *'v clauses* ⇒ *'st* **and**
add-init-cls :: *'v clause* ⇒ *'st* ⇒ *'st*
begin

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

This invariant holds all the invariant related to the strategy. See the structural invariant in *cdcl*$_W$*-all-struct-inv*

When we add a new clause, we reduce the trail until we get to tho first literal included in C. Then we can mark the conflict.

fun *cut-trail-wrt-clause* **where**
cut-trail-wrt-clause C [] *S* = *S* |
cut-trail-wrt-clause C (*Decided L* # *M*) *S* =
(*if* −*L* ∈# *C then S*
else cut-trail-wrt-clause C M (*tl-trail S*)) |
cut-trail-wrt-clause C (*Propagated L* - # *M*) *S* =
(*if* −*L* ∈# *C then S*
else cut-trail-wrt-clause C M (*tl-trail S*))

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

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

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

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

⟨*proof*⟩

lemma *trail-cut-trail-wrt-clause*:
∃ *M*. *trail S* = *M* @ *trail* (*cut-trail-wrt-clause C* (*trail S*) *S*)
⟨*proof*⟩

lemma *n-dup-no-dup-trail-cut-trail-wrt-clause*[*simp*]:
assumes *n-d*: *no-dup* (*trail T*)
shows *no-dup* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*))
⟨*proof*⟩

lemma *cut-trail-wrt-clause-backtrack-lvl-length-decided*:
assumes
backtrack-lvl T = *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*:
((∀ *L* ∈# *C*. −*L* ∉ *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 $cdcl_W$*-restart-s* or add a clause. Remark that we use $cdcl_W$*-restart-s* to avoid an explicit *skip*, *resolve*, and *backtrack* normalisation to get rid of the conflict C if possible.

inductive *incremental-cdcl*$_W$ :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* **where**
add-confl:
trail S ⊨*asm init-clss S* ⟹ *distinct-mset C* ⟹ *conflicting S* = *None* ⟹
trail S ⊨*as CNot C* ⟹
full cdcl$_W$*-stgy*
(*update-conflicting* (*Some C*)
(*add-init-cls C* (*cut-trail-wrt-clause C* (*trail S*) *S*))) *T* ⟹
incremental-cdcl$_W$ *S T* |
add-no-confl:
trail S ⊨*asm init-clss S* ⟹ *distinct-mset C* ⟹ *conflicting S* = *None* ⟹
¬*trail S* ⊨*as CNot C* ⟹
full cdcl$_W$*-stgy* (*add-init-cls C S*) *T* ⟹
incremental-cdcl$_W$ *S T*

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

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 *incremental-cdcl$_W$-inv*:
  assumes
    inc: *incremental-cdcl$_W$ S T* **and**
    inv: *cdcl$_W$-all-struct-inv S* **and**
    s-inv: *cdcl$_W$-stgy-invariant S* **and**
    learned-entailed: ‹*cdcl$_W$-learned-clauses-entailed-by-init S*›
  shows
    cdcl$_W$-all-struct-inv T **and**
    cdcl$_W$-stgy-invariant T **and**
    learned-entailed: ‹*cdcl$_W$-learned-clauses-entailed-by-init T*›
  ⟨*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* **and**
    learned-entailed: ‹*cdcl$_W$-learned-clauses-entailed-by-init S*›
  shows
    cdcl$_W$-all-struct-inv T **and**
    cdcl$_W$-stgy-invariant T **and**
    ‹*cdcl$_W$-learned-clauses-entailed-by-init T*›
    ⟨*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* **and**
    learned-entailed: ‹*cdcl$_W$-learned-clauses-entailed-by-init S*›
  shows *conflicting T* = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss T*))
    ∨ *conflicting T* = *None* ∧ *trail T* $\models$*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* **and**
    learned-entailed: ‹*cdcl$_W$-learned-clauses-entailed-by-init S*›
  shows *conflicting T* = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss T*))
    ∨ *conflicting T* = *None* ∧ *trail T* $\models$*asm init-clss T* ∧ *satisfiable* (*set-mset* (*init-clss T*))
  ⟨*proof*⟩

end

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

Chapter 4

List-based Implementation of DPLL and CDCL

We can now reuse all the theorems to go towards an implementation using 2-watched literals:

- `CDCL_W_Abstract_State.thy` defines a better-suited state: the operation operating on it are more constrained, allowing simpler proofs and less edge cases later.

4.1 Simple List-Based Implementation of the DPLL and CDCL

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

4.1.1 Common Rules

Propagation

The following theorem holds:

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

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

definition *is-unit-clause* :: *'a literal list* ⇒ (*'a*, *'b*) *ann-lits* ⇒ *'a literal option*
where
is-unit-clause l M =
(*case List.filter* (λ*a*. *atm-of a* ∉ *atm-of* ' *lits-of-l M*) *l of*
a # [] ⇒ *if M* ⊨*as CNot* (*mset l* − {#*a*#}) *then Some a else None*
| - ⇒ *None*)

definition *is-unit-clause-code* :: *'a literal list* ⇒ (*'a*, *'b*) *ann-lits*
⇒ *'a literal option* **where**
is-unit-clause-code l M =
(*case List.filter* (λ*a*. *atm-of a* ∉ *atm-of* ' *lits-of-l M*) *l of*
a # [] ⇒ *if* (∀ *c* ∈*set* (*remove1 a l*). −*c* ∈ *lits-of-l M*) *then Some a else None*
| - ⇒ *None*)

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

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

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

lemma *is-unit-clause-some-in*: *is-unit-clause l M = Some a ⟹ a ∈ set l*
  ⟨*proof*⟩

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

Unit propagation for all clauses

Finding the first clause to propagate

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

lemma *find-first-unit-clause-some*:
  find-first-unit-clause l M = Some (a, c)
  ⟹ c ∈ set l ∧ M ⊨as CNot (mset c − {#a#}) ∧ undefined-lit M a ∧ a ∈ set c
  ⟨*proof*⟩

lemma *propagate-is-unit-clause-not-None*:
  assumes
  M: *M ⊨as CNot (mset c − {#a#})* **and**
  undef: *undefined-lit M a* **and**
  ac: *a ∈ set c*
  shows *is-unit-clause c M ≠ None*
⟨*proof*⟩

lemma *find-first-unit-clause-none*:
  c ∈ set l ⟹ M ⊨as CNot (mset c − {#a#}) ⟹ undefined-lit M a ⟹ a ∈ set c
  ⟹ find-first-unit-clause l M ≠ None
  ⟨*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* (*λlit. lit ∉ M ∧ −lit ∉ M*) *a of*
    None ⇒ find-first-unused-var l M
  | *Some a ⇒ Some a*) |

find-first-unused-var [] - = *None*

lemma *find-none*[*iff*]:
List.find (λ*lit*. *lit* ∉ *M* ∧ −*lit* ∉ *M*) *a* = *None* ⟷ *atm-of* ‘ *set a* ⊆ *atm-of* ‘ *M*
⟨*proof*⟩

lemma *find-some*: *List.find* (λ*lit*. *lit* ∉ *M* ∧ −*lit* ∉ *M*) *a* = *Some b* ⟹ *b* ∈ *set a* ∧ *b* ∉ *M* ∧ −*b* ∉ *M*
⟨*proof*⟩

lemma *find-first-unused-var-None*[*iff*]:
find-first-unused-var l M = *None* ⟷ (∀ *a* ∈ *set l*. *atm-of* ‘ *set a* ⊆ *atm-of* ‘ *M*)
⟨*proof*⟩

lemma *find-first-unused-var-Some-not-all-incl*:
assumes *find-first-unused-var l M* = *Some c*
shows ¬(∀ *a* ∈ *set l*. *atm-of* ‘ *set a* ⊆ *atm-of* ‘ *M*)
⟨*proof*⟩

lemma *find-first-unused-var-Some*:
find-first-unused-var l M = *Some a* ⟹ (∃ *m* ∈ *set l*. *a* ∈ *set m* ∧ *a* ∉ *M* ∧ −*a* ∉ *M*)
⟨*proof*⟩

lemma *find-first-unused-var-undefined*:
find-first-unused-var l (*lits-of-l Ms*) = *Some a* ⟹ *undefined-lit Ms a*
⟨*proof*⟩

4.1.2 CDCL specific functions

Level

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

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

lemma [*code*]:
fixes *M* :: (′*a*, ′*b*) *ann-lits*
shows *get-maximum-level M* (*mset D*) = *maximum-level-code D M*
⟨*proof*⟩

Backjumping

fun *find-level-decomp* **where**
find-level-decomp M [] *D k* = *None* |
find-level-decomp M (*L* # *Ls*) *D k* =
(*case* (*get-level M L*, *maximum-level-code* (*D* @ *Ls*) *M*) *of*
(*i*, *j*) ⇒ *if i* = *k* ∧ *j* < *i then Some* (*L*, *j*) *else find-level-decomp M Ls* (*L*#*D*) *k*
)

lemma *find-level-decomp-some*:
assumes *find-level-decomp M Ls D k* = *Some* (*L*, *j*)
shows *L* ∈ *set Ls* ∧ *get-maximum-level M* (*mset* (*remove1 L* (*Ls* @ *D*))) = *j* ∧ *get-level M L* = *k*

⟨*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*, *Some D*) =
(*case find-level-decomp M D* [] (*count-decided M*) *of*
None ⇒ (*M*, *N*, *U*, *Some D*)
| *Some* (*L*, *j*) ⇒
(*case bt-cut j M of*
Some (*Decided* - # *Ls*) ⇒ (*Propagated L D* # *Ls*, *N*, *D* # *U*, *None*)
| - ⇒ (*M*, *N*, *U*, *Some D*))
) |
do-backtrack-step S = *S*

end
theory *DPLL-W-Implementation*
imports *DPLL-CDCL-W-Implementation DPLL-W HOL−Library.Code-Target-Numeral*
begin

4.1.3 Simple Implementation of DPLL

Combining the propagate and decide: a DPLL step

definition *DPLL-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 $toS \equiv \lambda(Ms::(int, unit)\ ann\text{-}lits)$
$(N::\ int\ literal\ list\ list).\ (Ms,\ mset\ (map\ mset\ N))$
abbreviation $toS' \equiv \lambda(Ms::(int, unit)\ ann\text{-}lits,$
$N::\ int\ literal\ list\ list).\ (Ms,\ mset\ (map\ mset\ N))$

Proof of correctness of *DPLL-step*

lemma *DPLL-step-is-a-dpll$_W$-step*:
assumes *step*: $(Ms',\ N') = DPLL\text{-}step\ (Ms,\ N)$
and *neq*: $(Ms,\ N) \neq (Ms',\ N')$
shows $dpll_W\ (toS\ Ms\ N)\ (toS\ Ms'\ N')$
⟨*proof*⟩

lemma *DPLL-step-stuck-final-state*:
assumes *step*: $(Ms,\ N) = DPLL\text{-}step\ (Ms,\ N)$
shows *conclusive-dpll$_W$-state* $(toS\ Ms\ N)$
⟨*proof*⟩

Adding invariants

Invariant tested in the function **function** *DPLL-ci* :: *int dpll$_W$-ann-lits* ⇒ *int literal list list* ⇒ *int dpll$_W$-ann-lits* × *int literal list list* **where**
DPLL-ci Ms N =
(*if* ¬*dpll$_W$-all-inv* (*Ms*, *mset* (*map mset N*))
then (*Ms*, *N*)
else
let (*Ms'*, *N'*) = *DPLL-step* (*Ms*, *N*) *in*
if (*Ms'*, *N'*) = (*Ms*, *N*) *then* (*Ms*, *N*) *else DPLL-ci Ms' N*)
⟨*proof*⟩
termination
⟨*proof*⟩

No invariant tested **function** (*domintros*) *DPLL-part*:: *int dpll$_W$-ann-lits* ⇒ *int literal list list* ⇒ *int dpll$_W$-ann-lits* × *int literal list list* **where**
DPLL-part Ms N =
(*let* (*Ms'*, *N'*) = *DPLL-step* (*Ms*, *N*) *in*
if (*Ms'*, *N'*) = (*Ms*, *N*) *then* (*Ms*, *N*) *else DPLL-part Ms' N*)
⟨*proof*⟩

lemma *snd-DPLL-step*[*simp*]:
snd (*DPLL-step* (*Ms*, *N*)) = *N*
⟨*proof*⟩

lemma *dpll*$_W$*-all-inv-implieS-2-eq3-and-dom*:
assumes *dpll*$_W$*-all-inv* (*Ms*, *mset* (*map mset N*))
shows *DPLL-ci Ms N* = *DPLL-part Ms N* ∧ *DPLL-part-dom* (*Ms*, *N*)
⟨*proof*⟩

lemma *DPLL-ci-dpll*$_W$*-rtranclp*:
assumes *DPLL-ci Ms N* = (*Ms'*, *N'*)
shows *dpll*$_W$** (*toS Ms N*) (*toS Ms' N*)
⟨*proof*⟩

lemma *dpll*$_W$*-all-inv-dpll*$_W$*-tranclp-irrefl*:
assumes *dpll*$_W$*-all-inv* (*Ms*, *N*)
and *dpll*$_W$$^{++}$ (*Ms*, *N*) (*Ms*, *N*)
shows *False*
⟨*proof*⟩

lemma *DPLL-ci-final-state*:
assumes *step*: *DPLL-ci Ms N* = (*Ms*, *N*)
and *inv*: *dpll*$_W$*-all-inv* (*toS Ms N*)
shows *conclusive-dpll*$_W$*-state* (*toS Ms N*)
⟨*proof*⟩

lemma *DPLL-step-obtains*:
obtains *Ms'* **where** (*Ms'*, *N*) = *DPLL-step* (*Ms*, *N*)
⟨*proof*⟩

lemma *DPLL-ci-obtains*:
obtains *Ms'* **where** (*Ms'*, *N*) = *DPLL-ci Ms N*
⟨*proof*⟩

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

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

Embedding the invariant into the type

Defining the type **typedef** *dpll*$_W$*-state* =
{(*M*::(*int*, *unit*) *ann-lits*, *N*::*int literal list list*).
dpll$_W$*-all-inv* (*toS M N*)}
morphisms *rough-state-of state-of*

⟨*proof*⟩

lemma
DPLL-part-dom ([], *N*)
⟨*proof*⟩

Some type classes **instantiation** $dpll_W$-*state* :: *equal*
begin
definition *equal-*$dpll_W$*-state* :: $dpll_W$-*state* ⇒ $dpll_W$-*state* ⇒ *bool* **where**
equal-$dpll_W$*-state S S′* = (*rough-state-of S* = *rough-state-of S′*)
instance
⟨*proof*⟩
end

DPLL **definition** *DPLL-step′* :: $dpll_W$-*state* ⇒ $dpll_W$-*state* **where**
DPLL-step′ S = *state-of* (*DPLL-step* (*rough-state-of S*))

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

lemma *DPLL-step-*$dpll_W$*-conc-inv*:
DPLL-step (*rough-state-of S*) ∈ {(*M*, *N*). $dpll_W$-*all-inv* (*toS M N*)}
⟨*proof*⟩

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

function *DPLL-tot*:: $dpll_W$-*state* ⇒ $dpll_W$-*state* **where**
DPLL-tot S =
(*let S′* = *DPLL-step′ S in*
if S′ = *S then S else DPLL-tot S′*)
⟨*proof*⟩
termination
⟨*proof*⟩

lemma [*code*]:
DPLL-tot S =
(*let S′* = *DPLL-step′ S in*
if S′ = *S then S else DPLL-tot S′*) ⟨*proof*⟩

lemma *DPLL-tot-DPLL-step-DPLL-tot*[*simp*]: *DPLL-tot* (*DPLL-step′ S*) = *DPLL-tot S*
⟨*proof*⟩

lemma *DOPLL-step′-DPLL-tot*[*simp*]:
DPLL-step′ (*DPLL-tot S*) = *DPLL-tot S*
⟨*proof*⟩

lemma *DPLL-tot-final-state*:
assumes *DPLL-tot S* = *S*
shows *conclusive-*$dpll_W$*-state* (*toS′* (*rough-state-of S*))
⟨*proof*⟩

lemma *DPLL-tot-star*:

assumes *rough-state-of* (*DPLL-tot S*) = *S′*
shows *dpll*$_W$** (*toS′* (*rough-state-of S*)) (*toS′ S′*)
⟨*proof*⟩

lemma *rough-state-of-rough-state-of-Nil*[*simp*]:
rough-state-of (*state-of* ([], *N*)) = ([], *N*)
⟨*proof*⟩

Theorem of correctness

lemma *DPLL-tot-correct*:
assumes *rough-state-of* (*DPLL-tot* (*state-of* (([], *N*)))) = (*M*, *N′*)
and (*M′*, *N″*) = *toS′* (*M*, *N′*)
shows *M′* ⊨*asm* *N″* ⟷ *satisfiable* (*set-mset N″*)
⟨*proof*⟩

Code export

A conversion to *DPLL-W-Implementation.dpll*$_W$*-state* **definition** *Con* :: (*int*, *unit*) *ann-lits* × *int literal list list*
⇒ *dpll*$_W$*-state* **where**
Con xs = *state-of* (*if dpll*$_W$*-all-inv* (*toS* (*fst xs*) (*snd xs*)) *then xs else* ([], []))
lemma [*code abstype*]:
Con (*rough-state-of S*) = *S*
⟨*proof*⟩

declare *rough-state-of-DPLL-step′-DPLL-step*[*code abstract*]

lemma *Con-DPLL-step-rough-state-of-state-of*[*simp*]:
Con (*DPLL-step* (*rough-state-of s*)) = *state-of* (*DPLL-step* (*rough-state-of s*))
⟨*proof*⟩

A slightly different version of *DPLL-tot* where the returned boolean indicates the result.

definition *DPLL-tot-rep* **where**
DPLL-tot-rep S =
(*let* (*M*, *N*) = (*rough-state-of* (*DPLL-tot S*)) *in* (∀ *A* ∈ *set N*. (∃ *a*∈*set A*. *a* ∈ *lits-of-l M*), *M*))

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

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

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

end
theory *CDCL-W-Implementation*
imports *DPLL-CDCL-W-Implementation CDCL-W-Termination*
HOL−Library.Code-Target-Numeral
begin

4.1.4 List-based CDCL Implementation

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

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

Types and Instantiation

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

type-synonym *'a cdcl$_W$-restart-mark = 'a clause*

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

abbreviation *raw-trail :: 'a × 'b × 'c × 'd ⇒ 'a* **where**
raw-trail ≡ (λ(M, -). M)

abbreviation *raw-cons-trail :: 'a ⇒ 'a list × 'b × 'c × 'd ⇒ 'a list × 'b × 'c × 'd*
where
raw-cons-trail ≡ (λL (M, S). (L#M, S))

abbreviation *raw-tl-trail :: 'a list × 'b × 'c × 'd ⇒ 'a list × 'b × 'c × 'd* **where**
raw-tl-trail ≡ (λ(M, S). (tl M, S))

abbreviation *raw-init-clss :: 'a × 'b × 'c × 'd ⇒ 'b* **where**
raw-init-clss ≡ λ(M, N, -). N

abbreviation *raw-learned-clss :: 'a × 'b × 'c × 'd ⇒ 'c* **where**
raw-learned-clss ≡ λ(M, N, U, -). U

abbreviation *raw-conflicting :: 'a × 'b × 'c × 'd ⇒ 'd* **where**
raw-conflicting ≡ λ(M, N, U, D). D

abbreviation *raw-update-conflicting :: 'd ⇒ 'a × 'b × 'c × 'd ⇒ 'a × 'b × 'c × 'd*
where
raw-update-conflicting ≡ λS (M, N, U, -). (M, N, U, S)

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

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

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

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

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

definition *state* **where**
‹*state S* = (*raw-trail S*, *raw-init-clss S*, *raw-learned-clss S*, *raw-conflicting S*, ())›

interpretation $state_W$
(=)
state
raw-trail raw-init-clss raw-learned-clss raw-conflicting
λ*L* (*M*, *S*). (*L* # *M*, *S*)
λ(*M*, *S*). (*tl M*, *S*)
λ*C* (*M*, *N*, *U*, *S*). (*M*, *N*, *add-mset C U*, *S*)
λ*C* (*M*, *N*, *U*, *S*). (*M*, *removeAll-mset C N*, *removeAll-mset C U*, *S*)
λ*D* (*M*, *N*, *U*, -). (*M*, *N*, *U*, *D*)
λ*N*. ([], *N*, {#}, *None*)
⟨*proof*⟩

declare *state-simp*[*simp del*]

interpretation $conflict\text{-}driven\text{-}clause\text{-}learning_W$
(=) *state*
raw-trail raw-init-clss raw-learned-clss
raw-conflicting
λ*L* (*M*, *S*). (*L* # *M*, *S*)
λ(*M*, *S*). (*tl M*, *S*)
λ*C* (*M*, *N*, *U*, *S*). (*M*, *N*, *add-mset C U*, *S*)
λ*C* (*M*, *N*, *U*, *S*). (*M*, *removeAll-mset C N*, *removeAll-mset C U*, *S*)
λ*D* (*M*, *N*, *U*, -). (*M*, *N*, *U*, *D*)
λ*N*. ([], *N*, {#}, *None*)
⟨*proof*⟩

declare *clauses-def*[*simp*]

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

lemma *reduce-trail-to′*:
reduce-trail-to F S =
((*if length* (*raw-trail S*) ≥ *length F*
then drop (*length* (*raw-trail S*) − *length F*) (*raw-trail S*)
else []), *raw-init-clss S*, *raw-learned-clss S*, *raw-conflicting S*)
(**is** *?S* = -)
⟨*proof*⟩

Definition of the rules

Types **lemma** *true-raw-init-clss-remdups*[*simp*]:
I ⊨*s* (*mset* ∘ *remdups*) ‘ *N* ⟷ *I* ⊨*s mset* ‘ *N*
⟨*proof*⟩

lemma *true-clss-raw-remdups-mset-mset*[*simp*]:
‹*I* ⊨*s* (λ*L*. *remdups-mset* (*mset L*)) ‘ *N′* ⟷ *I* ⊨*s mset* ‘ *N′*›

⟨*proof*⟩

declare *satisfiable-carac*[*iff del*]
lemma *satisfiable-mset-remdups*[*simp*]:
satisfiable ((*mset* ∘ *remdups*) ‘ *N*) ⟷ *satisfiable* (*mset* ‘ *N*)
satisfiable ((λ*L*. *remdups-mset* (*mset L*)) ‘ *N'*) ⟷ *satisfiable* (*mset* ‘ *N'*)
⟨*proof*⟩

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

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

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

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

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

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

lemma *is-decided-convert-is-decided*[*simp*]: ‹(*is-decided* ∘ *convert*) = (*is-decided*)›
⟨*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*⟩

lemma *count-decided-convert*[*simp*]:
‹*count-decided* (*map convert M*) = *count-decided M*›
⟨*proof*⟩

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

lemma *no-dup-convert*[*simp*]:
‹*no-dup* (*map convert M*) = *no-dup M*›
⟨*proof*⟩

Conversion function

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

Definition an abstract type

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

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

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

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

lemma *true-annot-map-convert*[*simp*]: *map convert M* ⊨*a N* ⟷ *M* ⊨*a N*
⟨*proof*⟩

lemma *true-annots-map-convert*[*simp*]: *map convert M* ⊨*as N* ⟷ *M* ⊨*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*, *None*)) (*toS* (*Propagated L C* # *M*, *N*, *U*, *None*))
⟨*proof*⟩

The Transitions

Propagate **definition** *do-propagate-step* :: ‹*'v cdcl$_W$-restart-state-inv-st* ⇒ *'v cdcl$_W$-restart-state-inv-st*›
where
do-propagate-step S =
(*case S of*
(*M*, *N*, *U*, *None*) ⇒
(*case find-first-unit-clause* (*N* @ *U*) *M of*
Some (*L*, *C*) ⇒ (*Propagated L C* # *M*, *N*, *U*, *None*)
| *None* ⇒ (*M*, *N*, *U*, *None*))
| *S* ⇒ *S*)

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

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

lemma *do-propagate-step-no-step*:
assumes *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* ⊨*as CNot* (*mset N*)
⟨*proof*⟩

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

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

definition *do-conflict-step* :: ‹'*v cdcl*$_W$*-restart-state-inv-st* ⇒ '*v cdcl*$_W$*-restart-state-inv-st*› **where**
do-conflict-step S =
(*case S of*
(*M*, *N*, *U*, *None*) ⇒
(*case find-conflict M* (*N* @ *U*) *of*
Some a ⇒ (*M*, *N*, *U*, *Some a*)
| *None* ⇒ (*M*, *N*, *U*, *None*))
| *S* ⇒ *S*)

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

Skip **fun** *do-skip-step* :: '*v cdcl*$_W$*-restart-state-inv-st* ⇒ '*v cdcl*$_W$*-restart-state-inv-st* **where**
do-skip-step (*Propagated L C* # *Ls*, *N*, *U*, *Some D*) =

(*if* $-L \notin$ *set* $D \wedge D \neq []$
then (Ls, N, U, *Some* D)
else (*Propagated* L C #Ls, N, U, *Some* D)) |
do-skip-step $S = S$

lemma *do-skip-step*:
do-skip-step $S \neq S \Longrightarrow$ *skip* (*toS* S) (*toS* (*do-skip-step* S))
⟨*proof*⟩

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

lemma *do-skip-step-raw-trail-is-None*[*iff*]:
do-skip-step $S = (a,\ b,\ c,\ None) \longleftrightarrow S = (a,\ b,\ c,\ None)$
⟨*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*$_W$*-restart-state-inv-st* ⇒ *'v cdcl*$_W$*-restart-state-inv-st* **where**
do-resolve-step (*Propagated* L C # Ls, N, U, *Some* D) =
(*if* $-L \in$ *set* $D \wedge$ *maximum-level-code* (*remove1* ($-L$) D) (*Propagated* L C # Ls) = *count-decided Ls*
then (Ls, N, U, *Some* (*remdups* (*remove1* L C @ *remove1* ($-L$) D)))
else (*Propagated* L C # Ls, N, U, *Some* D)) |
do-resolve-step $S = S$

lemma *do-resolve-step*:
cdcl$_W$*-all-struct-inv* (*toS* S) $\Longrightarrow$ *do-resolve-step* $S \neq S$
$\Longrightarrow$ *resolve* (*toS* S) (*toS* (*do-resolve-step* S))
⟨*proof*⟩

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

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

lemma *do-resolve-step-raw-trail-is-None*[*iff*]:
do-resolve-step $S = (a,\ b,\ c,\ None) \longleftrightarrow S = (a,\ b,\ c,\ None)$
⟨*proof*⟩

Backjumping **lemma** *get-all-ann-decomposition-map-convert*:
(*get-all-ann-decomposition* (*map convert* M)) =
map (λ(a, b). (*map convert* a, *map convert* b)) (*get-all-ann-decomposition* M)
⟨*proof*⟩

lemma *do-backtrack-step*:
  assumes
    db: *do-backtrack-step S* ≠ *S* **and**
    inv: *cdcl*$_W$*-all-struct-inv* (*toS S*)
  shows *backtrack* (*toS S*) (*toS* (*do-backtrack-step S*))
⟨*proof*⟩

lemma *map-eq-list-length*:
  map f L = *L'* ⟹ *length L* = *length L'*
  ⟨*proof*⟩

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'*
  ⟨*proof*⟩

lemma *Decided-convert-iff*:
  Decided K = *convert za* ⟷ *za* = *Decided K*
  ⟨*proof*⟩

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

lemma *do-backtrack-step-no*:
  assumes
    db: *do-backtrack-step S* = *S* **and**
    inv: *cdcl*$_W$*-all-struct-inv* (*toS S*) **and**
    ns: ‹*no-step skip* (*toS S*)› ‹*no-step resolve* (*toS S*)›
  shows *no-step backtrack* (*toS S*)
⟨*proof*⟩

lemma *rough-state-of-state-of-backtrack*[*simp*]:
  assumes *inv*: *cdcl*$_W$*-all-struct-inv* (*toS S*)
  shows *rough-state-of* (*state-of* (*do-backtrack-step S*)) = *do-backtrack-step S*
⟨*proof*⟩

Decide **fun** *do-decide-step* **where**
do-decide-step (*M*, *N*, *U*, *None*) =
  (*case find-first-unused-var N* (*lits-of-l M*) *of*
    None ⇒ (*M*, *N*, *U*, *None*)
  | *Some L* ⇒ (*Decided L* # *M*, *N*, *U*, *None*)) |
do-decide-step S = *S*

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

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

lemma *rough-state-of-state-of-do-decide-step*[*simp*]:

$cdcl_W$-all-struct-inv (toS S) ⟹ rough-state-of (state-of (do-decide-step S)) = do-decide-step S
⟨proof⟩

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

Code generation

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

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

lemma [code abstype]:
Con (rough-state-of S) = S
⟨proof⟩

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

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

instantiation $cdcl_W$-restart-state-inv-from-init-state :: (type) equal
begin
definition equal-$cdcl_W$-restart-state-inv-from-init-state :: 'v $cdcl_W$-restart-state-inv-from-init-state ⇒
'v $cdcl_W$-restart-state-inv-from-init-state ⇒ bool **where**
equal-$cdcl_W$-restart-state-inv-from-init-state S S′ ⟷
(rough-state-from-init-state-of S = rough-state-from-init-state-of S′)
instance
⟨proof⟩
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-$cdcl_W$-restart (raw-init-clss (toS S))) (toS S) then S else ([], [], [], None))

lemma [code abstype]:
ConI (rough-state-from-init-state-of S) = S
⟨proof⟩

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

lemma [code abstract]:
rough-state-of (id-of-I-to S) = rough-state-from-init-state-of S
⟨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*⟩

The other rules **fun** *do-if-not-equal* **where**
do-if-not-equal [] *S* = *S* |
do-if-not-equal (*f* # *fs*) *S* =
(*let T* = *f S in*
if T ≠ *S then T else do-if-not-equal fs S*)

fun *do-cdcl-step* **where**
do-cdcl-step S =
do-if-not-equal [*do-conflict-step*, *do-propagate-step*, *do-skip-step*, *do-resolve-step*,
do-backtrack-step, *do-decide-step*] *S*

lemma *do-cdcl-step*:
assumes *inv*: *cdcl*$_W$*-all-struct-inv* (*toS S*) **and**
st: *do-cdcl-step S* ≠ *S*
shows *cdcl*$_W$*-stgy* (*toS S*) (*toS* (*do-cdcl-step S*))
⟨*proof*⟩

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

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

definition *do-cdcl*$_W$*-stgy-step* :: *'v cdcl*$_W$*-restart-state-inv* ⇒ *'v cdcl*$_W$*-restart-state-inv* **where**
do-cdcl$_W$*-stgy-step S* =
state-of (*do-cdcl-step* (*rough-state-of S*))

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

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

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

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-cp-step-neq-raw-trail-increase*:
∃ *c*. *raw-trail* (*do-cp-step S*) = *c* @ *raw-trail S* ∧ (∀ *m* ∈ *set c*. ¬ *is-decided m*)
⟨*proof*⟩

lemma *do-cp-step-raw-conflicting*:
raw-conflicting (*rough-state-of S*) ≠ *None* ⟹ *do-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 ≠ *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* ≠ *None* **and**
do-decide-step S ≠ *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 *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* ⟷ *raw-conflicting S* = *None*
⟨*proof*⟩

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

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

lemma *raw-conflicting-do-backtrack-step-imp*[*simp*]:
do-backtrack-step S ≠ *S* ⟹ *raw-conflicting* (*do-backtrack-step S*) = *None*
⟨*proof*⟩

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

lemma *do-decide-step-eq-iff-raw-trail-eq*:
do-decide-step S = *S* ⟷ *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* ⟷ *raw-trail* (*do-backtrack-step S*) = *raw-trail S*
⟨*proof*⟩

lemma *do-resolve-step-eq-iff-raw-trail-eq*:
do-resolve-step S = *S* ⟷ *raw-trail* (*do-resolve-step S*) = *raw-trail 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$_W$-stgy-is-rtranclp-cdcl$_W$-restart*:
cdcl$_W$-stgy S T ⟹ *cdcl$_W$-restart*** *S T*
⟨*proof*⟩

lemma *cdcl$_W$-stgy-init-raw-init-clss*:
cdcl$_W$-stgy S T ⟹ *cdcl$_W$-M-level-inv S* ⟹ *raw-init-clss S* = *raw-init-clss T*
⟨*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*:
(⋀*S*. (*do-cdcl$_W$-stgy-step′ S* ≠ *S* ⟹ *P* (*do-cdcl$_W$-stgy-step′ S*)) ⟹ *P S*) ⟹ *P a0*
⟨*proof*⟩

lemma *no-step-cdcl$_W$-stgy-cdcl$_W$-restart-all*:
fixes *S* :: *'a cdcl$_W$-restart-state-inv-from-init-state*
shows *no-step cdcl$_W$-stgy* (*toS* (*rough-state-from-init-state-of* (*do-all-cdcl$_W$-stgy S*)))
⟨*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, [], None)))) = S **and**
    S: *(M', N', U', E) = toS S*
  shows *(E ≠ Some {#} ∧ satisfiable (set (map mset N)))*
    ∨ (E = Some {#} ∧ unsatisfiable (set (map mset N)))
⟨*proof*⟩

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.

⟨*proof*⟩
theory *CDCL-Abstract-Clause-Representation*
imports *Entailment-Definition.Partial-Herbrand-Interpretation*
begin

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

4.1.5 Abstract Clause Representation

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

We assume the following:

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

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

The two following locales are the *exact same* locale, but we need two different locales. Otherwise, instantiating *raw-clss* would lead to duplicate constants.

locale *abstract-with-index* =
  fixes
    get-lit :: *'a ⇒ 'it ⇒ 'conc option* **and**
    convert-to-mset :: *'a ⇒ 'conc multiset*
  assumes
    in-clss-mset-cls[dest]:
      get-lit Cs a = Some e ⟹ e ∈# convert-to-mset Cs **and**
    in-mset-cls-exists-preimage:
      b ∈# convert-to-mset Cs ⟹ ∃ b'. get-lit Cs b' = Some b

locale *abstract-with-index2* =
  fixes
    get-lit :: *'a* ⇒ *'it* ⇒ *'conc option* **and**
    convert-to-mset :: *'a* ⇒ *'conc multiset*
  assumes
    in-clss-mset-clss[*dest*]:
      get-lit Cs a = *Some e* ⟹ *e* ∈# *convert-to-mset Cs* **and**
    in-mset-clss-exists-preimage:
      b ∈# *convert-to-mset Cs* ⟹ ∃ *b'*. *get-lit Cs b'* = *Some b*

locale *raw-clss* =
  abstract-with-index get-lit mset-cls +
  abstract-with-index2 get-cls mset-clss
  for
    get-lit :: *'cls* ⇒ *'lit* ⇒ *'v literal option* **and**
    mset-cls :: *'cls* ⇒ *'v clause* **and**

    get-cls :: *'clss* ⇒ *'cls-it* ⇒ *'cls option* **and**
    mset-clss:: *'clss* ⇒ *'cls multiset*
begin

definition *cls-lit* :: *'cls* ⇒ *'lit* ⇒ *'v literal* (**infix** ↓ *49*) **where**
C ↓ *a* ≡ *the* (*get-lit C a*)

definition *clss-cls* :: *'clss* ⇒ *'cls-it* ⇒ *'cls* (**infix** ⇓ *49*) **where**
C ⇓ *a* ≡ *the* (*get-cls C a*)

definition *in-cls* :: *'lit* ⇒ *'cls* ⇒ *bool* (**infix** ∈↓ *49*) **where**
a ∈↓ *Cs* ≡ *get-lit Cs a* ≠ *None*

definition *in-clss* :: *'cls-it* ⇒ *'clss* ⇒ *bool* (**infix** ∈⇓ *49*) **where**
a ∈⇓ *Cs* ≡ *get-cls Cs a* ≠ *None*

definition *raw-clss* **where**
raw-clss S ≡ *image-mset mset-cls* (*mset-clss S*)

end

experiment
begin
  fun *safe-nth* **where**
  safe-nth (*x* # -) *0* = *Some x* |
  safe-nth (- # *xs*) (*Suc n*) = *safe-nth xs n* |
  safe-nth [] - = *None*

  lemma *safe-nth-nth*: *n* < *length l* ⟹ *safe-nth l n* = *Some* (*nth l n*)
    ⟨*proof*⟩

  lemma *safe-nth-None*: *n* ≥ *length l* ⟹ *safe-nth l n* = *None*
    ⟨*proof*⟩

  lemma *safe-nth-Some-iff*: *safe-nth l n* = *Some m* ⟷ *n* < *length l* ∧ *m* = *nth l n*
    ⟨*proof*⟩

  lemma *safe-nth-None-iff*: *safe-nth l n* = *None* ⟷ *n* ≥ *length l*

⟨*proof*⟩

interpretation *abstract-with-index*
safe-nth
mset
⟨*proof*⟩

interpretation *abstract-with-index2*
safe-nth
mset
⟨*proof*⟩

interpretation *list-cls*: *raw-clss*
safe-nth mset
safe-nth mset
⟨*proof*⟩
end

end
theory *CDCL-W-Abstract-State*
imports *CDCL-W-Full CDCL-W-Restart*

begin

4.2 Instantiation of Weidenbach's CDCL by Multisets

We first instantiate the locale of Weidenbach's locale. Then we refine it to a 2-WL program.

type-synonym *'v cdcl$_W$-restart-mset* = (*'v*, *'v clause*) *ann-lit list* ×
'v clauses ×
'v clauses ×
'v clause option

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

fun *trail* :: *'v cdcl$_W$-restart-mset* ⇒ (*'v*, *'v clause*) *ann-lit list* **where**
trail (*M*, -) = *M*

fun *init-clss* :: *'v cdcl$_W$-restart-mset* ⇒ *'v clauses* **where**
init-clss (-, *N*, -) = *N*

fun *learned-clss* :: *'v cdcl$_W$-restart-mset* ⇒ *'v clauses* **where**
learned-clss (-, -, *U*, -) = *U*

fun *conflicting* :: *'v cdcl$_W$-restart-mset* ⇒ *'v clause option* **where**
conflicting (-, -, -, *C*) = *C*

fun *cons-trail* :: (*'v*, *'v clause*) *ann-lit* ⇒ *'v cdcl$_W$-restart-mset* ⇒ *'v cdcl$_W$-restart-mset* **where**
cons-trail L (*M*, *R*) = (*L* # *M*, *R*)

fun *tl-trail* **where**
tl-trail (*M*, *R*) = (*tl M*, *R*)

fun *add-learned-cls* **where**
add-learned-cls C (*M*, *N*, *U*, *R*) = (*M*, *N*, {#*C*#} + *U*, *R*)

fun *remove-cls* **where**
remove-cls C (*M*, *N*, *U*, *R*) = (*M*, *removeAll-mset C N*, *removeAll-mset C U*, *R*)

fun *update-conflicting* **where**
update-conflicting D (*M*, *N*, *U*, -) = (*M*, *N*, *U*, *D*)

fun *init-state* **where**
init-state N = ([], *N*, {#}, *None*)

declare *trail.simps*[*simp del*] *cons-trail.simps*[*simp del*] *tl-trail.simps*[*simp del*]
add-learned-cls.simps[*simp del*] *remove-cls.simps*[*simp del*]
update-conflicting.simps[*simp del*] *init-clss.simps*[*simp del*] *learned-clss.simps*[*simp del*]
conflicting.simps[*simp del*] *init-state.simps*[*simp del*]

lemmas *cdcl*$_W$*-restart-mset-state* = *trail.simps cons-trail.simps tl-trail.simps add-learned-cls.simps*
remove-cls.simps update-conflicting.simps init-clss.simps learned-clss.simps
conflicting.simps init-state.simps

definition *state* **where**
‹*state S* = (*trail S*, *init-clss S*, *learned-clss S*, *conflicting S*, ())›

interpretation *cdcl*$_W$*-restart-mset*: *state*$_W$*-ops* **where**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**

cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
⟨*proof*⟩

definition *state-eq* :: *'v cdcl*$_W$*-restart-mset* ⇒ *'v cdcl*$_W$*-restart-mset* ⇒ *bool* (**infix** ∼*m* 50) **where**
‹*S* ∼*m T* ⟷ *state S* = *state T*›

interpretation *cdcl*$_W$*-restart-mset*: *state*$_W$ **where**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
state-eq = *state-eq* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
⟨*proof*⟩

abbreviation *backtrack-lvl* :: *'v cdcl*$_W$*-restart-mset* ⇒ *nat* **where**

backtrack-lvl ≡ *cdcl$_W$-restart-mset.backtrack-lvl*

interpretation *cdcl$_W$-restart-mset*: *conflict-driven-clause-learning$_W$* **where**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**

state-eq = *state-eq* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
⟨*proof*⟩

lemma *cdcl$_W$-restart-mset-state-eq-eq*: *state-eq* = (=)
⟨*proof*⟩

lemma *clauses-def*: ‹*cdcl$_W$-restart-mset.clauses* (*M*, *N*, *U*, *C*) = *N* + *U*›
⟨*proof*⟩

lemma *cdcl$_W$-restart-mset-reduce-trail-to*:
cdcl$_W$-restart-mset.reduce-trail-to F S =
((*if length* (*trail S*) ≥ *length F*
then drop (*length* (*trail S*) − *length F*) (*trail S*)
else []), *init-clss S*, *learned-clss S*, *conflicting S*)
(**is** *?S* = -)
⟨*proof*⟩

lemma *full-cdcl$_W$-init-state*:
‹*full cdcl$_W$-restart-mset.cdcl$_W$-stgy* (*init-state* {#}) *S* ⟷ *S* = *init-state* {#}›
⟨*proof*⟩

locale *twl-restart-ops* =
fixes
f :: ‹*nat* ⇒ *nat*›
begin

interpretation *cdcl$_W$-restart-mset*: *cdcl$_W$-restart-restart-ops* **where**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**

state-eq = *state-eq* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**

init-state = *init-state* **and**
f = *f*
⟨*proof*⟩

end

locale *twl-restart* =
twl-restart-ops f **for** *f* :: ‹*nat* ⇒ *nat*› +
assumes
f: ‹*unbounded f*›
begin

interpretation $cdcl_W$*-restart-mset*: $cdcl_W$*-restart-restart* **where**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**

state-eq = *state-eq* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state* **and**
f = *f*
⟨*proof*⟩

end

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

lemma $distinct\text{-}cdcl_W\text{-}state\text{-}alt\text{-}def$:
‹$distinct\text{-}cdcl_W\text{-}state\ S$ =
((∀ *T*. *conflicting S* = *Some T* ⟶ *distinct-mset T*) ∧
distinct-mset-mset (*clauses S*) ∧
(∀ *L mark*. *Propagated L mark* ∈ *set* (*trail S*) ⟶ *distinct-mset mark*))›
⟨*proof*⟩
end

lemma $cdcl_W\text{-}stgy\text{-}cdcl_W\text{-}init\text{-}state\text{-}empty\text{-}no\text{-}step$:
‹$cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}stgy$ (*init-state* {#}) *S* ⟷ *False*›
⟨*proof*⟩

lemma $cdcl_W\text{-}stgy\text{-}cdcl_W\text{-}init\text{-}state$:
‹$cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}stgy^{**}$ (*init-state* {#}) *S* ⟷ *S* = *init-state* {#}›
⟨*proof*⟩

end