

Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

July 17, 2023

Contents

| | | |
|----------|---|-----------|
| 0.1 | Weidenbach's DPLL | 3 |
| 0.1.1 | Rules | 3 |
| 0.1.2 | Invariants | 3 |
| 0.1.3 | Termination | 6 |
| 0.1.4 | Final States | 7 |
| 1 | Weidenbach's CDCL | 13 |
| 1.1 | Weidenbach's CDCL with Multisets | 13 |
| 1.1.1 | The State | 13 |
| 1.1.2 | CDCL Rules | 22 |
| 1.1.3 | Structural Invariants | 28 |
| 1.1.4 | CDCL Strong Completeness | 40 |
| 1.1.5 | Higher level strategy | 40 |
| 1.1.6 | Structural Invariant | 48 |
| 1.1.7 | Strategy-Specific Invariant | 49 |
| 1.1.8 | Additional Invariant: No Smaller Propagation | 50 |
| 1.1.9 | More Invariants: Conflict is False if no decision | 51 |
| 1.1.10 | Some higher level use on the invariants | 53 |
| 1.1.11 | Termination | 54 |
| 1.2 | Merging backjump rules | 61 |
| 1.2.1 | Inclusion of the States | 62 |
| 1.2.2 | More lemmas about Conflict, Propagate and Backjumping | 63 |
| 1.2.3 | CDCL with Merging | 64 |
| 1.2.4 | CDCL with Merge and Strategy | 66 |
| 2 | NOT's CDCL and DPLL | 67 |
| 2.1 | Measure | 67 |
| 2.2 | NOT's CDCL | 69 |
| 2.2.1 | Auxiliary Lemmas and Measure | 69 |
| 2.2.2 | Initial Definitions | 69 |
| 2.2.3 | DPLL with Backjumping | 74 |
| 2.2.4 | CDCL | 81 |
| 2.2.5 | CDCL with Restarts | 92 |
| 2.2.6 | Merging backjump and learning | 96 |
| 2.2.7 | Instantiations | 102 |
| 2.3 | Link between Weidenbach's and NOT's CDCL | 108 |
| 2.3.1 | Inclusion of the states | 108 |
| 2.3.2 | Inclusion of Weidenbach's CDCL without Strategy | 110 |
| 2.3.3 | Additional Lemmas between NOT and W states | 111 |
| 2.3.4 | Inclusion of Weidenbach's CDCL in NOT's CDCL | 111 |

| | | |
|----------|---|------------|
| 2.3.5 | Inclusion of Weidenbach's CDCL with Strategy | 113 |
| 3 | Extensions on Weidenbach's CDCL | 117 |
| 3.1 | Restarts | 117 |
| 3.2 | Incremental SAT solving | 122 |
| 4 | List-based Implementation of DPLL and CDCL | 131 |
| 4.1 | Simple List-Based Implementation of the DPLL and CDCL | 131 |
| 4.1.1 | Common Rules | 131 |
| 4.1.2 | CDCL specific functions | 133 |
| 4.1.3 | Simple Implementation of DPLL | 134 |
| 4.1.4 | List-based CDCL Implementation | 139 |
| 4.1.5 | Abstract Clause Representation | 150 |
| 4.2 | Instantiation of Weidenbach's CDCL by Multisets | 152 |
| 4.2.1 | Restart with Replay | 156 |
| 5 | Pragmatic CDCL | 159 |
| 5.1 | Conversion | 160 |
| 5.2 | The old rules | 162 |
| 5.3 | The new rules | 163 |
| 5.4 | Invariants | 167 |
| 5.5 | Relation to CDCL | 167 |
| 5.6 | Higher-level rules | 176 |
| 5.6.1 | Simplify unit clauses | 176 |
| 5.6.2 | Subsumption resolution | 178 |
| 5.7 | Variable elimination | 179 |
| 5.8 | Backtrack for unit clause | 179 |
| 5.9 | Subsume and promote | 180 |
| 5.10 | Termination | 180 |
| 5.11 | Restarts | 182 |

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*

$\implies \text{dpll}_W S (\text{Propagated } L () \# \text{trail } S, \text{clauses } S) \mid$
decided: undefined-lit (trail S) L \implies atm-of L \in atms-of-mm (clauses S)
 $\implies \text{dpll}_W S (\text{Decided } L \# \text{trail } S, \text{clauses } S) \mid$
backtrack: backtrack-split (trail S) = (M', L # M) \implies is-decided L \implies D \in # clauses S
 $\implies \text{trail } S \models_{\text{as}} \text{CNot } D \implies \text{dpll}_W S (\text{Propagated } (- (\text{lit-of } L)) () \# M, \text{clauses } S)$

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)) \subseteq atms-of-mm (clauses S)*
shows *atm-of ' (lits-of-l (trail S')) \subseteq atms-of-mm (clauses S')*
<proof>

lemma *atms-of-ms-lit-of-atms-of: atms-of-ms (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) \subseteq 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) \subseteq atms-of-mm (clauses S)*
shows *set-mset (clauses S') \cup \{\{\#lit-of L#\} \mid L. is-decided L \wedge L \in set (trail S')\}*
 $\models_{\text{ps}} \text{unmark ' } \bigcup (\text{set ' snd ' set (get-all-ann-decomposition (trail S')))$
<proof>

theorem 2.8.5 page 86 of Weidenbach's book

lemma *only-propagated-vars-unsat:*

assumes *decided: $\forall x \in \text{set } M. \neg \text{is-decided } x$*
and *DN: D \in N and D: M $\models_{\text{as}} \text{CNot } D$*
and *inv: all-decomposition-implies N (get-all-ann-decomposition M)*
and *atm-incl: atm-of ' lits-of-l M \subseteq atms-of-ms N*
shows *unsatisfiable N*
<proof>

lemma *dpll_W-same-clauses*:
assumes *dpll_W S S'*
shows *clauses S = clauses S'*
 ⟨*proof*⟩

lemma *rtranclp-dpll_W-inv*:
assumes *rtranclp dpll_W S S'*
and *inv*: *all-decomposition-implies-m (clauses S) (get-all-ann-decomposition (trail S))*
and *atm-incl*: *atm-of ' lits-of-l (trail S) ⊆ atms-of-mm (clauses S)*
and *consistent-interp (lits-of-l (trail S))*
and *no-dup (trail S)*
shows *all-decomposition-implies-m (clauses S') (get-all-ann-decomposition (trail S'))*
and *atm-of ' lits-of-l (trail S') ⊆ atms-of-mm (clauses S')*
and *clauses S = clauses S'*
and *consistent-interp (lits-of-l (trail S'))*
and *no-dup (trail S')*
 ⟨*proof*⟩

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

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

lemma *rtranclp-dpll_W-all-inv*:
assumes *rtranclp dpll_W S S'*
and *dpll_W-all-inv S*
shows *dpll_W-all-inv S'*
 ⟨*proof*⟩

lemma *dpll_W-all-inv*:
assumes *dpll_W S S'*
and *dpll_W-all-inv S*
shows *dpll_W-all-inv S'*
 ⟨*proof*⟩

lemma *rtranclp-dpll_W-inv-starting-from-0*:
assumes *rtranclp dpll_W S S'*
and *inv*: *trail S = []*
shows *dpll_W-all-inv S'*
 ⟨*proof*⟩

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

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

theorem 2.8.7 page 87 of Weidenbach's book

lemma *dpll_W-strong-completeness*:

assumes set $M \models_{sm} N$
and *consistent-interp* (set M)
and *distinct* M
and *atm-of* ' (set M) \subseteq *atms-of-mm* N
shows $dpll_W^{**} (\ [], N) (\text{map } \text{Decided } M, N)$
and *conclusive-dpll_W-state* ($\text{map } \text{Decided } M, N$)

$\langle \text{proof} \rangle$

theorem 2.8.6 page 86 of Weidenbach's book

lemma *dpll_W-sound*:

assumes
rtranclp $dpll_W (\ [], N) (M, N)$ **and**
 $\forall S. \neg dpll_W (M, N) S$
shows $M \models_{asm} N \longleftrightarrow \text{satisfiable } (\text{set-mset } N) (\text{is } ?A \longleftrightarrow ?B)$

$\langle \text{proof} \rangle$

0.1.3 Termination

definition *dpll_W-mes* M $n =$

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

lemma *length-dpll_W-mes*:

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

$\langle \text{proof} \rangle$

lemma *distinctcard-atm-of-lit-of-eq-length*:

assumes *no-dup* S
shows $\text{card } (\text{atm-of } ' \text{ lits-of-l } S) = \text{length } S$

$\langle \text{proof} \rangle$

lemma *Cons-lexn-iff*:

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

$\langle \text{proof} \rangle$

declare *append-same-lexn*[simp] *prepend-same-lexn*[simp] *Cons-lexn-iff*[simp]

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

lemma *dpll_W-card-decrease*:

assumes
dpll: $dpll_W \ S \ S'$ **and**
 [simp]: $\text{length } (\text{trail } S') \leq \text{card vars}$ **and**
 $\text{length } (\text{trail } S) \leq \text{card vars}$

shows

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

$\langle \text{proof} \rangle$

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 *wf-dpll_W*:

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 $\forall S'. \neg dpll_W S S'$

shows *conclusive-dpll_W-state S*

⟨*proof*⟩

lemma *dpll_W-conclusive-state-correct*:

assumes *dpll_W^{**} ([], N) (M, N)* **and** *conclusive-dpll_W-state (M, N)*

shows $M \models_{asm} N \iff \text{satisfiable (set-mset } N)$ **(is** *?A ↔ ?B*)

⟨*proof*⟩

lemma *dpll_W-trail-after-step1*:

assumes ⟨*dpll_W S T*⟩

shows

⟨ $\exists K' M1 M2' M2''.$

$(\text{rev (trail } T) = \text{rev (trail } S) @ M2' \wedge M2' \neq []) \vee$

$(\text{rev (trail } S) = M1 @ \text{Decided } (-K') \# M2' \wedge$

$\text{rev (trail } T) = M1 @ \text{Propagated } K' () \# M2'' \wedge$

$\text{Suc (length } M1) \leq \text{length (trail } S))$ ⟩

⟨*proof*⟩

lemma *tranclp-dpll_W-trail-after-step*:

assumes ⟨*dpll_W⁺⁺ S T*⟩

shows


```

⟨∃ K' M1 M2' M2''.
  (rev (trail T) = rev (trail S) @ M2' ∧ M2' ≠ []) ∨
  (rev (trail S) = M1 @ Decided (-K') # M2' ∧
   rev (trail T) = M1 @ Propagated K' () # M2'' ∧ Suc (length M1) ≤ length (trail S))⟩
⟨proof⟩

```

This theorem is an important (although rather obvious) property: the model induced by trails are not repeated.

```

lemma tranclp-dpllW-no-dup-trail:
  assumes ⟨dpllW++ S T⟩ and ⟨dpllW-all-inv S⟩
  shows ⟨set (trail S) ≠ set (trail T)⟩
⟨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 (λS. 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 $get\text{-}level\ M\ L > n$
shows $atm\text{-}of\ L \in atm\text{-}of\ 'lits\text{-}of\text{-}l\ M$
 $\langle proof \rangle$

In $get\text{-}level$ (resp. $get\text{-}level$), the beginning (resp. the end) can be skipped if the literal is not in the beginning (resp. the end).

lemma $get\text{-}level\text{-}skip[simp]$:
assumes $undefined\text{-}lit\ M\ L$
shows $get\text{-}level\ (M\ @\ M')\ L = get\text{-}level\ M'\ L$
 $\langle proof \rangle$

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

lemma $get\text{-}level\text{-}skip\text{-}end[simp]$:
assumes $defined\text{-}lit\ M\ L$
shows $get\text{-}level\ (M\ @\ M')\ L = get\text{-}level\ M\ L + count\text{-}decided\ M'$
 $\langle proof \rangle$

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

lemma $get\text{-}level\text{-}take\text{-}beginning[simp]$:
assumes $atm\text{-}of\ L' = atm\text{-}of\ (lit\text{-}of\ K)$
shows $get\text{-}level\ (K\ \#\ M)\ L' = count\text{-}decided\ (K\ \#\ M)$
 $\langle proof \rangle$

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

lemma $get\text{-}level\text{-}skip\text{-}beginning\text{-}not\text{-}decided[simp]$:
assumes $undefined\text{-}lit\ S\ L$
and $\forall s \in set\ S. \neg is\text{-}decided\ s$
shows $get\text{-}level\ (M\ @\ S)\ L = get\text{-}level\ M\ L$
 $\langle proof \rangle$

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

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

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

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

lemma $get\text{-}maximum\text{-}level\text{-}empty[simp]$:
 $get\text{-}maximum\text{-}level\ M\ \{\#\} = 0$

⟨proof⟩

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

$D \neq \{\#\} \implies \exists L \in \# D. \text{get-level } M L = \text{get-maximum-level } M D$

⟨proof⟩

lemma *get-maximum-level-empty-list[simp]:*

$\text{get-maximum-level } [] D = 0$

⟨proof⟩

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

$\text{get-maximum-level } M (\text{add-mset } L D) = \max (\text{get-level } M L) (\text{get-maximum-level } M D)$

⟨proof⟩

lemma *get-level-append-if:*

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

⟨proof⟩

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

lemma *get-maximum-level-single:*

$\langle \text{get-maximum-level } M \{\#x\# \} = \text{get-level } M x \rangle$

⟨proof⟩

lemma *get-maximum-level-plus:*

$\text{get-maximum-level } M (D + D') = \max (\text{get-maximum-level } M D) (\text{get-maximum-level } M D')$

⟨proof⟩

lemma *get-maximum-level-cong:*

assumes $\langle \forall L \in \# D. \text{get-level } M L = \text{get-level } M' L \rangle$

shows $\langle \text{get-maximum-level } M D = \text{get-maximum-level } M' D \rangle$

⟨proof⟩

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

assumes $n: n > 0$

and $\text{max}: \text{get-maximum-level } M D = n$

shows $\exists L \in \# D. \text{get-level } M L = n$

⟨proof⟩

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

assumes $\text{atm-of } (\text{lit-of } K) \notin \text{atms-of } D$

shows $\text{get-maximum-level } (K \# M) D = \text{get-maximum-level } M D$

⟨proof⟩

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

assumes $DH: \forall x \in \# D. \text{undefined-lit } c x$

shows $\text{get-maximum-level } (c @ H) D = \text{get-maximum-level } H D$

⟨proof⟩

lemma *get-maximum-level-D-single-propagated:*

$\text{get-maximum-level } [\text{Propagated } x21 \ x22] D = 0$

⟨proof⟩

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

$\text{get-maximum-level } M (A \cup \# B) = \text{get-maximum-level } M (A + B)$

⟨proof⟩

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

lemma *count-decided-ge-get-level*:
 $count-decided M \geq get-level M L$
 ⟨proof⟩

lemma *count-decided-ge-get-maximum-level*:
 $count-decided M \geq get-maximum-level M D$
 ⟨proof⟩

lemma *get-level-last-decided-ge*:
 $\langle defined-lit (c @ [Decided K]) L' \implies 0 < get-level (c @ [Decided K]) L' \rangle$
 ⟨proof⟩

lemma *get-maximum-level-mono*:
 $\langle D \subseteq \# D' \implies get-maximum-level M D \leq get-maximum-level M D' \rangle$
 ⟨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*:
 $\langle M \neq [] \implies is-proped (hd M) \implies get-all-mark-of-propagated (tl M) = tl (get-all-mark-of-propagated M) \rangle$
 ⟨proof⟩

Properties about the levels

lemma *atm-lit-of-set-lits-of-l*:
 $(\lambda l. atm-of (lit-of l)) \text{ ' set } xs = atm-of \text{ ' lits-of-l } xs$
 ⟨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 $\langle no-dup M \rangle$
shows $\langle i < count-decided M \longleftrightarrow (\exists c K c'. M = c @ Decided K \# c' \wedge get-level M K = Suc i) \rangle$
 (**is** $?A \longleftrightarrow ?B$)
 ⟨proof⟩

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

lemma
fixes K
defines $\langle M \equiv replicate 3 (Decided K) \rangle$

defines $\langle i \equiv 1 \rangle$

assumes $\langle i < \text{count-decided } M \longleftrightarrow (\exists c K c'. M = c @ \text{Decided } K \# c' \wedge \text{get-level } M K = \text{Suc } i) \rangle$

shows *False*

$\langle \text{proof} \rangle$

lemma *Suc-count-decided-gt-get-level:*

$\langle \text{get-level } M L < \text{Suc } (\text{count-decided } M) \rangle$

$\langle \text{proof} \rangle$

lemma *get-level-neg-Suc-count-decided[simp]:*

$\langle \text{get-level } M L \neq \text{Suc } (\text{count-decided } M) \rangle$

$\langle \text{proof} \rangle$

lemma *length-get-all-ann-decomposition:* $\langle \text{length } (\text{get-all-ann-decomposition } M) = 1 + \text{count-decided } M \rangle$

$\langle \text{proof} \rangle$

lemma *get-maximum-level-remove-non-max-lvl:*

$\langle \text{get-level } M a < \text{get-maximum-level } M D \implies$

$\text{get-maximum-level } M (\text{remove1-mset } a D) = \text{get-maximum-level } M D \rangle$

$\langle \text{proof} \rangle$

lemma *exists-lit-max-level-in-negate-ann-lits:*

$\langle \text{negate-ann-lits } M \neq \{\#\} \implies \exists L \in \# \text{negate-ann-lits } M. \text{get-level } M L = \text{count-decided } M \rangle$

$\langle \text{proof} \rangle$

lemma *get-maximum-level-eq-count-decided-iff:*

$\langle ya \neq \{\#\} \implies \text{get-maximum-level } xa ya = \text{count-decided } xa \longleftrightarrow (\exists L \in \# ya. \text{get-level } xa L = \text{count-decided } xa) \rangle$

$\langle \text{proof} \rangle$

definition *card-max-lvl where*

$\langle \text{card-max-lvl } M C \equiv \text{size } (\text{filter-mset } (\lambda L. \text{get-level } M L = \text{count-decided } M) C) \rangle$

lemma *card-max-lvl-add-mset:* $\langle \text{card-max-lvl } M (\text{add-mset } L C) =$

$(\text{if } \text{get-level } M L = \text{count-decided } M \text{ then } 1 \text{ else } 0) +$

$\text{card-max-lvl } M C \rangle$

$\langle \text{proof} \rangle$

lemma *card-max-lvl-empty[simp]:* $\langle \text{card-max-lvl } M \{\#\} = 0 \rangle$

$\langle \text{proof} \rangle$

lemma *card-max-lvl-all-poss:*

$\langle \text{card-max-lvl } M C = \text{card-max-lvl } M (\text{poss } (\text{atm-of } \{\#\} C)) \rangle$

$\langle \text{proof} \rangle$

lemma *card-max-lvl-distinct-cong:*

assumes

$\langle \bigwedge L. \text{get-level } M (\text{Pos } L) = \text{count-decided } M \implies (L \in \text{atms-of } C) \implies (L \in \text{atms-of } C') \rangle$ **and**

$\langle \bigwedge L. \text{get-level } M (\text{Pos } L) = \text{count-decided } M \implies (L \in \text{atms-of } C') \implies (L \in \text{atms-of } C) \rangle$ **and**

$\langle \text{distinct-mset } C \rangle \langle \neg \text{tautology } C \rangle$ **and**

$\langle \text{distinct-mset } C' \rangle \langle \neg \text{tautology } C' \rangle$

shows $\langle \text{card-max-lvl } M C = \text{card-max-lvl } M C' \rangle$

$\langle \text{proof} \rangle$

lemma *get-maximum-level-card-max-lvl-ge1:*

$\langle \text{count-decided } xa > 0 \implies \text{get-maximum-level } xa ya = \text{count-decided } xa \longleftrightarrow \text{card-max-lvl } xa ya > 0 \rangle$

$\langle \text{proof} \rangle$

lemma *card-max-lvl-remove-hd-trail-iff*:

$\langle xa \neq [] \implies - \text{lit-of } (hd\ xa) \in\# ya \implies 0 < \text{card-max-lvl } xa\ (\text{remove1-mset } (- \text{lit-of } (hd\ xa))\ ya) \longleftrightarrow \text{Suc } 0 < \text{card-max-lvl } xa\ ya \rangle$
 $\langle \text{proof} \rangle$

lemma *card-max-lvl-Cons*:

assumes $\langle \text{no-dup } (L\ \# a) \rangle \langle \text{distinct-mset } y \rangle \langle \neg \text{tautology } y \rangle \langle \neg \text{is-decided } L \rangle$

shows $\langle \text{card-max-lvl } (L\ \# a)\ y =$

$(\text{if } (\text{lit-of } L \in\# y \vee -\text{lit-of } L \in\# y) \wedge \text{count-decided } a \neq 0 \text{ then } \text{card-max-lvl } a\ y + 1$
 $\text{else } \text{card-max-lvl } a\ y) \rangle$

$\langle \text{proof} \rangle$

lemma *card-max-lvl-tl*:

assumes $\langle a \neq [] \rangle \langle \text{distinct-mset } y \rangle \langle \neg \text{tautology } y \rangle \langle \neg \text{is-decided } (hd\ a) \rangle \langle \text{no-dup } a \rangle$

$\langle \text{count-decided } a \neq 0 \rangle$

shows $\langle \text{card-max-lvl } (tl\ a)\ y =$

$(\text{if } (\text{lit-of } (hd\ a) \in\# y \vee -\text{lit-of } (hd\ a) \in\# y)$
 $\text{then } \text{card-max-lvl } a\ y - 1 \text{ else } \text{card-max-lvl } a\ y) \rangle$

$\langle \text{proof} \rangle$

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 prove 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 refinements: one from NOT's to Weidenbach's CDCL and another to W's CDCL with strategy.

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

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

1.1 Weidenbach's CDCL with Multisets

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

1.1.1 The State

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

```

locale stateW-ops =
fixes
  state :: 'st ⇒ ('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-clss :: 'v clause ⇒ 'st ⇒ 'st and
  remove-clss :: '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-clss :: ⟨'a literal ⇒ 'a clause ⇒ 'a clause ⇒ 'a clause⟩ where
  resolve-clss 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 stateW-no-state =
  stateW-ops
  state
  — functions about the state:
    — getter:
  trail init-cls learned-cls 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-cls :: 'st ⇒ 'v clauses and
  learned-cls :: 'st ⇒ 'v clauses and
  conflicting :: 'st ⇒ 'v clause option and

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

  init-state :: 'v clauses ⇒ 'st +
assumes
  state-eq-ref[simp, intro]: ⟨S ~ S⟩ and
  state-eq-sym: ⟨S ~ T ⟷ T ~ S⟩ and
  state-eq-trans: ⟨S ~ T ⟹ T ~ U' ⟹ S ~ U'⟩ and
  state-eq-state: ⟨S ~ T ⟹ state S = state T⟩ and

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

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

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

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

  update-conflicting:

```

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

init-state:

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

cons-trail-state-eq:

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

tl-trail-state-eq:

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

add-learned-cls-state-eq:

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

remove-cls-state-eq:

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

update-conflicting-state-eq:

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

tl-trail-add-learned-cls-commute:

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

tl-trail-update-conflicting:

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

update-conflicting-update-conflicting:

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

update-conflicting-itself:

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

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-lits \times 'v clauses \times 'v clauses \times 'v clause option \times 'b **and**

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

init-clss :: 'st \Rightarrow 'v clauses **and**

learned-clss :: 'st \Rightarrow 'v clauses **and**

conflicting :: 'st \Rightarrow 'v clause option **and**

cons-trail :: ('v, 'v clause) ann-lit \Rightarrow 'st \Rightarrow 'st **and**

tl-trail :: 'st \Rightarrow 'st **and**

add-learned-cls :: 'v clause \Rightarrow 'st \Rightarrow 'st **and**

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

init-state :: 'v clauses \Rightarrow 'st +

assumes

state-prop[simp]:

\langle state $S = (\text{trail } S, \text{init-clss } S, \text{learned-clss } S, \text{conflicting } S, \text{additional-info } S)$ \rangle

begin

lemma

trail-cons-trail[simp]:

trail (*cons-trail* L st) = $L \# \text{trail } st$ **and**

trail-tl-trail[simp]: *trail* (*tl-trail* st) = tl (*trail* st) **and**

trail-add-learned-cls[simp]:

trail (*add-learned-cls* C st) = *trail* st **and**

trail-remove-cls[simp]:

trail (*remove-cls* C st) = *trail* st **and**

trail-update-conflicting[simp]: *trail* (*update-conflicting* E st) = *trail* st **and**

init-clss-cons-trail[simp]:

init-clss (*cons-trail* M st) = *init-clss* st

and

init-clss-tl-trail[simp]:

init-clss (*tl-trail* st) = *init-clss* st **and**

init-clss-add-learned-cls[simp]:

init-clss (*add-learned-cls* C st) = *init-clss* st **and**

init-clss-remove-cls[simp]:

init-clss (*remove-cls* C st) = *removeAll-mset* C (*init-clss* st) **and**

init-clss-update-conflicting[simp]:

init-clss (*update-conflicting* E st) = *init-clss* st **and**

learned-clss-cons-trail[simp]:

learned-clss (*cons-trail* M st) = *learned-clss* st **and**

learned-clss-tl-trail[simp]:

learned-clss (*tl-trail* st) = *learned-clss* st **and**

learned-clss-add-learned-cls[simp]:

learned-clss (*add-learned-cls* C st) = $\{\#C\#$ $\} + \text{learned-clss } st$ **and**

learned-clss-remove-cls[simp]:

learned-clss (*remove-cls* C st) = *removeAll-mset* C (*learned-clss* st) **and**

learned-clss-update-conflicting[simp]:

learned-clss (*update-conflicting* E st) = *learned-clss* st **and**

conflicting-cons-trail[simp]:

conflicting (*cons-trail* M st) = *conflicting* st **and**

conflicting-tl-trail[simp]:

conflicting (*tl-trail* st) = *conflicting* st **and**

conflicting-add-learned-cls[simp]:

conflicting (*add-learned-cls* C st) = *conflicting* st

and

conflicting-remove-cls[simp]:

conflicting (*remove-cls* C st) = *conflicting* st **and**

conflicting-update-conflicting[simp]:

conflicting (*update-conflicting* E st) = E **and**

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

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

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

lemma
shows

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

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

clauses-add-learned-clss-unfolded:

$\text{clauses } (\text{add-learned-clss } U S) = \{\#U\# \} + \text{learned-clss } S + \text{init-clss } S$

and

clauses-update-conflicting[simp]: $\text{clauses } (\text{update-conflicting } D S) = \text{clauses } S$ **and**

clauses-remove-clss[simp]:

$\text{clauses } (\text{remove-clss } C S) = \text{removeAll-mset } C (\text{clauses } S)$ **and**

clauses-add-learned-clss[simp]:

$\text{clauses } (\text{add-learned-clss } C S) = \{\#C\# \} + \text{clauses } S$ **and**

clauses-init-state[simp]: $\text{clauses } (\text{init-state } N) = N$

⟨proof⟩

lemma *state-eq-trans'*: $\langle S \sim S' \implies T \sim S' \implies T \sim S \rangle$

⟨proof⟩

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

⟨*backtrack-lvl* $S \equiv \text{count-decided } (\text{trail } S)$ ⟩

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

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

lemma
shows

state-eq-trail[state-simp]: $S \sim T \implies \text{trail } S = \text{trail } T$ **and**

state-eq-init-clss[state-simp]: $S \sim T \implies \text{init-clss } S = \text{init-clss } T$ **and**

state-eq-learned-clss[state-simp]: $S \sim T \implies \text{learned-clss } S = \text{learned-clss } T$ **and**

state-eq-conflicting[state-simp]: $S \sim T \implies \text{conflicting } S = \text{conflicting } T$ **and**

state-eq-clauses[state-simp]: $S \sim T \implies \text{clauses } S = \text{clauses } T$ **and**

state-eq-undefined-lit[state-simp]: $S \sim T \implies \text{undefined-lit } (\text{trail } S) L = \text{undefined-lit } (\text{trail } T) L$ **and**

state-eq-backtrack-lvl[state-simp]: $S \sim T \implies \text{backtrack-lvl } S = \text{backtrack-lvl } T$

⟨proof⟩

lemma *state-eq-conflicting-None*:

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

⟨proof⟩

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

declare *state-simp*[simp]

function *reduce-trail-to* :: 'a list \Rightarrow 'st \Rightarrow 'st **where**

reduce-trail-to $F S =$

(if $\text{length } (\text{trail } S) = \text{length } F \vee \text{trail } S = []$ then S else *reduce-trail-to* $F (\text{tl-trail } S)$)

⟨proof⟩

termination

⟨proof⟩

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

lemma *reduce-trail-to-induct*:

assumes

⟨ $\bigwedge F S. \text{length} (\text{trail } S) = \text{length } F \implies P F S$ ⟩ **and**

⟨ $\bigwedge F S. \text{trail } S = [] \implies P F S$ ⟩ **and**

⟨ $\bigwedge F S. \text{length} (\text{trail } S) \neq \text{length } F \implies \text{trail } S \neq [] \implies P F (\text{tl-trail } S) \implies P F S$ ⟩

shows

⟨ $P F S$ ⟩

⟨proof⟩

lemma

shows

reduce-trail-to-Nil[simp]: $\text{trail } S = [] \implies \text{reduce-trail-to } F S = S$ **and**

reduce-trail-to-eq-length[simp]: $\text{length} (\text{trail } S) = \text{length } F \implies \text{reduce-trail-to } F S = S$

⟨proof⟩

lemma *reduce-trail-to-length-ne*:

$\text{length} (\text{trail } S) \neq \text{length } F \implies \text{trail } S \neq [] \implies$

$\text{reduce-trail-to } F S = \text{reduce-trail-to } F (\text{tl-trail } S)$

⟨proof⟩

lemma *trail-reduce-trail-to-length-le*:

assumes $\text{length } F > \text{length} (\text{trail } S)$

shows $\text{trail} (\text{reduce-trail-to } F S) = []$

⟨proof⟩

lemma *trail-reduce-trail-to-Nil*[simp]:

$\text{trail} (\text{reduce-trail-to } [] S) = []$

⟨proof⟩

lemma *clauses-reduce-trail-to-Nil*:

$\text{clauses} (\text{reduce-trail-to } [] S) = \text{clauses } S$

⟨proof⟩

lemma *reduce-trail-to-skip-beginning*:

assumes $\text{trail } S = F' @ F$

shows $\text{trail} (\text{reduce-trail-to } F S) = F$

⟨proof⟩

lemma *clauses-reduce-trail-to*[simp]:

$\text{clauses} (\text{reduce-trail-to } F S) = \text{clauses } S$

⟨proof⟩

lemma *conflicting-update-trail*[simp]:

$\text{conflicting} (\text{reduce-trail-to } F S) = \text{conflicting } S$

⟨proof⟩

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

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

⟨proof⟩

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

$\text{learned-clss} (\text{reduce-trail-to } F S) = \text{learned-clss } S$

⟨proof⟩

lemma *conflicting-reduce-trail-to*[simp]:

$\text{conflicting } (\text{reduce-trail-to } F S) = \text{None} \longleftrightarrow \text{conflicting } S = \text{None}$

⟨proof⟩

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

$\text{trail } S = \text{trail } T \implies \text{trail } (\text{reduce-trail-to } F S) = \text{trail } (\text{reduce-trail-to } F T)$

⟨proof⟩

lemma *reduce-trail-to-trail-tl-trail-decomp*[simp]:

$\text{trail } S = F' @ \text{Decided } K \# F \implies \text{trail } (\text{reduce-trail-to } F S) = F$

⟨proof⟩

lemma *reduce-trail-to-add-learned-cls*[simp]:

$\text{trail } (\text{reduce-trail-to } F (\text{add-learned-cls } C S)) = \text{trail } (\text{reduce-trail-to } F S)$

⟨proof⟩

lemma *reduce-trail-to-remove-learned-cls*[simp]:

$\text{trail } (\text{reduce-trail-to } F (\text{remove-cls } C S)) = \text{trail } (\text{reduce-trail-to } F S)$

⟨proof⟩

lemma *reduce-trail-to-update-conflicting*[simp]:

$\text{trail } (\text{reduce-trail-to } F (\text{update-conflicting } C S)) = \text{trail } (\text{reduce-trail-to } F S)$

⟨proof⟩

lemma *reduce-trail-to-length*:

$\text{length } M = \text{length } M' \implies \text{reduce-trail-to } M S = \text{reduce-trail-to } M' S$

⟨proof⟩

lemma *trail-reduce-trail-to-drop*:

$\text{trail } (\text{reduce-trail-to } F S) =$
 (if $\text{length } (\text{trail } S) \geq \text{length } F$
 then $\text{drop } (\text{length } (\text{trail } S) - \text{length } F) (\text{trail } S)$
 else $[]$)

⟨proof⟩

lemma *in-get-all-ann-decomposition-trail-update-trail*[simp]:

assumes $H: (L \# M1, M2) \in \text{set } (\text{get-all-ann-decomposition } (\text{trail } S))$

shows $\text{trail } (\text{reduce-trail-to } M1 S) = M1$

⟨proof⟩

lemma *reduce-trail-to-state-eq*:

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

⟨proof⟩

lemma *conflicting-cons-trail-conflicting*[iff]:

$\text{conflicting } (\text{cons-trail } L S) = \text{None} \longleftrightarrow \text{conflicting } S = \text{None}$

⟨proof⟩

lemma *conflicting-add-learned-cls-conflicting*[iff]:

$\text{conflicting } (\text{add-learned-cls } C S) = \text{None} \longleftrightarrow \text{conflicting } S = \text{None}$

⟨proof⟩

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

assumes $\langle \text{length } M < \text{length } (\text{trail } M') \rangle$

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

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

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

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

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

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

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

lemma *compow-tl-trail-add-learned-cls-swap:*

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

lemma *reduce-trail-to-add-learned-cls-state-eq:*

$\langle \text{length } M \leq \text{length } (\text{trail } S) \implies$
 $\text{reduce-trail-to } M (\text{add-learned-cls } D S) \sim \text{add-learned-cls } D (\text{reduce-trail-to } M S) \rangle$
 $\langle \text{proof} \rangle$

lemma *compow-tl-trail-update-conflicting-swap:*

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

lemma *reduce-trail-to-update-conflicting-state-eq:*

$\langle \text{length } M \leq \text{length } (\text{trail } S) \implies$
 $\text{reduce-trail-to } M (\text{update-conflicting } D S) \sim \text{update-conflicting } D (\text{reduce-trail-to } M S) \rangle$
 $\langle \text{proof} \rangle$

lemma

additional-info-cons-trail[simp]:
 $\langle \text{additional-info } (\text{cons-trail } L S) = \text{additional-info } S \rangle$ **and**
additional-info-tl-trail[simp]:
 $\text{additional-info } (\text{tl-trail } S) = \text{additional-info } S$ **and**
additional-info-add-learned-cls-unfolded:
 $\text{additional-info } (\text{add-learned-cls } U S) = \text{additional-info } S$ **and**
additional-info-update-conflicting[simp]:
 $\text{additional-info } (\text{update-conflicting } D S) = \text{additional-info } S$ **and**
additional-info-remove-cls[simp]:
 $\text{additional-info } (\text{remove-cls } C S) = \text{additional-info } S$ **and**
additional-info-add-learned-cls[simp]:
 $\text{additional-info } (\text{add-learned-cls } C S) = \text{additional-info } S$
 $\langle \text{proof} \rangle$

lemma *additional-info-reduce-trail-to[simp]:*

$\langle \text{additional-info } (\text{reduce-trail-to } F S) = \text{additional-info } S \rangle$
 $\langle \text{proof} \rangle$

lemma *reduce-trail-to:*

$\text{state } (\text{reduce-trail-to } F S) =$

((if length (trail S) ≥ length F
 then drop (length (trail S) – length F) (trail S)
 else []), init-cls S, learned-cls 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-cls learned-cls 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-cls :: 'st ⇒ 'v clauses **and**
learned-cls :: '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 ⇒

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

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

inductive *backtrack* :: 'st \Rightarrow 'st \Rightarrow bool **for** S :: 'st **where**

backtrack-rule:

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

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

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

inductive *simple-backtrack* :: 'st \Rightarrow 'st \Rightarrow bool **for** S :: 'st **where**

simple-backtrack-rule:

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

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

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

inductive *backtrackg* :: 'st \Rightarrow 'st \Rightarrow bool **for** S :: 'st **where**

backtrackg-rule:

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

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-cls 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 $\neg M \models N + U$. 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 $C \notin \# \text{init-cls } S$, to maintain consistency even without the strategy.

inductive *forget* :: '*st* ⇒ '*st* ⇒ *bool* **where**

forget-rule:

conflicting S = None ⇒

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

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

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

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

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

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

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

lemma *rtranclp-propagate-is-rtranclp-cdcl_W-restart*:
 $\text{propagate}^{**} S S' \implies \text{cdcl}_W\text{-restart}^{**} S S'$
 $\langle \text{proof} \rangle$

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

lemma *cdcl_W-cdcl_W-restart*:
 $\text{cdcl}_W S T \implies \text{cdcl}_W\text{-restart } S T$
 $\langle \text{proof} \rangle$

lemma *rtranclp-cdcl_W-cdcl_W-restart*:
 $\langle \text{cdcl}_W^{**} S T \implies \text{cdcl}_W\text{-restart}^{**} S T \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl_W-restart-all-rules-induct*[*consumes 1, case-names propagate conflict forget restart decide skip resolve backtrack*]:
fixes $S :: 'st$
assumes
cdcl_W-restart: $\text{cdcl}_W\text{-restart } S S'$ **and**
propagate: $\bigwedge T. \text{propagate } S T \implies P S T$ **and**
conflict: $\bigwedge T. \text{conflict } S T \implies P S T$ **and**
forget: $\bigwedge T. \text{forget } S T \implies P S T$ **and**

restart: $\bigwedge T. \text{ restart } S \ T \implies P \ S \ T$ **and**
decide: $\bigwedge T. \text{ decide } S \ T \implies P \ S \ T$ **and**
skip: $\bigwedge T. \text{ skip } S \ T \implies P \ S \ T$ **and**
resolve: $\bigwedge T. \text{ resolve } S \ T \implies P \ S \ T$ **and**
backtrack: $\bigwedge T. \text{ backtrack } S \ T \implies P \ S \ T$
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: $\bigwedge C \ L \ T. \text{ conflicting } S = \text{None} \implies$

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

$L \in \# \ C \implies$

$\text{trail } S \models_{\text{as}} \text{CNot } (\text{remove1-mset } L \ C) \implies$

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

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

$P \ S \ T$ **and**

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

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

$\text{trail } S \models_{\text{as}} \text{CNot } D \implies$

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

$P \ S \ T$ **and**

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

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

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

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

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

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

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

$P \ S \ T$ **and**

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

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

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

$P \ S \ T$ **and**

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

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

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

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

$P \ S \ T$ **and**

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

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

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

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

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

$P \ S \ T$ **and**

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

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

$L \in \# \ E \implies$

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

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

$\neg L \in \# \ D \implies$

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

$T \sim \text{update-conflicting}$

$(\text{Some } (\text{resolve-cls } L D E)) (\text{tl-trail } S) \implies$
P S T and
 $\text{backtrackH}: \bigwedge L D K i M1 M2 T D'.$
 $\text{conflicting } S = \text{Some } (\text{add-mset } L D) \implies$
 $(\text{Decided } K \# M1, M2) \in \text{set } (\text{get-all-ann-decomposition } (\text{trail } S)) \implies$
 $\text{get-level } (\text{trail } S) L = \text{backtrack-lvl } S \implies$
 $\text{get-level } (\text{trail } S) L = \text{get-maximum-level } (\text{trail } S) (\text{add-mset } L D') \implies$
 $\text{get-maximum-level } (\text{trail } S) D' \equiv i \implies$
 $\text{get-level } (\text{trail } S) K = i+1 \implies$
 $D' \subseteq\# D \implies$
 $\text{clauses } S \models_{\text{pm}} \text{add-mset } L D' \implies$
 $T \sim \text{cons-trail } (\text{Propagated } L (\text{add-mset } L D'))$
 $(\text{reduce-trail-to } M1$
 $(\text{add-learned-cls } (\text{add-mset } L D')$
 $(\text{update-conflicting } \text{None } S))) \implies$
P S T
shows P S S'
 $\langle \text{proof} \rangle$

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

fixes $S :: 'st$

assumes $\text{cdcl}_W\text{-restart}: \text{cdcl}_W\text{-o } S T$ **and**

$\text{decideH}: \bigwedge L T. \text{conflicting } S = \text{None} \implies \text{undefined-lit } (\text{trail } S) L$
 $\implies \text{atm-of } L \in \text{atms-of-mm } (\text{init-cls } S)$
 $\implies T \sim \text{cons-trail } (\text{Decided } L) S$
 $\implies \text{P S T and}$

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

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

P S T and

$\text{resolveH}: \bigwedge L E M D T.$

$\text{trail } S = \text{Propagated } L E \# M \implies$
 $L \in\# E \implies$
 $\text{hd-trail } S = \text{Propagated } L E \implies$
 $\text{conflicting } S = \text{Some } D \implies$
 $-L \in\# D \implies$
 $\text{get-maximum-level } (\text{trail } S) ((\text{remove1-mset } (-L) D)) = \text{backtrack-lvl } S \implies$
 $T \sim \text{update-conflicting}$
 $(\text{Some } (\text{resolve-cls } L D E)) (\text{tl-trail } S) \implies$

P S T and

$\text{backtrackH}: \bigwedge L D K i M1 M2 T D'.$

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

P S T

shows $P S T$
 $\langle \text{proof} \rangle$

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**
 $\bigwedge T. \text{decide } S T \implies P S T$ **and**
 $\bigwedge T. \text{backtrack } S T \implies P S T$ **and**
 $\bigwedge T. \text{skip } S T \implies P S T$ **and**
 $\bigwedge T. \text{resolve } S T \implies P S T$
shows $P S T$
 $\langle \text{proof} \rangle$

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**
 $\text{decide } S T \implies P$ **and**
 $\text{backtrack } S T \implies P$ **and**
 $\text{skip } S T \implies P$ **and**
 $\text{resolve } S T \implies P$
shows P
 $\langle \text{proof} \rangle$

lemma *backtrack-backtrackg*:
 $\langle \text{backtrack } S T \implies \text{backtrackg } S T \rangle$
 $\langle \text{proof} \rangle$

lemma *simple-backtrack-backtrackg*:
 $\langle \text{simple-backtrack } S T \implies \text{backtrackg } S T \rangle$
 $\langle \text{proof} \rangle$

1.1.3 Structural Invariants

Properties of the trail

We here establish that:

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

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

lemma *backtrack-lit-skipped*:
assumes
 $L: \text{get-level } (\text{trail } S) L = \text{backtrack-lvl } S$ **and**
 $M1: (\text{Decided } K \# M1, M2) \in \text{set } (\text{get-all-ann-decomposition } (\text{trail } S))$ **and**
 $\text{no-dup: no-dup } (\text{trail } S)$ **and**
 $\text{lev-K: get-level } (\text{trail } S) K = i + 1$

shows *undefined-lit M1 L*
<proof>

lemma *cdcl_W-restart-distinctiv-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 \longleftrightarrow
 consistent-interp (lits-of-l (trail S))
 \wedge *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: $i < \text{backtrack-lvl } S$

shows $\exists K M1 M2. (\text{Decided } K \# M1, M2) \in \text{set } (\text{get-all-ann-decomposition } (\text{trail } S)) \wedge$
get-level (*trail S*) $K = \text{Suc } i$

<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

conf: *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** \neg *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 \implies *skip S T'* \implies $T \sim T'$

<proof>

lemma *resolve-unique*:

resolve S T \implies *resolve S T'* \implies $T \sim T'$

<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 $init-clss\ S = init-clss\ S'$
 $\langle proof \rangle$

lemma *tranclp-cdcl_W-o-no-more-init-clss*:

assumes
 $cdcl_W-o^{++}\ S\ S'$ **and**
 $inv: cdcl_W-M-level-inv\ S$
shows $init-clss\ S = init-clss\ S'$
 $\langle proof \rangle$

lemma *rtranclp-cdcl_W-o-no-more-init-clss*:

assumes
 $cdcl_W-o^{**}\ S\ S'$ **and**
 $inv: cdcl_W-M-level-inv\ S$
shows $init-clss\ S = init-clss\ S'$
 $\langle proof \rangle$

lemma *cdcl_W-restart-init-clss*:

assumes
 $cdcl_W-restart\ S\ T$
shows $init-clss\ S = init-clss\ T$
 $\langle proof \rangle$

lemma *rtranclp-cdcl_W-restart-init-clss*:

$cdcl_W-restart^{**}\ S\ T \implies init-clss\ S = init-clss\ T$
 $\langle proof \rangle$

lemma *tranclp-cdcl_W-restart-init-clss*:

$cdcl_W-restart^{++}\ S\ T \implies init-clss\ S = init-clss\ T$
 $\langle proof \rangle$

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* :: $\langle 'st \Rightarrow bool \rangle$ **where**
 $\langle reasons-in-clauses\ (S :: 'st) \longleftrightarrow$
 $(set\ (get-all-mark-of-propagated\ (trail\ S)) \subseteq set-mset\ (clauses\ S)) \rangle$

definition (in *state_W-ops*) *cdcl_W-learned-clause* :: $\langle 'st \Rightarrow bool \rangle$ **where**
 $cdcl_W-learned-clause\ (S :: 'st) \longleftrightarrow$
 $((\forall T. conflicting\ S = Some\ T \longrightarrow clauses\ S \models_{pm}\ T)$
 $\wedge reasons-in-clauses\ S)$

lemma *cdcl_W-learned-clause-alt-def*:

$\langle cdcl_W-learned-clause\ (S :: 'st) \longleftrightarrow$
 $((\forall T. conflicting\ S = Some\ T \longrightarrow clauses\ S \models_{pm}\ T)$
 $\wedge set\ (get-all-mark-of-propagated\ (trail\ S)) \subseteq set-mset\ (clauses\ S)) \rangle$

$\langle proof \rangle$

lemma *reasons-in-clauses-init-state*[simp]: $\langle reasons-in-clauses (init-state N) \rangle$
 $\langle proof \rangle$

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

lemma *cdcl_W-learned-clause-S0-cdcl_W-restart*[simp]:
cdcl_W-learned-clause (init-state N)
 $\langle proof \rangle$

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'
 $\langle proof \rangle$

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'
 $\langle proof \rangle$

lemma *cdcl_W-restart-reasons-in-clauses*:
assumes
 cdcl_W-restart S S' **and**
 learned: reasons-in-clauses S
shows *reasons-in-clauses* S'
 $\langle proof \rangle$

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'
 $\langle proof \rangle$

No alien atom in the state

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

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

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

shows $\text{conflicting } S = \text{Some } T \implies \text{atms-of } T \subseteq \text{atms-of-mm } (\text{init-clss } S)$
and $(\forall L \text{ mark. Propagated } L \text{ mark} \in \text{set } (\text{trail } S) \longrightarrow \text{atms-of mark} \subseteq \text{atms-of-mm } (\text{init-clss } S))$
and $\text{atms-of-mm } (\text{learned-clss } S) \subseteq \text{atms-of-mm } (\text{init-clss } S)$
and $\text{atm-of } ' (\text{lits-of-l } (\text{trail } S)) \subseteq \text{atms-of-mm } (\text{init-clss } S)$
 $\langle \text{proof} \rangle$

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

lemma *propagate-no-strange-atm-inv*:

assumes
propagate S T **and**
alien: *no-strange-atm* S
shows *no-strange-atm* T
 $\langle \text{proof} \rangle$

lemma *atms-of-ms-learned-clss-restart-state-in-atms-of-ms-learned-clssI*:

$\text{atms-of-mm } (\text{learned-clss } S) \subseteq \text{atms-of-mm } (\text{init-clss } S) \implies$
 $x \in \text{atms-of-mm } (\text{learned-clss } T) \implies$
 $\text{learned-clss } T \subseteq \# \text{learned-clss } S \implies$
 $x \in \text{atms-of-mm } (\text{init-clss } S)$
 $\langle \text{proof} \rangle$

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

lemma *cdcl_W-restart-no-strange-atm-explicit*:

assumes
cdcl_W-restart S S' **and**
lev: *cdcl_W-M-level-inv* S **and**
conf: $\forall T. \text{conflicting } S = \text{Some } T \longrightarrow \text{atms-of } T \subseteq \text{atms-of-mm } (\text{init-clss } S)$ **and**
decided: $\forall L \text{ mark. Propagated } L \text{ mark} \in \text{set } (\text{trail } S)$
 $\longrightarrow \text{atms-of mark} \subseteq \text{atms-of-mm } (\text{init-clss } S)$ **and**
learned: $\text{atms-of-mm } (\text{learned-clss } S) \subseteq \text{atms-of-mm } (\text{init-clss } S)$ **and**
trail: $\text{atm-of } ' (\text{lits-of-l } (\text{trail } S)) \subseteq \text{atms-of-mm } (\text{init-clss } S)$
shows
 $(\forall T. \text{conflicting } S' = \text{Some } T \longrightarrow \text{atms-of } T \subseteq \text{atms-of-mm } (\text{init-clss } S')) \wedge$
 $(\forall L \text{ mark. Propagated } L \text{ mark} \in \text{set } (\text{trail } S') \longrightarrow \text{atms-of mark} \subseteq \text{atms-of-mm } (\text{init-clss } S')) \wedge$
 $\text{atms-of-mm } (\text{learned-clss } S') \subseteq \text{atms-of-mm } (\text{init-clss } S') \wedge$
 $\text{atm-of } ' (\text{lits-of-l } (\text{trail } S')) \subseteq \text{atms-of-mm } (\text{init-clss } S')$
(is ?C S' \wedge ?M S' \wedge ?U S' \wedge ?V S')
 $\langle \text{proof} \rangle$

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'
 $\langle \text{proof} \rangle$

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'
 $\langle \text{proof} \rangle$

No Duplicates all Around

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

definition *distinct-cdcl_W-state* ($S :: 'st$)
 $\longleftrightarrow ((\forall T. \text{conflicting } S = \text{Some } T \longrightarrow \text{distinct-mset } T)$
 $\wedge \text{distinct-mset-mset } (\text{learned-cls } S)$
 $\wedge \text{distinct-mset-mset } (\text{init-cls } S)$
 $\wedge (\forall L \text{ mark. } (\text{Propagated } L \text{ mark} \in \text{set } (\text{trail } S) \longrightarrow \text{distinct-mset } \text{mark})))$

lemma *distinct-cdcl_W-state-decomp*:

assumes *distinct-cdcl_W-state* S

shows

$\forall T. \text{conflicting } S = \text{Some } T \longrightarrow \text{distinct-mset } T$ **and**
 $\text{distinct-mset-mset } (\text{learned-cls } S)$ **and**
 $\text{distinct-mset-mset } (\text{init-cls } S)$ **and**
 $\forall L \text{ mark. } (\text{Propagated } L \text{ mark} \in \text{set } (\text{trail } S) \longrightarrow \text{distinct-mset } \text{mark})$
 $\langle \text{proof} \rangle$

lemma *distinct-cdcl_W-state-decomp-2*:

assumes *distinct-cdcl_W-state* S **and** *conflicting* $S = \text{Some } T$

shows *distinct-mset* T

$\langle \text{proof} \rangle$

lemma *distinct-cdcl_W-state-S0-cdcl_W-restart[simp]*:

distinct-mset-mset $N \implies \text{distinct-cdcl}_W\text{-state } (\text{init-state } N)$

$\langle \text{proof} \rangle$

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'

$\langle \text{proof} \rangle$

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'

$\langle \text{proof} \rangle$

Conflicts and Annotations

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

abbreviation *every-mark-is-a-conflict* $:: 'st \Rightarrow \text{bool}$ **where**

every-mark-is-a-conflict $S \equiv$

$\forall L \text{ mark } a \ b. a \ @ \ \text{Propagated } L \ \text{mark} \ \# \ b = (\text{trail } S)$
 $\longrightarrow (b \models_{\text{as}} \text{CNot } (\text{mark} - \{\#L\}) \wedge L \in \# \ \text{mark})$

definition *cdcl_W-conflicting* $:: 'st \Rightarrow \text{bool}$ **where**

$cdcl_W$ -conflicting $S \longleftrightarrow$
 $(\forall T. \text{ conflicting } S = \text{Some } T \longrightarrow \text{trail } S \models_{as} \text{CNot } T) \wedge \text{every-mark-is-a-conflict } S$

lemma *backtrack-atms-of-D-in-M1*:

fixes $S T :: 'st$ **and** $D D' :: \langle 'v \text{ clause} \rangle$ **and** $K L :: \langle 'v \text{ literal} \rangle$ **and** $i :: nat$ **and**
 $M1 M2 :: \langle ('v, 'v \text{ clause}) \text{ ann-lits} \rangle$

assumes

inv: *no-dup* (*trail* S) **and**

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

decomp: (*Decided* $K \# M1, M2$)

$\in \text{set } (\text{get-all-ann-decomposition } (\text{trail } S))$ **and**

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

S-confl: *conflicting* $S = \text{Some } D$ **and**

lev-K: *get-level* (*trail* S) $K = \text{Suc } i$ **and**

$T: T \sim \text{cons-trail } K''$

(*reduce-trail-to* $M1$

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

(*update-conflicting* *None* S))) **and**

confl: $\forall T. \text{ conflicting } S = \text{Some } T \longrightarrow \text{trail } S \models_{as} \text{CNot } T$ **and**

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

shows $\text{atms-of } D' \subseteq \text{atm-of } \langle \text{lits-of-l } (\text{tl } (\text{trail } T)) \rangle$

<proof>

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

assumes

a1: *no-dup* ($M @ M'$) **and**

a2: $\text{atms-of } D \subseteq \text{atm-of } \langle \text{lits-of-l } M' \rangle$ **and**

a3: $x \in \text{atms-of } D$

shows $x \notin \text{atm-of } \langle \text{lits-of-l } M \rangle$

<proof>

lemma *backtrack-M1-CNot-D'*:

fixes $S T :: 'st$ **and** $D D' :: \langle 'v \text{ clause} \rangle$ **and** $K L :: \langle 'v \text{ literal} \rangle$ **and** $i :: nat$ **and**

$M1 M2 :: \langle ('v, 'v \text{ clause}) \text{ ann-lits} \rangle$

assumes

inv: *no-dup* (*trail* S) **and**

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

decomp: (*Decided* $K \# M1, M2$)

$\in \text{set } (\text{get-all-ann-decomposition } (\text{trail } S))$ **and**

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

S-confl: *conflicting* $S = \text{Some } D$ **and**

lev-K: *get-level* (*trail* S) $K = \text{Suc } i$ **and**

$T: T \sim \text{cons-trail } K''$

(*reduce-trail-to* $M1$

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

(*update-conflicting* *None* S))) **and**

confl: $\forall T. \text{ conflicting } S = \text{Some } T \longrightarrow \text{trail } S \models_{as} \text{CNot } T$ **and**

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

shows $M1 \models_{as} \text{CNot } D'$ **and**

$\langle \text{atm-of } (\text{lit-of } K'') = \text{atm-of } L \implies \text{no-dup } (\text{trail } T) \rangle$

<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**
conf: $\forall T$. conflicting $S = \text{Some } T \longrightarrow \text{trail } S \models_{as} CNot\ T$ **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**
conf: $\forall T$. conflicting $S = \text{Some } T \longrightarrow \text{trail } S \models_{as} CNot\ T$ **and**
alien: no-strange-atm S **and**
mark-conf: 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**
conf-_{inv}: $\forall T$. conflicting $S = \text{Some } T \longrightarrow \text{trail } S \models_{as} CNot\ T$ **and**
decided-conf: $\forall L\ \text{mark } a\ b$. $a @ \text{Propagated } L\ \text{mark } \# b = (\text{trail } S) \longrightarrow (b \models_{as} CNot (\text{mark} - \{\#L\}) \wedge L \in \# \text{mark})$ **and**
dist: distinct- $cdcl_W$ -state S
shows $\forall T$. conflicting $S' = \text{Some } T \longrightarrow \text{trail } S' \models_{as} CNot\ T$
 ⟨proof⟩

lemma $cdcl_W$ -conflicting-decomp:

assumes $cdcl_W$ -conflicting S
shows
 $\forall T$. conflicting $S = \text{Some } T \longrightarrow \text{trail } S \models_{as} CNot\ T$ **and**
 $\forall L\ \text{mark } a\ b$. $a @ \text{Propagated } L\ \text{mark } \# b = (\text{trail } S) \longrightarrow (b \models_{as} CNot (\text{mark} - \{\#L\}) \wedge L \in \# \text{mark})$
 ⟨proof⟩

lemma $cdcl_W$ -conflicting-decomp2:

assumes $cdcl_W$ -conflicting S **and** conflicting $S = \text{Some } T$
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 $S \longleftrightarrow \text{init-clss } S \models_{psm} \text{learned-clss } S$ ⟩

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: \langle *cdcl_W-learned-clauses-entailed-by-init S* \rangle

shows \langle *cdcl_W-learned-clauses-entailed-by-init S'* \rangle
\langle proof \rangle

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: \langle *cdcl_W-learned-clauses-entailed-by-init S* \rangle

shows \langle *cdcl_W-learned-clauses-entailed-by-init S'* \rangle
\langle proof \rangle

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'

\langle proof \rangle

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'

\langle proof \rangle

lemma *all-invariant-S0-cdcl_W-restart*:

assumes *distinct-mset-mset N*

shows

all-decomposition-implies-m (*init-cls* (*init-state N*))
(*get-all-ann-decomposition* (*trail* (*init-state N*))) **and**
cdcl_W-learned-clause (*init-state N*) **and**
 $\forall T. \text{conflicting} (\text{init-state } N) = \text{Some } T \longrightarrow (\text{trail} (\text{init-state } N)) \models_{as} \text{CNot } T$ **and**
no-strange-atm (*init-state N*) **and**
consistent-interp (*lits-of-l* (*trail* (*init-state N*))) **and**
 $\forall L \text{ mark } a \ b. \ a \ @ \ \text{Propagated } L \ \text{mark } \# \ b = \text{trail} (\text{init-state } N) \longrightarrow$
 $(b \models_{as} \text{CNot} (\text{mark} - \{\#L\}) \wedge L \in \# \text{ mark})$ **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: $\forall x \in \text{set } M. \neg \text{is-decided } x$ **and**
DN: $D \in \# \text{ clauses } S$ **and**
D: $M \models_{as} \text{CNot } D$ **and**
inv: *all-decomposition-implies-m* ($N + U$) (*get-all-ann-decomposition M*) **and**
state: $\text{state } S = (M, N, U, k, C)$ **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 $\forall m \in \text{set } M. \neg \text{is-decided } m$
shows *set-mset N* $\models_{ps} \text{unmark-l } M$

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

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

<proof>

definition *final-cdcl_W-restart-state* ($S :: 'st$)

$\longleftrightarrow (\text{trail } S \models_{asm} \text{init-clss } S$

$\vee ((\forall L \in \text{set } (\text{trail } S). \neg \text{is-decided } L) \wedge$

$(\exists C \in \# \text{ init-clss } S. \text{trail } S \models_{as} C \text{Not } C)))$

definition *termination-cdcl_W-restart-state* ($S :: 'st$)

$\longleftrightarrow (\text{trail } S \models_{asm} \text{init-clss } S$

$\vee ((\forall L \in \text{atms-of-mm } (\text{init-clss } S). L \in \text{atm-of ' lits-of-l } (\text{trail } S))$

$\wedge (\exists C \in \# \text{ init-clss } S. \text{trail } S \models_{as} C \text{Not } C)))$

1.1.4 CDCL Strong Completeness

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

assumes

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

distinct M **and**

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

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

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

<proof>

theorem 2.9.11 page 98 of Weidenbach's book

lemma *cdcl_W-restart-strong-completeness*:

assumes

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

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

dist: *distinct* M **and**

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

obtains S **where**

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

$\text{tranclp conflict } S \ S' \Longrightarrow \text{conflict } S \ S'$
 $\langle \text{proof} \rangle$

lemma *no-chained-conflict*:

assumes $\text{conflict } S \ S'$ **and** $\text{conflict } S' \ S''$
shows False
 $\langle \text{proof} \rangle$

lemma *tranclp-conflict-iff*:

$\text{full1 conflict } S \ S' \longleftrightarrow \text{conflict } S \ S'$
 $\langle \text{proof} \rangle$

lemma *no-conflict-after-conflict*:

$\text{conflict } S \ T \Longrightarrow \neg \text{conflict } T \ U$
 $\langle \text{proof} \rangle$

lemma *no-propagate-after-conflict*:

$\text{conflict } S \ T \Longrightarrow \neg \text{propagate } T \ U$
 $\langle \text{proof} \rangle$

inductive $\text{cdcl}_W\text{-stgy} :: 'st \Rightarrow 'st \Rightarrow \text{bool}$ **for** $S :: 'st$ **where**

conflict': $\text{conflict } S \ S' \Longrightarrow \text{cdcl}_W\text{-stgy } S \ S' \mid$

propagate': $\text{propagate } S \ S' \Longrightarrow \text{cdcl}_W\text{-stgy } S \ S' \mid$

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

lemma $\text{cdcl}_W\text{-stgy-cdcl}_W$: $\text{cdcl}_W\text{-stgy } S \ T \Longrightarrow \text{cdcl}_W \ S \ T$

$\langle \text{proof} \rangle$

lemma $\text{cdcl}_W\text{-stgy-induct}$ [*consumes 1, case-names conflict propagate decide skip resolve backtrack*]:

assumes

$\langle \text{cdcl}_W\text{-stgy } S \ T \rangle$ **and**

$\langle \bigwedge T. \text{conflict } S \ T \Longrightarrow P \ T \rangle$ **and**

$\langle \bigwedge T. \text{propagate } S \ T \Longrightarrow P \ T \rangle$ **and**

$\langle \bigwedge T. \text{no-step conflict } S \Longrightarrow \text{no-step propagate } S \Longrightarrow \text{decide } S \ T \Longrightarrow P \ T \rangle$ **and**

$\langle \bigwedge T. \text{no-step conflict } S \Longrightarrow \text{no-step propagate } S \Longrightarrow \text{skip } S \ T \Longrightarrow P \ T \rangle$ **and**

$\langle \bigwedge T. \text{no-step conflict } S \Longrightarrow \text{no-step propagate } S \Longrightarrow \text{resolve } S \ T \Longrightarrow P \ T \rangle$ **and**

$\langle \bigwedge T. \text{no-step conflict } S \Longrightarrow \text{no-step propagate } S \Longrightarrow \text{backtrack } S \ T \Longrightarrow P \ T \rangle$

shows

$\langle P \ T \rangle$

$\langle \text{proof} \rangle$

lemma $\text{cdcl}_W\text{-stgy-cases}$ [*consumes 1, case-names conflict propagate decide skip resolve backtrack*]:

assumes

$\langle \text{cdcl}_W\text{-stgy } S \ T \rangle$ **and**

$\langle \text{conflict } S \ T \Longrightarrow P \rangle$ **and**

$\langle \text{propagate } S \ T \Longrightarrow P \rangle$ **and**

$\langle \text{no-step conflict } S \Longrightarrow \text{no-step propagate } S \Longrightarrow \text{decide } S \ T \Longrightarrow P \rangle$ **and**

$\langle \text{no-step conflict } S \Longrightarrow \text{no-step propagate } S \Longrightarrow \text{skip } S \ T \Longrightarrow P \rangle$ **and**

$\langle \text{no-step conflict } S \Longrightarrow \text{no-step propagate } S \Longrightarrow \text{resolve } S \ T \Longrightarrow P \rangle$ **and**

$\langle \text{no-step conflict } S \Longrightarrow \text{no-step propagate } S \Longrightarrow \text{backtrack } S \ T \Longrightarrow P \rangle$

shows

$\langle P \rangle$

$\langle \text{proof} \rangle$

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* $S = \text{init-clss } S'$

<proof>

lemma *rtranclp-cdcl_W-stgy-no-more-init-clss*:

assumes *cdcl_W-stgy*** $S S'$

shows *init-clss* $S = \text{init-clss } S'$

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

conflict-is-false-with-level $S \equiv \forall D. \text{conflicting } S = \text{Some } D \longrightarrow D \neq \{\#\}$

$\longrightarrow (\exists L \in \# D. \text{get-level } (\text{trail } S) L = \text{backtrack-lvl } S)$

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

Literal of highest level in decided literals

definition *mark-is-false-with-level* :: $'st \Rightarrow \text{bool}$ **where**

mark-is-false-with-level $S' \equiv$

$\forall D M1 M2 L. M1 @ \text{Propagated } L D \# M2 = \text{trail } S' \longrightarrow D - \{\#L\} \neq \{\#\}$

$\longrightarrow (\exists L. L \in \# D \wedge \text{get-level } (\text{trail } S') L = \text{count-decided } M1)$

lemma *backtrack_W-rule*:

assumes

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

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

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

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

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

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

T: $\langle T \sim \text{cons-trail } (\text{Propagated } L (\text{add-mset } L D))$

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

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

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

lev-inv: *cdcl_W-M-level-inv* S **and**

conf: $\langle \text{cdcl}_W\text{-conflicting } S \rangle$ **and**

learned: $\langle \text{cdcl}_W\text{-learned-clause } S \rangle$

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

<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 $\exists S'. \text{cdcl}_W\text{-o } S \ S' \ \exists S'. \text{backtrack } S \ S'$

<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 \neq []>* **and**
L': *<hd-trail S = Decided L'>* **and**
nempty: *<C \neq {#}>* **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 \neq {#}>* **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**

conft: *cdcl_W-conflicting S* **and**

conft-k: *conflict-is-false-with-level S*

shows (*conflicting S = Some {#} \wedge unsatisfiable (set-mset (clauses S))*)
 \vee (*conflicting S = None \wedge trail S \models as set-mset (clauses S)*)

<proof>

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**

conft: *cdcl_W-conflicting S* **and**

conft-k: *conflict-is-false-with-level S* **and**

learned-entailed: *<cdcl_W-learned-clauses-entailed-by-init S>*

shows (*conflicting S = Some {#} \wedge unsatisfiable (set-mset (init-cls S))*)
 \vee (*conflicting S = None \wedge trail S \models as set-mset (init-cls S)*)

<proof>

lemma *cdcl_W-stgy-tranclp-cdcl_W-restart*:

cdcl_W-stgy S S' \implies cdcl_W-restart⁺⁺ S S'

<proof>

lemma *tranclp-cdcl_W-stgy-tranclp-cdcl_W-restart*:

cdcl_W-stgy⁺⁺ S S' \implies cdcl_W-restart⁺⁺ S S'

<proof>

lemma *rtranclp-cdcl_W-stgy-rtranclp-cdcl_W-restart*:

*cdcl_W-stgy^{**} S S' \implies 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**

conft-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 $S = (M', N', U, \text{None})$ **and**
state-butlast $T = (\text{Propagated } L (C + \{\#L\#\}) \# M', N', U, \text{None})$ **and**
 $C + \{\#L\#\} \in \# \text{ local.clauses } S$ **and**
 $M' \models_{as} C \text{Not } C$ **and**
undefined-lit (*trail* S) L
 <proof>

lemma *cdcl_W-propagate-conflict-completeness:*

assumes

$MN: \text{set } M \models_s \text{set-mset } N$ **and**
 $\text{cons: consistent-interp } (\text{set } M)$ **and**
 $\text{tot: total-over-m } (\text{set } M) (\text{set-mset } N)$ **and**
 $\text{lits-of-l } (\text{trail } S) \subseteq \text{set } M$ **and**
 $\text{init-cls } S = N$ **and**
 $\text{propagate}^{**} S S'$ **and**
 $\text{learned-cls } S = \{\#\}$

shows $\text{length } (\text{trail } S) \leq \text{length } (\text{trail } S') \wedge \text{lits-of-l } (\text{trail } S') \subseteq \text{set } M$

<proof>

lemma

assumes $\text{propagate}^{**} S X$

shows

$\text{rtranclp-propagate-init-cls: init-cls } X = \text{init-cls } S$ **and**
 $\text{rtranclp-propagate-learned-cls: learned-cls } X = \text{learned-cls } S$

<proof>

lemma *cdcl_W-stgy-strong-completeness-n:*

assumes

$MN: \text{set } M \models_s \text{set-mset } N$ **and**
 $\text{cons: consistent-interp } (\text{set } M)$ **and**
 $\text{tot: total-over-m } (\text{set } M) (\text{set-mset } N)$ **and**
 $\text{atm-incl: atm-of ' } (\text{set } M) \subseteq \text{atms-of-mm } N$ **and**
 $\text{distM: distinct } M$ **and**
 $\text{length: } n \leq \text{length } M$

shows

$\exists M' S. \text{length } M' \geq n \wedge$
 $\text{lits-of-l } M' \subseteq \text{set } M \wedge$
 $\text{no-dup } M' \wedge$
 $\text{state-butlast } S = (M', N, \{\#\}, \text{None}) \wedge$
 $\text{cdcl}_W\text{-stgy}^{**} (\text{init-state } N) S$

<proof>

lemma *cdcl_W-stgy-strong-completeness':*

assumes

$MN: \text{set } M \models_s \text{set-mset } N$ **and**
 $\text{cons: consistent-interp } (\text{set } M)$ **and**
 $\text{tot: total-over-m } (\text{set } M) (\text{set-mset } N)$ **and**
 $\text{atm-incl: atm-of ' } (\text{set } M) \subseteq \text{atms-of-mm } N$ **and**
 $\text{distM: distinct } M$

shows

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

<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-conflict (S :: 'st) \equiv*

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

lemma *no-smaller-conflict-init-sate[simp]:*

no-smaller-conflict (init-state N) <proof>

lemma *cdcl_W-o-no-smaller-conflict-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-conflict S

shows *no-smaller-conflict S'*

<proof>

lemma *conflict-no-smaller-conflict-inv:*

assumes *conflict S S'*

and *no-smaller-conflict S*

shows *no-smaller-conflict S'*

<proof>

lemma *propagate-no-smaller-conflict-inv:*

assumes *propagate: propagate S S'*

and *n-l: no-smaller-conflict S*

shows *no-smaller-conflict S'*

<proof>

lemma *cdcl_W-stgy-no-smaller-conflict:*

assumes *cdcl_W-stgy S S'*

and *n-l: no-smaller-conflict S*

and *conflict-is-false-with-level S*

and *cdcl_W-M-level-inv S*

shows *no-smaller-conflict S'*

<proof>

lemma *conflict-conflict-is-false-with-level*:

assumes

conflict: *conflict* S T **and**

smaller: *no-smaller-conf* 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-conf* 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-conf-inv*:

assumes

*cdcl_W-stgy*** S S' **and**

n-l: *no-smaller-conf* 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-conf* $S' \wedge$ *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* $\{\#\}$ \wedge *unsatisfiable* (*set-mset* (*init-cls* S')))

\vee (*conflicting* $S' =$ *None* \wedge *trail* $S' \models_{asm}$ *init-cls* S')

<proof>

lemma *cdcl_W-o-fst-empty-conflicting-false*:

assumes

cdcl_W-o S S' **and**

trail $S = []$ **and**

conflicting $S \neq$ *None*

shows *False*

<proof>

lemma *cdcl_W-stgy-fst-empty-conflicting-false*:

assumes

$cdcl_W\text{-stgy } S \ S'$ **and**
 $trail \ S = []$ **and**
 $conflicting \ S \neq None$
shows $False$
 $\langle proof \rangle$

lemma $cdcl_W\text{-o-conflicting-is-false}$:
 $cdcl_W\text{-o } S \ S' \implies conflicting \ S = Some \ \{\#\} \implies False$
 $\langle proof \rangle$

lemma $cdcl_W\text{-stgy-conflicting-is-false}$:
 $cdcl_W\text{-stgy } S \ S' \implies conflicting \ S = Some \ \{\#\} \implies False$
 $\langle proof \rangle$

lemma $rtranclp\text{-}cdcl_W\text{-stgy-conflicting-is-false}$:
 $cdcl_W\text{-stgy}^* \ S \ S' \implies conflicting \ S = Some \ \{\#\} \implies S' = S$
 $\langle proof \rangle$

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

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

lemma $conflict\text{-or-propagate-intros}$:
 $conflict \ S \ T \implies conflict\text{-or-propagate} \ S \ T$
 $propagate \ S \ T \implies conflict\text{-or-propagate} \ S \ T$
 $\langle proof \rangle$

theorem 2.9.9 page 97 of Weidenbach's book

lemma $full\text{-}cdcl_W\text{-stgy-final-state-conclusive-from-init-state}$:
fixes $S' :: 'st$
assumes $full$: $full \ cdcl_W\text{-stgy} \ (init\text{-state} \ N) \ S'$
and $no\text{-d}$: $distinct\text{-mset-mset} \ N$
shows $(conflicting \ S' = Some \ \{\#\} \wedge unsatisfiable \ (set\text{-mset} \ N))$
 $\vee (conflicting \ S' = None \wedge trail \ S' \models_{asm} \ N \wedge satisfiable \ (set\text{-mset} \ N))$
 $\langle proof \rangle$

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

lemma $cdcl_W\text{-all-struct-inv-cong}$:
 $\langle S \sim T \implies cdcl_W\text{-all-struct-inv} \ S \longleftrightarrow cdcl_W\text{-all-struct-inv} \ T \rangle$
 $\langle proof \rangle$

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'
 \langle *proof* \rangle

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'
 \langle *proof* \rangle

lemma *cdcl_W-stgy-cdcl_W-all-struct-inv*:
cdcl_W-stgy $S T \implies$ *cdcl_W-all-struct-inv* $S \implies$ *cdcl_W-all-struct-inv* T
 \langle *proof* \rangle

lemma *rtranclp-cdcl_W-stgy-cdcl_W-all-struct-inv*:
*cdcl_W-stgy*** $S T \implies$ *cdcl_W-all-struct-inv* $S \implies$ *cdcl_W-all-struct-inv* T
 \langle *proof* \rangle

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

1.1.7 Strategy-Specific Invariant

definition *cdcl_W-stgy-invariant* **where**
cdcl_W-stgy-invariant $S \longleftrightarrow$
conflict-is-false-with-level S
 \wedge *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
 \langle *proof* \rangle

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
 \langle *proof* \rangle

lemma *full-cdcl_W-stgy-inv-normal-form*:
assumes
full: *full cdcl_W-stgy* $S T$ **and**
inv-s: *cdcl_W-stgy-invariant* S **and**
inv: *cdcl_W-all-struct-inv* S **and**
learned-entailed: \langle *cdcl_W-learned-clauses-entailed-by-init* S \rangle

shows $\text{conflicting } T = \text{Some } \{\#\} \wedge \text{unsatisfiable } (\text{set-mset } (\text{init-cls } S))$
 $\vee \text{conflicting } T = \text{None} \wedge \text{trail } T \models_{\text{asm}} \text{init-cls } S \wedge \text{satisfiable } (\text{set-mset } (\text{init-cls } S))$
 ⟨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 $\text{conflicting } T = \text{Some } \{\#\} \wedge \text{unsatisfiable } (\text{set-mset } (\text{clauses } T))$

$\vee \text{conflicting } T = \text{None} \wedge \text{satisfiable } (\text{set-mset } (\text{clauses } T))$

⟨proof⟩

1.1.8 Additional Invariant: No Smaller Propagation

definition *no-smaller-propa* :: $\langle 'st \Rightarrow \text{bool} \rangle$ **where**

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

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

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

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

M' # [] \wedge *Ms = tl M' @ Decided K # M* \wedge *hd M' = Propagated L E*

⟨proof⟩

lemma *in-get-all-mark-of-propagated-in-trail*:

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

⟨proof⟩

lemma *no-smaller-propa-tl*:

assumes

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

$\langle \text{trail } S \neq [] \rangle$ **and**

$\langle \neg \text{is-decided}(\text{hd-trail } S) \rangle$ **and**

$\langle \text{trail } U = \text{tl } (\text{trail } S) \rangle$ **and**

$\langle \text{clauses } U = \text{clauses } S \rangle$

shows

$\langle \text{no-smaller-propa } U \rangle$

⟨proof⟩

lemmas *rulesE* =

skipE resolveE backtrackE propagateE conflictE decideE restartE forgetE backtrackgE

lemma *decide-no-smaller-step*:

assumes *dec*: $\langle \text{decide } S T \rangle$ **and** *smaller-propa*: $\langle \text{no-smaller-propa } S \rangle$ **and**

n-s: $\langle \text{no-step propagate } S \rangle$

shows $\langle \text{no-smaller-propa } T \rangle$

⟨proof⟩

lemma *no-smaller-propa-reduce-trail-to*:

$\langle \text{no-smaller-propa } S \implies \text{no-smaller-propa } (\text{reduce-trail-to } M1 S) \rangle$

⟨proof⟩

lemma *backtrackg-no-smaller-propa*:

assumes o : $\langle \text{backtrackg } S \ T \rangle$ **and** smaller-propa : $\langle \text{no-smaller-propa } S \rangle$ **and**
 $n\text{-d}$: $\langle \text{no-dup } (\text{trail } S) \rangle$ **and**
 $n\text{-s}$: $\langle \text{no-step propagate } S \rangle$ **and**
 tr-CNot : $\langle \text{trail } S \models_{\text{as}} \text{CNot } (\text{the } (\text{conflicting } S)) \rangle$
shows $\langle \text{no-smaller-propa } T \rangle$
 $\langle \text{proof} \rangle$

lemmas $\text{backtrack-no-smaller-propa} = \text{backtrackg-no-smaller-propa}[\text{OF backtrack-backtrackg}]$

lemma $\text{cdcl}_W\text{-stgy-no-smaller-propa}$:
assumes
 cdcl : $\langle \text{cdcl}_W\text{-stgy } S \ T \rangle$ **and**
 smaller-propa : $\langle \text{no-smaller-propa } S \rangle$ **and**
 inv : $\langle \text{cdcl}_W\text{-all-struct-inv } S \rangle$
shows $\langle \text{no-smaller-propa } T \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{rtranclp-cdcl}_W\text{-stgy-no-smaller-propa}$:
assumes
 cdcl : $\langle \text{cdcl}_W\text{-stgy}^{**} \ S \ T \rangle$ **and**
 smaller-propa : $\langle \text{no-smaller-propa } S \rangle$ **and**
 inv : $\langle \text{cdcl}_W\text{-all-struct-inv } S \rangle$
shows $\langle \text{no-smaller-propa } T \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{hd-trail-level-ge-1-length-gt-1}$:
fixes $S :: 'st$
defines $M[\text{symmetric, simp}]$: $\langle M \equiv \text{trail } S \rangle$
defines $L[\text{symmetric, simp}]$: $\langle L \equiv \text{hd } M \rangle$
assumes
 smaller : $\langle \text{no-smaller-propa } S \rangle$ **and**
 struct : $\langle \text{cdcl}_W\text{-all-struct-inv } S \rangle$ **and**
 dec : $\langle \text{count-decided } M \geq 1 \rangle$ **and**
 proped : $\langle \text{is-proped } L \rangle$
shows $\langle \text{size } (\text{mark-of } L) > 1 \rangle$
 $\langle \text{proof} \rangle$

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 $\text{conflict-non-zero-unless-level-0} :: \langle 'st \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{conflict-non-zero-unless-level-0 } S \longleftrightarrow$
 $(\text{conflicting } S = \text{Some } \{\#\} \longrightarrow \text{count-decided } (\text{trail } S) = 0) \rangle$

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

lemma $\text{cdcl}_W\text{-restart-no-false-clause}$:
assumes
 $\langle \text{cdcl}_W\text{-restart } S \ T \rangle$
 $\langle \text{no-false-clause } S \rangle$
shows $\langle \text{no-false-clause } T \rangle$
 $\langle \text{proof} \rangle$

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-empty: ⟨*M* ∼ = []⟩

shows

clauses *S* ⊨_{pm} *D'* **and**

⟨− *lit-of* (*hd* *M*) ∈ # *D'*⟩

⟨*proof*⟩

lemma *literals-of-level0-entailed*:

assumes

struct-invs: $\langle \text{cdcl}_W\text{-all-struct-inv } S \rangle$ **and**
in-trail: $\langle L \in \text{lits-of-l } (\text{trail } S) \rangle$ **and**
lev: $\langle \text{get-level } (\text{trail } S) \ L = 0 \rangle$
shows
 $\langle \text{clauses } S \models_{pm} \{ \#L \# \} \rangle$
 $\langle \text{proof} \rangle$

1.1.10 Some higher level use on the invariants

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

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

$\langle \text{conflict } S \ T \implies$
 $\text{no-smaller-confl } S \implies$
 $\text{cdcl}_W\text{-all-struct-inv } S \implies$
 $\text{conflict-is-false-with-level } T \rangle$
 $\langle \text{proof} \rangle$

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

assumes
 $\text{cdcl}_W\text{-stgy } S \ S'$ **and**
 $n\text{-l: no-smaller-confl } S$ **and**
 $\text{conflict-is-false-with-level } S$ **and**
 $\text{cdcl}_W\text{-all-struct-inv } S$
shows $\text{conflict-is-false-with-level } S'$
 $\langle \text{proof} \rangle$

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

assumes
 $\langle \text{cdcl}_W\text{-o } S \ T \rangle$
 $\langle \text{cdcl}_W\text{-all-struct-inv } S \rangle$
 $\langle \text{conflict-is-false-with-level } S \rangle$
shows $\langle \text{conflict-is-false-with-level } T \rangle$
 $\langle \text{proof} \rangle$

lemma *no-step-cdcl_W-total*:

assumes
 $\langle \text{no-step } \text{cdcl}_W \ S \rangle$
 $\langle \text{conflicting } S = \text{None} \rangle$
 $\langle \text{no-strange-atm } S \rangle$
shows $\langle \text{total-over-m } (\text{lits-of-l } (\text{trail } S)) \ (\text{set-mset } (\text{clauses } S)) \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl_W-Ex-cdcl_W-stgy*:

assumes
 $\langle \text{cdcl}_W \ S \ T \rangle$
shows $\langle \text{Ex } (\text{cdcl}_W\text{-stgy } S) \rangle$
 $\langle \text{proof} \rangle$

lemma *no-step-skip-hd-in-conflicting*:

assumes
 $\text{inv-s: } \langle \text{cdcl}_W\text{-stgy-invariant } S \rangle$ **and**

```

  inv: ⟨cdclW-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: ⟨cdclW-all-struct-inv S⟩ and
  stgy: ⟨cdclW-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: ⟨cdclW-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: $\langle \text{backtrack } S \ T \rangle$ **and**
inv: $\langle \text{cdcl}_W\text{-all-struct-inv } S \rangle$ **and**
smaller: $\langle \text{no-smaller-propa } S \rangle$ **and**
smaller-conf: $\langle \text{no-smaller-conf } S \rangle$ **and**
E-subset: $\langle E \subset\# \text{ mark-of } (\text{hd-trail } T) \rangle$
shows $\langle E \notin\# \text{ clauses } S \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl_W-stgy-no-relearned-highest-subres-clause*:

assumes
cdcl: $\langle \text{backtrack } S \ T \rangle$ **and**
inv: $\langle \text{cdcl}_W\text{-all-struct-inv } S \rangle$ **and**
smaller: $\langle \text{no-smaller-propa } S \rangle$ **and**
smaller-conf: $\langle \text{no-smaller-conf } S \rangle$ **and**
E-subset: $\langle \text{mark-of } (\text{hd-trail } T) = \text{add-mset } (\text{lit-of } (\text{hd-trail } T)) \ E \rangle$
shows $\langle \text{add-mset } (- \text{lit-of } (\text{hd-trail } T)) \ E \notin\# \text{ clauses } S \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl_W-stgy-distinct-mset*:

assumes
cdcl: $\langle \text{cdcl}_W\text{-stgy } S \ T \rangle$ **and**
inv: $\langle \text{cdcl}_W\text{-all-struct-inv } S \rangle$ **and**
smaller: $\langle \text{no-smaller-propa } S \rangle$ **and**
dist: $\langle \text{distinct-mset } (\text{clauses } S) \rangle$
shows
 $\langle \text{distinct-mset } (\text{clauses } T) \rangle$
 $\langle \text{proof} \rangle$

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: $\langle \text{cdcl}_W\text{-stgy } S \ T \rangle$ **and**
inv: $\langle \text{cdcl}_W\text{-all-struct-inv } S \rangle$ **and**
smaller: $\langle \text{no-smaller-propa } S \rangle$ **and**
dist: $\langle \text{distinct-mset } (\text{learned-clss } S + \text{remdups-mset } (\text{init-clss } S)) \rangle$
shows
 $\langle \text{distinct-mset } (\text{learned-clss } T + \text{remdups-mset } (\text{init-clss } T)) \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-cdcl_W-stgy-distinct-mset-clauses*:

assumes
st: $\langle \text{cdcl}_W\text{-stgy}^{**} \ R \ S \rangle$ **and**
invR: $\langle \text{cdcl}_W\text{-all-struct-inv } R \rangle$ **and**
dist: $\langle \text{distinct-mset } (\text{clauses } R) \rangle$ **and**
no-smaller: $\langle \text{no-smaller-propa } R \rangle$
shows $\langle \text{distinct-mset } (\text{clauses } S) \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-cdcl_W-stgy-distinct-mset-learned-clauses*:

assumes
st: $\langle \text{cdcl}_W\text{-stgy}^{**} \ R \ S \rangle$ **and**
invR: $\langle \text{cdcl}_W\text{-all-struct-inv } R \rangle$ **and**
dist: $\langle \text{distinct-mset } (\text{learned-clss } R + \text{remdups-mset } (\text{init-clss } R)) \rangle$ **and**

no-smaller: $\langle \text{no-smaller-propa } R \rangle$
shows *distinct-mset* (*learned-clss* $S + \text{remdups-mset} (\text{init-clss } S)$)
 $\langle \text{proof} \rangle$

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)
 $\langle \text{proof} \rangle$

lemma *cdcl_W-stgy-learned-distinct-mset-new*:

assumes
cdcl: $\langle \text{cdcl}_W\text{-stgy } S \ T \rangle$ **and**
inv: *cdcl_W-all-struct-inv* S **and**
smaller: $\langle \text{no-smaller-propa } S \rangle$ **and**
dist: $\langle \text{distinct-mset} (\text{learned-clss } S - A) \rangle$
shows $\langle \text{distinct-mset} (\text{learned-clss } T - A) \rangle$
 $\langle \text{proof} \rangle$

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: $\langle \text{no-smaller-propa } R \rangle$ **and**
 $\langle \text{distinct-mset} (\text{learned-clss } R - A) \rangle$
shows *distinct-mset* (*learned-clss* $S - A$)
 $\langle \text{proof} \rangle$

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: $\langle \text{no-smaller-propa } R \rangle$
shows *distinct-mset* (*learned-clss* $S - \text{learned-clss } R$)
 $\langle \text{proof} \rangle$

Decrease of a Measure

fun *cdcl_W-restart-measure* **where**

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

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 $\text{length} (\text{trail } S) \leq \text{card} (\text{atms-of-mm} (\text{init-clss } S))$
 $\langle \text{proof} \rangle$

lemma *length-model-le-vars-all-inv*:
assumes *cdcl_W-all-struct-inv S*
shows $\text{length } (\text{trail } S) \leq \text{card } (\text{atms-of-mm } (\text{init-clss } S))$
<proof>

lemma *learned-clss-less-upper-bound*:
fixes $S :: 'st$
assumes
distinct-cdcl_W-state S and
 $\forall s \in \# \text{ learned-clss } S. \neg \text{tautology } s$
shows $\text{card}(\text{set-mset } (\text{learned-clss } S)) \leq 3 \wedge \text{card } (\text{atms-of-mm } (\text{learned-clss } S))$
<proof>

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

lemma *cdcl_W-stgy-step-decreasing*:
fixes $S T :: 'st$
assumes
cdcl: $\langle \text{cdcl}_W\text{-stgy } S T \rangle$ **and**
struct-inv: $\langle \text{cdcl}_W\text{-all-struct-inv } S \rangle$ **and**
smaller: $\langle \text{no-smaller-propa } S \rangle$
shows $(\text{cdcl}_W\text{-restart-measure } T, \text{cdcl}_W\text{-restart-measure } S) \in \text{lexn less-than } 3$
<proof>

lemma *empty-trail-no-smaller-propa:* $\langle \text{trail } R = [] \implies \text{no-smaller-propa } R \rangle$
<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: $\text{trail } R = []$ **and**
cdcl_W-all-struct-inv R
shows $(\text{cdcl}_W\text{-restart-measure } S, \text{cdcl}_W\text{-restart-measure } R) \in \text{lexn less-than } 3$
<proof>

lemma *tranclp-cdcl_W-stgy-S0-decreasing*:
fixes $R S T :: 'st$

assumes

pl: $cdcl_W\text{-stgy}^{++}$ (*init-state* *N*) *S* **and**
no-dup: *distinct-mset-mset* *N*

shows ($cdcl_W\text{-restart-measure } S, cdcl_W\text{-restart-measure } (init\text{-state } N) \in \text{learn less-than } 3$)
<proof>

lemma *wf-tranclp-cdcl_W-stgy*:

wf $\{(S::'st, init\text{-state } N) \mid S \ N. \ distinct\text{-mset-mset } N \wedge cdcl_W\text{-stgy}^{++} (init\text{-state } N) \ S\}$
<proof>

The following theorem 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 than $(2::'a)^n$.

lemma *no-conflict-after-decide*:

assumes

dec: $\langle decide \ S \ T \rangle$ **and**
inv: $\langle cdcl_W\text{-all-struct-inv } T \rangle$ **and**
smaller: $\langle no\text{-smaller-propa } T \rangle$ **and**
smaller-conf: $\langle no\text{-smaller-conf } T \rangle$

shows $\langle \neg \text{conflict } T \ U \rangle$

<proof>

abbreviation *list-weight-propa-trail* :: $\langle ('v \ \text{literal}, 'v \ \text{literal}, 'v \ \text{literal multiset}) \ \text{annotated-lit list} \Rightarrow \text{bool list} \rangle$ **where**

$\langle list\text{-weight-propa-trail } M \equiv \text{map } is\text{-proped } M \rangle$

definition *comp-list-weight-propa-trail* :: $\langle nat \Rightarrow ('v \ \text{literal}, 'v \ \text{literal}, 'v \ \text{literal multiset}) \ \text{annotated-lit list} \Rightarrow \text{bool list} \rangle$ **where**

$\langle comp\text{-list-weight-propa-trail } b \ M \equiv \text{replicate } (b - \text{length } M) \ \text{False} \ @ \ list\text{-weight-propa-trail } M \rangle$

lemma *comp-list-weight-propa-trail-append[simp]*:

$\langle comp\text{-list-weight-propa-trail } b \ (M \ @ \ M') =$
 $comp\text{-list-weight-propa-trail } (b - \text{length } M') \ M \ @ \ list\text{-weight-propa-trail } M' \rangle$
<proof>

lemma *comp-list-weight-propa-trail-append-single[simp]*:

$\langle comp\text{-list-weight-propa-trail } b \ (M \ @ \ [K]) =$
 $comp\text{-list-weight-propa-trail } (b - 1) \ M \ @ \ [is\text{-proped } K] \rangle$
<proof>

lemma *comp-list-weight-propa-trail-cons[simp]*:

$\langle comp\text{-list-weight-propa-trail } b \ (K \ \# \ M') =$
 $comp\text{-list-weight-propa-trail } (b - \text{Suc } (\text{length } M')) \ [] \ @ \ is\text{-proped } K \ \# \ list\text{-weight-propa-trail } M' \rangle$
<proof>

fun *of-list-weight* :: $\langle \text{bool list} \Rightarrow nat \rangle$ **where**

$\langle of\text{-list-weight } [] = 0 \rangle$

$\mid \langle of\text{-list-weight } (b \ \# \ xs) = (\text{if } b \ \text{then } 1 \ \text{else } 0) + 2 * of\text{-list-weight } xs \rangle$

lemma *of-list-weight-append[simp]*:

$\langle of\text{-list-weight } (a \ @ \ b) = of\text{-list-weight } a + 2^{\text{length } a} * of\text{-list-weight } b \rangle$
<proof>

lemma *of-list-weight-append-single[simp]*:

$\langle of\text{-list-weight } (a \ @ \ [b]) = of\text{-list-weight } a + 2^{\text{length } a} * (\text{if } b \ \text{then } 1 \ \text{else } 0) \rangle$

⟨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) = 2ⁿ - 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 $S \rightarrow \dots \rightarrow T$. The bookkeeping part:

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

Then the simple part:

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

Comments on the proof:

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

Comments on the bound:

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

lemma *cdcl-pow2-n-learned-clauses:*

assumes

cdcl: $\langle \text{cdcl}_W^{**} S T \rangle$ **and**

confl: $\langle \text{conflicting } S = \text{None} \rangle$ **and**

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

shows $\langle \text{size}(\text{learned-clss } T) \leq \text{size}(\text{learned-clss } S) + 2^{\wedge}(\text{card}(\text{atms-of-mm}(\text{init-clss } S))) \rangle$
(is $\langle - \leq - + ?b \rangle$)

$\langle \text{proof} \rangle$

Application of the previous theorem to an initial state:

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

assumes

cdcl: $\langle \text{cdcl}_W^{**}(\text{init-state } N) T \rangle$ **and**

inv: $\langle \text{cdcl}_W\text{-all-struct-inv}(\text{init-state } N) \rangle$

shows $\langle \text{size}(\text{learned-clss } T) \leq 2^{\wedge}(\text{card}(\text{atms-of-mm } N)) \rangle$

$\langle \text{proof} \rangle$

A rather obvious theorem, but can be handy when talking about CDCL with inclusion of new rules.

lemma *cdcl_W-enlarge-clauses:*

assumes

$\langle \text{cdcl}_W S S' \rangle$ **and**

$\langle \text{trail } T = \text{trail } S \wedge \text{init-clss } T = \text{init-clss } S + N' \wedge$

$\text{learned-clss } T = \text{learned-clss } S + U' \wedge \text{conflicting } T = \text{conflicting } S \rangle$

shows $\langle \exists T'. \text{trail } T' = \text{trail } S' \wedge \text{init-clss } T' = \text{init-clss } S' + N' \wedge$

$\text{learned-clss } T' = \text{learned-clss } S' + U' \wedge \text{conflicting } T' = \text{conflicting } S' \wedge$

$\text{cdcl}_W T T' \rangle$

$\langle \text{proof} \rangle$

lemma *rtranclp-cdcl_W-enlarge-clauses:*

assumes $\langle \text{trail } T = \text{trail } S \wedge \text{init-clss } T = \text{init-clss } S + N' \wedge$

$\langle \text{learned-clss } T = \text{learned-clss } S + U' \wedge \text{conflicting } T = \text{conflicting } S \rangle$ **and**
 $\langle \text{rtranclp } \text{cdcl}_W S S' \rangle$
shows $\langle \exists T'. \text{trail } T' = \text{trail } S' \wedge \text{init-clss } T' = \text{init-clss } S' + N' \wedge$
 $\text{learned-clss } T' = \text{learned-clss } S' + U' \wedge \text{conflicting } T' = \text{conflicting } S' \wedge$
 $\text{cdcl}_W^{**} T T' \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{cdcl}_W\text{-clauses-cong}$:

assumes

$\langle \text{cdcl}_W S S' \rangle$ **and**

$\langle \text{trail } T = \text{trail } S \wedge \text{set-mset } (\text{init-clss } T) = \text{set-mset } (\text{init-clss } S) \wedge$

$\text{set-mset } (\text{learned-clss } T) = \text{set-mset } (\text{learned-clss } S) \wedge \text{conflicting } T = \text{conflicting } S \rangle$

shows $\langle \exists T'. \text{trail } T' = \text{trail } S' \wedge \text{set-mset } (\text{init-clss } T') = \text{set-mset } (\text{init-clss } S') \wedge$

$\text{learned-clss } T' = \text{learned-clss } T + (\text{learned-clss } S' - \text{learned-clss } S) \wedge \text{conflicting } T' = \text{conflicting}$

$S' \wedge$

$\text{cdcl}_W T T' \rangle$

$\langle \text{proof} \rangle$

lemma $\text{cdcl}_W\text{-learned-clss-mono}$: $\langle \text{cdcl}_W S T \implies \text{learned-clss } S \subseteq\# \text{learned-clss } T \rangle$

$\langle \text{proof} \rangle$

lemma $\text{rtranclp-cdcl}_W\text{-learned-clauses-mono}$: $\langle \text{cdcl}_W^{**} S T \implies \text{learned-clss } S \subseteq\# \text{learned-clss } T \rangle$

$\langle \text{proof} \rangle$

lemma $\text{rtranclp-cdcl}_W\text{-clauses-cong}$:

assumes $\langle \text{trail } T = \text{trail } S \wedge \text{set-mset } (\text{init-clss } T) = \text{set-mset } (\text{init-clss } S) \wedge$

$\text{set-mset } (\text{learned-clss } T) = \text{set-mset } (\text{learned-clss } S) \wedge \text{conflicting } T = \text{conflicting } S \rangle$ **and**

$\langle \text{rtranclp } \text{cdcl}_W S S' \rangle$

shows $\langle \exists T'. \text{trail } T' = \text{trail } S' \wedge \text{set-mset } (\text{init-clss } T') = \text{set-mset } (\text{init-clss } S') \wedge$

$\text{learned-clss } T' = \text{learned-clss } T + (\text{learned-clss } S' - \text{learned-clss } S) \wedge \text{conflicting } T' = \text{conflicting}$

$S' \wedge$

$\text{cdcl}_W^{**} T T' \rangle$

$\langle \text{proof} \rangle$

lemma $\text{cdcl}_W\text{-all-struct-inv-clauses-cong}$:

assumes

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

$\langle \text{trail } T = \text{trail } S \wedge \text{set-mset } (\text{init-clss } T) = \text{set-mset } (\text{init-clss } S) \wedge$

$\text{set-mset } (\text{learned-clss } T) = \text{set-mset } (\text{learned-clss } S) \wedge \text{conflicting } T = \text{conflicting } S \rangle$

shows $\langle \text{cdcl}_W\text{-all-struct-inv } T \rangle$

$\langle \text{proof} \rangle$

end

end

1.2 Merging backjump rules

theory CDCL-W-Merge

imports CDCL-W

begin

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

1. *conflict-driven-clause-learning_W.conflict* to find the conflict
2. the conflict is analysed by repetitive application of *conflict-driven-clause-learning_W.resolve* and *conflict-driven-clause-learning_W.skip*,
3. finally *conflict-driven-clause-learning_W.backtrack* is used to backtrack.

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

1.2.1 Inclusion of the States

context *conflict-driven-clause-learning_W*
begin

declare *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 \neg *backtrack V T*
 \langle *proof* \rangle

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

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

lemma *rtranclp-cdcl_W-bj-skip-or-resolve-backtrack*:
assumes *cdcl_W-bj** S U*
shows *skip-or-resolve** S U* \vee ($\exists T$. *skip-or-resolve** S T* \wedge *backtrack T U*)
 \langle *proof* \rangle

lemma *rtranclp-skip-or-resolve-rtranclp-cdcl_W-restart*:
*skip-or-resolve** S T* \Longrightarrow *cdcl_W-restart** S T*
 \langle *proof* \rangle

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

lemma *wf-skip-or-resolve*:
wf $\{(T, S)$. *skip-or-resolve S T* $\}$
 \langle *proof* \rangle

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

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

context *conflict-driven-clause-learning_W*
begin

1.2.2 More lemmas about Conflict, Propagate and Backjumping

Termination

lemma *cdcl_W-bj-measure*:

assumes *cdcl_W-bj* $S T$

shows $\text{length}(\text{trail } S) + (\text{if conflicting } S = \text{None then } 0 \text{ else } 1)$
 $> \text{length}(\text{trail } T) + (\text{if conflicting } T = \text{None then } 0 \text{ else } 1)$

$\langle \text{proof} \rangle$

lemma *wf-cdcl_W-bj*:

wf $\{(b,a). \text{cdcl}_W\text{-bj } a \ b\}$

$\langle \text{proof} \rangle$

lemma *cdcl_W-bj-exists-normal-form*:

shows $\exists T. \text{full } \text{cdcl}_W\text{-bj } S T$

$\langle \text{proof} \rangle$

lemma *rtranclp-skip-state-decomp*:

assumes *skip*** $S T$

shows

$\exists M. \text{trail } S = M @ \text{trail } T \wedge (\forall m \in \text{set } M. \neg \text{is-decided } m)$

init-clss $S = \text{init-clss } T$

learned-clss $S = \text{learned-clss } T$

backtrack-lvl $S = \text{backtrack-lvl } T$

conflicting $S = \text{conflicting } T$

$\langle \text{proof} \rangle$

Analysing is confluent

lemma *backtrack-reduce-trail-to-state-eq*:

assumes

V-T: $\langle V \sim \text{tl-trail } T \rangle$ **and**

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

shows $\langle \text{reduce-trail-to } M1(\text{add-learned-clss } E(\text{update-conflicting } \text{None } V))$

$\sim \text{reduce-trail-to } M1(\text{add-learned-clss } E(\text{update-conflicting } \text{None } T)) \rangle$

$\langle \text{proof} \rangle$

lemma *rtranclp-skip-backtrack-reduce-trail-to-state-eq*:

assumes

V-T: $\langle \text{skip** } T V \rangle$ **and**

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

shows $\langle \text{reduce-trail-to } M1(\text{add-learned-clss } E(\text{update-conflicting } \text{None } T))$

$\sim \text{reduce-trail-to } M1(\text{add-learned-clss } E(\text{update-conflicting } \text{None } V)) \rangle$

$\langle \text{proof} \rangle$

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* $S W$

$\langle \text{proof} \rangle$

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: $cdcl_W\text{-all-struct-inv } S$
shows $backtrack T W$
 $\langle proof \rangle$

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

1.2.3 CDCL with Merging

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

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

lemma $cdcl_W\text{-merge-restart-cdcl}_W\text{-restart}$:
assumes $cdcl_W\text{-merge-restart } S T$
shows $cdcl_W\text{-restart}^{**} S T$
 $\langle proof \rangle$

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

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 $cdcl_W\text{-merge-cdcl}_W\text{-merge-restart}$:
 $cdcl_W\text{-merge } S T \Longrightarrow cdcl_W\text{-merge-restart } S T$
 $\langle proof \rangle$

lemma $rtranclp\text{-}cdcl_W\text{-merge-tranclp-cdcl}_W\text{-merge-restart}$:
 $cdcl_W\text{-merge}^{**} S T \Longrightarrow cdcl_W\text{-merge-restart}^{**} S T$
 $\langle proof \rangle$

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

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

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 $(\lambda S T. \text{cdcl}_W\text{-all-struct-inv } S \wedge \text{cdcl}_W\text{-merge } S T)^{++} b a$

<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* \wedge *conflicting* *V = None*)

$\vee (\exists T U. \text{cdcl}_W\text{-merge-restart}^{**} S T \wedge \text{conflicting } V \neq \text{None} \wedge \text{conflict } T U \wedge \text{cdcl}_W\text{-bj}^{**} U V)$

<proof>

lemma *no-step-cdcl_W-restart-no-step-cdcl_W-merge-restart*:

no-step cdcl_W-restart *S* \implies *no-step cdcl_W-merge-restart* *S*

<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* \neq *None*, we cannot say anything.

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

lemma *conflicting-true-full-cdcl_W-restart-iff-full-cdcl_W-merge*:

assumes *conf*: *conflicting* *S = None* **and** *lev*: *cdcl_W-M-level-inv* *S*

shows *full cdcl_W-restart* *S V* \longleftrightarrow *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* \longleftrightarrow *full cdcl_W-merge-restart* (*init-state* *N*) *V*

<proof>

1.2.4 CDCL with Merge and Strategy

The intermediate step

inductive $cdcl_W-s' :: 'st \Rightarrow 'st \Rightarrow bool$ for $S :: 'st$ where

conflict': $conflict\ S\ S' \Longrightarrow cdcl_W-s'\ S\ S' \mid$

propagate': $propagate\ S\ S' \Longrightarrow cdcl_W-s'\ S\ S' \mid$

decide': $no-step\ conflict\ S \Longrightarrow no-step\ propagate\ S \Longrightarrow decide\ S\ S' \Longrightarrow cdcl_W-s'\ S\ S' \mid$

bj': $full1\ cdcl_W-bj\ S\ S' \Longrightarrow cdcl_W-s'\ S\ S'$

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

lemma *rtranclp-cdcl_W-bj-full1-cdclp-cdcl_W-stgy*:

$cdcl_W-bj^{**}\ S\ S' \Longrightarrow cdcl_W-stgy^{**}\ S\ S'$

<proof>

lemma *cdcl_W-s'-is-rtranclp-cdcl_W-stgy*:

$cdcl_W-s'\ S\ T \Longrightarrow cdcl_W-stgy^{**}\ S\ T$

<proof>

lemma *cdcl_W-stgy-cdcl_W-s'-no-step*:

assumes $cdcl_W-stgy\ S\ U$ **and** $cdcl_W-all-struct-inv\ S$ **and** $no-step\ cdcl_W-bj\ U$

shows $cdcl_W-s'\ S\ U$

<proof>

lemma *rtranclp-cdcl_W-stgy-connected-to-rtranclp-cdcl_W-s'*:

assumes $cdcl_W-stgy^{**}\ S\ U$ **and** inv : $cdcl_W-M-level-inv\ S$

shows $cdcl_W-s'^{**}\ S\ U \vee (\exists T. cdcl_W-s'^{**}\ S\ T \wedge cdcl_W-bj^{++}\ T\ U \wedge conflicting\ U \neq None)$

<proof>

lemma *n-step-cdcl_W-stgy-iff-no-step-cdcl_W-restart-cl-cdcl_W-o*:

assumes inv : $cdcl_W-all-struct-inv\ S$

shows $no-step\ cdcl_W-s'\ S \longleftrightarrow no-step\ cdcl_W-stgy\ S$ (**is** $?S'\ S \longleftrightarrow ?C\ S$)

<proof>

lemma *cdcl_W-s'-tranclp-cdcl_W-restart*:

assumes $cdcl_W-s'\ S\ S'$ **shows** $cdcl_W-restart^{++}\ S\ S'$

<proof>

lemma *tranclp-cdcl_W-s'-tranclp-cdcl_W-restart*:

$cdcl_W-s'^{++}\ S\ S' \Longrightarrow cdcl_W-restart^{++}\ S\ S'$

<proof>

lemma *rtranclp-cdcl_W-s'-rtranclp-cdcl_W-restart*:

$cdcl_W-s'^{**}\ S\ S' \Longrightarrow cdcl_W-restart^{**}\ S\ S'$

<proof>

lemma *full-cdcl_W-stgy-iff-full-cdcl_W-s'*:

assumes inv : $cdcl_W-all-struct-inv\ S$

shows $full\ cdcl_W-stgy\ S\ T \longleftrightarrow full\ cdcl_W-s'\ S\ T$ (**is** $?S \longleftrightarrow ?S'$)

<proof>

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

lemma $\mu_C\text{-Nil}[simp]$:

$$\mu_C s b [] = 0$$

<proof>

lemma $\mu_C\text{-single}[simp]$:

$$\mu_C s b [L] = L * b^\wedge(s - \text{Suc } 0)$$

<proof>

lemma $\text{set-sum-atLeastLessThan-add}$:

$$(\sum i=k..<k+(b::\text{nat}). f i) = (\sum i=0..<b. f (k+i))$$

<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*:

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

<proof>

lemma *μ_C -append*:

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

<proof>

lemma *μ_C -cons-non-empty-inf*:

assumes *M-ge-1*: $\forall i \in \text{set}\ M. i \geq 1$ **and** *M*: $M \neq []$
shows $\mu_C\ s\ b\ M \geq b^{\wedge}(s - \text{length}\ M)$

<proof>

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

lemma *sum-of-powers*: $0 \leq k \implies (k - 1) * (\sum i=0..<n. k^i) = k^{\wedge}n - (1::nat)$

<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: $\forall i < \text{length}\ M. M!i < b$ **and**
 $s \geq \text{length}\ M$
shows $\mu_C\ s\ b\ M < b^{\wedge}s$

<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: $\forall i < \text{length}\ M. M!i < b$ **and**
 $s \geq \text{length}\ M$
 $b > 0$
shows $\mu_C\ s\ b\ M < b^{\wedge}s$

<proof>

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

lemma *μ_C -base-0*:

assumes $\text{length}\ M \leq s$
shows $\mu_C\ s\ 0\ M \leq M!0$

<proof>

lemma *finite-bounded-pair-list*:

fixes $b :: nat$
shows *finite* $\{(ys, xs). \text{length}\ xs < s \wedge \text{length}\ ys < s \wedge$

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

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

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

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*:
 $\langle \text{atms-of } \{\# \text{ -lit-of } x. x \in \# A \# \} = \text{atm-of } \langle \text{lit-of } \langle \text{set-mset } A \rangle \rangle$
 ⟨proof⟩

lemma *atms-of-ms-single-image-atm-of-lit-of*:
 $\langle \text{atms-of-ms } (\text{unmark-s } A) = \text{atm-of } \langle \text{lit-of } \langle A \rangle$
 ⟨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 :: $\langle 'st \Rightarrow ('v, \text{unit}) \text{ ann-lits} \rangle$ **and**
clauses_{NOT} :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**
prepend-trail :: $\langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
tl-trail :: $\langle 'st \Rightarrow 'st \rangle$ **and**
add-cl_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
remove-cl_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$

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

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

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

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

$\text{reduce-trail-to}_{NOT}\text{-Nil}[simp]: \langle \text{trail } S = [] \implies \text{reduce-trail-to}_{NOT} F S = S \rangle$ **and**
 $\text{reduce-trail-to}_{NOT}\text{-eq-length}[simp]: \langle \text{length } (\text{trail } S) = \text{length } F \implies \text{reduce-trail-to}_{NOT} F S = S \rangle$
 $\langle \text{proof} \rangle$

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

$\langle \text{length } (\text{trail } S) \neq \text{length } F \implies \text{trail } S \neq [] \implies$
 $\text{reduce-trail-to}_{NOT} F S = \text{reduce-trail-to}_{NOT} F (\text{tl-trail } S) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{trail-reduce-trail-to}_{NOT}\text{-length-le}:$

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

lemma $\text{trail-reduce-trail-to}_{NOT}\text{-Nil}[simp]:$

$\langle \text{trail } (\text{reduce-trail-to}_{NOT} [] S) = [] \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{clauses-reduce-trail-to}_{NOT}\text{-Nil}:$

$\langle \text{clauses}_{NOT} (\text{reduce-trail-to}_{NOT} [] S) = \text{clauses}_{NOT} S \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{trail-reduce-trail-to}_{NOT}\text{-drop}:$

$\langle \text{trail } (\text{reduce-trail-to}_{NOT} F S) =$
 $(\text{if } \text{length } (\text{trail } S) \geq \text{length } F$
 $\text{then } \text{drop } (\text{length } (\text{trail } S) - \text{length } F) (\text{trail } S)$
 $\text{else } []) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{reduce-trail-to}_{NOT}\text{-skip-beginning}:$

assumes $\langle \text{trail } S = F' @ F \rangle$
shows $\langle \text{trail } (\text{reduce-trail-to}_{NOT} F S) = F \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{reduce-trail-to}_{NOT}\text{-clauses}[simp]:$

$\langle \text{clauses}_{NOT} (\text{reduce-trail-to}_{NOT} F S) = \text{clauses}_{NOT} S \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{trail-eq-reduce-trail-to}_{NOT}\text{-eq}:$

$\langle \text{trail } S = \text{trail } T \implies \text{trail } (\text{reduce-trail-to}_{NOT} F S) = \text{trail } (\text{reduce-trail-to}_{NOT} F T) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{trail-reduce-trail-to}_{NOT}\text{-add-cl}_{NOT}[simp]:$

$\langle \text{no-dup } (\text{trail } S) \implies$
 $\text{trail } (\text{reduce-trail-to}_{NOT} F (\text{add-cl}_{NOT} C S)) = \text{trail } (\text{reduce-trail-to}_{NOT} F S) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{reduce-trail-to}_{NOT}\text{-trail-tl-trail-decomp}[simp]:$

$\langle \text{trail } S = F' @ \text{Decided } K \# F \implies$
 $\text{trail } (\text{reduce-trail-to}_{NOT} F (\text{tl-trail } S)) = F \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{reduce-trail-to}_{NOT}\text{-length}:$

$\langle \text{length } M = \text{length } M' \implies \text{reduce-trail-to}_{NOT} M S = \text{reduce-trail-to}_{NOT} M' S \rangle$
 $\langle \text{proof} \rangle$

abbreviation *trail-weight* **where**

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

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

definition *state-eq_{NOT}* :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ (**infix** \sim 50) **where**
 $\langle S \sim T \iff \text{trail } S = \text{trail } T \wedge \text{clauses}_{NOT} S = \text{clauses}_{NOT} T \rangle$

lemma *state-eq_{NOT}-ref*[*intro, simp*]:

$\langle S \sim S \rangle$
 $\langle \text{proof} \rangle$

lemma *state-eq_{NOT}-sym*:

$\langle S \sim T \iff T \sim S \rangle$
 $\langle \text{proof} \rangle$

lemma *state-eq_{NOT}-trans*:

$\langle S \sim T \implies T \sim U \implies S \sim U \rangle$
 $\langle \text{proof} \rangle$

lemma

shows

state-eq_{NOT}-trail: $\langle S \sim T \implies \text{trail } S = \text{trail } T \rangle$ **and**

state-eq_{NOT}-clauses: $\langle S \sim T \implies \text{clauses}_{NOT} S = \text{clauses}_{NOT} T \rangle$

$\langle \text{proof} \rangle$

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

lemma *reduce-trail-to_{NOT}-state-eq_{NOT}-compatible*:

assumes *ST*: $\langle S \sim T \rangle$

shows $\langle \text{reduce-trail-to}_{NOT} F S \sim \text{reduce-trail-to}_{NOT} F T \rangle$

$\langle \text{proof} \rangle$

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-cl_{NOT}* *remove-cl_{NOT}*
for

trail :: $\langle 'v, \text{unit} \rangle \text{ann-lits} \rangle$ **and**

clauses_{NOT} :: $\langle 'v \text{ clauses} \rangle$ **and**

prepend-trail :: $\langle ('v, \text{unit}) \text{ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

tl-trail :: $\langle 'st \Rightarrow 'st \rangle$ **and**

add-cl_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

remove-cl_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ +

fixes

propagate-conds :: $\langle ('v, \text{unit}) \text{ann-lit} \Rightarrow 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$

begin

inductive *propagate_{NOT}* :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **where**

$propagate_{NOT}[intro]: \langle add\text{-}mset\ L\ C\ \in\#\ \text{clauses}_{NOT}\ S\ \Rightarrow\ \text{trail}\ S\ \models_{as}\ C\ \text{Not}\ C$
 $\Rightarrow\ \text{undefined}\text{-}lit\ (\text{trail}\ S)\ L$
 $\Rightarrow\ \text{propagate}\text{-}conds\ (\text{Propagated}\ L\ ())\ S\ T$
 $\Rightarrow\ T\ \sim\ \text{prepend}\text{-}trail\ (\text{Propagated}\ L\ ())\ S$
 $\Rightarrow\ \text{propagate}_{NOT}\ S\ T \rangle$

inductive-cases $propagate_{NOT}E[elim]: \langle \text{propagate}_{NOT}\ S\ T \rangle$

end

locale $decide\text{-}ops =$

$dpll\text{-}state\ \text{trail}\ \text{clauses}_{NOT}\ \text{prepend}\text{-}trail\ \text{tl}\text{-}trail\ \text{add}\text{-}cls_{NOT}\ \text{remove}\text{-}cls_{NOT}$

for

$\text{trail} :: \langle 'st\ \Rightarrow\ ('v,\ \text{unit})\ \text{ann}\text{-}lits \rangle\ \mathbf{and}$
 $\text{clauses}_{NOT} :: \langle 'st\ \Rightarrow\ 'v\ \text{clauses} \rangle\ \mathbf{and}$
 $\text{prepend}\text{-}trail :: \langle ('v,\ \text{unit})\ \text{ann}\text{-}lit\ \Rightarrow\ 'st\ \Rightarrow\ 'st \rangle\ \mathbf{and}$
 $\text{tl}\text{-}trail :: \langle 'st\ \Rightarrow\ 'st \rangle\ \mathbf{and}$
 $\text{add}\text{-}cls_{NOT} :: \langle 'v\ \text{clause}\ \Rightarrow\ 'st\ \Rightarrow\ 'st \rangle\ \mathbf{and}$
 $\text{remove}\text{-}cls_{NOT} :: \langle 'v\ \text{clause}\ \Rightarrow\ 'st\ \Rightarrow\ 'st \rangle +$

fixes

$\text{decide}\text{-}conds :: \langle 'st\ \Rightarrow\ 'st\ \Rightarrow\ \text{bool} \rangle$

begin

inductive $decide_{NOT} :: \langle 'st\ \Rightarrow\ 'st\ \Rightarrow\ \text{bool} \rangle\ \mathbf{where}$

$decide_{NOT}[intro]:$

$\langle \text{undefined}\text{-}lit\ (\text{trail}\ S)\ L\ \Rightarrow$
 $\text{atm}\text{-}of\ L\ \in\ \text{atms}\text{-}of\text{-}mm\ (\text{clauses}_{NOT}\ S)\ \Rightarrow$
 $T\ \sim\ \text{prepend}\text{-}trail\ (\text{Decided}\ L)\ S\ \Rightarrow$
 $\text{decide}\text{-}conds\ S\ T\ \Rightarrow$
 $\text{decide}_{NOT}\ S\ T \rangle$

inductive-cases $decide_{NOT}E[elim]: \langle \text{decide}_{NOT}\ S\ S' \rangle$

end

locale $\text{backjumping}\text{-}ops =$

$dpll\text{-}state\ \text{trail}\ \text{clauses}_{NOT}\ \text{prepend}\text{-}trail\ \text{tl}\text{-}trail\ \text{add}\text{-}cls_{NOT}\ \text{remove}\text{-}cls_{NOT}$

for

$\text{trail} :: \langle 'st\ \Rightarrow\ ('v,\ \text{unit})\ \text{ann}\text{-}lits \rangle\ \mathbf{and}$
 $\text{clauses}_{NOT} :: \langle 'st\ \Rightarrow\ 'v\ \text{clauses} \rangle\ \mathbf{and}$
 $\text{prepend}\text{-}trail :: \langle ('v,\ \text{unit})\ \text{ann}\text{-}lit\ \Rightarrow\ 'st\ \Rightarrow\ 'st \rangle\ \mathbf{and}$
 $\text{tl}\text{-}trail :: \langle 'st\ \Rightarrow\ 'st \rangle\ \mathbf{and}$
 $\text{add}\text{-}cls_{NOT} :: \langle 'v\ \text{clause}\ \Rightarrow\ 'st\ \Rightarrow\ 'st \rangle\ \mathbf{and}$
 $\text{remove}\text{-}cls_{NOT} :: \langle 'v\ \text{clause}\ \Rightarrow\ 'st\ \Rightarrow\ 'st \rangle +$

fixes

$\text{backjump}\text{-}conds :: \langle 'v\ \text{clause}\ \Rightarrow\ 'v\ \text{clause}\ \Rightarrow\ 'v\ \text{literal}\ \Rightarrow\ 'st\ \Rightarrow\ 'st\ \Rightarrow\ \text{bool} \rangle$

begin

inductive $\text{backjump}\ \mathbf{where}$

$\langle \text{trail}\ S = F' @ \text{Decided}\ K \# F$

$\Rightarrow\ T\ \sim\ \text{prepend}\text{-}trail\ (\text{Propagated}\ L\ ())\ (\text{reduce}\text{-}trail\text{-}to_{NOT}\ F\ S)$

$\Rightarrow\ C\ \in\#\ \text{clauses}_{NOT}\ S$

$\Rightarrow\ \text{trail}\ S\ \models_{as}\ C\ \text{Not}\ C$

$\Rightarrow\ \text{undefined}\text{-}lit\ F\ L$

$\Rightarrow\ \text{atm}\text{-}of\ L\ \in\ \text{atms}\text{-}of\text{-}mm\ (\text{clauses}_{NOT}\ S) \cup \text{atm}\text{-}of\ ' (\text{lits}\text{-}of\text{-}l\ (\text{trail}\ S))$

$\Rightarrow\ \text{clauses}_{NOT}\ S\ \models_{pm}\ \text{add}\text{-}mset\ L\ C'$

$\Rightarrow\ F\ \models_{as}\ C\ \text{Not}\ C'$

$\Rightarrow\ \text{backjump}\text{-}conds\ C\ C'\ L\ S\ T$

$\Rightarrow\ \text{backjump}\ S\ T \rangle$

inductive-cases *backjumpE*: $\langle \text{backjump } S \ T \rangle$

The condition *atm-of* $L \in \text{atms-of-mm } (\text{clauses}_{NOT} \ S) \cup \text{atm-of ' lits-of-l } (\text{trail } S)$ is not implied by the the condition $\text{clauses}_{NOT} \ S \models_{pm} \text{add-mset } L \ C'$ (no negation).

end

2.2.3 DPLL with Backjumping

locale *dpll-with-backjumping-ops* =

propagate-ops *trail* *clauses*_{NOT} *prepend-trail* *tl-trail* *add-cl*_{NOT} *remove-cl*_{NOT} *propagate-conds* +
decide-ops *trail* *clauses*_{NOT} *prepend-trail* *tl-trail* *add-cl*_{NOT} *remove-cl*_{NOT} *decide-conds* +
backjumping-ops *trail* *clauses*_{NOT} *prepend-trail* *tl-trail* *add-cl*_{NOT} *remove-cl*_{NOT} *backjump-conds*

for

trail :: $\langle 'st \Rightarrow ('v, \text{unit}) \text{ ann-lits} \rangle$ **and**
*clauses*_{NOT} :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**
prepend-trail :: $\langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
tl-trail :: $\langle 'st \Rightarrow 'st \rangle$ **and**
*add-cl*_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
*remove-cl*_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
inv :: $\langle 'st \Rightarrow \text{bool} \rangle$ **and**
decide-conds :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **and**
backjump-conds :: $\langle 'v \text{ clause} \Rightarrow 'v \text{ clause} \Rightarrow 'v \text{ literal} \Rightarrow 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **and**
propagate-conds :: $\langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ +

assumes

bj-can-jump:

$\langle \bigwedge S \ C \ F' \ K \ F \ L.$

inv $S \Longrightarrow$

trail $S = F' @ \text{Decided } K \ \# \ F \Longrightarrow$

$C \in \# \text{clauses}_{NOT} \ S \Longrightarrow$

trail $S \models_{as} \text{CNot } C \Longrightarrow$

undefined-lit $F \ L \Longrightarrow$

atm-of $L \in \text{atms-of-mm } (\text{clauses}_{NOT} \ S) \cup \text{atm-of ' } (\text{lits-of-l } (F' @ \text{Decided } K \ \# \ F)) \Longrightarrow$

$\text{clauses}_{NOT} \ S \models_{pm} \text{add-mset } L \ C' \Longrightarrow$

$F \models_{as} \text{CNot } C' \Longrightarrow$

$\neg \text{no-step } \text{backjump } S \rangle$ **and**

can-propagate-or-decide-or-backjump:

$\langle \text{atm-of } L \in \text{atms-of-mm } (\text{clauses}_{NOT} \ S) \Longrightarrow$

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

satisfiable $(\text{set-mset } (\text{clauses}_{NOT} \ S)) \Longrightarrow$

inv $S \Longrightarrow$

no-dup $(\text{trail } S) \Longrightarrow$

$\exists T. \text{decide}_{NOT} \ S \ T \vee \text{propagate}_{NOT} \ S \ T \vee \text{backjump } S \ T \rangle$

begin

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

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

Definition

We define *dpll* with backjumping:

inductive *dpll-bj* :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **for** $S :: 'st$ **where**

*bj-decide*_{NOT}: $\langle \text{decide}_{NOT} \ S \ S' \Longrightarrow \text{dpll-bj } S \ S' \rangle$ |

*bj-propagate*_{NOT}: $\langle \text{propagate}_{NOT} S S' \implies \text{dpll-bj } S S' \rangle$ |
bj-backjump: $\langle \text{backjump } S S' \implies \text{dpll-bj } S S' \rangle$

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* :: $\langle 'st \rangle$

assumes

$\langle \text{dpll-bj } S T \rangle$ **and**

$\langle \text{inv } S \rangle$

$\langle \bigwedge L T. \text{undefined-lit } (\text{trail } S) L \implies \text{atm-of } L \in \text{atms-of-mm } (\text{clauses}_{NOT} S)$

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

$\implies P S T \rangle$ **and**

$\langle \bigwedge C L T. \text{add-mset } L C \in \# \text{ clauses}_{NOT} S \implies \text{trail } S \models_{as} C \text{Not } C \implies \text{undefined-lit } (\text{trail } S) L$

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

$\implies P S T \rangle$ **and**

$\langle \bigwedge C F' K F L C' T. C \in \# \text{ clauses}_{NOT} S \implies F' @ \text{Decided } K \# F \models_{as} C \text{Not } C$

$\implies \text{trail } S = F' @ \text{Decided } K \# F$

$\implies \text{undefined-lit } F L$

$\implies \text{atm-of } L \in \text{atms-of-mm } (\text{clauses}_{NOT} S) \cup \text{atm-of } ' (\text{lits-of-l } (F' @ \text{Decided } K \# F))$

$\implies \text{clauses}_{NOT} S \models_{pm} \text{add-mset } L C'$

$\implies F \models_{as} C \text{Not } C'$

$\implies T \sim \text{prepend-trail } (\text{Propagated } L ()) (\text{reduce-trail-to}_{NOT} F S)$

$\implies P S T \rangle$

shows $\langle P S T \rangle$

$\langle \text{proof} \rangle$

Basic properties

First, some better suited induction principle **lemma** *dpll-bj-clauses*:

assumes $\langle \text{dpll-bj } S T \rangle$ **and** $\langle \text{inv } S \rangle$

shows $\langle \text{clauses}_{NOT} S = \text{clauses}_{NOT} T \rangle$

$\langle \text{proof} \rangle$

No duplicates in the trail **lemma** *dpll-bj-no-dup*:

assumes $\langle \text{dpll-bj } S T \rangle$ **and** $\langle \text{inv } S \rangle$

and $\langle \text{no-dup } (\text{trail } S) \rangle$

shows $\langle \text{no-dup } (\text{trail } T) \rangle$

$\langle \text{proof} \rangle$

Valuations **lemma** *dpll-bj-sat-iff*:

assumes $\langle \text{dpll-bj } S T \rangle$ **and** $\langle \text{inv } S \rangle$

shows $\langle I \models_{sm} \text{clauses}_{NOT} S \longleftrightarrow I \models_{sm} \text{clauses}_{NOT} T \rangle$

$\langle \text{proof} \rangle$

Clauses **lemma** *dpll-bj-atms-of-ms-clauses-inv*:

assumes

$\langle \text{dpll-bj } S T \rangle$ **and**

$\langle \text{inv } S \rangle$

shows $\langle \text{atms-of-mm } (\text{clauses}_{NOT} S) = \text{atms-of-mm } (\text{clauses}_{NOT} T) \rangle$

$\langle \text{proof} \rangle$

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

assumes

$\langle \text{dpll-bj } S T \rangle$ **and**

$\langle \text{inv } S \rangle$ **and**
 $\langle \text{atm-of } ' (\text{lits-of-l } (\text{trail } S)) \subseteq \text{atms-of-mm } (\text{clauses}_{NOT} S) \rangle$
shows $\langle \text{atm-of } ' (\text{lits-of-l } (\text{trail } T)) \subseteq \text{atms-of-mm } (\text{clauses}_{NOT} S) \rangle$
 $\langle \text{proof} \rangle$

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

assumes $\langle \text{dpll-bj } S T \rangle$ **and**
 $\langle \text{inv } S \rangle$ **and**
 $\langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq A \rangle$ **and**
 $\langle \text{atm-of } ' (\text{lits-of-l } (\text{trail } S)) \subseteq A \rangle$
shows $\langle \text{atm-of } ' (\text{lits-of-l } (\text{trail } T)) \subseteq A \rangle$
 $\langle \text{proof} \rangle$

lemma *dpll-bj-all-decomposition-implies-inv:*

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

Termination

Using a proper measure lemma *length-get-all-ann-decomposition-append-Decided:*

$\langle \text{length } (\text{get-all-ann-decomposition } (F' @ \text{Decided } K \# F)) =$
 $\text{length } (\text{get-all-ann-decomposition } F')$
 $+ \text{length } (\text{get-all-ann-decomposition } (\text{Decided } K \# F))$
 $- 1 \rangle$
 $\langle \text{proof} \rangle$

lemma *take-length-get-all-ann-decomposition-decided-sandwich:*

$\langle \text{take } (\text{length } (\text{get-all-ann-decomposition } F))$
 $(\text{map } (f \circ \text{snd}) (\text{rev } (\text{get-all-ann-decomposition } (F' @ \text{Decided } K \# F))))$
 $=$
 $\text{map } (f \circ \text{snd}) (\text{rev } (\text{get-all-ann-decomposition } F))$
 \rangle
 $\langle \text{proof} \rangle$

lemma *length-get-all-ann-decomposition-length:*

$\langle \text{length } (\text{get-all-ann-decomposition } M) \leq 1 + \text{length } M \rangle$
 $\langle \text{proof} \rangle$

lemma *length-in-get-all-ann-decomposition-bounded:*

assumes $i: \langle i \in \text{set } (\text{trail-weight } S) \rangle$
shows $\langle i \leq \text{Suc } (\text{length } (\text{trail } S)) \rangle$
 $\langle \text{proof} \rangle$

Well-foundedness The bounds are the following:

- $1 + \text{card } (\text{atms-of-ms } A)$: $\text{card } (\text{atms-of-ms } A)$ is an upper bound on the length of the list. As *get-all-ann-decomposition* appends an possibly empty couple at the end, adding one is needed.
- $2 + \text{card } (\text{atms-of-ms } A)$: $\text{card } (\text{atms-of-ms } A)$ is an upper bound on the number of elements, where adding one is necessary for the same reason as for the bound on the list,

and one is needed to have a strict bound.

abbreviation *unassigned-lit* :: $\langle 'b \text{ clause set} \Rightarrow 'a \text{ list} \Rightarrow \text{nat} \rangle$ **where**
 $\langle \text{unassigned-lit } N \ M \equiv \text{card } (\text{atms-of-ms } N) - \text{length } M \rangle$

lemma *dpll-bj-trail-mes-increasing-prop*:

fixes $M :: \langle 'v, \text{unit} \rangle \text{ ann-lits} \rangle$ **and** $N :: \langle 'v \text{ clauses} \rangle$

assumes

$\langle \text{dpll-bj } S \ T \rangle$ **and**

$\langle \text{inv } S \rangle$ **and**

$N\text{-}A: \langle \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \subseteq \text{atms-of-ms } A \rangle$ **and**

$M\text{-}A: \langle \text{atm-of } ' \text{ lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A \rangle$ **and**

$n\text{-}d: \langle \text{no-dup } (\text{trail } S) \rangle$ **and**

$\text{finite}: \langle \text{finite } A \rangle$

shows $\langle \mu_C (1 + \text{card } (\text{atms-of-ms } A)) (2 + \text{card } (\text{atms-of-ms } A)) (\text{trail-weight } T) \rangle$

$\rangle \mu_C (1 + \text{card } (\text{atms-of-ms } A)) (2 + \text{card } (\text{atms-of-ms } A)) (\text{trail-weight } S) \rangle$

$\langle \text{proof} \rangle$

lemma *dpll-bj-trail-mes-decreasing-prop*:

assumes *dpll*: $\langle \text{dpll-bj } S \ T \rangle$ **and** *inv*: $\langle \text{inv } S \rangle$ **and**

$N\text{-}A: \langle \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \subseteq \text{atms-of-ms } A \rangle$ **and**

$M\text{-}A: \langle \text{atm-of } ' \text{ lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A \rangle$ **and**

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

$\text{fin}\text{-}A: \langle \text{finite } A \rangle$

shows $\langle (2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A)) \rangle$

$\quad - \mu_C (1 + \text{card } (\text{atms-of-ms } A)) (2 + \text{card } (\text{atms-of-ms } A)) (\text{trail-weight } T) \rangle$

$\quad < (2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A)) \rangle$

$\quad - \mu_C (1 + \text{card } (\text{atms-of-ms } A)) (2 + \text{card } (\text{atms-of-ms } A)) (\text{trail-weight } S) \rangle$

$\langle \text{proof} \rangle$

lemma *wf-dpll-bj*:

assumes *fin*: $\langle \text{finite } A \rangle$

shows $\langle \text{wf } \{(T, S), \text{dpll-bj } S \ T\} \rangle$

$\quad \wedge \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \subseteq \text{atms-of-ms } A \wedge \text{atm-of } ' \text{ lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A$

$\quad \wedge \text{no-dup } (\text{trail } S) \wedge \text{inv } S \rangle$

$(\text{is } \langle \text{wf } ?A \rangle)$

$\langle \text{proof} \rangle$

Alternative termination proof **abbreviation** *DPLL-mes_W* **where**

$\langle \text{DPLL-mes}_W \ A \ M \equiv$

$\quad \text{map } (\lambda L. \text{if is-decided } L \text{ then } 2::\text{nat} \text{ else } 1) (\text{rev } M) \ @ \ \text{replicate } (\text{card } A - \text{length } M) \ 3 \rangle$

lemma *distinctcard-atm-of-lit-of-eq-length*:

assumes *no-dup* S

shows $\text{card } (\text{atm-of } ' \text{ lits-of-l } S) = \text{length } S$

$\langle \text{proof} \rangle$

lemma *dpll-bj-trail-mes-decreasing-less-than*:

assumes *dpll*: $\langle \text{dpll-bj } S \ T \rangle$ **and** *inv*: $\langle \text{inv } S \rangle$ **and**

$N\text{-}A: \langle \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \subseteq \text{atms-of-ms } A \rangle$ **and**

$M\text{-}A: \langle \text{atm-of } ' \text{ lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A \rangle$ **and**

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

$\text{fin}\text{-}A: \langle \text{finite } A \rangle$

shows $\langle (\text{DPLL-mes}_W (\text{atms-of-ms } A) (\text{trail } T), \text{DPLL-mes}_W (\text{atms-of-ms } A) (\text{trail } S)) \in$

$\quad \text{lexn less-than } (\text{card } ((\text{atms-of-ms } A))) \rangle$

⟨proof⟩

lemma

assumes $fin[simp]$: ⟨finite A ⟩

shows ⟨wf $\{(T, S), dpll-bj\} S T$

$\wedge atms-of-mm (clauses_{NOT} S) \subseteq atms-of-ms A \wedge atm-of \text{ ' lits-of-l } (trail S) \subseteq atms-of-ms A$

$\wedge no-dup (trail S) \wedge inv S \rangle$

(**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 that $satisfiable\ N, \neg M \models_{as} N$ and there is no remaining step is incompatible.

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

The assumption are saying that we have a finite upper bound A for the literals, that we cannot do any step $\forall S'. \neg dpll-bj\ S S'$

theorem *dpll-backjump-final-state*:

fixes $A :: \langle 'v\ clause\ set \rangle$ **and** $S\ T :: \langle 'st \rangle$

assumes

⟨ $atms-of-mm (clauses_{NOT} S) \subseteq atms-of-ms A$ ⟩ **and**

⟨ $atm-of \text{ ' lits-of-l } (trail S) \subseteq atms-of-ms A$ ⟩ **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 ⟨ $unsatisfiable (set-mset (clauses_{NOT} S))$

$\vee (trail\ S \models_{asm}\ clauses_{NOT}\ S \wedge satisfiable (set-mset (clauses_{NOT} S)))$ ⟩

⟨proof⟩

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

locale *dpll-with-backjumping* =

dpll-with-backjumping-ops $trail\ clauses_{NOT}\ prepend-trail\ tl-trail\ add-cl_{NOT}\ remove-cl_{NOT}\ inv$
 $decide-conds\ backjump-conds\ propagate-conds$

for

$trail :: \langle 'st \Rightarrow ('v, unit)\ ann-lits \rangle$ **and**

$clauses_{NOT} :: \langle 'st \Rightarrow 'v\ clauses \rangle$ **and**

$prepend-trail :: \langle ('v, unit)\ ann-lit \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

$tl-trail :: \langle 'st \Rightarrow 'st \rangle$ **and**

$add_cls_{NOT} :: \langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
 $remove_cls_{NOT} :: \langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
 $inv :: \langle 'st \Rightarrow bool \rangle$ **and**
 $decide_conds :: \langle 'st \Rightarrow 'st \Rightarrow bool \rangle$ **and**
 $backjump_conds :: \langle 'v \text{ clause} \Rightarrow 'v \text{ clause} \Rightarrow 'v \text{ literal} \Rightarrow 'st \Rightarrow 'st \Rightarrow bool \rangle$ **and**
 $propagate_conds :: \langle ('v, unit) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \Rightarrow bool \rangle$
+
assumes $dpll\text{-}bj\text{-}inv: \langle \bigwedge S T. dpll\text{-}bj S T \Longrightarrow inv S \Longrightarrow inv T \rangle$
begin

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

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

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

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

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

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

lemma $rtranclp\text{-}dpll\text{-}bj\text{-}all\text{-}decomposition\text{-}implies\text{-}inv$:
assumes
 $\langle dpll\text{-}bj^{**} S T \rangle$ **and**
 $\langle inv S \rangle$
 $\langle all\text{-}decomposition\text{-}implies\text{-}m (clauses_{NOT} S) (get\text{-}all\text{-}ann\text{-}decomposition (trail S)) \rangle$
shows $\langle all\text{-}decomposition\text{-}implies\text{-}m (clauses_{NOT} T) (get\text{-}all\text{-}ann\text{-}decomposition (trail T)) \rangle$
 $\langle proof \rangle$

lemma *rtranclp-dpll-bj-inv-incl-dpll-bj-inv-trancl*:

$\langle \{(T, S). \text{dpll-bj}^{++} S T$
 $\wedge \text{atms-of-mm} (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A \wedge \text{atm-of ' lits-of-l} (\text{trail } S) \subseteq \text{atms-of-ms } A$
 $\wedge \text{no-dup} (\text{trail } S) \wedge \text{inv } S\}$
 $\subseteq \{(T, S). \text{dpll-bj } S T \wedge \text{atms-of-mm} (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A$
 $\wedge \text{atm-of ' lits-of-l} (\text{trail } S) \subseteq \text{atms-of-ms } A \wedge \text{no-dup} (\text{trail } S) \wedge \text{inv } S\}^+ \rangle$
(is $\langle ?A \subseteq ?B^+ \rangle$
 $\langle \text{proof} \rangle$

lemma *wf-tranclp-dpll-bj*:

assumes *fin*: $\langle \text{finite } A \rangle$
shows $\langle \text{wf } \{(T, S). \text{dpll-bj}^{++} S T$
 $\wedge \text{atms-of-mm} (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A \wedge \text{atm-of ' lits-of-l} (\text{trail } S) \subseteq \text{atms-of-ms } A$
 $\wedge \text{no-dup} (\text{trail } S) \wedge \text{inv } S\}$
 $\langle \text{proof} \rangle$

lemma *dpll-bj-sat-ext-iff*:

$\langle \text{dpll-bj } S T \implies \text{inv } S \implies I \models_{\text{sextm}} \text{clauses}_{NOT} S \iff I \models_{\text{sextm}} \text{clauses}_{NOT} T \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-dpll-bj-sat-ext-iff*:

$\langle \text{dpll-bj}^{**} S T \implies \text{inv } S \implies I \models_{\text{sextm}} \text{clauses}_{NOT} S \iff I \models_{\text{sextm}} \text{clauses}_{NOT} T \rangle$
 $\langle \text{proof} \rangle$

theorem *full-dpll-backjump-final-state*:

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

corollary *full-dpll-backjump-final-state-from-init-state*:

fixes *A* :: $\langle 'v \text{ clause set} \rangle$ **and** *S T* :: $\langle 'st \rangle$
assumes
full: $\langle \text{full dpll-bj } S T \rangle$ **and**
 $\langle \text{trail } S = [] \rangle$ **and**
 $\langle \text{clauses}_{NOT} S = N \rangle$ **and**
 $\langle \text{inv } S \rangle$
shows $\langle \text{unsatisfiable} (\text{set-mset } N) \vee (\text{trail } T \models_{\text{asm}} N \wedge \text{satisfiable} (\text{set-mset } N)) \rangle$
 $\langle \text{proof} \rangle$

lemma *tranclp-dpll-bj-trail-mes-decreasing-prop*:

assumes *dpll*: $\langle \text{dpll-bj}^{++} S T \rangle$ **and** *inv*: $\langle \text{inv } S \rangle$ **and**
N-A: $\langle \text{atms-of-mm} (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A \rangle$ **and**
M-A: $\langle \text{atm-of ' lits-of-l} (\text{trail } S) \subseteq \text{atms-of-ms } A \rangle$ **and**
n-d: $\langle \text{no-dup} (\text{trail } S) \rangle$ **and**
fin-A: $\langle \text{finite } A \rangle$
shows $\langle (2 + \text{card} (\text{atms-of-ms } A)) \wedge (1 + \text{card} (\text{atms-of-ms } A)) \rangle$

$$\begin{aligned}
& - \mu_C (1 + \text{card} (\text{atms-of-ms } A)) (2 + \text{card} (\text{atms-of-ms } A)) (\text{trail-weight } T) \\
& < (2 + \text{card} (\text{atms-of-ms } A)) \wedge (1 + \text{card} (\text{atms-of-ms } A)) \\
& - \mu_C (1 + \text{card} (\text{atms-of-ms } A)) (2 + \text{card} (\text{atms-of-ms } A)) (\text{trail-weight } S) \rangle \\
\langle \text{proof} \rangle
\end{aligned}$$

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

2.2.4 CDCL

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

Learn and Forget

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

locale *learn-ops* =
dpll-state *trail* *clauses*_{NOT} *prepend-trail* *tl-trail* *add-cl*_{NOT} *remove-cl*_{NOT}
for
trail :: $\langle 'st \Rightarrow ('v, \text{unit}) \text{ ann-lits} \rangle$ **and**
*clauses*_{NOT} :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**
prepend-trail :: $\langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
tl-trail :: $\langle 'st \Rightarrow 'st \rangle$ **and**
*add-cl*_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
*remove-cl*_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle +$
fixes
learn-conds :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow \text{bool} \rangle$
begin
inductive *learn* :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **where**
*learn*_{NOT}-rule: $\langle \text{clauses}_{\text{NOT}} S \models_{\text{pm}} C \Rightarrow$
 $\text{atms-of } C \subseteq \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \cup \text{atm-of } ' (\text{lits-of-l } (\text{trail } S)) \Rightarrow$
 $\text{learn-conds } C S \Rightarrow$
 $T \sim \text{add-cl}_{\text{NOT}} C S \Rightarrow$
 $\text{learn } S T \rangle$
inductive-cases *learn*_{NOT}E: $\langle \text{learn } S T \rangle$
lemma *learn- μ_C -stable*:
assumes $\langle \text{learn } S T \rangle$ **and** $\langle \text{no-dup } (\text{trail } S) \rangle$
shows $\langle \mu_C A B (\text{trail-weight } S) = \mu_C A B (\text{trail-weight } T) \rangle$
 $\langle \text{proof} \rangle$
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-cl*_{NOT} *remove-cl*_{NOT}
for
trail :: $\langle 'st \Rightarrow ('v, \text{unit}) \text{ ann-lits} \rangle$ **and**
*clauses*_{NOT} :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**
prepend-trail :: $\langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
tl-trail :: $\langle 'st \Rightarrow 'st \rangle$ **and**
*add-cl*_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
*remove-cl*_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle +$
fixes

forget-conds :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow \text{bool} \rangle$

begin

inductive *forget_{NOT}* :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **where**

forget_{NOT}:

$\langle \text{removeAll-mset } C(\text{clauses}_{\text{NOT}} S) \models_{\text{pm}} C \Rightarrow$

forget-conds $C S \Rightarrow$

$C \in \# \text{ clauses}_{\text{NOT}} S \Rightarrow$

$T \sim \text{remove-cls}_{\text{NOT}} C S \Rightarrow$

forget_{NOT} $S T \rangle$

inductive-cases *forget_{NOT}E*: $\langle \text{forget}_{\text{NOT}} S T \rangle$

lemma *forget- μ_C -stable*:

assumes $\langle \text{forget}_{\text{NOT}} S T \rangle$

shows $\langle \mu_C A B (\text{trail-weight } S) = \mu_C A B (\text{trail-weight } T) \rangle$

$\langle \text{proof} \rangle$

end

locale *learn-and-forget_{NOT}* =

learn-ops *trail* *clauses_{NOT}* *prepend-trail* *tl-trail* *add-cls_{NOT}* *remove-cls_{NOT}* *learn-conds* +

forget-ops *trail* *clauses_{NOT}* *prepend-trail* *tl-trail* *add-cls_{NOT}* *remove-cls_{NOT}* *forget-conds*

for

trail :: $\langle 'st \Rightarrow ('v, \text{unit}) \text{ ann-lits} \rangle$ **and**

clauses_{NOT} :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**

prepend-trail :: $\langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

tl-trail :: $\langle 'st \Rightarrow 'st \rangle$ **and**

add-cls_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

remove-cls_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

learn-conds *forget-conds* :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow \text{bool} \rangle$

begin

inductive *learn-and-forget_{NOT}* :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$

where

lf-learn: $\langle \text{learn } S T \Rightarrow \text{learn-and-forget}_{\text{NOT}} S T \rangle$ |

lf-forget: $\langle \text{forget}_{\text{NOT}} S T \Rightarrow \text{learn-and-forget}_{\text{NOT}} S T \rangle$

end

Definition of CDCL

locale *conflict-driven-clause-learning-ops* =

dpll-with-backjumping-ops *trail* *clauses_{NOT}* *prepend-trail* *tl-trail* *add-cls_{NOT}* *remove-cls_{NOT}*

inv *decide-conds* *backjump-conds* *propagate-conds* +

learn-and-forget_{NOT} *trail* *clauses_{NOT}* *prepend-trail* *tl-trail* *add-cls_{NOT}* *remove-cls_{NOT}* *learn-conds*

forget-conds

for

trail :: $\langle 'st \Rightarrow ('v, \text{unit}) \text{ ann-lits} \rangle$ **and**

clauses_{NOT} :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**

prepend-trail :: $\langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

tl-trail :: $\langle 'st \Rightarrow 'st \rangle$ **and**

add-cls_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

remove-cls_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

inv :: $\langle 'st \Rightarrow \text{bool} \rangle$ **and**

decide-conds :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **and**

backjump-conds :: $\langle 'v \text{ clause} \Rightarrow 'v \text{ clause} \Rightarrow 'v \text{ literal} \Rightarrow 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **and**

propagate-conds :: $\langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **and**

learn-conds *forget-conds* :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow \text{bool} \rangle$

begin

inductive $cdcl_{NOT} :: \langle 'st \Rightarrow 'st \Rightarrow bool \rangle$ **for** $S :: 'st$ **where**

$c\text{-dpll-bj}: \langle dpll\text{-bj } S S' \Longrightarrow cdcl_{NOT} S S' \rangle |$

$c\text{-learn}: \langle learn S S' \Longrightarrow cdcl_{NOT} S S' \rangle |$

$c\text{-forget}_{NOT}: \langle forget_{NOT} S S' \Longrightarrow cdcl_{NOT} S S' \rangle$

lemma $cdcl_{NOT}\text{-all-induct}[consumes 1, case-names dpll-bj learn forget_{NOT}]$:

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:

$\langle \bigwedge C T. clauses_{NOT} S \models_{pm} C \Longrightarrow$

$atms\text{-of } C \subseteq atms\text{-of-mm } (clauses_{NOT} S) \cup atm\text{-of } ' (lits\text{-of-l } (trail S)) \Longrightarrow$

$T \sim add\text{-cls}_{NOT} C S \Longrightarrow$

$P S T \rangle$ **and**

forgetting: $\langle \bigwedge C T. removeAll\text{-mset } C (clauses_{NOT} S) \models_{pm} C \Longrightarrow$

$C \in \# clauses_{NOT} S \Longrightarrow$

$T \sim remove\text{-cls}_{NOT} C S \Longrightarrow$

$P S T \rangle$

shows $\langle P S T \rangle$

$\langle proof \rangle$

lemma $cdcl_{NOT}\text{-no-dup}$:

assumes

$\langle cdcl_{NOT} S T \rangle$ **and**

$\langle inv S \rangle$ **and**

$\langle no\text{-dup } (trail S) \rangle$

shows $\langle no\text{-dup } (trail T) \rangle$

$\langle proof \rangle$

Consistency of the trail lemma $cdcl_{NOT}\text{-consistent}$:

assumes

$\langle cdcl_{NOT} S T \rangle$ **and**

$\langle inv S \rangle$ **and**

$\langle no\text{-dup } (trail S) \rangle$

shows $\langle consistent\text{-interp } (lits\text{-of-l } (trail T)) \rangle$

$\langle proof \rangle$

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

lemma $cdcl_{NOT}\text{-atms-of-ms-clauses-decreasing}$:

assumes $\langle cdcl_{NOT} S T \rangle$ **and** $\langle inv S \rangle$

shows $\langle atms\text{-of-mm } (clauses_{NOT} T) \subseteq atms\text{-of-mm } (clauses_{NOT} S) \cup atm\text{-of } ' (lits\text{-of-l } (trail S)) \rangle$

$\langle proof \rangle$

lemma $cdcl_{NOT}\text{-atms-in-trail}$:

assumes $\langle cdcl_{NOT} S T \rangle$ **and** $\langle inv S \rangle$

and $\langle atm\text{-of } ' (lits\text{-of-l } (trail S)) \subseteq atms\text{-of-mm } (clauses_{NOT} S) \rangle$

shows $\langle atm\text{-of } ' (lits\text{-of-l } (trail T)) \subseteq atms\text{-of-mm } (clauses_{NOT} S) \rangle$

$\langle proof \rangle$

lemma $cdcl_{NOT}\text{-atms-in-trail-in-set}$:

assumes

$\langle cdcl_{NOT} S T \rangle$ **and** $\langle inv S \rangle$ **and**
 $\langle atms-of-mm (clauses_{NOT} S) \subseteq A \rangle$ **and**
 $\langle atm-of '(lits-of-l (trail S)) \subseteq A \rangle$
shows $\langle atm-of '(lits-of-l (trail T)) \subseteq A \rangle$
 $\langle proof \rangle$

lemma *cdcl_{NOT}-all-decomposition-implies*:
assumes $\langle cdcl_{NOT} S T \rangle$ **and** $\langle inv S \rangle$ **and**
 $\langle all-decomposition-implies-m (clauses_{NOT} S) (get-all-ann-decomposition (trail S)) \rangle$
shows
 $\langle all-decomposition-implies-m (clauses_{NOT} T) (get-all-ann-decomposition (trail T)) \rangle$
 $\langle proof \rangle$

Extension of models lemma *cdcl_{NOT}-bj-sat-ext-iff*:
assumes $\langle cdcl_{NOT} S T \rangle$ **and** $\langle inv S \rangle$
shows $\langle I \models sextm clauses_{NOT} S \longleftrightarrow I \models sextm clauses_{NOT} T \rangle$
 $\langle proof \rangle$

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*: $\langle \bigwedge S T. cdcl_{NOT} S T \implies inv S \implies inv T \rangle$
begin
sublocale *dpll-with-backjumping*
 $\langle proof \rangle$

lemma *rtranclp-cdcl_{NOT}-inv*:
 $\langle cdcl_{NOT}^{**} S T \implies inv S \implies inv T \rangle$
 $\langle proof \rangle$

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

lemma *rtranclp-cdcl_{NOT}-trail-clauses-bound*:
assumes
cdcl: $\langle cdcl_{NOT}^{**} S T \rangle$ **and**
inv: $\langle inv S \rangle$ **and**
atms-clauses-S: $\langle atms-of-mm (clauses_{NOT} S) \subseteq A \rangle$ **and**
atms-trail-S: $\langle atm-of '(lits-of-l (trail S)) \subseteq A \rangle$
shows $\langle atm-of '(lits-of-l (trail T)) \subseteq A \wedge atms-of-mm (clauses_{NOT} T) \subseteq A \rangle$
 $\langle proof \rangle$

lemma *rtranclp-cdcl_{NOT}-all-decomposition-implies*:
assumes $\langle cdcl_{NOT}^{**} S T \rangle$ **and** $\langle inv S \rangle$ **and** $\langle no-dup (trail S) \rangle$ **and**
 $\langle all-decomposition-implies-m (clauses_{NOT} S) (get-all-ann-decomposition (trail S)) \rangle$
shows
 $\langle all-decomposition-implies-m (clauses_{NOT} T) (get-all-ann-decomposition (trail T)) \rangle$
 $\langle proof \rangle$

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

assumes $\langle cdcl_{NOT}^{**} S T \rangle$ **and** $\langle inv S \rangle$
shows $\langle I \models_{sextm} clauses_{NOT} S \longleftrightarrow I \models_{sextm} clauses_{NOT} T \rangle$
 $\langle proof \rangle$

definition $cdcl_{NOT-NOT-all-inv}$ **where**

$\langle cdcl_{NOT-NOT-all-inv} A S \longleftrightarrow (finite A \wedge inv S \wedge atms-of-mm (clauses_{NOT} S) \subseteq atms-of-ms A$
 $\wedge atm-of \text{ ' lits-of-l } (trail S) \subseteq atms-of-ms A \wedge no-dup (trail S)) \rangle$

lemma $cdcl_{NOT-NOT-all-inv}$:

assumes $\langle cdcl_{NOT}^{**} S T \rangle$ **and** $\langle cdcl_{NOT-NOT-all-inv} A S \rangle$
shows $\langle cdcl_{NOT-NOT-all-inv} A T \rangle$
 $\langle proof \rangle$

abbreviation $learn-or-forget$ **where**

$\langle learn-or-forget S T \equiv learn S T \vee forget_{NOT} S T \rangle$

lemma $rtranclp-learn-or-forget-cdcl_{NOT}$:

$\langle learn-or-forget^{**} S T \implies cdcl_{NOT}^{**} S T \rangle$
 $\langle proof \rangle$

lemma $learn-or-forget-dpll-\mu_C$:

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

lemma $infinite-cdcl_{NOT}-exists-learn-and-forget-infinite-chain$:

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

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

assumes
 $no-infinite-lf$: $\langle \bigwedge f j. \neg (\forall i \geq j. learn-or-forget (f i) (f (Suc i))) \rangle$
shows $\langle wf \{(T, S). cdcl_{NOT} S T \wedge cdcl_{NOT-NOT-all-inv} A S\} \rangle$
(is $\langle wf \{(T, S). cdcl_{NOT} S T \wedge ?inv S\} \rangle$)
 $\langle proof \rangle$

lemma $inv-and-tranclp-cdcl_{NOT}-tranclp-cdcl_{NOT}-and-inv$:

$\langle cdcl_{NOT}^{++} S T \wedge cdcl_{NOT-NOT-all-inv} A S \longleftrightarrow (\lambda S T. cdcl_{NOT} S T \wedge cdcl_{NOT-NOT-all-inv} A$
 $S)^{++} S T \rangle$
(is $\langle ?A \wedge ?I \longleftrightarrow ?B \rangle$)
 $\langle proof \rangle$

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

assumes
 $no-infinite-lf$: $\langle \bigwedge f j. \neg (\forall i \geq j. learn-or-forget (f i) (f (Suc i))) \rangle$

shows $\langle wf \{(T, S). cdcl_{NOT}^{++} S T \wedge cdcl_{NOT-NOT-all-inv} A S\} \rangle$
 $\langle proof \rangle$

lemma *cdcl_{NOT}-final-state*:

assumes

n-s: $\langle no-step \ cdcl_{NOT} S \rangle$ **and**

inv: $\langle cdcl_{NOT-NOT-all-inv} A S \rangle$ **and**

decomp: $\langle all-decomposition-implies-m \ (clauses_{NOT} S) \ (get-all-ann-decomposition \ (trail \ S)) \rangle$

shows $\langle unsatisfiable \ (set-mset \ (clauses_{NOT} S)) \rangle$

$\vee \ (trail \ S \models_{asm} \ clauses_{NOT} \ S \wedge \ satisfiable \ (set-mset \ (clauses_{NOT} S))) \rangle$

$\langle proof \rangle$

lemma *full-cdcl_{NOT}-final-state*:

assumes

full: $\langle full \ cdcl_{NOT} S T \rangle$ **and**

inv: $\langle cdcl_{NOT-NOT-all-inv} A S \rangle$ **and**

n-d: $\langle no-dup \ (trail \ S) \rangle$ **and**

decomp: $\langle all-decomposition-implies-m \ (clauses_{NOT} S) \ (get-all-ann-decomposition \ (trail \ S)) \rangle$

shows $\langle unsatisfiable \ (set-mset \ (clauses_{NOT} T)) \rangle$

$\vee \ (trail \ T \models_{asm} \ clauses_{NOT} \ T \wedge \ satisfiable \ (set-mset \ (clauses_{NOT} T))) \rangle$

$\langle proof \rangle$

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

Termination

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

Restricting learn and forget

locale *conflict-driven-clause-learning-learning-before-backjump-only-distinct-learnt* =

dpll-state trail clauses_{NOT} prepend-trail tl-trail add-cl_{NOT} remove-cl_{NOT} +
conflict-driven-clause-learning trail clauses_{NOT} prepend-trail tl-trail add-cl_{NOT} remove-cl_{NOT}
inv decide-conds backjump-conds propagate-conds

$\langle \lambda C S. \ distinct-mset \ C \wedge \neg tautology \ C \wedge \ learn-restrictions \ C \ S \wedge$

$(\exists F K d F' C' L. \ trail \ S = F' @ Decided \ K \ \# \ F \wedge \ C = add-mset \ L \ C' \wedge \ F \models_{as} \ CNot \ C'$
 $\wedge \ add-mset \ L \ C' \notin \# \ clauses_{NOT} \ S) \rangle$

$\langle \lambda C S. \ \neg(\exists F' F K d L. \ trail \ S = F' @ Decided \ K \ \# \ F \wedge \ F \models_{as} \ CNot \ (remove1-mset \ L \ C))$
 $\wedge \ forget-restrictions \ C \ S \rangle$

for

trail :: $\langle 'st \Rightarrow ('v, unit) \ ann-lits \rangle$ **and**

clauses_{NOT} :: $\langle 'st \Rightarrow 'v \ clauses \rangle$ **and**

prepend-trail :: $\langle ('v, unit) \ ann-lit \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

tl-trail :: $\langle 'st \Rightarrow 'st \rangle$ **and**

add-cl_{NOT} :: $\langle 'v \ clause \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

remove-cl_{NOT} :: $\langle 'v \ clause \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

inv :: $\langle 'st \Rightarrow bool \rangle$ **and**

decide-conds :: $\langle 'st \Rightarrow 'st \Rightarrow bool \rangle$ **and**

backjump-conds :: $\langle 'v \ clause \Rightarrow 'v \ clause \Rightarrow 'v \ literal \Rightarrow 'st \Rightarrow 'st \Rightarrow bool \rangle$ **and**

propagate-conds :: $\langle ('v, unit) \ ann-lit \Rightarrow 'st \Rightarrow 'st \Rightarrow bool \rangle$ **and**

learn-restrictions forget-restrictions :: $\langle 'v \ clause \Rightarrow 'st \Rightarrow bool \rangle$

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**

$dpll: \langle \bigwedge T. dpll\text{-bj } S T \implies P S T \rangle$ **and**

learning:

$\langle \bigwedge C F K F' C' L T. clauses_{NOT} S \models_{pm} C \implies$
 $atms\text{-of } C \subseteq atms\text{-of}\text{-mm } (clauses_{NOT} S) \cup atm\text{-of } ' (lits\text{-of}\text{-l } (trail S)) \implies$
 $distinct\text{-mset } C \implies$

$\neg tautology C \implies$

$learn\text{-restrictions } C S \implies$

$trail S = F' @ Decided K \# F \implies$

$C = add\text{-mset } L C' \implies$

$F \models_{as} CNot C' \implies$

$add\text{-mset } L C' \notin \# clauses_{NOT} S \implies$

$T \sim add\text{-cls}_{NOT} C S \implies$

$P S T \rangle$ **and**

forgetting: $\langle \bigwedge C T. removeAll\text{-mset } C (clauses_{NOT} S) \models_{pm} C \implies$

$C \in \# clauses_{NOT} S \implies$

$\neg (\exists F' F K L. trail S = F' @ Decided K \# F \wedge F \models_{as} CNot (C - \{\#L\#})) \implies$

$T \sim remove\text{-cls}_{NOT} C S \implies$

$forget\text{-restrictions } C S \implies$

$P S T \rangle$

shows $\langle P S T \rangle$

$\langle proof \rangle$

lemma *rtranclp-cdcl_{NOT}-inv:*

$\langle cdcl_{NOT}^{**} S T \implies inv S \implies inv T \rangle$

$\langle proof \rangle$

lemma *learn-always-simple-clauses:*

assumes

learn: $\langle learn S T \rangle$ **and**

n-d: $\langle no\text{-dup } (trail S) \rangle$

shows $\langle set\text{-mset } (clauses_{NOT} T - clauses_{NOT} S)$

$\subseteq simple\text{-clss } (atms\text{-of}\text{-mm } (clauses_{NOT} S) \cup atm\text{-of } ' (lits\text{-of}\text{-l } (trail S))) \rangle$

$\langle proof \rangle$

definition $\langle conflicting\text{-bj-clss } S \equiv$

$\{C + \{\#L\#\} \mid C L. C + \{\#L\#\} \in \# clauses_{NOT} S \wedge distinct\text{-mset } (C + \{\#L\#\})$

$\wedge \neg tautology (C + \{\#L\#\})$

$\wedge (\exists F' K F. trail S = F' @ Decided K \# F \wedge F \models_{as} CNot C) \rangle$

lemma *conflicting-bj-clss-remove-cls_{NOT}[simp]:*

$\langle conflicting\text{-bj-clss } (remove\text{-cls}_{NOT} C S) = conflicting\text{-bj-clss } S - \{C\} \rangle$

$\langle proof \rangle$

lemma *conflicting-bj-clss-remove-cls_{NOT}'[simp]:*

$\langle T \sim remove\text{-cls}_{NOT} C S \implies conflicting\text{-bj-clss } T = conflicting\text{-bj-clss } S - \{C\} \rangle$

$\langle proof \rangle$

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 $\langle \text{conflicting-bj-clss } T$
 $= \text{conflicting-bj-clss } S$
 $\cup (\text{if } \exists C L. C' = \text{add-mset } L C \wedge \text{distinct-mset } (\text{add-mset } L C) \wedge \neg \text{tautology } (\text{add-mset } L C)$
 $\wedge (\exists F' K d F. \text{trail } S = F' @ \text{Decided } K \# F \wedge F \models_{\text{as}} C \text{Not } C)$
 $\text{then } \{C'\} \text{ else } \{\}) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{conflicting-bj-clss-add-clss}_{\text{NOT}}$:
 $\langle \text{no-dup } (\text{trail } S) \implies$
 $\text{conflicting-bj-clss } (\text{add-clss}_{\text{NOT}} C' S)$
 $= \text{conflicting-bj-clss } S$
 $\cup (\text{if } \exists C L. C' = C + \{\#L\# \} \wedge \text{distinct-mset } (C + \{\#L\# \}) \wedge \neg \text{tautology } (C + \{\#L\# \})$
 $\wedge (\exists F' K d F. \text{trail } S = F' @ \text{Decided } K \# F \wedge F \models_{\text{as}} C \text{Not } C)$
 $\text{then } \{C'\} \text{ else } \{\}) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{conflicting-bj-clss-incl-clauses}$:
 $\langle \text{conflicting-bj-clss } S \subseteq \text{set-mset } (\text{clauses}_{\text{NOT}} S) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{finite-conflicting-bj-clss}[\text{simp}]$:
 $\langle \text{finite } (\text{conflicting-bj-clss } S) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{learn-conflicting-increasing}$:
 $\langle \text{no-dup } (\text{trail } S) \implies \text{learn } S T \implies \text{conflicting-bj-clss } S \subseteq \text{conflicting-bj-clss } T \rangle$
 $\langle \text{proof} \rangle$

abbreviation $\langle \text{conflicting-bj-clss-yet } b S \equiv$
 $\exists \wedge b - \text{card } (\text{conflicting-bj-clss } S) \rangle$

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

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

lemma $\text{forget-}\mu_L\text{-decrease}$:
assumes $\text{forget}_{\text{NOT}}: \langle \text{forget}_{\text{NOT}} S T \rangle$
shows $\langle (\mu_L b T, \mu_L b S) \in \text{less-than } \langle * \text{lex} * \rangle \text{ less-than} \rangle$
 $\langle \text{proof} \rangle$

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

lemma set-insert-neq :
 $\langle A \neq \text{insert } a A \longleftrightarrow a \notin A \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{learn-}\mu_L\text{-decrease}$:
assumes $\text{learnST}: \langle \text{learn } S T \rangle$ **and** $n\text{-d}: \langle \text{no-dup } (\text{trail } S) \rangle$ **and**
 $A: \langle \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \cup \text{atm-of } ' \text{lits-of-l } (\text{trail } S) \subseteq A \rangle$ **and**
 $\text{fin-A}: \langle \text{finite } A \rangle$

shows $\langle \mu_L (\text{card } A) T, \mu_L (\text{card } A) S \rangle \in \text{less-than } \langle *lex* \rangle \text{ less-than}$
 $\langle \text{proof} \rangle$

We have to assume the following:

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

definition μ_{CDCL} **where**

$\langle \mu_{CDCL} A T \equiv ((2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A))$
 $\quad - \mu_C (1 + \text{card } (\text{atms-of-ms } A)) (2 + \text{card } (\text{atms-of-ms } A)) (\text{trail-weight } T),$
 $\quad \text{conflicting-bj-clss-yet } (\text{card } (\text{atms-of-ms } A)) T, \text{card } (\text{set-mset } (\text{clauses}_{NOT} T))) \rangle$

lemma *cdcl_{NOT}-decreasing-measure*:

assumes

$\langle \text{cdcl}_{NOT} S T \rangle$ **and**

inv: $\langle \text{inv } S \rangle$ **and**

atm-clss: $\langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A \rangle$ **and**

atm-lits: $\langle \text{atm-of } \text{‘} \textit{lits-of-l (trail S)} \subseteq \text{atms-of-ms } A \rangle$ **and**

n-d: $\langle \text{no-dup } (\text{trail } S) \rangle$ **and**

fin-A: $\langle \text{finite } A \rangle$

shows $\langle \mu_{CDCL} A T, \mu_{CDCL} A S \rangle$

$\in \text{less-than } \langle *lex* \rangle (\text{less-than } \langle *lex* \rangle \text{ less-than})$

$\langle \text{proof} \rangle$

lemma *wf-cdcl_{NOT}-restricted-learning*:

assumes $\langle \text{finite } A \rangle$

shows $\langle \text{wf } \{(T, S).$

$(\text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A \wedge \text{atm-of } \text{‘} \textit{lits-of-l (trail S)} \subseteq \text{atms-of-ms } A$

$\wedge \text{no-dup } (\text{trail } S)$

$\wedge \text{inv } S)$

$\wedge \text{cdcl}_{NOT} S T \}$

$\langle \text{proof} \rangle$

definition μ_C' :: $\langle \text{‘}v \text{ clause set} \Rightarrow \text{‘}st \Rightarrow \text{nat} \rangle$ **where**

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

definition μ_{CDCL}' :: $\langle \text{‘}v \text{ clause set} \Rightarrow \text{‘}st \Rightarrow \text{nat} \rangle$ **where**

$\langle \mu_{CDCL}' A T \equiv$

$((2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A)) - \mu_C' A T) * (1 + 3^{\text{card } (\text{atms-of-ms } A)}) * 2$

$+ \text{conflicting-bj-clss-yet } (\text{card } (\text{atms-of-ms } A)) T * 2$

$+ \text{card } (\text{set-mset } (\text{clauses}_{NOT} T)) \rangle$

lemma *cdcl_{NOT}-decreasing-measure'*:

assumes

$\langle \text{cdcl}_{NOT} S T \rangle$ **and**

inv: $\langle \text{inv } S \rangle$ **and**

atms-clss: $\langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A \rangle$ **and**

atms-trail: $\langle \text{atm-of } \text{‘} \textit{lits-of-l (trail S)} \subseteq \text{atms-of-ms } A \rangle$ **and**

n-d: $\langle \text{no-dup } (\text{trail } S) \rangle$ **and**

fin-A: $\langle \text{finite } A \rangle$

shows $\langle \mu_{CDCL}' A T < \mu_{CDCL}' A S \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl_{NOT}-clauses-bound*:

assumes

$\langle \text{cdcl}_{NOT} S T \rangle$ **and**
 $\langle \text{inv } S \rangle$ **and**
 $\langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq A \rangle$ **and**
 $\langle \text{atm-of } (\text{lits-of-l } (\text{trail } S)) \subseteq A \rangle$ **and**
 $n\text{-d: } \langle \text{no-dup } (\text{trail } S) \rangle$ **and**
 $\text{fin-A[simp]: } \langle \text{finite } A \rangle$

shows $\langle \text{set-mset } (\text{clauses}_{NOT} T) \subseteq \text{set-mset } (\text{clauses}_{NOT} S) \cup \text{simple-cls } A \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-cdcl_{NOT}-clauses-bound*:

assumes

$\langle \text{cdcl}_{NOT}^{**} S T \rangle$ **and**
 $\langle \text{inv } S \rangle$ **and**
 $\langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq A \rangle$ **and**
 $\langle \text{atm-of } (\text{lits-of-l } (\text{trail } S)) \subseteq A \rangle$ **and**
 $n\text{-d: } \langle \text{no-dup } (\text{trail } S) \rangle$ **and**
 $\text{finite: } \langle \text{finite } A \rangle$

shows $\langle \text{set-mset } (\text{clauses}_{NOT} T) \subseteq \text{set-mset } (\text{clauses}_{NOT} S) \cup \text{simple-cls } A \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-cdcl_{NOT}-card-clauses-bound*:

assumes

$\langle \text{cdcl}_{NOT}^{**} S T \rangle$ **and**
 $\langle \text{inv } S \rangle$ **and**
 $\langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq A \rangle$ **and**
 $\langle \text{atm-of } (\text{lits-of-l } (\text{trail } S)) \subseteq A \rangle$ **and**
 $n\text{-d: } \langle \text{no-dup } (\text{trail } S) \rangle$ **and**
 $\text{finite: } \langle \text{finite } A \rangle$

shows $\langle \text{card } (\text{set-mset } (\text{clauses}_{NOT} T)) \leq \text{card } (\text{set-mset } (\text{clauses}_{NOT} S)) + 3 \wedge (\text{card } A) \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-cdcl_{NOT}-card-clauses-bound'*:

assumes

$\langle \text{cdcl}_{NOT}^{**} S T \rangle$ **and**
 $\langle \text{inv } S \rangle$ **and**
 $\langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq A \rangle$ **and**
 $\langle \text{atm-of } (\text{lits-of-l } (\text{trail } S)) \subseteq A \rangle$ **and**
 $n\text{-d: } \langle \text{no-dup } (\text{trail } S) \rangle$ **and**
 $\text{finite: } \langle \text{finite } A \rangle$

shows $\langle \text{card } \{C \mid C \in \# \text{clauses}_{NOT} T \wedge (\text{tautology } C \vee \neg \text{distinct-mset } C)\} \leq \text{card } \{C \mid C \in \# \text{clauses}_{NOT} S \wedge (\text{tautology } C \vee \neg \text{distinct-mset } C)\} + 3 \wedge (\text{card } A) \rangle$
 $(\text{is } \langle \text{card } ?T \leq \text{card } ?S + \rightarrow \rangle)$
 $\langle \text{proof} \rangle$

lemma *rtranclp-cdcl_{NOT}-card-simple-clauses-bound*:

assumes

$\langle \text{cdcl}_{NOT}^{**} S T \rangle$ **and**
 $\langle \text{inv } S \rangle$ **and**
 $NA: \langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq A \rangle$ **and**
 $MA: \langle \text{atm-of } (\text{lits-of-l } (\text{trail } S)) \subseteq A \rangle$ **and**
 $n\text{-d: } \langle \text{no-dup } (\text{trail } S) \rangle$ **and**

finite: $\langle \text{finite } A \rangle$
shows $\langle \text{card } (\text{set-mset } (\text{clauses}_{NOT} T))$
 $\leq \text{card } \{C. C \in \# \text{ clauses}_{NOT} S \wedge (\text{tautology } C \vee \neg \text{distinct-mset } C)\} + 3 \wedge (\text{card } A) \rangle$
(is $\langle \text{card } ?T \leq \text{card } ?S + \cdot \rangle$
proof)

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

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

lemma $r\text{tranclp-cdcl}_{NOT}\text{-}\mu_{CDCL}'\text{-bound-reduce-trail-to}_{NOT}$:

assumes
 $\langle \text{cdcl}_{NOT}^{**} S T \rangle$ **and**
 $\langle \text{inv } S \rangle$ **and**
 $\langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $\langle \text{atm-of } (\text{lits-of-l } (\text{trail } S)) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $n\text{-d}: \langle \text{no-dup } (\text{trail } S) \rangle$ **and**
finite: $\langle \text{finite } (\text{atms-of-ms } A) \rangle$ **and**
 $U: \langle U \sim \text{reduce-trail-to}_{NOT} M T \rangle$
shows $\langle \mu_{CDCL}' A U \leq \mu_{CDCL}'\text{-bound } A S \rangle$
proof)

lemma $r\text{tranclp-cdcl}_{NOT}\text{-}\mu_{CDCL}'\text{-bound}$:

assumes
 $\langle \text{cdcl}_{NOT}^{**} S T \rangle$ **and**
 $\langle \text{inv } S \rangle$ **and**
 $\langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $\langle \text{atm-of } (\text{lits-of-l } (\text{trail } S)) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $n\text{-d}: \langle \text{no-dup } (\text{trail } S) \rangle$ **and**
finite: $\langle \text{finite } (\text{atms-of-ms } A) \rangle$
shows $\langle \mu_{CDCL}' A T \leq \mu_{CDCL}'\text{-bound } A S \rangle$
proof)

lemma $r\text{tranclp-}\mu_{CDCL}'\text{-bound-decreasing}$:

assumes
 $\langle \text{cdcl}_{NOT}^{**} S T \rangle$ **and**
 $\langle \text{inv } S \rangle$ **and**
 $\langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $\langle \text{atm-of } (\text{lits-of-l } (\text{trail } S)) \subseteq \text{atms-of-ms } A \rangle$ **and**
 $n\text{-d}: \langle \text{no-dup } (\text{trail } S) \rangle$ **and**
finite[*simp*]: $\langle \text{finite } (\text{atms-of-ms } A) \rangle$
shows $\langle \mu_{CDCL}'\text{-bound } A T \leq \mu_{CDCL}'\text{-bound } A S \rangle$
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

$cdcl_{NOT} :: \langle 'st \Rightarrow 'st \Rightarrow bool \rangle$ **and**
 $restart :: \langle 'st \Rightarrow 'st \Rightarrow bool \rangle$

begin

inductive *cdcl_{NOT}-raw-restart* :: $\langle 'st \Rightarrow 'st \Rightarrow bool \rangle$ **where**

$\langle cdcl_{NOT} S T \Longrightarrow cdcl_{NOT}\text{-raw-restart } S T \rangle$ |
 $\langle restart S T \Longrightarrow cdcl_{NOT}\text{-raw-restart } S T \rangle$

end

locale *conflict-driven-clause-learning-with-restarts* =

conflict-driven-clause-learning *trail* *clauses_{NOT}* *prepend-trail* *tl-trail* *add-cl_{NOT}* *remove-cl_{NOT}*
inv *decide-conds* *backjump-conds* *propagate-conds* *learn-conds* *forget-conds*

for

$trail :: \langle 'st \Rightarrow ('v, unit) \text{ ann-lits} \rangle$ **and**
 $clauses_{NOT} :: \langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**
 $prepend-trail :: \langle ('v, unit) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
 $tl-trail :: \langle 'st \Rightarrow 'st \rangle$ **and**
 $add-cl_{NOT} :: \langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
 $remove-cl_{NOT} :: \langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
 $inv :: \langle 'st \Rightarrow bool \rangle$ **and**
 $decide-conds :: \langle 'st \Rightarrow 'st \Rightarrow bool \rangle$ **and**
 $backjump-conds :: \langle 'v \text{ clause} \Rightarrow 'v \text{ clause} \Rightarrow 'v \text{ literal} \Rightarrow 'st \Rightarrow 'st \Rightarrow bool \rangle$ **and**
 $propagate-conds :: \langle ('v, unit) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \Rightarrow bool \rangle$ **and**
 $learn-conds \text{ forget-conds} :: \langle 'v \text{ clause} \Rightarrow 'st \Rightarrow bool \rangle$

begin

lemma *cdcl_{NOT}-iff-cdcl_{NOT}-raw-restart-no-restarts*:

$\langle cdcl_{NOT} S T \longleftrightarrow restart\text{-ops}.cdcl_{NOT}\text{-raw-restart } cdcl_{NOT} (\lambda - . False) S T \rangle$
(is $\langle ?C S T \longleftrightarrow ?R S T \rangle$)

<proof>

lemma *cdcl_{NOT}-cdcl_{NOT}-raw-restart*:

$\langle cdcl_{NOT} S T \Longrightarrow restart\text{-ops}.cdcl_{NOT}\text{-raw-restart } cdcl_{NOT} restart S T \rangle$

<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 assumptions to prove termination:

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

- we also assume that the measure decrease after any $cdcl_{NOT}$ step.
- an invariant on the states $cdcl_{NOT}$ -inv that also holds after restarts.
- it is *not required* that the measure decrease with respect to restarts, but the measure has to be bound by some function 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 :: $\langle 'st \Rightarrow 'st \Rightarrow bool \rangle$ **and**
 $cdcl_{NOT}$:: $\langle 'st \Rightarrow 'st \Rightarrow bool \rangle$ +
fixes
 f :: $\langle nat \Rightarrow nat \rangle$ **and**
 bound-inv :: $\langle 'bound \Rightarrow 'st \Rightarrow bool \rangle$ **and**
 μ :: $\langle 'bound \Rightarrow 'st \Rightarrow nat \rangle$ **and**
 $cdcl_{NOT}$ -inv :: $\langle 'st \Rightarrow bool \rangle$ **and**
 μ -bound :: $\langle 'bound \Rightarrow 'st \Rightarrow nat \rangle$
assumes
 f: $\langle unbounded\ f \rangle$ **and**
 f-ge-1: $\langle \bigwedge n. n \geq 1 \implies f\ n \neq 0 \rangle$ **and**
 bound-inv: $\langle \bigwedge A\ S\ T. cdcl_{NOT}$ -inv $S \implies bound$ -inv $A\ S \implies cdcl_{NOT}\ S\ T \implies bound$ -inv $A\ T \rangle$ **and**
 $cdcl_{NOT}$ -measure: $\langle \bigwedge A\ S\ T. cdcl_{NOT}$ -inv $S \implies bound$ -inv $A\ S \implies cdcl_{NOT}\ S\ T \implies \mu\ A\ T < \mu$
 $A\ S \rangle$ **and**
 measure-bound2: $\langle \bigwedge A\ T\ U. cdcl_{NOT}$ -inv $T \implies bound$ -inv $A\ T \implies cdcl_{NOT}^{**}\ T\ U$
 $\implies \mu\ A\ U \leq \mu$ -bound $A\ T \rangle$ **and**
 measure-bound4: $\langle \bigwedge A\ T\ U. cdcl_{NOT}$ -inv $T \implies bound$ -inv $A\ T \implies cdcl_{NOT}^{**}\ T\ U$
 $\implies \mu$ -bound $A\ U \leq \mu$ -bound $A\ T \rangle$ **and**
 $cdcl_{NOT}$ -restart-inv: $\langle \bigwedge A\ U\ V. cdcl_{NOT}$ -inv $U \implies restart\ U\ V \implies bound$ -inv $A\ U \implies bound$ -inv
 $A\ V \rangle$
and
 exists-bound: $\langle \bigwedge R\ S. cdcl_{NOT}$ -inv $R \implies restart\ R\ S \implies \exists A. bound$ -inv $A\ S \rangle$ **and**
 $cdcl_{NOT}$ -inv: $\langle \bigwedge S\ T. cdcl_{NOT}$ -inv $S \implies cdcl_{NOT}\ S\ T \implies cdcl_{NOT}$ -inv $T \rangle$ **and**
 $cdcl_{NOT}$ -inv-restart: $\langle \bigwedge S\ T. cdcl_{NOT}$ -inv $S \implies restart\ S\ T \implies cdcl_{NOT}$ -inv $T \rangle$

begin

lemma $cdcl_{NOT}$ - $cdcl_{NOT}$ -inv:

assumes
 $\langle (cdcl_{NOT} \widetilde{\widetilde{n}}) S\ T \rangle$ **and**
 $\langle cdcl_{NOT}$ -inv $S \rangle$
shows $\langle cdcl_{NOT}$ -inv $T \rangle$
 $\langle proof \rangle$

lemma $cdcl_{NOT}$ -bound-inv:

assumes
 $\langle (cdcl_{NOT} \widetilde{\widetilde{n}}) S\ T \rangle$ **and**
 $\langle cdcl_{NOT}$ -inv $S \rangle$
 $\langle bound$ -inv $A\ S \rangle$
shows $\langle bound$ -inv $A\ T \rangle$
 $\langle proof \rangle$

lemma $rtranclp$ - $cdcl_{NOT}$ - $cdcl_{NOT}$ -inv:

assumes
 $\langle cdcl_{NOT}^{**}\ S\ T \rangle$ **and**
 $\langle cdcl_{NOT}$ -inv $S \rangle$
shows $\langle cdcl_{NOT}$ -inv $T \rangle$

$\langle \text{proof} \rangle$

lemma *rtranclp-cdcl_{NOT}-bound-inv*:

assumes
 $\langle \text{cdcl}_{NOT}^{**} S T \rangle$ **and**
 $\langle \text{bound-inv } A S \rangle$ **and**
 $\langle \text{cdcl}_{NOT}\text{-inv } S \rangle$
shows $\langle \text{bound-inv } A T \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl_{NOT}-comp-n-le*:

assumes
 $\langle (\text{cdcl}_{NOT} \widetilde{\sim} (\text{Suc } n)) S T \rangle$ **and**
 $\langle \text{bound-inv } A S \rangle$
 $\langle \text{cdcl}_{NOT}\text{-inv } S \rangle$
shows $\langle \mu A T < \mu A S - n \rangle$
 $\langle \text{proof} \rangle$

lemma *wf-cdcl_{NOT}*:

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

lemma *rtranclp-cdcl_{NOT}-measure*:

assumes
 $\langle \text{cdcl}_{NOT}^{**} S T \rangle$ **and**
 $\langle \text{bound-inv } A S \rangle$ **and**
 $\langle \text{cdcl}_{NOT}\text{-inv } S \rangle$
shows $\langle \mu A T \leq \mu A S \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl_{NOT}-comp-bounded*:

assumes
 $\langle \text{bound-inv } A S \rangle$ **and** $\langle \text{cdcl}_{NOT}\text{-inv } S \rangle$ **and** $\langle m \geq 1 + \mu A S \rangle$
shows $\langle \neg(\text{cdcl}_{NOT} \widetilde{\sim} m) S T \rangle$
 $\langle \text{proof} \rangle$

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

inductive *cdcl_{NOT}-restart where*

restart-step: $\langle (\text{cdcl}_{NOT} \widetilde{\sim} m) S T \implies m \geq f n \implies \text{restart } T U$
 $\implies \text{cdcl}_{NOT}\text{-restart } (S, n) (U, \text{Suc } n) \rangle$ |

restart-full: $\langle \text{full1 } \text{cdcl}_{NOT} S T \implies \text{cdcl}_{NOT}\text{-restart } (S, n) (T, \text{Suc } n) \rangle$

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*:

$\langle \text{cdcl}_{NOT}\text{-restart } S T \implies \text{cdcl}_{NOT}\text{-raw-restart}^{**} (fst S) (fst T) \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl_{NOT}-with-restart-bound-inv*:

assumes
 $\langle \text{cdcl}_{NOT}\text{-restart } S T \rangle$ **and**
 $\langle \text{bound-inv } A (fst S) \rangle$ **and**
 $\langle \text{cdcl}_{NOT}\text{-inv } (fst S) \rangle$

shows $\langle \text{bound-inv } A \text{ (fst } T) \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl_{NOT}-with-restart-cdcl_{NOT}-inv:*

assumes
 $\langle \text{cdcl}_{NOT}\text{-restart } S \ T \rangle$ **and**
 $\langle \text{cdcl}_{NOT}\text{-inv (fst } S) \rangle$
shows $\langle \text{cdcl}_{NOT}\text{-inv (fst } T) \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-cdcl_{NOT}-with-restart-cdcl_{NOT}-inv:*

assumes
 $\langle \text{cdcl}_{NOT}\text{-restart}^{**} S \ T \rangle$ **and**
 $\langle \text{cdcl}_{NOT}\text{-inv (fst } S) \rangle$
shows $\langle \text{cdcl}_{NOT}\text{-inv (fst } T) \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-cdcl_{NOT}-with-restart-bound-inv:*

assumes
 $\langle \text{cdcl}_{NOT}\text{-restart}^{**} S \ T \rangle$ **and**
 $\langle \text{cdcl}_{NOT}\text{-inv (fst } S) \rangle$ **and**
 $\langle \text{bound-inv } A \text{ (fst } S) \rangle$
shows $\langle \text{bound-inv } A \text{ (fst } T) \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl_{NOT}-with-restart-increasing-number:*

$\langle \text{cdcl}_{NOT}\text{-restart } S \ T \implies \text{snd } T = 1 + \text{snd } S \rangle$
 $\langle \text{proof} \rangle$

end

locale *cdcl_{NOT}-increasing-restarts =*

*cdcl_{NOT}-increasing-restarts-ops restart cdcl_{NOT} f bound-inv μ cdcl_{NOT}-inv μ -bound +
dpll-state trail clauses_{NOT} prepend-trail tl-trail add-cl_{NOT} remove-cl_{NOT}*

for

trail :: $\langle 'st \Rightarrow ('v, \text{unit}) \text{ ann-lits} \rangle$ **and**
clauses_{NOT} :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**
prepend-trail :: $\langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
tl-trail :: $\langle 'st \Rightarrow 'st \rangle$ **and**
add-cl_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
remove-cl_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
f :: $\langle \text{nat} \Rightarrow \text{nat} \rangle$ **and**
restart :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **and**
bound-inv :: $\langle 'bound \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **and**
 μ :: $\langle 'bound \Rightarrow 'st \Rightarrow \text{nat} \rangle$ **and**
cdcl_{NOT} :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **and**
cdcl_{NOT}-inv :: $\langle 'st \Rightarrow \text{bool} \rangle$ **and**
 μ -bound :: $\langle 'bound \Rightarrow 'st \Rightarrow \text{nat} \rangle$ +

assumes

measure-bound: $\langle \bigwedge A \ T \ V \ n. \text{cdcl}_{NOT}\text{-inv } T \implies \text{bound-inv } A \ T$
 $\implies \text{cdcl}_{NOT}\text{-restart } (T, n) \ (V, \text{Suc } n) \implies \mu \ A \ V \leq \mu\text{-bound } A \ T \rangle$ **and**
cdcl_{NOT}-raw-restart- μ -bound:
 $\langle \text{cdcl}_{NOT}\text{-restart } (T, a) \ (V, b) \implies \text{cdcl}_{NOT}\text{-inv } T \implies \text{bound-inv } A \ T$
 $\implies \mu\text{-bound } A \ V \leq \mu\text{-bound } A \ T \rangle$

begin

lemma *rtranclp-cdcl_{NOT}-raw-restart- μ -bound:*

$\langle \text{cdcl}_{NOT}\text{-restart}^{**} (T, a) (V, b) \implies \text{cdcl}_{NOT}\text{-inv } T \implies \text{bound-inv } A T$
 $\implies \mu\text{-bound } A V \leq \mu\text{-bound } A T \rangle$
 <proof>

lemma *cdcl_{NOT}-raw-restart-measure-bound:*

$\langle \text{cdcl}_{NOT}\text{-restart} (T, a) (V, b) \implies \text{cdcl}_{NOT}\text{-inv } T \implies \text{bound-inv } A T$
 $\implies \mu A V \leq \mu\text{-bound } A T \rangle$
 <proof>

lemma *rtranclp-cdcl_{NOT}-raw-restart-measure-bound:*

$\langle \text{cdcl}_{NOT}\text{-restart}^{**} (T, a) (V, b) \implies \text{cdcl}_{NOT}\text{-inv } T \implies \text{bound-inv } A T$
 $\implies \mu A V \leq \mu\text{-bound } A T \rangle$
 <proof>

lemma *wf-cdcl_{NOT}-restart:*

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

lemma *cdcl_{NOT}-restart-steps-bigger-than-bound:*

assumes
 $\langle \text{cdcl}_{NOT}\text{-restart } S T \rangle$ **and**
 $\langle \text{bound-inv } A (fst S) \rangle$ **and**
 $\langle \text{cdcl}_{NOT}\text{-inv } (fst S) \rangle$ **and**
 $\langle f (snd S) > \mu\text{-bound } A (fst S) \rangle$
shows $\langle \text{full1 } \text{cdcl}_{NOT} (fst S) (fst T) \rangle$
 <proof>

lemma *rtranclp-cdcl_{NOT}-with-inv-inv-rtranclp-cdcl_{NOT}:*

assumes
 $inv: \langle \text{cdcl}_{NOT}\text{-inv } S \rangle$ **and**
 $binv: \langle \text{bound-inv } A S \rangle$
shows $\langle (\lambda S T. \text{cdcl}_{NOT} S T \wedge \text{cdcl}_{NOT}\text{-inv } S \wedge \text{bound-inv } A S)^{**} S T \longleftrightarrow \text{cdcl}_{NOT}^{**} S T \rangle$
 (is <?A^{**} S T \longleftrightarrow ?B^{**} S T>)
 <proof>

lemma *no-step-cdcl_{NOT}-restart-no-step-cdcl_{NOT}:*

assumes
 $n\text{-s}: \langle \text{no-step } \text{cdcl}_{NOT}\text{-restart } S \rangle$ **and**
 $inv: \langle \text{cdcl}_{NOT}\text{-inv } (fst S) \rangle$ **and**
 $binv: \langle \text{bound-inv } A (fst S) \rangle$
shows $\langle \text{no-step } \text{cdcl}_{NOT} (fst S) \rangle$
 <proof>

end

2.2.6 Merging backjump and learning

locale *cdcl_{NOT}-merge-bj-learn-ops =*

$decide\text{-ops } trail \text{ clauses}_{NOT} \text{ prepend-trail } tl\text{-trail } add\text{-cls}_{NOT} \text{ remove-cls}_{NOT} \text{ decide-conds } +$
 $forget\text{-ops } trail \text{ clauses}_{NOT} \text{ prepend-trail } tl\text{-trail } add\text{-cls}_{NOT} \text{ remove-cls}_{NOT} \text{ forget-conds } +$
 $propagate\text{-ops } trail \text{ clauses}_{NOT} \text{ prepend-trail } tl\text{-trail } add\text{-cls}_{NOT} \text{ remove-cls}_{NOT} \text{ propagate-conds}$
for

$trail :: \langle 'st \Rightarrow ('v, unit) \text{ ann-lits} \rangle$ **and**
 $\text{clauses}_{NOT} :: \langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**
 $\text{prepend-trail} :: \langle ('v, unit) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
 $tl\text{-trail} :: \langle 'st \Rightarrow 'st \rangle$ **and**

```

add-clsNOT :: ⟨'v clause ⇒ 'st ⇒ 'st⟩ and
remove-clsNOT :: ⟨'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 ~ prepend-trail (Propagated L ()) (reduce-trail-toNOT F (add-clsNOT C'' S))
  ⇒ C ∈# clausesNOT S
  ⇒ trail S ⊨as CNot C
  ⇒ undefined-lit F L
  ⇒ atm-of L ∈ atms-of-mm (clausesNOT S) ∪ atm-of ' (lits-of-l (trail S))
  ⇒ clausesNOT S ⊨pm add-mset L C'
  ⇒ C'' = add-mset L C'
  ⇒ F ⊨as CNot C'
  ⇒ backjump-l-cond C C' L S T
  ⇒ backjump-l S T⟩

```

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

```

declare reduce-trail-toNOT-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-toNOT-length-ne[simp] Set.Un-iff[simp] Set.insert-iff[simp]

```

```

inductive cdclNOT-merged-bj-learn :: ⟨'st ⇒ 'st ⇒ bool⟩ for S :: 'st where
cdclNOT-merged-bj-learn-decideNOT: ⟨decideNOT S S' ⇒ cdclNOT-merged-bj-learn S S'⟩ |
cdclNOT-merged-bj-learn-propagateNOT: ⟨propagateNOT S S' ⇒ cdclNOT-merged-bj-learn S S'⟩ |
cdclNOT-merged-bj-learn-backjump-l: ⟨backjump-l S S' ⇒ cdclNOT-merged-bj-learn S S'⟩ |
cdclNOT-merged-bj-learn-forgetNOT: ⟨forgetNOT S S' ⇒ cdclNOT-merged-bj-learn S S'⟩

```

```

lemma cdclNOT-merged-bj-learn-no-dup-inv:
  ⟨cdclNOT-merged-bj-learn S T ⇒ no-dup (trail S) ⇒ no-dup (trail T)⟩
  ⟨proof⟩
end

```

```

locale cdclNOT-merge-bj-learn-proxy =
  cdclNOT-merge-bj-learn-ops trail clausesNOT prepend-trail tl-trail add-clsNOT remove-clsNOT
  decide-conds propagate-conds forget-conds
  ⟨λC C' L' S T. backjump-l-cond C C' L' S T
  ∧ distinct-mset C' ∧ L' ∉# C' ∧ ¬tautology (add-mset L' C')⟩
for
  trail :: ⟨'st ⇒ ('v, unit) ann-lits⟩ and
  clausesNOT :: ⟨'st ⇒ 'v clauses⟩ and
  prepend-trail :: ⟨('v, unit) ann-lit ⇒ 'st ⇒ 'st⟩ and
  tl-trail :: ⟨'st ⇒ 'st⟩ and
  add-clsNOT :: ⟨'v clause ⇒ 'st ⇒ 'st⟩ and
  remove-clsNOT :: ⟨'v clause ⇒ 'st ⇒ 'st⟩ and
  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 ≡ λC C' L' S T. distinct-mset C' ∧ L' ∉# C' ∧ ¬tautology (add-mset L' C')⟩

sublocale backjumping-ops trail clausesNOT prepend-trail tl-trail add-clNOT remove-clNOT
    backjump-conds
    ⟨proof⟩

end

locale cdclNOT-merge-bj-learn =
    cdclNOT-merge-bj-learn-proxy trail clausesNOT prepend-trail tl-trail add-clNOT remove-clNOT
    decide-conds propagate-conds forget-conds backjump-l-cond inv
for
    trail :: ⟨'st ⇒ ('v, unit) ann-lits⟩ and
    clausesNOT :: ⟨'st ⇒ 'v clauses⟩ and
    prepend-trail :: ⟨('v, unit) ann-lit ⇒ 'st ⇒ 'st⟩ and
    tl-trail :: ⟨'st ⇒ 'st⟩ and
    add-clNOT :: ⟨'v clause ⇒ 'st ⇒ 'st⟩ and
    remove-clNOT :: ⟨'v clause ⇒ 'st ⇒ 'st⟩ and
    decide-conds :: ⟨'st ⇒ 'st ⇒ bool⟩ and
    propagate-conds :: ⟨('v, unit) ann-lit ⇒ 'st ⇒ 'st ⇒ bool⟩ and
    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 ∈# clausesNOT S
        ⇒ trail S ⊨as CNot C
        ⇒ undefined-lit F L
        ⇒ atm-of L ∈ atms-of-mm (clausesNOT S) ∪ atm-of ' (lits-of-l (F' @ Decided K # F))
        ⇒ clausesNOT S ⊨pm add-mset L C'
        ⇒ F ⊨as CNot C'
        ⇒ ¬no-step backjump-l S⟩ and
    cdcl-merged-inv: ⟨∧ S T. cdclNOT-merged-bj-learn S T ⇒ inv S ⇒ inv T⟩ and
    can-propagate-or-decide-or-backjump-l:
    ⟨atm-of L ∈ atms-of-mm (clausesNOT S) ⇒
        undefined-lit (trail S) L ⇒
        inv S ⇒
        satisfiable (set-mset (clausesNOT S)) ⇒
        ∃ T. decideNOT S T ∨ propagateNOT S T ∨ backjump-l S T⟩
begin

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

lemma tautology-single-add:

```

$\langle \text{tautology } (L + \{\#a\}) \longleftrightarrow \text{tautology } L \vee -a \in \# L \rangle$
 $\langle \text{proof} \rangle$

lemma *backjump-l-implies-exists-backjump*:

assumes *bj*: $\langle \text{backjump-l } S T \rangle$ **and** $\langle \text{inv } S \rangle$ **and** *n-d*: $\langle \text{no-dup } (\text{trail } S) \rangle$
shows $\langle \exists U. \text{backjump } S U \rangle$

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

sublocale *conflict-driven-clause-learning-ops* *trail clauses_{NOT} prepend-trail tl-trail add-cls_{NOT}*
remove-cls_{NOT} inv decide-conds backjump-conds propagate-conds

$\langle \lambda C -. \text{distinct-mset } C \wedge \neg \text{tautology } C \rangle$

forget-conds

$\langle \text{proof} \rangle$

lemma *backjump-l-learn-backjump*:

assumes *bt*: $\langle \text{backjump-l } S T \rangle$ **and** *inv*: $\langle \text{inv } S \rangle$

shows $\langle \exists C' L D. \text{learn } S (\text{add-cls}_{\text{NOT}} D S) \rangle$

$\wedge D = \text{add-mset } L C'$

$\wedge \text{backjump } (\text{add-cls}_{\text{NOT}} D S) T$

$\wedge \text{atms-of } (\text{add-mset } L C') \subseteq \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \cup \text{atm-of } ' (\text{lits-of-l } (\text{trail } S)) \rangle$

$\langle \text{proof} \rangle$

lemma *backjump-l-backjump-learn*:

assumes *bt*: $\langle \text{backjump-l } S T \rangle$ **and** *inv*: $\langle \text{inv } S \rangle$

shows $\langle \exists C' L D S'. \text{backjump } S S' \rangle$

$\wedge \text{learn } S' T$

$\wedge D = (\text{add-mset } L C')$

$\wedge T \sim \text{add-cls}_{\text{NOT}} D S'$

$\wedge \text{atms-of } (\text{add-mset } L C') \subseteq \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \cup \text{atm-of } ' (\text{lits-of-l } (\text{trail } S)) \rangle$

$\wedge \text{clauses}_{\text{NOT}} S \models_{\text{pm}} D \rangle$

$\langle \text{proof} \rangle$

lemma *cdcl_{NOT}-merged-bj-learn-is-tranclp-cdcl_{NOT}*:

$\langle \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn } S T \implies \text{inv } S \implies \text{cdcl}_{\text{NOT}}^{++} S T \rangle$

$\langle \text{proof} \rangle$

lemma *rtranclp-cdcl_{NOT}-merged-bj-learn-is-rtranclp-cdcl_{NOT}-and-inv*:

$\langle \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn}^{**} S T \implies \text{inv } S \implies \text{cdcl}_{\text{NOT}}^{**} S T \wedge \text{inv } T \rangle$

$\langle \text{proof} \rangle$

lemma *rtranclp-cdcl_{NOT}-merged-bj-learn-is-rtranclp-cdcl_{NOT}*:

$\langle \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn}^{**} S T \implies \text{inv } S \implies \text{cdcl}_{\text{NOT}}^{**} S T \rangle$

$\langle \text{proof} \rangle$

lemma *rtranclp-cdcl_{NOT}-merged-bj-learn-inv*:

$\langle \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn}^{**} S T \implies \text{inv } S \implies \text{inv } T \rangle$

$\langle \text{proof} \rangle$

lemma *rtranclp-cdcl_{NOT}-merged-bj-learn-no-dup-inv*:

$\langle \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn}^{**} S T \implies \text{no-dup } (\text{trail } S) \implies \text{no-dup } (\text{trail } T) \rangle$

$\langle \text{proof} \rangle$

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

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

definition $\mu_{CDCL}'\text{-merged} :: \langle 'v \text{ clause set} \Rightarrow 'st \Rightarrow nat \rangle$ **where**

$\langle \mu_{CDCL}'\text{-merged } A T \equiv ((2 + \text{card} (\text{atms-of-ms } A)) \wedge (1 + \text{card} (\text{atms-of-ms } A)) - \mu_C' A T) * 2 + \text{card} (\text{set-mset} (\text{clauses}_{NOT} T)) \rangle$

lemma $\text{cdcl}_{NOT}\text{-decreasing-measure}'$:

assumes

$\langle \text{cdcl}_{NOT}\text{-merged-bj-learn } S T \rangle$ **and**

$\langle \text{inv } S \rangle$ **and**

$\langle \text{atm-cls} : \langle \text{atms-of-mm} (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A \rangle$ **and**

$\langle \text{atm-trail} : \langle \text{atm-of } ' \text{ lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A \rangle$ **and**

$\langle n\text{-d} : \langle \text{no-dup } (\text{trail } S) \rangle$ **and**

$\langle \text{fin-A} : \langle \text{finite } A \rangle$

shows $\langle \mu_{CDCL}'\text{-merged } A T < \mu_{CDCL}'\text{-merged } A S \rangle$

$\langle \text{proof} \rangle$

lemma $\text{wf-cdcl}_{NOT}\text{-merged-bj-learn}$:

assumes

$\langle \text{fin-A} : \langle \text{finite } A \rangle$

shows $\langle \text{wf } \{(T, S)\}$

$(\text{inv } S \wedge \text{atms-of-mm} (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A \wedge \text{atm-of } ' \text{ lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A$

$\wedge \text{no-dup } (\text{trail } S))$

$\wedge \text{cdcl}_{NOT}\text{-merged-bj-learn } S T \rangle$

$\langle \text{proof} \rangle$

lemma $\text{in-atms-neg-defined} : \langle x \in \text{atms-of } C' \implies F \models_{\text{as}} \text{CNot } C' \implies x \in \text{atm-of } ' \text{ lits-of-l } F \rangle$

$\langle \text{proof} \rangle$

lemma $\text{cdcl}_{NOT}\text{-merged-bj-learn-atms-of-ms-clauses-decreasing}$:

assumes $\langle \text{cdcl}_{NOT}\text{-merged-bj-learn } S T \rangle$ **and** $\langle \text{inv } S \rangle$

shows $\langle \text{atms-of-mm} (\text{clauses}_{NOT} T) \subseteq \text{atms-of-mm} (\text{clauses}_{NOT} S) \cup \text{atm-of } ' (\text{lits-of-l } (\text{trail } S)) \rangle$

$\langle \text{proof} \rangle$

lemma $\text{cdcl}_{NOT}\text{-merged-bj-learn-atms-in-trail-in-set}$:

assumes

$\langle \text{cdcl}_{NOT}\text{-merged-bj-learn } S T \rangle$ **and** $\langle \text{inv } S \rangle$ **and**

$\langle \text{atms-of-mm} (\text{clauses}_{NOT} S) \subseteq A \rangle$ **and**

$\langle \text{atm-of } ' (\text{lits-of-l } (\text{trail } S)) \subseteq A \rangle$

shows $\langle \text{atm-of } ' (\text{lits-of-l } (\text{trail } T)) \subseteq A \rangle$

$\langle \text{proof} \rangle$

lemma $\text{rtranclp-cdcl}_{NOT}\text{-merged-bj-learn-trail-clauses-bound}$:

assumes

$\text{cdcl} : \langle \text{cdcl}_{NOT}\text{-merged-bj-learn}^{**} S T \rangle$ **and**

$\text{inv} : \langle \text{inv } S \rangle$ **and**

$\text{atms-clauses-S} : \langle \text{atms-of-mm} (\text{clauses}_{NOT} S) \subseteq A \rangle$ **and**

$\text{atms-trail-S} : \langle \text{atm-of } ' (\text{lits-of-l } (\text{trail } S)) \subseteq A \rangle$

shows $\langle \text{atm-of } ' (\text{lits-of-l } (\text{trail } T)) \subseteq A \wedge \text{atms-of-mm} (\text{clauses}_{NOT} T) \subseteq A \rangle$

$\langle \text{proof} \rangle$

lemma $\text{cdcl}_{NOT}\text{-merged-bj-learn-trail-clauses-bound}$:

assumes

cdcl: $\langle \text{cdcl}_{\text{NOT-merged-bj-learn}} S T \rangle$ **and**
inv: $\langle \text{inv } S \rangle$ **and**
atms-clauses-S: $\langle \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \subseteq A \rangle$ **and**
atms-trail-S: $\langle \text{atm-of } (\text{lits-of-l } (\text{trail } S)) \subseteq A \rangle$
shows $\langle \text{atm-of } (\text{lits-of-l } (\text{trail } T)) \subseteq A \wedge \text{atms-of-mm } (\text{clauses}_{\text{NOT}} T) \subseteq A \rangle$
 $\langle \text{proof} \rangle$

lemma *tranclp-cdcl_{NOT}-cdcl_{NOT}-tranclp*:

assumes

$\langle \text{cdcl}_{\text{NOT-merged-bj-learn}}^{++} S T \rangle$ **and**

inv: $\langle \text{inv } S \rangle$ **and**

atm-cls: $\langle \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \subseteq \text{atms-of-ms } A \rangle$ **and**

atm-trail: $\langle \text{atm-of } (\text{lits-of-l } (\text{trail } S)) \subseteq \text{atms-of-ms } A \rangle$ **and**

n-d: $\langle \text{no-dup } (\text{trail } S) \rangle$ **and**

fin-A[simp]: $\langle \text{finite } A \rangle$

shows $\langle (T, S) \in \{(T, S)\} \rangle$.

$(\text{inv } S \wedge \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \subseteq \text{atms-of-ms } A \wedge \text{atm-of } (\text{lits-of-l } (\text{trail } S)) \subseteq \text{atms-of-ms } A$
 $\wedge \text{no-dup } (\text{trail } S))$

$\wedge \text{cdcl}_{\text{NOT-merged-bj-learn}} S T \rangle$ (**is** $\langle - \in ?P^+ \rangle$)

$\langle \text{proof} \rangle$

lemma *wf-tranclp-cdcl_{NOT}-merged-bj-learn*:

assumes $\langle \text{finite } A \rangle$

shows $\langle \text{wf } \{(T, S)\} \rangle$.

$(\text{inv } S \wedge \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \subseteq \text{atms-of-ms } A \wedge \text{atm-of } (\text{lits-of-l } (\text{trail } S)) \subseteq \text{atms-of-ms } A$
 $\wedge \text{no-dup } (\text{trail } S))$

$\wedge \text{cdcl}_{\text{NOT-merged-bj-learn}}^{++} S T \rangle$

$\langle \text{proof} \rangle$

lemma *cdcl_{NOT}-merged-bj-learn-final-state*:

fixes $A :: \langle 'v \text{ clause set} \rangle$ **and** $S T :: \langle 'st \rangle$

assumes

n-s: $\langle \text{no-step } \text{cdcl}_{\text{NOT-merged-bj-learn}} S \rangle$ **and**

atms-S: $\langle \text{atms-of-mm } (\text{clauses}_{\text{NOT}} S) \subseteq \text{atms-of-ms } A \rangle$ **and**

atms-trail: $\langle \text{atm-of } (\text{lits-of-l } (\text{trail } S)) \subseteq \text{atms-of-ms } A \rangle$ **and**

n-d: $\langle \text{no-dup } (\text{trail } S) \rangle$ **and**

$\langle \text{finite } A \rangle$ **and**

inv: $\langle \text{inv } S \rangle$ **and**

decomp: $\langle \text{all-decomposition-implies-m } (\text{clauses}_{\text{NOT}} S) (\text{get-all-ann-decomposition } (\text{trail } S)) \rangle$

shows $\langle \text{unsatisfiable } (\text{set-mset } (\text{clauses}_{\text{NOT}} S))$

$\vee (\text{trail } S \models_{\text{asm}} \text{clauses}_{\text{NOT}} S \wedge \text{satisfiable } (\text{set-mset } (\text{clauses}_{\text{NOT}} S))) \rangle$

$\langle \text{proof} \rangle$

lemma *cdcl_{NOT}-merged-bj-learn-all-decomposition-implies*:

assumes $\langle \text{cdcl}_{\text{NOT-merged-bj-learn}} S T \rangle$ **and** **inv**: $\langle \text{inv } S \rangle$

$\langle \text{all-decomposition-implies-m } (\text{clauses}_{\text{NOT}} S) (\text{get-all-ann-decomposition } (\text{trail } S)) \rangle$

shows

$\langle \text{all-decomposition-implies-m } (\text{clauses}_{\text{NOT}} T) (\text{get-all-ann-decomposition } (\text{trail } T)) \rangle$

$\langle \text{proof} \rangle$

lemma *rtranclp-cdcl_{NOT}-merged-bj-learn-all-decomposition-implies*:

assumes $\langle \text{cdcl}_{\text{NOT-merged-bj-learn}}^{**} S T \rangle$ **and** **inv**: $\langle \text{inv } S \rangle$

$\langle \text{all-decomposition-implies-m } (\text{clauses}_{\text{NOT}} S) (\text{get-all-ann-decomposition } (\text{trail } S)) \rangle$

shows

$\langle \text{all-decomposition-implies-m } (\text{clauses}_{\text{NOT}} T) (\text{get-all-ann-decomposition } (\text{trail } T)) \rangle$

$\langle \text{proof} \rangle$

lemma *full-cdcl_{NOT}-merged-bj-learn-final-state*:

fixes $A :: \langle 'v \text{ clause set} \rangle$ **and** $S T :: \langle 'st \rangle$

assumes

full: $\langle \text{full-cdcl}_{NOT}\text{-merged-bj-learn } S T \rangle$ **and**

atms-S: $\langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A \rangle$ **and**

atms-trail: $\langle \text{atm-of } ' \text{ lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A \rangle$ **and**

n-d: $\langle \text{no-dup } (\text{trail } S) \rangle$ **and**

$\langle \text{finite } A \rangle$ **and**

inv: $\langle \text{inv } S \rangle$ **and**

decomp: $\langle \text{all-decomposition-implies-m } (\text{clauses}_{NOT} S) (\text{get-all-ann-decomposition } (\text{trail } S)) \rangle$

shows $\langle \text{unsatisfiable } (\text{set-mset } (\text{clauses}_{NOT} T)) \rangle$

$\vee (\text{trail } T \models_{asm} \text{clauses}_{NOT} T \wedge \text{satisfiable } (\text{set-mset } (\text{clauses}_{NOT} T)))$

<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-learn

trail clauses_{NOT} prepend-trail tl-trail add-cl_{NOT} remove-cl_{NOT}

inv decide-conds backjump-conds propagate-conds learn-restrictions forget-restrictions

for

trail :: $\langle 'st \Rightarrow ('v, \text{unit}) \text{ ann-lits} \rangle$ **and**

clauses_{NOT} :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**

prepend-trail :: $\langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

tl-trail :: $\langle 'st \Rightarrow 'st \rangle$ **and**

add-cl_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

remove-cl_{NOT} :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

inv :: $\langle 'st \Rightarrow \text{bool} \rangle$ **and**

decide-conds :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **and**

backjump-conds :: $\langle 'v \text{ clause} \Rightarrow 'v \text{ clause} \Rightarrow 'v \text{ literal} \Rightarrow 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **and**

propagate-conds :: $\langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **and**

learn-restrictions forget-restrictions :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow \text{bool} \rangle$

+

fixes $f :: \langle \text{nat} \Rightarrow \text{nat} \rangle$

assumes

unbounded: $\langle \text{unbounded } f \rangle$ **and** *f-ge-1*: $\langle \bigwedge n. n \geq 1 \Rightarrow f n \geq 1 \rangle$ **and**

inv-restart: $\langle \bigwedge S T. \text{inv } S \Rightarrow T \sim \text{reduce-trail-to}_{NOT} (\square :: 'a \text{ list}) S \Rightarrow \text{inv } T \rangle$

begin

lemma *bound-inv-inv*:

assumes

$\langle \text{inv } S \rangle$ **and**

n-d: $\langle \text{no-dup } (\text{trail } S) \rangle$ **and**

atms-cl_{NOT}-S-A: $\langle \text{atms-of-mm } (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A \rangle$ **and**

atms-trail-S-A: $\langle \text{atm-of } ' \text{ lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A \rangle$ **and**

$\langle \text{finite } A \rangle$ **and**

cdcl_{NOT}: $\langle \text{cdcl}_{NOT} S T \rangle$

shows

$\langle \text{atms-of-mm } (\text{clauses}_{NOT} T) \subseteq \text{atms-of-ms } A \rangle$ **and**

$\langle \text{atm-of } ' \text{ lits-of-l } (\text{trail } T) \subseteq \text{atms-of-ms } A \rangle$ **and**

$\langle \text{finite } A \rangle$
 $\langle \text{proof} \rangle$

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

lemma $\text{cdcl}_{NOT}\text{-with-restart-}\mu_{CDCL}'\text{-le-}\mu_{CDCL}'\text{-bound}$:

assumes

cdcl_{NOT} : $\langle \text{cdcl}_{NOT}\text{-restart} (T, a) (V, b) \rangle$ **and**

$\text{cdcl}_{NOT}\text{-inv}$:

$\langle \text{inv } T \rangle$

$\langle \text{no-dup} (\text{trail } T) \rangle$ **and**

bound-inv :

$\langle \text{atms-of-mm} (\text{clauses}_{NOT} T) \subseteq \text{atms-of-ms } A \rangle$

$\langle \text{atm-of ' lits-of-l} (\text{trail } T) \subseteq \text{atms-of-ms } A \rangle$

$\langle \text{finite } A \rangle$

shows $\langle \mu_{CDCL}' A V \leq \mu_{CDCL}'\text{-bound } A T \rangle$

$\langle \text{proof} \rangle$

lemma $\text{cdcl}_{NOT}\text{-with-restart-}\mu_{CDCL}'\text{-bound-le-}\mu_{CDCL}'\text{-bound}$:

assumes

cdcl_{NOT} : $\langle \text{cdcl}_{NOT}\text{-restart} (T, a) (V, b) \rangle$ **and**

$\text{cdcl}_{NOT}\text{-inv}$:

$\langle \text{inv } T \rangle$

$\langle \text{no-dup} (\text{trail } T) \rangle$ **and**

bound-inv :

$\langle \text{atms-of-mm} (\text{clauses}_{NOT} T) \subseteq \text{atms-of-ms } A \rangle$

$\langle \text{atm-of ' lits-of-l} (\text{trail } T) \subseteq \text{atms-of-ms } A \rangle$

$\langle \text{finite } A \rangle$

shows $\langle \mu_{CDCL}'\text{-bound } A V \leq \mu_{CDCL}'\text{-bound } A T \rangle$

$\langle \text{proof} \rangle$

sublocale $\text{cdcl}_{NOT}\text{-increasing-restarts}$ - - - - -

f

$\langle \lambda S T. T \sim \text{reduce-trail-to}_{NOT} ([\]::'a \text{ list}) S \rangle$

$\langle \lambda A S. \text{atms-of-mm} (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms } A$

$\wedge \text{atm-of ' lits-of-l} (\text{trail } S) \subseteq \text{atms-of-ms } A \wedge \text{finite } A \rangle$

$\mu_{CDCL}' \text{cdcl}_{NOT}$

$\langle \lambda S. \text{inv } S \wedge \text{no-dup} (\text{trail } S) \rangle$

$\mu_{CDCL}'\text{-bound}$

$\langle \text{proof} \rangle$

lemma $\text{cdcl}_{NOT}\text{-restart-all-decomposition-implies}$:

assumes $\langle \text{cdcl}_{NOT}\text{-restart } S T \rangle$ **and**

$\langle \text{inv} (fst S) \rangle$ **and**

$\langle \text{no-dup} (\text{trail} (fst S)) \rangle$

$\langle \text{all-decomposition-implies-m} (\text{clauses}_{NOT} (fst S)) (\text{get-all-ann-decomposition} (\text{trail} (fst S))) \rangle$

shows

$\langle \text{all-decomposition-implies-m} (\text{clauses}_{NOT} (fst T)) (\text{get-all-ann-decomposition} (\text{trail} (fst T))) \rangle$

$\langle \text{proof} \rangle$

lemma $\text{rtranclp-cdcl}_{NOT}\text{-restart-all-decomposition-implies}$:

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

lemma $\text{cdcl}_{NOT}\text{-restart-sat-ext-iff}$:

assumes
 st : $\langle \text{cdcl}_{NOT}\text{-restart} S T \rangle$ **and**
 no-d : $\langle \text{no-dup} (\text{trail} (fst S)) \rangle$ **and**
 inv : $\langle \text{inv} (fst S) \rangle$
shows $\langle I \models_{\text{sextm}} \text{clauses}_{NOT} (fst S) \longleftrightarrow I \models_{\text{sextm}} \text{clauses}_{NOT} (fst T) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{rtranclp-cdcl}_{NOT}\text{-restart-sat-ext-iff}$:

fixes $S T :: \langle 'st \times \text{nat} \rangle$
assumes
 st : $\langle \text{cdcl}_{NOT}\text{-restart}^{**} S T \rangle$ **and**
 no-d : $\langle \text{no-dup} (\text{trail} (fst S)) \rangle$ **and**
 inv : $\langle \text{inv} (fst S) \rangle$
shows $\langle I \models_{\text{sextm}} \text{clauses}_{NOT} (fst S) \longleftrightarrow I \models_{\text{sextm}} \text{clauses}_{NOT} (fst T) \rangle$
 $\langle \text{proof} \rangle$

theorem $\text{full-cdcl}_{NOT}\text{-restart-backjump-final-state}$:

fixes $A :: \langle 'v \text{ clause set} \rangle$ **and** $S T :: \langle 'st \rangle$
assumes
 full : $\langle \text{full cdcl}_{NOT}\text{-restart} (S, n) (T, m) \rangle$ **and**
 atms-S : $\langle \text{atms-of-mm} (\text{clauses}_{NOT} S) \subseteq \text{atms-of-ms} A \rangle$ **and**
 atms-trail : $\langle \text{atm-of} ' \text{lits-of-l} (\text{trail} S) \subseteq \text{atms-of-ms} A \rangle$ **and**
 no-d : $\langle \text{no-dup} (\text{trail} S) \rangle$ **and**
 fin-A[simp] : $\langle \text{finite} A \rangle$ **and**
 inv : $\langle \text{inv} S \rangle$ **and**
 decomp : $\langle \text{all-decomposition-implies-m} (\text{clauses}_{NOT} S) (\text{get-all-ann-decomposition} (\text{trail} S)) \rangle$
shows $\langle \text{unsatisfiable} (\text{set-mset} (\text{clauses}_{NOT} S))$
 $\vee (\text{lits-of-l} (\text{trail} T) \models_{\text{sextm}} \text{clauses}_{NOT} S \wedge \text{satisfiable} (\text{set-mset} (\text{clauses}_{NOT} S))) \rangle$
 $\langle \text{proof} \rangle$

end — End of the locale $\text{cdcl}_{NOT}\text{-with-backtrack-and-restarts}$.

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 $\text{cdcl}_{NOT}\text{-merge-bj-learn-with-backtrack-restarts} =$

$\text{cdcl}_{NOT}\text{-merge-bj-learn}$ trail clauses_{NOT} prepend-trail tl-trail add-cl_{NOT} remove-cl_{NOT}
 decide-conds propagate-conds forget-conds
 $\langle \lambda C C' L' S T. \text{distinct-mset} C' \wedge L' \notin \# C' \wedge \text{backjump-l-cond} C C' L' S T \rangle$ inv
for
 $\text{trail} :: \langle 'st \Rightarrow ('v, \text{unit}) \text{ ann-lits} \rangle$ **and**
 $\text{clauses}_{NOT} :: \langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**
 $\text{prepend-trail} :: \langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
 $\text{tl-trail} :: \langle 'st \Rightarrow 'st \rangle$ **and**
 $\text{add-cl}_{NOT} :: \langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
 $\text{remove-cl}_{NOT} :: \langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**
 $\text{decide-conds} :: \langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **and**
 $\text{propagate-conds} :: \langle ('v, \text{unit}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ **and**

$inv :: \langle 'st \Rightarrow bool \rangle$ **and**
 $forget-conds :: \langle 'v \text{ clause} \Rightarrow 'st \Rightarrow bool \rangle$ **and**
 $backjump-l-cond :: \langle 'v \text{ clause} \Rightarrow 'v \text{ clause} \Rightarrow 'v \text{ literal} \Rightarrow 'st \Rightarrow 'st \Rightarrow bool \rangle$
 +
fixes $f :: \langle nat \Rightarrow nat \rangle$
assumes
 $unbounded: \langle unbounded \ f \rangle$ **and** $f\text{-ge-1}: \langle \bigwedge n. n \geq 1 \implies f \ n \geq 1 \rangle$ **and**
 $inv\text{-restart}: \langle \bigwedge S \ T. inv \ S \implies T \sim reduce\text{-trail-to}_{NOT} \ [] \ S \implies inv \ T \rangle$
begin

definition $not\text{-simplified-cl} :: \langle 'b \text{ clause multiset} \Rightarrow 'b \text{ clauses} \rangle$

where

$\langle not\text{-simplified-cl} \ A \equiv \{ \# C \in \# A. C \notin simple\text{-clss} \ (atms\text{-of-mm} \ A) \# \}$

lemma $not\text{-simplified-cl}\text{-tautology-distinct-mset}$:

$\langle not\text{-simplified-cl} \ A = \{ \# C \in \# A. tautology \ C \vee \neg distinct\text{-mset} \ C \# \}$
 $\langle proof \rangle$

lemma $simple\text{-clss-or-not-simplified-cl}$:

assumes $\langle atms\text{-of-mm} \ (clauses_{NOT} \ S) \subseteq atms\text{-of-ms} \ A \rangle$ **and**

$\langle x \in \# clauses_{NOT} \ S \rangle$ **and** $\langle finite \ A \rangle$

shows $\langle x \in simple\text{-clss} \ (atms\text{-of-ms} \ A) \vee x \in \# not\text{-simplified-cl} \ (clauses_{NOT} \ S) \rangle$

$\langle proof \rangle$

lemma $cdcl_{NOT}\text{-merged-bj-learn-clauses-bound}$:

assumes

$\langle cdcl_{NOT}\text{-merged-bj-learn} \ S \ T \rangle$ **and**

$inv: \langle inv \ S \rangle$ **and**

$atms\text{-clss}: \langle atms\text{-of-mm} \ (clauses_{NOT} \ S) \subseteq atms\text{-of-ms} \ A \rangle$ **and**

$atms\text{-trail}: \langle atm\text{-of} \ (lits\text{-of-l} \ (trail \ S)) \subseteq atms\text{-of-ms} \ A \rangle$ **and**

$fin\text{-A}[simp]: \langle finite \ A \rangle$

shows $\langle set\text{-mset} \ (clauses_{NOT} \ T) \subseteq set\text{-mset} \ (not\text{-simplified-cl} \ (clauses_{NOT} \ S))$

$\cup simple\text{-clss} \ (atms\text{-of-ms} \ A) \rangle$

$\langle proof \rangle$

lemma $cdcl_{NOT}\text{-merged-bj-learn-not-simplified-decreasing}$:

assumes $\langle cdcl_{NOT}\text{-merged-bj-learn} \ S \ T \rangle$

shows $\langle not\text{-simplified-cl} \ (clauses_{NOT} \ T) \subseteq \# not\text{-simplified-cl} \ (clauses_{NOT} \ S) \rangle$

$\langle proof \rangle$

lemma $rtranclp\text{-cdcl}_{NOT}\text{-merged-bj-learn-not-simplified-decreasing}$:

assumes $\langle cdcl_{NOT}\text{-merged-bj-learn}^{**} \ S \ T \rangle$

shows $\langle not\text{-simplified-cl} \ (clauses_{NOT} \ T) \subseteq \# not\text{-simplified-cl} \ (clauses_{NOT} \ S) \rangle$

$\langle proof \rangle$

lemma $rtranclp\text{-cdcl}_{NOT}\text{-merged-bj-learn-clauses-bound}$:

assumes

$\langle cdcl_{NOT}\text{-merged-bj-learn}^{**} \ S \ T \rangle$ **and**

$\langle inv \ S \rangle$ **and**

$\langle atms\text{-of-mm} \ (clauses_{NOT} \ S) \subseteq atms\text{-of-ms} \ A \rangle$ **and**

$\langle atm\text{-of} \ (lits\text{-of-l} \ (trail \ S)) \subseteq atms\text{-of-ms} \ A \rangle$ **and**

$finite[simp]: \langle finite \ A \rangle$

shows $\langle set\text{-mset} \ (clauses_{NOT} \ T) \subseteq set\text{-mset} \ (not\text{-simplified-cl} \ (clauses_{NOT} \ S))$

$\cup simple\text{-clss} \ (atms\text{-of-ms} \ A) \rangle$

$\langle proof \rangle$

abbreviation μ_{CDCL}' -bound where

$$\langle \mu_{CDCL}'\text{-bound } A \ T \equiv ((2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A))) * 2 \\ + \text{card } (\text{set-mset } (\text{not-simplified-cl}(\text{clauses}_{NOT} \ T))) \\ + 3 \wedge \text{card } (\text{atms-of-ms } A) \rangle$$

lemma $rtrancpl$ - $cdcl_{NOT}$ -merged-bj-learn-clauses-bound-card:

assumes

$$\langle cdcl_{NOT}\text{-merged-bj-learn}^{**} \ S \ T \rangle \ \mathbf{and} \\ \langle inv \ S \rangle \ \mathbf{and} \\ \langle \text{atms-of-mm } (\text{clauses}_{NOT} \ S) \subseteq \text{atms-of-ms } A \rangle \ \mathbf{and} \\ \langle \text{atm-of } ' \text{lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A \rangle \ \mathbf{and} \\ \text{finite: } \langle \text{finite } A \rangle$$

shows $\langle \mu_{CDCL}'\text{-merged } A \ T \leq \mu_{CDCL}'\text{-bound } A \ S \rangle$

$\langle \text{proof} \rangle$

sublocale $cdcl_{NOT}$ -increasing-restarts-ops $\langle \lambda S \ T. \ T \sim \text{reduce-trail-to}_{NOT} \ ([::'a \ \text{list}] \ S) \rangle$

$$cdcl_{NOT}\text{-merged-bj-learn } f \\ \langle \lambda A \ S. \ \text{atms-of-mm } (\text{clauses}_{NOT} \ S) \subseteq \text{atms-of-ms } A \\ \wedge \text{atm-of } ' \text{lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A \wedge \text{finite } A \rangle \\ \mu_{CDCL}'\text{-merged} \\ \langle \lambda S. \ inv \ S \wedge \text{no-dup } (\text{trail } S) \rangle \\ \mu_{CDCL}'\text{-bound} \\ \langle \text{proof} \rangle$$

lemma $cdcl_{NOT}$ -restart- μ_{CDCL}' -merged-le- μ_{CDCL}' -bound:

assumes

$$\langle cdcl_{NOT}\text{-restart } T \ V \rangle \\ \langle inv \ (\text{fst } T) \rangle \ \mathbf{and} \\ \langle \text{no-dup } (\text{trail } (\text{fst } T)) \rangle \ \mathbf{and} \\ \langle \text{atms-of-mm } (\text{clauses}_{NOT} \ (\text{fst } T)) \subseteq \text{atms-of-ms } A \rangle \ \mathbf{and} \\ \langle \text{atm-of } ' \text{lits-of-l } (\text{trail } (\text{fst } T)) \subseteq \text{atms-of-ms } A \rangle \ \mathbf{and} \\ \langle \text{finite } A \rangle$$

shows $\langle \mu_{CDCL}'\text{-merged } A \ (\text{fst } V) \leq \mu_{CDCL}'\text{-bound } A \ (\text{fst } T) \rangle$

$\langle \text{proof} \rangle$

lemma $cdcl_{NOT}$ -restart- μ_{CDCL}' -bound-le- μ_{CDCL}' -bound:

assumes

$$\langle cdcl_{NOT}\text{-restart } T \ V \rangle \ \mathbf{and} \\ \langle \text{no-dup } (\text{trail } (\text{fst } T)) \rangle \ \mathbf{and} \\ \langle inv \ (\text{fst } T) \rangle \ \mathbf{and} \\ \text{fin: } \langle \text{finite } A \rangle$$

shows $\langle \mu_{CDCL}'\text{-bound } A \ (\text{fst } V) \leq \mu_{CDCL}'\text{-bound } A \ (\text{fst } T) \rangle$

$\langle \text{proof} \rangle$

sublocale $cdcl_{NOT}$ -increasing-restarts - - - - - f

$$\langle \lambda S \ T. \ T \sim \text{reduce-trail-to}_{NOT} \ ([::'a \ \text{list}] \ S) \rangle \\ \langle \lambda A \ S. \ \text{atms-of-mm } (\text{clauses}_{NOT} \ S) \subseteq \text{atms-of-ms } A \\ \wedge \text{atm-of } ' \text{lits-of-l } (\text{trail } S) \subseteq \text{atms-of-ms } A \wedge \text{finite } A \rangle \\ \mu_{CDCL}'\text{-merged } cdcl_{NOT}\text{-merged-bj-learn} \\ \langle \lambda S. \ inv \ S \wedge \text{no-dup } (\text{trail } S) \rangle \\ \langle \lambda A \ T. \ ((2 + \text{card } (\text{atms-of-ms } A)) \wedge (1 + \text{card } (\text{atms-of-ms } A))) * 2 \\ + \text{card } (\text{set-mset } (\text{not-simplified-cl}(\text{clauses}_{NOT} \ T))) \\ + 3 \wedge \text{card } (\text{atms-of-ms } A) \rangle \\ \langle \text{proof} \rangle$$

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

lemma *true-clss-ext-decrease-add-implied*:
assumes $\langle M \models_{\text{pm}} C \rangle$
shows $\langle I \models_{\text{sext}} \text{insert } C \text{ (set-mset } M) \longleftrightarrow I \models_{\text{sextm}} M \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl_{NOT}-merged-bj-learn-bj-sat-ext-iff*:
assumes $\langle \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn } S \ T \rangle$ **and** *inv*: $\langle \text{inv } S \rangle$
shows $\langle I \models_{\text{sextm}} \text{clauses}_{\text{NOT}} \ S \longleftrightarrow I \models_{\text{sextm}} \text{clauses}_{\text{NOT}} \ T \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-cdcl_{NOT}-merged-bj-learn-bj-sat-ext-iff*:
assumes $\langle \text{cdcl}_{\text{NOT}}\text{-merged-bj-learn}^{**} \ S \ T \rangle$ **and** *inv*: $\langle \text{inv } S \rangle$
shows $\langle I \models_{\text{sextm}} \text{clauses}_{\text{NOT}} \ S \longleftrightarrow I \models_{\text{sextm}} \text{clauses}_{\text{NOT}} \ T \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl_{NOT}-restart-eq-sat-iff*:
assumes
 $\langle \text{cdcl}_{\text{NOT}}\text{-restart } S \ T \rangle$ **and**
inv: $\langle \text{inv } (\text{fst } S) \rangle$
shows $\langle I \models_{\text{sextm}} \text{clauses}_{\text{NOT}} \ (\text{fst } S) \longleftrightarrow I \models_{\text{sextm}} \text{clauses}_{\text{NOT}} \ (\text{fst } T) \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-cdcl_{NOT}-restart-eq-sat-iff*:
assumes
 $\langle \text{cdcl}_{\text{NOT}}\text{-restart}^{**} \ S \ T \rangle$ **and**
inv: $\langle \text{inv } (\text{fst } S) \rangle$ **and** *n-d*: $\langle \text{no-dup}(\text{trail } (\text{fst } S)) \rangle$
shows $\langle I \models_{\text{sextm}} \text{clauses}_{\text{NOT}} \ (\text{fst } S) \longleftrightarrow I \models_{\text{sextm}} \text{clauses}_{\text{NOT}} \ (\text{fst } T) \rangle$
 $\langle \text{proof} \rangle$

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

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

lemma *full-cdcl_{NOT}-restart-normal-form*:
assumes
full: $\langle \text{full } \text{cdcl}_{\text{NOT}}\text{-restart } S \ T \rangle$ **and**
inv: $\langle \text{inv } (\text{fst } S) \rangle$ **and** *n-d*: $\langle \text{no-dup}(\text{trail } (\text{fst } S)) \rangle$ **and**

decomp: $\langle \text{all-decomposition-implies-}m \text{ (} \text{clauses}_{NOT} \text{ (fst } S))$
 $\text{(get-all-ann-decomposition (trail (fst } S))) \rangle$ **and**
atms-cl: $\langle \text{atms-of-mm (} \text{clauses}_{NOT} \text{ (fst } S)) \subseteq \text{atms-of-ms } A \rangle$ **and**
atms-trail: $\langle \text{atm-of ' lits-of-l (trail (fst } S)) \subseteq \text{atms-of-ms } A \rangle$ **and**
fin: $\langle \text{finite } A \rangle$
shows $\langle \text{unsatisfiable (set-mset (clauses}_{NOT} \text{ (fst } S))}$
 $\vee \text{ lits-of-l (trail (fst } T)) \models_{\text{sextm}} \text{clauses}_{NOT} \text{ (fst } S) \wedge$
 $\text{satisfiable (set-mset (clauses}_{NOT} \text{ (fst } S))) \rangle$
 $\langle \text{proof} \rangle$

corollary *full-cdcl_{NOT}-restart-normal-form-init-state*:

assumes

init-state: $\langle \text{trail } S = [] \rangle \langle \text{clauses}_{NOT} \text{ } S = N \rangle$ **and**

full: $\langle \text{full cdcl}_{NOT}\text{-restart (} S, 0) T \rangle$ **and**

inv: $\langle \text{inv } S \rangle$

shows $\langle \text{unsatisfiable (set-mset } N)$

$\vee \text{ lits-of-l (trail (fst } T)) \models_{\text{sextm}} N \wedge \text{satisfiable (set-mset } N) \rangle$

$\langle \text{proof} \rangle$

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*

\Rightarrow (*'v*, *unit*) *ann-lits* **where**

convert-trail-from-W $\equiv \text{map } \text{convert-ann-lit-from-W}$

lemma *lits-of-l-convert-trail-from-W[simp]*:

lits-of-l (*convert-trail-from-W* *M*) = *lits-of-l* *M*

$\langle \text{proof} \rangle$

lemma *lit-of-convert-trail-from-W[simp]*:

lit-of (*convert-ann-lit-from-W* *L*) = *lit-of* *L*

$\langle \text{proof} \rangle$

lemma *no-dup-convert-from-W[simp]*:

no-dup (*convert-trail-from-W* *M*) \longleftrightarrow *no-dup* *M*

$\langle \text{proof} \rangle$

lemma *convert-trail-from-W-true-annots[simp]*:

convert-trail-from-W *M* $\models_{\text{as}} C \longleftrightarrow M \models_{\text{as}} C$

⟨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-cl*s :: '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-cl |
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 = 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**

$add_cls_{NOT} = \lambda C S. add_learned_cls C S$ **and**
 $remove_cls_{NOT} = \lambda C S. remove_cls C S$
 ⟨proof⟩

sublocale $state_W \subseteq dpll_state$ **where**
 $trail = \lambda S. convert_trail_from_W (trail S)$ **and**
 $clauses_{NOT} = clauses$ **and**
 $prepend_trail = \lambda L S. cons_trail (convert_ann_lit_from_NOT L) S$ **and**
 $tl_trail = \lambda S. tl_trail S$ **and**
 $add_cls_{NOT} = \lambda C S. add_learned_cls C S$ **and**
 $remove_cls_{NOT} = \lambda C S. remove_cls C S$
 ⟨proof⟩

context $state_W$
begin
declare $state_simp_{NOT}[simp\ del]$
end

2.3.2 Inclusion of Weidendenbch's CDCL without Strategy

sublocale $conflict_driven_clause_learning_W \subseteq cdcl_{NOT}_merge_bj_learn_ops$ **where**
 $trail = \lambda S. convert_trail_from_W (trail S)$ **and**
 $clauses_{NOT} = clauses$ **and**
 $prepend_trail = \lambda L S. cons_trail (convert_ann_lit_from_NOT L) S$ **and**
 $tl_trail = \lambda S. tl_trail S$ **and**
 $add_cls_{NOT} = \lambda C S. add_learned_cls C S$ **and**
 $remove_cls_{NOT} = \lambda C S. remove_cls C S$ **and**
 $decide_conds = \lambda -. True$ **and**
 $propagate_conds = \lambda - -. True$ **and**
 $forget_conds = \lambda S. conflicting S = None$ **and**
 $backjump_l_cond = \lambda C C' L' S T. backjump_l_cond C C' L' S T$
 $\wedge distinct_mset C' \wedge L' \notin \# C' \wedge \neg tautology (add_mset L' C')$
 ⟨proof⟩

sublocale $conflict_driven_clause_learning_W \subseteq cdcl_{NOT}_merge_bj_learn_proxy$ **where**
 $trail = \lambda S. convert_trail_from_W (trail S)$ **and**
 $clauses_{NOT} = clauses$ **and**
 $prepend_trail = \lambda L S. cons_trail (convert_ann_lit_from_NOT L) S$ **and**
 $tl_trail = \lambda S. tl_trail S$ **and**
 $add_cls_{NOT} = \lambda C S. add_learned_cls C S$ **and**
 $remove_cls_{NOT} = \lambda C S. remove_cls C S$ **and**
 $decide_conds = \lambda -. True$ **and**
 $propagate_conds = \lambda - -. True$ **and**
 $forget_conds = \lambda S. conflicting S = None$ **and**
 $backjump_l_cond = backjump_l_cond$ **and**
 $inv = inv_{NOT}$
 ⟨proof⟩

sublocale $conflict_driven_clause_learning_W \subseteq cdcl_{NOT}_merge_bj_learn$ **where**
 $trail = \lambda S. convert_trail_from_W (trail S)$ **and**
 $clauses_{NOT} = clauses$ **and**
 $prepend_trail = \lambda L S. cons_trail (convert_ann_lit_from_NOT L) S$ **and**
 $tl_trail = \lambda S. tl_trail S$ **and**
 $add_cls_{NOT} = \lambda C S. add_learned_cls C S$ **and**
 $remove_cls_{NOT} = \lambda C S. remove_cls C S$ **and**
 $decide_conds = \lambda -. True$ **and**

propagate-conds = λ - - -. *True* **and**
forget-conds = λ - *S*. *conflicting* *S* = *None* **and**
backjump-l-cond = *backjump-l-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*:
 $trail\ S = trail\ T \implies trail\ (reduce-trail-to_{NOT}\ F\ S) = trail\ (reduce-trail-to_{NOT}\ F\ T)$
 ⟨*proof*⟩

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

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

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

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

lemma *skip-or-resolve-state-change*:
assumes *skip-or-resolve*** *S* *T*
shows
 $\exists M. trail\ S = M\ @\ trail\ T \wedge (\forall m \in set\ M. \neg is-decided\ m)$
 $clauses\ S = clauses\ T$
 $backtrack-lvl\ S = backtrack-lvl\ T$
 $init-clss\ S = init-clss\ T$
 $learned-clss\ S = learned-clss\ T$
 ⟨*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-step cdcl_W-merge $T \wedge$ conflicting $T \neq \text{None}$)
 ⟨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 $T \neq \text{None}$)

⟨proof⟩

abbreviation $\mu_{FW} :: 'st \Rightarrow nat$ **where**

$\mu_{FW} S \equiv$ (if no-step cdcl_W-merge S then 0 else $1 + \mu_{CDCL}$ '-merged (set-mset (init-cls S)) S)

lemma cdcl_W-merge- μ_{FW} -decreasing:

assumes

inv: cdcl_W-all-struct-inv S **and**

fw: cdcl_W-merge $S T$

shows $\mu_{FW} T < \mu_{FW} S$

⟨proof⟩

lemma wf-cdcl_W-merge: wf $\{(T, S). \text{cdcl}_W\text{-all-struct-inv } S \wedge \text{cdcl}_W\text{-merge } S T\}$

⟨proof⟩

lemma tranclp-cdcl_W-merge-cdcl_W-merge-trancl:

$\{(T, S). \text{cdcl}_W\text{-all-struct-inv } S \wedge \text{cdcl}_W\text{-merge}^{++} S T\}$

$\subseteq \{(T, S). \text{cdcl}_W\text{-all-struct-inv } S \wedge \text{cdcl}_W\text{-merge } S T\}^+$

⟨proof⟩

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

⟨proof⟩

lemma wf-cdcl_W-bj-all-struct: wf $\{(T, S). \text{cdcl}_W\text{-all-struct-inv } S \wedge \text{cdcl}_W\text{-bj } S T\}$

⟨proof⟩

lemma cdcl_W-conflicting-true-cdcl_W-merge-restart:

assumes cdcl_W $S V$ **and** confl: conflicting $S = \text{None}$

shows (cdcl_W-merge $S V \wedge$ conflicting $V = \text{None}$) \vee (conflicting $V \neq \text{None} \wedge$ 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 $S = \text{None}$

shows (cdcl_W-merge⁺⁺ $S V \wedge$ conflicting $V = \text{None}$)

\vee ($\exists T U. \text{cdcl}_W\text{-merge}^{**} S T \wedge$ conflicting $V \neq \text{None} \wedge$ conflict $T U \wedge \text{cdcl}_W\text{-bj}^{**} U V$)

⟨proof⟩

lemma wf-cdcl_W: wf $\{(T, S). \text{cdcl}_W\text{-all-struct-inv } S \wedge \text{cdcl}_W S T\}$

⟨proof⟩

lemma wf-cdcl_W-stgy:

⟨wf $\{(T, S). \text{cdcl}_W\text{-all-struct-inv } S \wedge \text{cdcl}_W\text{-stgy } S T\}$ ⟩

⟨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 $\equiv \lambda-. \text{propagate}$

abbreviation (*input*) *decide-conds* **where**

decide-conds $S T \equiv \text{decide } S T \wedge \text{no-step conflict } S \wedge \text{no-step propagate } S$

abbreviation *backjump-l-conds-stgy* :: '*v* clause \Rightarrow '*v* clause \Rightarrow '*v* literal \Rightarrow '*st* \Rightarrow '*st* \Rightarrow bool **where**

backjump-l-conds-stgy $C C' L S V \equiv$

$(\exists T U. \text{conflict } S T \wedge \text{full skip-or-resolve } T U \wedge \text{conflicting } T = \text{Some } C \wedge$
 $\text{mark-of } (\text{hd-trail } V) = \text{add-mset } L C' \wedge \text{backtrack } U V)$

abbreviation *inv_{NOT}-stgy* **where**

inv_{NOT}-stgy $S \equiv \text{conflicting } S = \text{None} \wedge \text{cdcl}_W\text{-all-struct-inv } S \wedge \text{no-smaller-propa } S \wedge$

$\text{cdcl}_W\text{-stgy-invariant } S \wedge \text{propagated-clauses-clauses } S$

interpretation *cdcl_W-with-strategy: cdcl_{NOT}-merge-bj-learn-ops* **where**

trail $= \lambda S. \text{convert-trail-from-}W (\text{trail } S)$ **and**

clauses_{NOT} $= \text{clauses}$ **and**

prepend-trail $= \lambda L S. \text{cons-trail } (\text{convert-ann-lit-from-NOT } L) S$ **and**

tl-trail $= \lambda S. \text{tl-trail } S$ **and**

add-cls_{NOT} $= \lambda C S. \text{add-learned-cls } C S$ **and**

remove-cls_{NOT} $= \lambda C S. \text{remove-cls } C S$ **and**

decide-conds $= \text{decide-conds}$ **and**

propagate-conds $= \text{propagate-conds}$ **and**

forget-conds $= \lambda-. \text{False}$ **and**

backjump-l-cond $= \lambda C C' L' S T. \text{backjump-l-conds-stgy } C C' L' S T$
 $\wedge \text{distinct-mset } C' \wedge L' \notin \# C' \wedge \neg \text{tautology } (\text{add-mset } L' C')$

$\langle \text{proof} \rangle$

interpretation *cdcl_W-with-strategy: cdcl_{NOT}-merge-bj-learn-proxy* **where**

trail $= \lambda S. \text{convert-trail-from-}W (\text{trail } S)$ **and**

clauses_{NOT} $= \text{clauses}$ **and**

prepend-trail $= \lambda L S. \text{cons-trail } (\text{convert-ann-lit-from-NOT } L) S$ **and**

tl-trail $= \lambda S. \text{tl-trail } S$ **and**

add-cls_{NOT} $= \lambda C S. \text{add-learned-cls } C S$ **and**

remove-cls_{NOT} $= \lambda C S. \text{remove-cls } C S$ **and**

decide-conds $= \text{decide-conds}$ **and**

propagate-conds $= \text{propagate-conds}$ **and**

forget-conds $= \lambda-. \text{False}$ **and**

backjump-l-cond $= \text{backjump-l-conds-stgy}$ **and**

inv $= \text{inv}_{\text{NOT-stgy}}$

$\langle \text{proof} \rangle$

lemma *cdcl_W-with-strategy-cdcl_{NOT}-merged-bj-learn-conflict:*

assumes

$\text{cdcl}_W\text{-with-strategy.cdcl}_{\text{NOT-merged-bj-learn}} S T$

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

shows

$\text{conflicting } T = \text{None}$

$\langle \text{proof} \rangle$

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 ⟹ ∃ f j. (∀ i < j. R (f i) (f (Suc i))) ∧ f 0 = a ∧ f j = b⟩*

⟨proof⟩

lemma *cdcl_W-bj-cdcl_W-stgy:* *⟨cdcl_W-bj S T ⟹ cdcl_W-stgy S T⟩*

⟨proof⟩

lemma *cdcl_W-restart-propagated-clauses-clauses:*

⟨cdcl_W-restart S T ⟹ propagated-clauses-clauses S ⟹ propagated-clauses-clauses T⟩

⟨proof⟩

lemma *rtranclp-cdcl_W-restart-propagated-clauses-clauses:*

*⟨cdcl_W-restart** S T ⟹ propagated-clauses-clauses S ⟹ propagated-clauses-clauses T⟩*

⟨proof⟩

lemma *rtranclp-cdcl_W-stgy-propagated-clauses-clauses:*

*⟨cdcl_W-stgy** S T ⟹ propagated-clauses-clauses S ⟹ propagated-clauses-clauses T⟩*

⟨proof⟩

lemma *conflicting-clause-bt-lvl-gt-0-backjump:*

assumes

inv: ⟨inv_{NOT}-stgy S⟩ and

C: ⟨C ∈ # clauses S⟩ and

tr-C: ⟨trail S |=_{as} CNot C⟩ and

bt: ⟨backtrack-lvl S > 0⟩

shows *⟨∃ T U V. conflict S T ∧ full skip-or-resolve T U ∧ 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 ⟷ propagate S T⟩ (is ?NOT ⟷ ?W)

⟨proof⟩

interpretation *cdcl_W-with-strategy: cdcl_{NOT}-merge-bj-learn where*

```

trail =  $\lambda S$ . convert-trail-from-W (trail S) and
clausesNOT = clauses and
prepend-trail =  $\lambda L S$ . cons-trail (convert-ann-lit-from-NOT L) S and
tl-trail =  $\lambda S$ . tl-trail S and
add-clsNOT =  $\lambda C S$ . add-learned-cls C S and
remove-clsNOT =  $\lambda C S$ . remove-cls C S and
decide-conds = decide-conds and
propagate-conds = propagate-conds and
forget-conds =  $\lambda$  -. False and
backjump-l-cond = backjump-l-conds-stgy and
inv = invNOT-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 $S T :: \langle 'st \times nat \rangle$ **where**

$\langle cdcl_W\text{-stgy } (fst S) (fst T) \implies snd S = snd T \implies cdcl_W\text{-restart-stgy } S T \rangle |$
 $\langle restart (fst S) (fst T) \implies snd T = Suc (snd S) \implies cdcl_W\text{-restart-stgy } S T \rangle$

lemma *cdcl_W-stgy-cdcl_W-restart*: $\langle cdcl_W\text{-stgy } S S' \implies cdcl_W\text{-restart } S S' \rangle$
<proof>

lemma *cdcl_W-restart-stgy-cdcl_W-restart*:
 $\langle cdcl_W\text{-restart-stgy } S T \implies cdcl_W\text{-restart } (fst S) (fst T) \rangle$
<proof>

lemma *rtranclp-cdcl_W-restart-stgy-cdcl_W-restart*:
 $\langle cdcl_W\text{-restart-stgy}^{**} S T \implies cdcl_W\text{-restart}^{**} (fst S) (fst T) \rangle$
<proof>

lemma *cdcl_W-stgy-cdcl_W-restart-stgy*:
 $\langle cdcl_W\text{-stgy } S T \implies cdcl_W\text{-restart-stgy } (S, n) (T, n) \rangle$
<proof>

lemma *rtranclp-cdcl_W-stgy-cdcl_W-restart-stgy*:
 $\langle cdcl_W\text{-stgy}^{**} S T \implies cdcl_W\text{-restart-stgy}^{**} (S, n) (T, n) \rangle$
<proof>

lemma *cdcl_W-restart-dcl_W-all-struct-inv*:
 $\langle cdcl_W\text{-restart-stgy } S T \implies cdcl_W\text{-all-struct-inv } (fst S) \implies cdcl_W\text{-all-struct-inv } (fst T) \rangle$
<proof>

lemma *rtranclp-cdcl_W-restart-dcl_W-all-struct-inv*:
 $\langle cdcl_W\text{-restart-stgy}^{**} S T \implies cdcl_W\text{-all-struct-inv } (fst S) \implies cdcl_W\text{-all-struct-inv } (fst T) \rangle$
<proof>

lemma *restart-cdcl_W-stgy-invariant*:

$\langle \text{restart } S \ T \implies \text{cdcl}_W\text{-stgy-invariant } T \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl_W-restart-dcl_W-stgy-invariant*:

$\langle \text{cdcl}_W\text{-restart-stgy } S \ T \implies \text{cdcl}_W\text{-all-struct-inv } (fst \ S) \implies \text{cdcl}_W\text{-stgy-invariant } (fst \ S) \implies$
 $\text{cdcl}_W\text{-stgy-invariant } (fst \ T) \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-cdcl_W-restart-dcl_W-stgy-invariant*:

$\langle \text{cdcl}_W\text{-restart-stgy}^* \ S \ T \implies \text{cdcl}_W\text{-all-struct-inv } (fst \ S) \implies \text{cdcl}_W\text{-stgy-invariant } (fst \ S) \implies$
 $\text{cdcl}_W\text{-stgy-invariant } (fst \ T) \rangle$
 $\langle \text{proof} \rangle$

end

locale *cdcl_W-restart-restart-ops* =

conflict-driven-clause-learning_W

state-eq

state

— functions for the state:

— access functions:

trail init-clss learned-clss conflicting

— changing state:

cons-trail tl-trail add-learned-cls remove-cls

update-conflicting

— get state:

init-state

for

state-eq :: $\langle 'st \Rightarrow 'st \Rightarrow \text{bool} \rangle$ (**infix** ~ 50) **and**

state :: $\langle 'st \Rightarrow ('v, 'v \text{ clause}) \text{ ann-lits} \times 'v \text{ clauses} \times 'v \text{ clauses} \times 'v \text{ clause option} \times$
 $'b \rangle$ **and**

trail :: $\langle 'st \Rightarrow ('v, 'v \text{ clause}) \text{ ann-lits} \rangle$ **and**

init-clss :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**

learned-clss :: $\langle 'st \Rightarrow 'v \text{ clauses} \rangle$ **and**

conflicting :: $\langle 'st \Rightarrow 'v \text{ clause option} \rangle$ **and**

cons-trail :: $\langle ('v, 'v \text{ clause}) \text{ ann-lit} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

tl-trail :: $\langle 'st \Rightarrow 'st \rangle$ **and**

add-learned-cls :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

remove-cls :: $\langle 'v \text{ clause} \Rightarrow 'st \Rightarrow 'st \rangle$ **and**

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

init-state :: $\langle 'v \text{ clauses} \Rightarrow 'st \rangle$ +

fixes

f :: $\langle \text{nat} \Rightarrow \text{nat} \rangle$

locale *cdcl_W-restart-restart* =

cdcl_W-restart-restart-ops +

assumes

f: $\langle \text{unbounded } f \rangle$

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 $\text{cdcl}_W\text{-stgy}^{\uparrow\downarrow} \ S \ T$: With a $\text{cdcl}_W\text{-stgy}^\downarrow \ S \ T$, 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:

$\langle (cdcl_W\text{-stgy} \sim (card (set\text{-mset} (learned\text{-class } T)) - card (set\text{-mset} (learned\text{-class } S)))) S T$
 $\implies card (set\text{-mset} (learned\text{-class } T)) - card (set\text{-mset} (learned\text{-class } S)) > f n$
 $\implies restart T U \implies cdcl_W\text{-merge-with-restart } (S, n) (U, Suc n) \rangle |$

restart-full: $\langle full1 cdcl_W\text{-stgy } S T \implies cdcl_W\text{-merge-with-restart } (S, n) (T, Suc n) \rangle$

lemma $cdcl_W$ -merge-with-restart-rtranclp- $cdcl_W$ -restart:

$\langle cdcl_W\text{-merge-with-restart } S T \implies cdcl_W\text{-restart}^{**} (fst S) (fst T) \rangle$
 $\langle proof \rangle$

lemma $cdcl_W$ -merge-with-restart-increasing-number:

$\langle cdcl_W\text{-merge-with-restart } S T \implies snd T = 1 + snd S \rangle$
 $\langle proof \rangle$

lemma $\langle full1 cdcl_W\text{-stgy } S T \implies cdcl_W\text{-merge-with-restart } (S, n) (T, Suc n) \rangle$

$\langle proof \rangle$

lemma $cdcl_W$ -all-struct-inv-learned-classes-bound:

assumes inv : $\langle cdcl_W\text{-all-struct-inv } S \rangle$

shows $\langle set\text{-mset} (learned\text{-class } S) \subseteq simple\text{-class} (atms\text{-of-mm} (init\text{-class } S)) \rangle$

$\langle proof \rangle$

lemma $cdcl_W$ -merge-with-restart-init-class:

$\langle cdcl_W\text{-merge-with-restart } S T \implies cdcl_W\text{-M-level-inv } (fst S) \implies$
 $init\text{-class } (fst S) = init\text{-class } (fst T) \rangle$

$\langle proof \rangle$

lemma (**in** $cdcl_W$ -restart-restart)

$\langle wf \{(T, S). cdcl_W\text{-all-struct-inv } (fst S) \wedge cdcl_W\text{-merge-with-restart } S T\} \rangle$

$\langle proof \rangle$

lemma $cdcl_W$ -merge-with-restart-distinct-mset-clauses:

assumes $invR$: $\langle cdcl_W\text{-all-struct-inv } (fst R) \rangle$ **and**

st : $\langle cdcl_W\text{-merge-with-restart } R S \rangle$ **and**

$dist$: $\langle distinct\text{-mset} (clauses (fst R)) \rangle$ **and**

R : $\langle no\text{-smaller-propa } (fst R) \rangle$

shows $\langle distinct\text{-mset} (clauses (fst S)) \rangle$

$\langle proof \rangle$

inductive $cdcl_W$ -restart-with-restart **where**

restart-step:

$\langle cdcl_W\text{-stgy}^{**} S T \implies$
 $card (set\text{-mset} (learned\text{-class } T)) - card (set\text{-mset} (learned\text{-class } S)) > f n \implies$
 $restart T U \implies$

$cdcl_W\text{-restart-with-restart } (S, n) (U, Suc n) \rangle |$

restart-full: $\langle full1 cdcl_W\text{-stgy } S T \implies cdcl_W\text{-restart-with-restart } (S, n) (T, Suc n) \rangle$

lemma $cdcl_W$ -restart-with-restart-rtranclp- $cdcl_W$ -restart:

$\langle cdcl_W\text{-restart-with-restart } S T \implies cdcl_W\text{-restart}^{**} (fst S) (fst T) \rangle$
 $\langle proof \rangle$

lemma $cdcl_W$ -restart-with-restart-increasing-number:

$\langle cdcl_W\text{-restart-with-restart } S T \implies snd T = 1 + snd S \rangle$

⟨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} - 1) ∨ i = 2^{k+1} - 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} ≤ i and i ≤ (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 : ⟨ $2^{(k - \text{Suc } 0)} \leq i$ ⟩ **and**

k' : ⟨ $i < 2^{k' - \text{Suc } 0}$ ⟩ **and**

$k-k'$: ⟨ $k > k'$ ⟩

shows ⟨*False*⟩

⟨proof⟩

lemma *luby-sequence-core-not-two-power-minus-one*:

assumes

$k-i$: ⟨ $2^{(k - 1)} \leq i$ ⟩ **and**

$i-k$: ⟨ $i < 2^k - 1$ ⟩

shows ⟨*luby-sequence-core* $i = \text{luby-sequence-core } (i - 2^{(k - 1)} + 1)$ ⟩

⟨proof⟩

lemma *unbounded-luby-sequence-core*: ⟨*unbounded luby-sequence-core*⟩

⟨proof⟩

abbreviation *luby-sequence* :: ⟨*nat* \Rightarrow *nat*⟩ **where**

⟨*luby-sequence* $n \equiv \text{ur} * \text{luby-sequence-core } n$ ⟩

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

⟨proof⟩

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

⟨proof⟩

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

⟨proof⟩

end

locale *luby-sequence-restart* =

luby-sequence *ur* +

*conflict-driven-clause-learning*_W

— functions for the state:

state-eq *state*

— access functions:

trail *init-clss* *learned-clss* *conflicting*

— changing state:

cons-trail *tl-trail* *add-learned-cl* *remove-cl*

update-conflicting

— get state:

init-state

for

ur :: *nat* **and**

state-eq :: ⟨*'st* \Rightarrow *'st* \Rightarrow *bool*⟩ (infix ~ 50) **and**

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

trail :: ⟨*'st* \Rightarrow (*'v*, *'v clause*) *ann-lits*⟩ **and**

```

hd-trail :: ⟨'st ⇒ ('v, 'v clause) ann-lit⟩ and
init-cls :: ⟨'st ⇒ 'v clauses⟩ and
learned-cls :: ⟨'st ⇒ 'v clauses⟩ and
conflicting :: ⟨'st ⇒ 'v clause option⟩ and

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

init-state :: ⟨'v clauses ⇒ 'st⟩
begin

sublocale cdclW-restart-restart where
  f = luby-sequence
  ⟨proof⟩

end

end
theory CDCL-W-Incremental
imports CDCL-W-Full
begin

```

3.2 Incremental SAT solving

```

locale stateW-adding-init-clause-no-state =
  stateW-no-state
  state-eq
  state
  — functions about the state:
  — getter:
  trail init-cls learned-cls 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-cls :: 'st ⇒ 'v clauses and
  learned-cls :: '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 stateW-adding-init-clause-ops =
stateW-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

locale stateW-adding-init-clause =
stateW-adding-init-clause-ops
state-eq
state
— functions about the state:
— getter:
trail init-clss learned-clss conflicting
— setter:
cons-trail tl-trail add-learned-cls remove-cls update-conflicting

— Some specific states:
init-state add-init-cls +
stateW
state-eq
state
— functions about the state:
— getter:

```

trail init-clss learned-clss conflicting

— setter:

cons-trail tl-trail add-learned-clss remove-clss 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-clss :: 'v clause ⇒ 'st ⇒ 'st **and**

remove-clss :: 'v clause ⇒ 'st ⇒ 'st **and**

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

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

add-init-clss :: 'v clause ⇒ 'st ⇒ 'st

begin

lemma

trail-add-init-clss[simp]:

trail (add-init-clss C st) = *trail* st **and**

init-clss-add-init-clss[simp]:

init-clss (add-init-clss C st) = {#C#} + *init-clss* st **and**

learned-clss-add-init-clss[simp]:

learned-clss (add-init-clss C st) = *learned-clss* st **and**

conflicting-add-init-clss[simp]:

conflicting (add-init-clss C st) = *conflicting* st

⟨proof⟩

lemma *clauses-add-init-clss*[simp]:

clauses (add-init-clss N S) = {#N#} + *init-clss* S + *learned-clss* S

⟨proof⟩

lemma *reduce-trail-to-add-init-clss*[simp]:

trail (reduce-trail-to F (add-init-clss C S)) = *trail* (reduce-trail-to F S)

⟨proof⟩

lemma *conflicting-add-init-clss-iff-conflicting*[simp]:

conflicting (add-init-clss 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-clss remove-clss update-conflicting

— get state:

init-state

— Adding a clause:

add-init-clss

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-clss :: 'v clause ⇒ 'st ⇒ 'st **and**

remove-clss :: 'v clause ⇒ 'st ⇒ 'st **and**

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

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

add-init-clss :: '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 the first literal included in C. Then we can mark the conflict.

fun (**in** *state_W*) *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 (**in** *state_W*) *reduce-trail-wrt-clause* :: 'v clause ⇒ 'st ⇒ 'st **where**

reduce-trail-wrt-clause C S = *update-conflicting* (Some C) (*cut-trail-wrt-clause* C (trail S) S)

definition *add-new-clause-and-update* :: 'v clause ⇒ 'st ⇒ 'st **where**

add-new-clause-and-update C S = *reduce-trail-wrt-clause* C (*add-init-clss* C S)

lemma (**in** *state_W*) *init-clss-cut-trail-wrt-clause[simp]*:

init-clss (*cut-trail-wrt-clause* C M S) = *init-clss* S

⟨*proof*⟩

lemma (**in** *state_W*) *learned-clss-cut-trail-wrt-clause[simp]*:

learned-clss (*cut-trail-wrt-clause* C M S) = *learned-clss* S

⟨proof⟩

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

lemma (in $state_W$) *clauses-cut-trail-wrt-clause[simp]*:
clauses (cut-trail-wrt-clause $C M S$) = clauses S
⟨proof⟩

lemma (in $state_W$) *trail-cut-trail-wrt-clause*:
 $\exists M. trail S = M @ trail (cut-trail-wrt-clause C (trail S) S)$
⟨proof⟩

lemma (in $state_W$) *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 *trail-cut-trail-wrt-clause-mono*:
⟨trail $S = trail T \implies trail (cut-trail-wrt-clause C M S) =$
 $trail (cut-trail-wrt-clause C M T)$ ⟩
⟨proof⟩

lemma *trail-cut-trail-wrt-clause-add-init-cls[simp]*:
⟨trail (cut-trail-wrt-clause $C M (add-init-cls C S)$) =
trail (cut-trail-wrt-clause $C M S$)⟩
⟨proof⟩

lemma (in $state_W$) *cut-trail-wrt-clause-CNot-trail*:
assumes $trail T \models_{as} CNot C$
shows
 $(trail ((cut-trail-wrt-clause C (trail T) T))) \models_{as} CNot C$
⟨proof⟩

lemma (in $state_W$) *cut-trail-wrt-clause-hd-trail-in-or-empty-trail*:
 $(\forall L \in \#C. -L \notin lits-of-l (trail T)) \wedge trail (cut-trail-wrt-clause C (trail T) T) = []$
 $\vee (-lit-of (hd (trail (cut-trail-wrt-clause C (trail T) T))) \in \# C$
 $\wedge length (trail (cut-trail-wrt-clause C (trail T) T)) \geq 1)$
⟨proof⟩

The following function allows to mark a conflict while backtrack at the correct position.

inductive (in $state_W$) *cdcl_W-OOO-conflict* :: ⟨ $'st \Rightarrow 'st \Rightarrow bool$ ⟩ for $S :: 'st$ where
cdcl_W-OOO-conflict-rule: ⟨ $cdcl_W\text{-OOO-conflict } S T$ ⟩

if

⟨ $trail S \models_{as} CNot C$ ⟩ and
⟨ $C \in \# clauses S$ ⟩ and
⟨ $conflicting S = None$ ⟩
⟨ $T \sim reduce-trail-wrt-clause C S$ ⟩

lemma (in *conflict-driven-clause-learning_W*)
cdcl_W-all-struct-inv-add-new-clause-and-update-cdcl_W-all-struct-inv:
assumes
 $inv-T: cdcl_W\text{-all-struct-inv } T$ and
 $tr-C[simp]: trail T \models_{as} CNot C$ and

[simp]: *distinct-mset C* **and**
C: ⟨C ∈# clauses T⟩
shows *cdcl_W-all-struct-inv (reduce-trail-wrt-clause C T) (is cdcl_W-all-struct-inv ?T)*
 ⟨proof⟩

lemma (in *conflict-driven-clause-learning_W*) *cdcl_W-OOO-conflict-is-conflict*:
assumes *⟨cdcl_W-OOO-conflict S U⟩*
shows *⟨conflict (cut-trail-wrt-clause (the (conflicting U)) (trail S) S) U⟩*
 ⟨proof⟩

lemma (in *conflict-driven-clause-learning_W*) *cdcl_W-OOO-conflict-all-struct-invs*:
assumes *⟨cdcl_W-OOO-conflict S T⟩* **and** *⟨cdcl_W-all-struct-inv S⟩*
shows *⟨cdcl_W-all-struct-inv T⟩*
 ⟨proof⟩

lemma (in *–*) *get-maximum-level-Cons-notin*:
⟨– lit-of L ∉# C ⟹ lit-of L ∉# C ⟹ get-maximum-level M C = get-maximum-level (L # M) C⟩
 ⟨proof⟩

lemma (in *state_W*) *backtrack-lvl-cut-trail-wrt-clause-get-maximum-level*:
⟨M = trail S ⟹ M ⊨_{as} CNot D ⟹ no-dup (trail S) ⟹
backtrack-lvl (cut-trail-wrt-clause D M S) = get-maximum-level M D⟩
 ⟨proof⟩

lemma (in *state_W*) *get-maximum-level-cut-trail-wrt-clause*:
⟨M = trail S ⟹ M ⊨_{as} CNot C ⟹ no-dup (trail S) ⟹
get-maximum-level (trail (cut-trail-wrt-clause C M S)) C =
get-maximum-level M C⟩
 ⟨proof⟩

lemma *cdcl_W-OOO-conflict-conflict-is-false-with-level*:
assumes *⟨cdcl_W-OOO-conflict S T⟩* **and** *⟨cdcl_W-all-struct-inv S⟩*
shows *⟨conflict-is-false-with-level T⟩*
 ⟨proof⟩

We can fully run *cdcl_W-stgy* or add a clause.

Compared to a previous I changed the order and replaced *update-conflicting (Some C) (add-init-cls C (cut-trail-wrt-clause C (trail S) S))* (like in my thesis) by *update-conflicting (Some C) (cut-trail-wrt-clause C (trail S) (add-init-cls C S))*. The motivation behind it is that adding clause first makes it fallback on conflict (with backtracking, but it is still a conflict) and, therefore, seems more regular than the opposite order.

inductive *incremental-cdcl_W* :: *'st ⇒ 'st ⇒ bool* **for** *S* **where**

add-conf:

trail S ⊨_{asm} init-cls S ⟹ distinct-mset C ⟹ conflicting S = None ⟹
trail S ⊨_{as} CNot C ⟹
full cdcl_W-stgy
(update-conflicting (Some C)
(cut-trail-wrt-clause C (trail S) (add-init-cls C S))) T ⟹
incremental-cdcl_W S T |

add-no-conf:

trail S ⊨_{asm} init-cls 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-init-cls*:

⟨*cdcl_W-all-struct-inv* (*T*) \implies *distinct-mset* *C* \implies *cdcl_W-all-struct-inv* (*add-init-cl* *C* *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* {*#*} \wedge *unsatisfiable* (*set-mset* (*init-clss* *T*))

\vee *conflicting* *T* = *None* \wedge *trail* *T* \models_{asm} *init-clss* *T* \wedge *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  $T = \text{Some } \{\#\} \wedge \text{unsatisfiable } (\text{set-mset } (\text{init-cls } T))$   
   $\vee \text{conflicting } T = \text{None} \wedge \text{trail } T \models_{\text{asm}} \text{init-cls } T \wedge \text{satisfiable } (\text{set-mset } (\text{init-cls } T))$   
<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 and simply iterate over-and-over on lists.

4.1.1 Common Rules

Propagation

The following theorem holds:

lemma *lits-of-l-unfold*:

$$(\forall c \in \text{set } C. -c \in \text{lits-of-l } Ms) \longleftrightarrow Ms \models_{\text{as}} \text{CNot } (\text{mset } C)$$

<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 \Rightarrow ('a, 'b) ann-lits \Rightarrow 'a literal option

where

is-unit-clause l M =

(case *List.filter* ($\lambda a. \text{atm-of } a \notin \text{atm-of ' lits-of-l } M$) l of
 a # [] \Rightarrow if M \models_{as} CNot (mset l - {#a#}) then Some a else None
 | - \Rightarrow None)

definition *is-unit-clause-code* :: 'a literal list \Rightarrow ('a, 'b) ann-lits

\Rightarrow 'a literal option **where**

is-unit-clause-code l M =

(case *List.filter* ($\lambda a. \text{atm-of } a \notin \text{atm-of ' lits-of-l } M$) l of
 a # [] \Rightarrow if ($\forall c \in \text{set } (\text{remove1 } a \text{ l}). -c \in \text{lits-of-l } M$) then Some a else None
 | - \Rightarrow None)

lemma *is-unit-clause-is-unit-clause-code*[code]:
is-unit-clause l $M = is-unit-clause-code$ 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 $M = Some$ $a \implies M \models_{as} CNot$ ($mset$ $l - \{ \#a\# \}$)
 ⟨proof⟩

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

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

Unit propagation for all clauses

Finding the first clause to propagate

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

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

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

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

Decide

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

find-first-unused-var [] - = None

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

List.find ($\lambda lit. lit \notin M \wedge \neg lit \notin M$) *a* = None \longleftrightarrow *atm-of* ' *set a* \subseteq *atm-of* ' *M*
<proof>

lemma *find-some*: *List.find* ($\lambda lit. lit \notin M \wedge \neg lit \notin M$) *a* = *Some b* $\implies b \in set\ a \wedge b \notin M \wedge \neg b \notin M$

<proof>

lemma *find-first-unused-var-None*[*iff*]:

find-first-unused-var l M = None $\longleftrightarrow (\forall a \in set\ l. atm-of\ ' set\ a \subseteq atm-of\ ' M)$
<proof>

lemma *find-first-unused-var-Some-not-all-incl*:

assumes *find-first-unused-var l M* = *Some c*

shows $\neg(\forall a \in set\ l. atm-of\ ' set\ a \subseteq atm-of\ ' M)$

<proof>

lemma *find-first-unused-var-Some*:

find-first-unused-var l M = *Some a* $\implies (\exists m \in set\ l. a \in set\ m \wedge a \notin M \wedge \neg a \notin M)$

<proof>

lemma *find-first-unused-var-undefined*:

find-first-unused-var l (lits-of-l Ms) = *Some a* $\implies undefined-lit\ Ms\ a$

<proof>

4.1.2 CDCL specific functions

Level

fun *maximum-level-code*:: 'a literal list \Rightarrow ('a, 'b) ann-lits \Rightarrow nat

where

maximum-level-code [] - = 0 |

maximum-level-code (L # Ls) M = max (get-level M L) (maximum-level-code Ls M)

lemma *maximum-level-code-eq-get-maximum-level*[*simp*]:

maximum-level-code D M = *get-maximum-level M (mset D)*

<proof>

lemma [*code*]:

fixes M :: ('a, 'b) ann-lits

shows *get-maximum-level M (mset D)* = *maximum-level-code D M*

<proof>

Backjumping

fun *find-level-decomp* **where**

find-level-decomp M [] D k = None |

find-level-decomp M (L # Ls) D k =

(case (get-level M L, maximum-level-code (D @ Ls) M) of

(i, j) \Rightarrow if i = k \wedge j < i then *Some (L, j)* else *find-level-decomp M Ls (L#D) k*

)

lemma *find-level-decomp-some*:

assumes *find-level-decomp M Ls D k* = *Some (L, j)*

shows L \in set Ls \wedge *get-maximum-level M (mset (remove1 L (Ls @ D)))* = j \wedge *get-level M L* = k

<proof>

lemma *find-level-decomp-none*:

assumes *find-level-decomp* $M\ Ls\ E\ k = None$ **and** $mset\ (L\#D) = mset\ (Ls\ @\ E)$

shows $\neg(L \in set\ Ls \wedge get\text{-}maximum\text{-}level\ M\ (mset\ D) < k \wedge k = get\text{-}level\ M\ L)$

<proof>

fun *bt-cut* **where**

bt-cut $i\ (Propagated\ -\ -\ \# Ls) = bt\text{-}cut\ i\ Ls\ |$

bt-cut $i\ (Decided\ K\ \# Ls) = (if\ count\text{-}decided\ Ls = i\ then\ Some\ (Decided\ K\ \# Ls)\ else\ bt\text{-}cut\ i\ Ls)\ |$

bt-cut $i\ [] = None$

lemma *bt-cut-some-decomp*:

assumes *no-dup* M **and** $bt\text{-}cut\ i\ M = Some\ M'$

shows $\exists K\ M2\ M1. M = M2\ @\ M' \wedge M' = Decided\ K\ \# M1 \wedge get\text{-}level\ M\ K = (i+1)$

<proof>

lemma *bt-cut-not-none*:

assumes *no-dup* M **and** $M = M2\ @\ Decided\ K\ \# M'$ **and** $get\text{-}level\ M\ K = (i+1)$

shows $bt\text{-}cut\ i\ M \neq None$

<proof>

lemma *get-all-ann-decomposition-ex*:

$\exists N. (Decided\ K\ \# M', N) \in set\ (get\text{-}all\text{-}ann\text{-}decomposition\ (M2@Decided\ K\ \# M'))$

<proof>

lemma *bt-cut-in-get-all-ann-decomposition*:

assumes *no-dup* M **and** $bt\text{-}cut\ i\ M = Some\ M'$

shows $\exists M2. (M', M2) \in set\ (get\text{-}all\text{-}ann\text{-}decomposition\ M)$

<proof>

fun *do-backtrack-step* **where**

do-backtrack-step $(M, N, U, Some\ D) =$

$(case\ find\text{-}level\text{-}decomp\ M\ D\ []\ (count\text{-}decided\ M)\ of$

$None \Rightarrow (M, N, U, Some\ D)$

$| Some\ (L, j) \Rightarrow$

$(case\ bt\text{-}cut\ j\ M\ of$

$Some\ (Decided\ -\ \# Ls) \Rightarrow (Propagated\ L\ D\ \# Ls, N, D\ \# U, None)$

$| - \Rightarrow (M, N, U, Some\ D))$

$)\ |$

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* $:: int\ dpll_W\text{-}ann\text{-}lits \times int\ literal\ list\ list$

$\Rightarrow int\ dpll_W\text{-}ann\text{-}lits \times int\ literal\ list\ list$ **where**

$DPLL\text{-}step = (\lambda(Ms, N).$

$(case\ find\text{-}first\text{-}unit\text{-}clause\ N\ Ms\ of$

$Some\ (L, -) \Rightarrow (Propagated\ L\ ()\ \# Ms, N)$


```

| - ⇒
  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 $toS \equiv \lambda(Ms::(int, unit) \text{ ann-lits})$
 $(N:: \text{int literal list list}). (Ms, mset (\text{map } mset N))$

abbreviation $toS' \equiv \lambda(Ms::(int, unit) \text{ ann-lits},$
 $N:: \text{int literal list list}). (Ms, mset (\text{map } mset N))$

Proof of correctness of *DPLL-step*

lemma *DPLL-step-is-a-dpll_W-step*:

assumes $step: (Ms', N') = DPLL\text{-step } (Ms, N)$

and $neg: (Ms, N) \neq (Ms', N')$

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 $conclusive\text{-dpll}_W\text{-state } (toS Ms N)$

<proof>

Adding invariants

Invariant tested in the function **function** $DPLL\text{-ci} :: \text{int } dpll_W\text{-ann-lits} \Rightarrow \text{int literal list list}$
 $\Rightarrow \text{int } dpll_W\text{-ann-lits} \times \text{int literal list list}$ **where**

$DPLL\text{-ci } Ms N =$

(if $\neg dpll_W\text{-all-inv } (Ms, mset (\text{map } mset N))$

then (Ms, N)

else

let $(Ms', N') = DPLL\text{-step } (Ms, N)$ in

if $(Ms', N') = (Ms, N)$ then (Ms, N) else $DPLL\text{-ci } Ms' N$)

<proof>

termination

<proof>

No invariant tested **function** (*domintros*) $DPLL\text{-part} :: \text{int } dpll_W\text{-ann-lits} \Rightarrow \text{int literal list list} \Rightarrow$
 $\text{int } dpll_W\text{-ann-lits} \times \text{int literal list list}$ **where**

$DPLL\text{-part } Ms N =$

(let $(Ms', N') = DPLL\text{-step } (Ms, N)$ in

if $(Ms', N') = (Ms, N)$ then (Ms, N) else $DPLL\text{-part } Ms' N$)

<proof>

lemma *snd-DPLL-step[simp]*:
snd (*DPLL-step* (*Ms*, *N*)) = *N*
 ⟨*proof*⟩

lemma *dpll_W-all-inv-implicS-2-eq3-and-dom*:
assumes *dpll_W-all-inv* (*Ms*, *mset* (*map mset N*))
shows *DPLL-ci Ms N* = *DPLL-part Ms N* \wedge *DPLL-part-dom* (*Ms*, *N*)
 ⟨*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*) \wedge *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 (\square , N)

<proof>

Some type classes instantiation *dpll_W-state* :: *equal*

begin

definition *equal-dpll_W-state* :: *dpll_W-state* \Rightarrow *dpll_W-state* \Rightarrow *bool* **where**

equal-dpll_W-state $S S' = (\text{rough-state-of } S = \text{rough-state-of } S')$

instance

<proof>

end

DPLL definition *DPLL-step'* :: *dpll_W-state* \Rightarrow *dpll_W-state* **where**

DPLL-step' $S = \text{state-of } (\text{DPLL-step } (\text{rough-state-of } S))$

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

lemma *DPLL-step-dpll_W-conc-inv*:

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

<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* \Rightarrow *dpll_W-state* **where**

DPLL-tot $S =$

(*let* $S' = \text{DPLL-step}' S$ *in*

if $S' = S$ *then* S *else* *DPLL-tot* $S')$

<proof>

termination

<proof>

lemma [*code*]:

DPLL-tot $S =$

(*let* $S' = \text{DPLL-step}' S$ *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* (\square , *N*)) = (\square , *N*)
 ⟨*proof*⟩

Theorem of correctness

lemma *DPLL-tot-correct*:
assumes *rough-state-of* (*DPLL-tot* (*state-of* (\square , *N*))) = (*M*, *N'*)
and (*M'*, *N''*) = *toS'* (*M*, *N'*)
shows *M'* \models_{asm} *N''* \longleftrightarrow *satisfiable* (*set-mset* *N''*)
 ⟨*proof*⟩

Code export

A conversion to DPLL-W-Implementation. *dpll_W-state* **definition** *Con* :: (*int*, *unit*) *ann-lits* \times *int literal list list*

\Rightarrow *dpll_W-state* **where**

Con *xs* = *state-of* (*if* *dpll_W-all-inv* (*toS* (*fst* *xs*) (*snd* *xs*)) *then* *xs* *else* (\square , \square))

lemma [*code abstype*]:
Con (*rough-state-of* *S*) = *S*
 ⟨*proof*⟩

declare *rough-state-of-DPLL-step'-DPLL-step*[*code abstract*]

lemma *Con-DPLL-step-rough-state-of-state-of*[*simp*]:
Con (*DPLL-step* (*rough-state-of* *s*)) = *state-of* (*DPLL-step* (*rough-state-of* *s*))
 ⟨*proof*⟩

A slightly different version of *DPLL-tot* where the returned boolean indicates the result.

definition *DPLL-tot-rep* **where**

DPLL-tot-rep *S* =

(*let* (*M*, *N*) = (*rough-state-of* (*DPLL-tot* *S*)) *in* ($\forall A \in \text{set } N. (\exists a \in \text{set } A. a \in \text{lits-of-l } M), M$))

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

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

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

end

theory *CDCL-W-Implementation*

imports *DPLL-CDCL-W-Implementation* *CDCL-W-Termination*
HOL-Library.Code-Target-Numeral

begin

4.1.4 List-based CDCL Implementation

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

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

Types and Instantiation

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

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

type-synonym *'v cdcl_W-restart-ann-lit* = (*'v*, *'v cdcl_W-restart-mark*) *ann-lit*

type-synonym *'v cdcl_W-restart-ann-lits* = (*'v*, *'v cdcl_W-restart-mark*) *ann-lits*

type-synonym *'v cdcl_W-restart-state* =
'v cdcl_W-restart-ann-lits × *'v clauses* × *'v clauses* × *'v clause option*

abbreviation *raw-trail* :: *'a* × *'b* × *'c* × *'d* ⇒ *'a where*

raw-trail ≡ (λ(*M*, -). *M*)

abbreviation *raw-cons-trail* :: *'a* ⇒ *'a list* × *'b* × *'c* × *'d* ⇒ *'a list* × *'b* × *'c* × *'d*

where

raw-cons-trail ≡ (λ*L* (*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 ≡ λ*S* (*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 ≡ λ*C* (*M*, *N*, *U*, *S*). (*M*, *N*, {#*C*#} + *U*, *S*)

abbreviation *raw-remove-cls where*

raw-remove-cls ≡ λ*C* (*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*:

$S = (\text{raw-trail } S, \text{raw-init-clss } S, \text{raw-learned-clss } S, \text{raw-conflicting } S)$
 $\langle \text{proof} \rangle$

definition *state where*

$\langle \text{state } S = (\text{raw-trail } S, \text{raw-init-clss } S, \text{raw-learned-clss } S, \text{raw-conflicting } S, ()) \rangle$

interpretation *state_W*

(=)
state
raw-trail raw-init-clss raw-learned-clss raw-conflicting
 $\lambda L (M, S). (L \# M, S)$
 $\lambda (M, S). (\text{tl } M, S)$
 $\lambda C (M, N, U, S). (M, N, \text{add-mset } C U, S)$
 $\lambda C (M, N, U, S). (M, \text{removeAll-mset } C N, \text{removeAll-mset } C U, S)$
 $\lambda D (M, N, U, -). (M, N, U, D)$
 $\lambda N. ([], N, \{\#\}, \text{None})$
 $\langle \text{proof} \rangle$

declare *state-simp*[*simp del*]

interpretation *conflict-driven-clause-learning_W*

(=) *state*
raw-trail raw-init-clss raw-learned-clss
raw-conflicting
 $\lambda L (M, S). (L \# M, S)$
 $\lambda (M, S). (\text{tl } M, S)$
 $\lambda C (M, N, U, S). (M, N, \text{add-mset } C U, S)$
 $\lambda C (M, N, U, S). (M, \text{removeAll-mset } C N, \text{removeAll-mset } C U, S)$
 $\lambda D (M, N, U, -). (M, N, U, D)$
 $\lambda N. ([], N, \{\#\}, \text{None})$
 $\langle \text{proof} \rangle$

declare *clauses-def*[*simp*]

lemma *reduce-trail-to-empty-trail*[*simp*]:

$\text{reduce-trail-to } F ([], aa, ab, b) = ([], aa, ab, b)$
 $\langle \text{proof} \rangle$

lemma *reduce-trail-to'*:

$\text{reduce-trail-to } F S =$
 $((\text{if length } (\text{raw-trail } S) \geq \text{length } F$
 $\text{then drop } (\text{length } (\text{raw-trail } S) - \text{length } F) (\text{raw-trail } S)$
 $\text{else } []), \text{raw-init-clss } S, \text{raw-learned-clss } S, \text{raw-conflicting } S)$
 $(\text{is } ?S = -)$

$\langle \text{proof} \rangle$

Definition of the rules

Types **lemma** *true-raw-init-clss-remdups*[*simp*]:

$I \models s (\text{mset} \circ \text{remdups}) ' N \longleftrightarrow I \models s \text{mset} ' N$
 $\langle \text{proof} \rangle$

lemma *true-clss-raw-remdups-mset-mset*[*simp*]:

$\langle I \models s (\lambda L. \text{remdups-mset } (\text{mset } L)) ' N' \longleftrightarrow I \models s \text{mset} ' N' \rangle$

⟨proof⟩

declare *satisfiable-carac*[*iff del*]

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

satisfiable ((*mset* ∘ *remdups*) ‘ *N*) \longleftrightarrow *satisfiable* (*mset* ‘ *N*)

satisfiable ((λ*L*. *remdups-mset* (*mset* *L*)) ‘ *N*) \longleftrightarrow *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 *atn-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 $M [] = \text{None}$ |

find-conflict $M (N \# Ns) = (\text{if } (\forall c \in \text{set } N. \neg c \in \text{ lits-of-}l\ M) \text{ then } \text{Some } N \text{ else } \text{find-conflict } M\ Ns)$

lemma *find-conflict-Some*:

find-conflict $M\ Ns = \text{Some } N \implies N \in \text{set } Ns \wedge M \models_{\text{as}} \text{CNot } (\text{mset } N)$

<proof>

lemma *find-conflict-None*:

find-conflict $M\ Ns = \text{None} \iff (\forall N \in \text{set } Ns. \neg M \models_{\text{as}} \text{CNot } (\text{mset } N))$

<proof>

lemma *find-conflict-None-no-conf*:

find-conflict $M (N @ U) = \text{None} \iff \text{no-step conflict } (\text{toS } (M, N, U, \text{None}))$

<proof>

definition *do-conflict-step* :: *'v cdcl_W-restart-state-inv-st \Rightarrow 'v cdcl_W-restart-state-inv-st* **where**

do-conflict-step $S =$

(*case* S of

$(M, N, U, \text{None}) \Rightarrow$

(*case* *find-conflict* $M (N @ U)$ of

$\text{Some } a \Rightarrow (M, N, U, \text{Some } a)$

| $\text{None} \Rightarrow (M, N, U, \text{None})$)

| $S \Rightarrow S$)

lemma *do-conflict-step*:

do-conflict-step $S \neq S \implies \text{conflict } (\text{toS } S) (\text{toS } (\text{do-conflict-step } S))$

<proof>

lemma *do-conflict-step-no-step*:

do-conflict-step $S = S \implies \text{no-step conflict } (\text{toS } S)$

<proof>

lemma *do-conflict-step-option[simp]*:

raw-conflicting $S \neq \text{None} \implies \text{do-conflict-step } S = S$

<proof>

lemma *do-conflict-step-raw-conflicting[dest]*:

do-conflict-step $S \neq S \implies \text{raw-conflicting } (\text{do-conflict-step } S) \neq \text{None}$

<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) $\implies \text{rough-state-of } (\text{state-of } S) = S$

<proof>

Skip fun *do-skip-step* :: *'v cdcl_W-restart-state-inv-st \Rightarrow 'v cdcl_W-restart-state-inv-st* **where**

do-skip-step (*Propagated* $L\ C \# Ls, N, U, \text{Some } D$) =

(if $-L \notin \text{set } D \wedge D \neq []$
 then $(Ls, N, U, \text{Some } D)$
 else $(\text{Propagated } L \ C \ \#Ls, N, U, \text{Some } D))$ |
 do-skip-step $S = S$

lemma do-skip-step:
 do-skip-step $S \neq S \implies \text{skip } (\text{toS } S) (\text{toS } (\text{do-skip-step } S))$
 <proof>

lemma do-skip-step-no:
 do-skip-step $S = S \implies \text{no-step skip } (\text{toS } S)$
 <proof>

lemma do-skip-step-raw-trail-is-None[iff]:
 do-skip-step $S = (a, b, c, \text{None}) \longleftrightarrow S = (a, b, c, \text{None})$
 <proof>

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

lemma maximum-level-code-eq-get-maximum-level[code, simp]:
 maximum-level-code $D \ M = \text{get-maximum-level } M (\text{mset } D)$
 <proof>

fun do-resolve-step :: 'v cdcl_W-restart-state-inv-st \Rightarrow 'v cdcl_W-restart-state-inv-st **where**
 do-resolve-step $(\text{Propagated } L \ C \ \# \ Ls, N, U, \text{Some } D) =$
 (if $-L \in \text{set } D \wedge \text{maximum-level-code } (\text{remove1 } (-L) \ D) (\text{Propagated } L \ C \ \# \ Ls) = \text{count-decided } Ls$
 then $(Ls, N, U, \text{Some } (\text{remdups } (\text{remove1 } L \ C \ @ \ \text{remove1 } (-L) \ D)))$
 else $(\text{Propagated } L \ C \ \# \ Ls, N, U, \text{Some } D))$ |
 do-resolve-step $S = S$

lemma do-resolve-step:
 cdcl_W-all-struct-inv $(\text{toS } S) \implies \text{do-resolve-step } S \neq S$
 $\implies \text{resolve } (\text{toS } S) (\text{toS } (\text{do-resolve-step } S))$
 <proof>

lemma do-resolve-step-no:
 do-resolve-step $S = S \implies \text{no-step resolve } (\text{toS } S)$
 <proof>

lemma rough-state-of-state-of-resolve[simp]:
 cdcl_W-all-struct-inv $(\text{toS } S) \implies$
 rough-state-of $(\text{state-of } (\text{do-resolve-step } S)) = \text{do-resolve-step } S$
 <proof>

lemma do-resolve-step-raw-trail-is-None[iff]:
 do-resolve-step $S = (a, b, c, \text{None}) \longleftrightarrow S = (a, b, c, \text{None})$
 <proof>

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

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

lemma *map-eq-list-length*:
 $\text{map } f \ L = L' \implies \text{length } L = \text{length } L'$
 \langle *proof* \rangle

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

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

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: \langle *no-step skip* (*toS* S) \rangle \langle *no-step resolve* (*toS* S) \rangle
shows *no-step backtrack* (*toS* S)
 \langle *proof* \rangle

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
 \langle *proof* \rangle

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

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

lemma *do-decide-step-no*:
 $\text{do-decide-step } S = S \implies \text{no-step decide } (\text{toS } S)$
 \langle *proof* \rangle

lemma *rough-state-of-state-of-do-decide-step*[*simp*]:

$cdcl_W\text{-all-struct-inv } (toS S) \implies \text{rough-state-of } (state-of (do-decide-step S)) = do-decide-step S$
 ⟨proof⟩

lemma *rough-state-of-state-of-do-skip-step*[simp]:

$cdcl_W\text{-all-struct-inv } (toS S) \implies \text{rough-state-of } (state-of (do-skip-step S)) = do-skip-step S$
 ⟨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-cls (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-cls (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:*

$c \in \text{set } (\text{raw-init-cls } (\text{rough-state-of } S) @ \text{raw-learned-cls } (\text{rough-state-of } S)) \implies \text{distinct } c$
 ⟨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 $T \neq S$ then T else *do-if-not-equal* $fs S$)

fun *do-cdcl-step* **where**

do-cdcl-step $S =$
do-if-not-equal [*do-conflict-step*, *do-propagate-step*, *do-skip-step*, *do-resolve-step*,
do-backtrack-step, *do-decide-step*] S

lemma *do-cdcl-step*:

assumes *inv*: *cdcl_W-all-struct-inv* (*toS* S) **and**
st: *do-cdcl-step* $S \neq S$
shows *cdcl_W-stgy* (*toS* S) (*toS* (*do-cdcl-step* S))
 ⟨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* \implies ' 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' $S = \text{state-from-init-state-of } (\text{rough-state-of } (\text{do-cdcl}_W\text{-stgy-step } (\text{id-of-I-to } S)))$

Correction of the transformation **lemma** *do-cdcl_W-stgy-step*:

assumes *do-cdcl_W-stgy-step* $S \neq S$
shows *cdcl_W-stgy* (*toS* (*rough-state-of* S)) (*toS* (*rough-state-of* (*do-cdcl_W-stgy-step* S)))
 ⟨proof⟩

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) $\neq \text{None} \iff \text{raw-conflicting } S \neq \text{None}$
 ⟨proof⟩

lemma *raw-trail-toS-neq-imp-raw-trail-neq*:

$raw-trail (toS S) \neq raw-trail (toS S') \implies raw-trail S \neq raw-trail S'$
(proof)

lemma *do-cp-step-neq-raw-trail-increase*:

$\exists c. raw-trail (do-cp-step S) = c @ raw-trail S \wedge (\forall m \in set c. \neg is-decided m)$
(proof)

lemma *do-cp-step-raw-conflicting*:

$raw-conflicting (rough-state-of S) \neq None \implies do-cp-step' S = S$
(proof)

lemma *do-decide-step-not-raw-conflicting-one-more-decide*:

assumes

$raw-conflicting S = None$ **and**

$do-decide-step S \neq S$

shows $Suc (length (filter is-decided (raw-trail S)))$

$= length (filter is-decided (raw-trail (do-decide-step S)))$

(proof)

lemma *do-decide-step-not-raw-conflicting-one-more-decide-bt*:

assumes $raw-conflicting S \neq None$ **and**

$do-decide-step S \neq S$

shows $length (filter is-decided (raw-trail S)) < length (filter is-decided (raw-trail (do-decide-step S)))$

(proof)

lemma *count-decided-raw-trail-toS*:

$count-decided (raw-trail (toS S)) = count-decided (raw-trail S)$

(proof)

lemma *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]*:

$raw-conflicting (do-resolve-step S) = None \iff raw-conflicting S = None$

(proof)

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

$raw-conflicting (do-skip-step S) = None \iff raw-conflicting S = None$

(proof)

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

$raw-conflicting (do-decide-step S) = None \iff raw-conflicting S = None$

(proof)

lemma *raw-conflicting-do-backtrack-step-imp[simp]*:

$do-backtrack-step S \neq S \implies raw-conflicting (do-backtrack-step S) = None$

(proof)

lemma *do-skip-step-eq-iff-raw-trail-eq*:

$do-skip-step S = S \iff raw-trail (do-skip-step S) = raw-trail S$

(proof)

lemma *do-decide-step-eq-iff-raw-trail-eq*:

$do-decide-step S = S \iff raw-trail (do-decide-step S) = raw-trail S$

(proof)

lemma *do-backtrack-step-eq-iff-raw-trail-eq:*

assumes *no-dup* (*raw-trail S*)

shows *do-backtrack-step S = S* \longleftrightarrow *raw-trail (do-backtrack-step S) = raw-trail S*

<proof>

lemma *do-resolve-step-eq-iff-raw-trail-eq:*

do-resolve-step S = S \longleftrightarrow *raw-trail (do-resolve-step S) = raw-trail S*

<proof>

lemma *do-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 \implies *cdcl_W-restart** S T*

<proof>

lemma *cdcl_W-stgy-init-raw-init-cls:*

cdcl_W-stgy S T \implies *cdcl_W-M-level-inv S* \implies *raw-init-cls S = raw-init-cls T*

<proof>

lemma *clauses-toS-rough-state-of-do-cdcl_W-stgy-step[simp]:*

raw-init-cls (toS (rough-state-of (do-cdcl_W-stgy-step (state-of (rough-state-from-init-state-of S))))))

= raw-init-cls (toS (rough-state-from-init-state-of S)) (is - = raw-init-cls (toS ?S))

<proof>

lemma *rough-state-from-init-state-of-do-cdcl_W-stgy-step'[code abstract]:*

rough-state-from-init-state-of (do-cdcl_W-stgy-step' S) =

rough-state-of (do-cdcl_W-stgy-step (id-of-I-to S))

<proof>

All rules together function *do-all-cdcl_W-stgy* **where**

do-all-cdcl_W-stgy S =

(let T = do-cdcl_W-stgy-step' S in

if T = S then S else do-all-cdcl_W-stgy T)

<proof>

termination

<proof>

thm *do-all-cdcl_W-stgy.induct*

lemma *do-all-cdcl_W-stgy-induct:*

($\bigwedge S. (do-cdcl_W-stgy-step' S \neq S \implies P (do-cdcl_W-stgy-step' S)) \implies P S) \implies P a0$

<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-cls* 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-cls-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-cl +
abstract-with-index2 get-cl *mset-clss*

for

get-lit :: 'cls ⇒ 'lit ⇒ 'v literal option **and**
mset-cl :: 'cls ⇒ 'v clause **and**

get-cl :: '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-cl* :: 'clss ⇒ 'cls-it ⇒ 'cls (**infix** ↓ 49) **where**
C ↓ *a* ≡ *the (get-cl 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-cl Cs a* ≠ *None*

definition *raw-clss* **where**

raw-clss S ≡ *image-mset mset-cl (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 CDCL-W-Incremental*

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 *add-init-cls* **where**

add-init-cls $C (M, N, U, R) = (M, \{\#C\# \} + N, U, R)$

fun *remove-cls* **where**

remove-cls $C (M, N, U, R) = (M, \text{removeAll-mset } C \ N, \text{removeAll-mset } C \ U, R)$

fun *update-conflicting* **where**

update-conflicting $D (M, N, U, -) = (M, N, U, D)$

fun *init-state* **where**

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

declare *trail.simps*[*simp del*] *cons-trail.simps*[*simp del*] *tl-trail.simps*[*simp del*]

add-learned-cls.simps[*simp del*] *remove-cls.simps*[*simp del*]

update-conflicting.simps[*simp del*] *init-clss.simps*[*simp del*] *learned-clss.simps*[*simp del*]

conflicting.simps[*simp del*] *init-state.simps*[*simp del*]

lemmas *cdcl_W-restart-mset-state* = *trail.simps* *cons-trail.simps* *tl-trail.simps* *add-learned-cls.simps*

remove-cls.simps *update-conflicting.simps* *init-clss.simps* *learned-clss.simps*

conflicting.simps *init-state.simps*

definition *state* **where**

$\langle \text{state } S = (\text{trail } S, \text{init-clss } S, \text{learned-clss } S, \text{conflicting } S, ()) \rangle$

interpretation *cdcl_W-restart-mset: state_W-ops* **where**

state = *state* **and**

trail = *trail* **and**

init-clss = *init-clss* **and**

learned-clss = *learned-clss* **and**

conflicting = *conflicting* **and**

cons-trail = *cons-trail* **and**

tl-trail = *tl-trail* **and**

add-learned-cls = *add-learned-cls* **and**

remove-cls = *remove-cls* **and**

update-conflicting = *update-conflicting* **and**

init-state = *init-state*

$\langle \text{proof} \rangle$

definition *state-eq* :: '*v* *cdcl_W-restart-mset* \Rightarrow '*v* *cdcl_W-restart-mset* \Rightarrow bool (**infix** \sim_m 50) **where**

$\langle S \sim_m T \longleftrightarrow \text{state } S = \text{state } T \rangle$

interpretation *cdcl_W-restart-mset: state_W* **where**

state = *state* **and**

trail = *trail* **and**

init-clss = *init-clss* **and**

learned-clss = *learned-clss* **and**

conflicting = *conflicting* **and**

state-eq = *state-eq* **and**

cons-trail = *cons-trail* **and**

tl-trail = *tl-trail* **and**

add-learned-cls = *add-learned-cls* **and**

remove-cls = *remove-cls* **and**

update-conflicting = *update-conflicting* **and**

init-state = *init-state*

$\langle \text{proof} \rangle$

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-cl = *add-learned-cl* **and**
remove-cl = *remove-cl* **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*⟩

interpretation *cdcl_W-restart-mset*: *state_W-adding-init-clause* **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-cl = *add-learned-cl* **and**
remove-cl = *remove-cl* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state* **and**
add-init-cl = *add-init-cl*
⟨*proof*⟩

interpretation *cdcl_W-restart-mset*: *conflict-driven-clause-learning-with-adding-init-clause_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* **and**
add-init-cls = *add-init-cls*
⟨*proof*⟩

lemma *full-cdcl_W-init-state*:

⟨*full cdcl_W-restart-mset.cdcl_W-stgy* (*init-state* {#}) *S* \longleftrightarrow *S* = *init-state* {#}⟩
⟨*proof*⟩

locale *twl-restart-ops* =
fixes
f :: ⟨*nat* \Rightarrow *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* \Rightarrow *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-driven-clause-learning_W*
begin

lemma *distinct-cdcl_W-state-alt-def*:

```

⟨distinct-cdclW-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-stgy-cdcl_W-init-state-empty-no-step*:

```

⟨cdclW-restart-mset.cdclW-stgy (init-state {#}) S ⟷ False⟩
⟨proof⟩

```

lemma *cdcl_W-stgy-cdcl_W-init-state*:

```

⟨cdclW-restart-mset.cdclW-stgy** (init-state {#}) S ⟷ S = init-state {#}⟩
⟨proof⟩

```

4.2.1 Restart with Replay

In rather unfortunate move, we named restart with reuse of the trail “fast restarts” due to the name the technique was presented first, before a reviewer pointed out that the “fast” refers to the interval between two successive restarts and not to the idea that restarts are faster to do. This error is still present in various lemmas throughout the formalization.

Anyway, the idea is that after a restart, parts of the trail are often reused (at least, for the VSIDS heuristic: VMTF moves too fast for that) and, therefore, it is better to not redo the propagations. A similar idea exists for backjump, chronological backtracking, but that technique is much more complicated to understand and implement properly.

TODO: fix

lemma *after-fast-restart-replay*:

assumes

```

inv: ⟨cdclW-restart-mset.cdclW-all-struct-inv (M', N, U, None)⟩ and
stgy-invs: ⟨cdclW-restart-mset.cdclW-stgy-invariant (M', N, U, None)⟩ and
smaller-propa: ⟨cdclW-restart-mset.no-smaller-propa (M', N, U, None)⟩ and
kept: ⟨∀ L E. Propagated L E ∈ set (drop (length M' - n) M') ⟶ E ∈ # N + U'⟩ and
U'-U: ⟨U' ⊆ # U⟩

```

shows

```

⟨cdclW-restart-mset.cdclW-stgy** ([], N, U', None) (drop (length M' - n) M', N, U', None)⟩
⟨proof⟩

```

lemma *after-fast-restart-replay-no-stgy*:

assumes

inv: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv } (M', N, U, \text{None}) \rangle$ **and**

kept: $\langle \forall L E. \text{Propagated } L E \in \text{set } (\text{drop } (\text{length } M' - n) M') \longrightarrow E \in \# N + U' \rangle$ **and**

U'-U: $\langle U' \subseteq \# U \rangle$

shows

$\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W^{**} ([], N, U', \text{None}) (\text{drop } (\text{length } M' - n) M', N, U', \text{None}) \rangle$

$\langle \text{proof} \rangle$

lemma (**in** *conflict-driven-clause-learning_W*) *cdcl_W-stgy-new-learned-in-all-simple-cls*:

assumes

st: $\langle \text{cdcl}_W\text{-stgy}^{**} R S \rangle$ **and**

invR: $\langle \text{cdcl}_W\text{-all-struct-inv } R \rangle$

shows $\langle \text{set-mset } (\text{learned-cls } S) \subseteq \text{simple-cls } (\text{atms-of-mm } (\text{init-cls } S)) \rangle$

$\langle \text{proof} \rangle$

end

theory *Pragmatic-CDCL*

imports *CDCL.CDCL-W-Restart CDCL.CDCL-W-Abstract-State*

begin

lemma *get-all-mark-of-propagated-alt-def*:

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

$\langle \text{proof} \rangle$

Chapter 5

Pragmatic CDCL

The idea of this calculus is to sit between the nice and abstract CDCL calculus and the first step towards the implementation, TWL. Pragmatic is not used to mean incomplete as Jasmin Blanchette for his superposition, but to mean that it is closer to the ideal behind SAT implementation. Therefore, the calculus will contain a few things that cannot be expressed in CDCL, but are important in a SAT solver:

1. To make it possible to express subsumption, we split our clauses in two parts: the subsumed clauses and the non-subsumed clauses. The CDCL calculus sees both of them together, but we only need the latter.
2. We also have components for special clauses that contains a literal set at level 0. We need that component anyway for clauses of length 1 when expressing two watched literals.
3. Adding clauses: if an init clause is subsumed by a learned clauses, it is better to add that clauses to the set of init clauses Armin Biere calls the non-subsumed initial clauses “irredundant”, because they cannot be removed anymore.

The “CDCL” operates on the non-subsumed clauses and we show that this is enough to connect it to a CDCL that operates on all clauses. The drawback of this approach is that we cannot remove literals, even if they do not appear anymore in the non-subsumed clauses. However, as these atoms will never appear in a conflict clause, they will very soon be at the end of the decision heuristic and, therefore, will not interfere nor slow down the solving process too much (they still occupy some memory locations, hence a small impact).

The second idea was already included in the formalization of TWL because of (i) clauses of length one do not have two literals to watch; (ii) moving them away makes garbage collecting easier, because no reason at level 0 on the trail is included in the set of clauses.

We also started to implement the ideas one and three, but we realised while thinking about restarts that separate rules are needed to avoid a lot of copy-and-paste. It was also not clear how to add subsumption detection on the fly without restart (like done right after a backtrack in CaDiCaL or in SPASS). In the end, we decided to go for a separate calculus that incorporates all these rules: The idea is to have CDCL as the core part of the calculus and other rules that are optional.

Termination still comes from the CDCL calculus: We do not want to apply the other rules exhaustively.

Non-satisfiability-preserving clause additions are possible if all models after adding a clause are also models from the set of clauses before adding that clause.

The calculus as expressed here does not deal with global transformations, like renumbering of variables or symmetry breaking. It only supports clause addition.

There are still a few things that are missing in this calculus and require further thinking:

1. Non-satisfiability-preserving clause elimination (blocked clause elimination, covered clauses). These transformations requires to reconstruct the model, and it also not clear how to find these clauses efficiently. Reconstruction is not that easy to express, and it is not so clear how to implement it in the SAT solver.
2. Variable Elimination. Here reconstruction is also required. This technique is however extremely powerful and likely a must for every SAT solver.

For the calculus we have three layers:

1. the core corresponds to the application to most CDCL rules. Correctness and completeness come from this level. The rules are applied until completion.
2. the terminating core (tcore) additionally includes rule that makes it possible to handle clauses of length 1. Termination comes from this level. The rules are optional.
3. the full calculus also includes rule to do inprocessing (and restarts). The rules are optional too.

In order to keep the core small, if backtrack learns a unit clause, we use one step from the tcore. At this level of the presentation, the special case of the unit clause does not matter. However, it is important for the calculus with watched literals, since these clauses cannot be watched.

TODO:

- model reconstruction

```
type-synonym 'v prag-st =
  ⟨('v, 'v clause) ann-lits × 'v clauses × 'v clauses ×
  'v clause option × 'v clauses × 'v clauses × 'v clauses × 'v clauses ×
  'v clauses × 'v clauses⟩
```

```
fun state-of :: ⟨'v prag-st ⇒ 'v cdclW-restart-mset⟩ where
  ⟨state-of (M, N, U, C, NE, UE, NS, US, N0, U0) =
  (M, N + NE + NS + N0, U + UE + US + U0, C)⟩
```

```
declare cdclW-restart-mset-state[simp]
```

```
named-theorems ptwl ⟨Theorems to simplify the state⟩
```

5.1 Conversion

```
fun pget-trail :: ⟨'v prag-st ⇒ ('v, 'v clause) ann-lit list⟩ where
  ⟨pget-trail (M, -, -, -, -, -, -) = M⟩
```

```
fun set-conflict :: ⟨'v clause ⇒ 'v prag-st ⇒ 'v prag-st⟩ where
  ⟨set-conflict D (M, N, U, -, NE, UE, NS, US) =
  (M, N, U, Some D, NE, UE, NS, US)⟩
```

```

fun pget-conflict :: ⟨'v prag-st ⇒ 'v clause option⟩ where
  ⟨pget-conflict (M, N, U, D, NE, UE, WS, Q) = D⟩

fun pget-clauses :: ⟨'v prag-st ⇒ 'v clauses⟩ where
  ⟨pget-clauses (M, N, U, D, NE, UE, NS, US) = N + U⟩

fun pget-init-clauses :: ⟨'v prag-st ⇒ 'v clauses⟩ where
  ⟨pget-init-clauses (M, N, U, D, NE, UE, NS, US) = N⟩

fun punit-clss :: ⟨'v prag-st ⇒ 'v clauses⟩ where
  ⟨punit-clss (M, N, U, D, NE, UE, NS, US) = NE + UE⟩

fun punit-init-clauses :: ⟨'v prag-st ⇒ 'v clauses⟩ where
  ⟨punit-init-clauses (M, N, U, D, NE, UE, NS, US) = NE⟩

fun punit-clauses :: ⟨'v prag-st ⇒ 'v clauses⟩ where
  ⟨punit-clauses (M, N, U, D, NE, UE, NS, US) = NE + UE⟩

fun pclauses0 :: ⟨'v prag-st ⇒ 'v clauses⟩ where
  ⟨pclauses0 (M, N, U, D, NE, UE, NS, US, N0, U0) = N0 + U0⟩

fun pinit-clauses0 :: ⟨'v prag-st ⇒ 'v clauses⟩ where
  ⟨pinit-clauses0 (M, N, U, D, NE, UE, NS, US, N0, U0) = N0⟩

fun plearned-clauses0 :: ⟨'v prag-st ⇒ 'v clauses⟩ where
  ⟨plearned-clauses0 (M, N, U, D, NE, UE, NS, US, N0, U0) = U0⟩

fun psubsumed-learned-clss :: ⟨'v prag-st ⇒ 'v clause multiset⟩ where
  ⟨psubsumed-learned-clss (M, N, U, D, NE, UE, NS, US, N0, U0) = US⟩

fun psubsumed-init-clauses :: ⟨'v prag-st ⇒ 'v clauses⟩ where
  ⟨psubsumed-init-clauses (M, N, U, D, NE, UE, NS, US) = NS⟩

fun pget-all-init-clss :: ⟨'v prag-st ⇒ 'v clause multiset⟩ where
  ⟨pget-all-init-clss (M, N, U, D, NE, UE, NS, US, N0, U0) = N + NE + NS + N0⟩

fun pget-learned-clss :: ⟨'v prag-st ⇒ 'v clauses⟩ where
  ⟨pget-learned-clss (M, N, U, D, NE, UE, NS, US) = U⟩

fun psubsumed-learned-clauses :: ⟨'v prag-st ⇒ 'v clauses⟩ where
  ⟨psubsumed-learned-clauses (M, N, U, D, NE, UE, NS, US, N0, U0) = US⟩

fun psubsumed-clauses :: ⟨'v prag-st ⇒ 'v clauses⟩ where
  ⟨psubsumed-clauses (M, N, U, D, NE, UE, NS, US, N0, U0) = NS + US⟩

fun pget-init-learned-clss :: ⟨'v prag-st ⇒ 'v clauses⟩ where
  ⟨pget-init-learned-clss (-, N, U, -, -, UE, NS, US) = UE⟩

fun pget-all-learned-clss :: ⟨'v prag-st ⇒ 'v clauses⟩ where
  ⟨pget-all-learned-clss (-, N, U, -, -, UE, NS, US, N0, U0) = U + UE + US + U0⟩

fun pget-all-clss :: ⟨'v prag-st ⇒ 'v clause multiset⟩ where
  ⟨pget-all-clss (M, N, U, D, NE, UE, NS, US, N0, U0) =
    N + NE + NS + N0 + U + UE + US + U0⟩

```

lemma *[ptwl]*:
 $\langle \text{trail } (\text{state-of } S) = \text{pget-trail } S \rangle$
 $\langle \text{proof} \rangle$

declare *ptwl[simp]*

5.2 The old rules

inductive *cdcl-propagate* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**
propagate:
 $\langle \text{cdcl-propagate } (M, N, U, \text{None}, \text{NE}, \text{UE}, \text{NS}, \text{US}, \text{N0}, \text{U0})$
 $(\text{Propagated } L' D \# M, N, U, \text{None}, \text{NE}, \text{UE}, \text{NS}, \text{US}, \text{N0}, \text{U0}) \rangle$
if
 $\langle L' \in \# D \rangle$ **and** $\langle M \models_{\text{as}} \text{CNot } (\text{remove1-mset } L' D) \rangle$ **and**
 $\langle \text{undefined-lit } M L' \rangle \langle D \in \# N + U \rangle$

inductive *cdcl-conflict* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**
conflict:
 $\langle \text{cdcl-conflict } (M, N, U, \text{None}, \text{NE}, \text{UE}, \text{NS}, \text{US}, \text{N0}, \text{U0})$
 $(M, N, U, \text{Some } D, \text{NE}, \text{UE}, \text{NS}, \text{US}, \text{N0}, \text{U0}) \rangle$
if
 $\langle M \models_{\text{as}} \text{CNot } D \rangle$ **and**
 $\langle D \in \# N + U \rangle$

inductive *cdcl-decide* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**
decide:
 $\langle \text{cdcl-decide } (M, N, U, \text{None}, \text{NE}, \text{UE}, \text{NS}, \text{US}, \text{N0}, \text{U0})$
 $(\text{Decided } L' \# M, N, U, \text{None}, \text{NE}, \text{UE}, \text{NS}, \text{US}, \text{N0}, \text{U0}) \rangle$
if
 $\langle \text{undefined-lit } M L' \rangle$ **and**
 $\langle \text{atm-of } L' \in \text{atms-of-mm } (N + \text{NE} + \text{NS} + \text{N0}) \rangle$

inductive *cdcl-skip* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**
skip:
 $\langle \text{cdcl-skip } (\text{Propagated } L' C \# M, N, U, \text{Some } D, \text{NE}, \text{UE}, \text{NS}, \text{US}, \text{N0}, \text{U0})$
 $(M, N, U, \text{Some } D, \text{NE}, \text{UE}, \text{NS}, \text{US}, \text{N0}, \text{U0}) \rangle$
if
 $\langle \neg L' \notin \# D \rangle$ **and**
 $\langle D \neq \{\#\} \rangle$

inductive *cdcl-resolve* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**
resolve:
 $\langle \text{cdcl-resolve } (\text{Propagated } L' C \# M, N, U, \text{Some } D, \text{NE}, \text{UE}, \text{NS}, \text{US}, \text{N0}, \text{U0})$
 $(M, N, U, \text{Some } (\text{cdcl}_W\text{-restart-mset.resolve-cls } L' D C), \text{NE}, \text{UE}, \text{NS}, \text{US}, \text{N0}, \text{U0}) \rangle$
if
 $\langle \neg L' \in \# D \rangle$ **and**
 $\langle \text{get-maximum-level } (\text{Propagated } L' C \# M) (\text{remove1-mset } (\neg L') D) = \text{count-decided } M \rangle$ **and**
 $\langle L' \in \# C \rangle$

inductive *cdcl-backtrack* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{cdcl-backtrack } (M, N, U, \text{Some } (\text{add-mset } L D), \text{NE}, \text{UE}, \text{NS}, \text{US}, \text{N0}, \text{U0})$
 $(\text{Propagated } L (\text{add-mset } L D') \# M1, N, \text{add-mset } (\text{add-mset } L D') U, \text{None}, \text{NE}, \text{UE}, \text{NS}, \text{US}, \text{N0}, \text{U0}) \rangle$

if
 ⟨(Decided $K \# M1, M2$) \in set (get-all-ann-decomposition M)⟩ **and**
 ⟨get-level $M L =$ count-decided M ⟩ **and**
 ⟨get-level $M L =$ get-maximum-level M (add-mset $L D'$)⟩ **and**
 ⟨get-maximum-level $M D' \equiv i$ ⟩ **and**
 ⟨get-level $M K = i + 1$ ⟩ **and**
 ⟨ $D' \subseteq\# D$ ⟩ **and**
 ⟨ $N + U + NE + UE + NS + US + N0 + U0 \models_{pm}$ add-mset $L D'$ ⟩

Restart are already slightly restricted: we cannot remove literals set at level 0.

inductive *cdcl-restart* :: ⟨'v prag-st \Rightarrow 'v prag-st \Rightarrow bool⟩ **where**
cdcl-restart:
 ⟨*cdcl-restart* ($M, N, U, None, NE, UE, NS, US, N0, U0$)
 ($M', N, U, None, NE, UE, NS, US, N0, U0$)⟩
if ⟨($M2, Decided K \# M1$) \in set (get-all-ann-decomposition M)⟩

inductive *cdcl-forget* :: ⟨'v prag-st \Rightarrow 'v prag-st \Rightarrow bool⟩ **where**
cdcl-forget:
 ⟨*cdcl-forget* ($M, N, add-mset C U, None, NE, UE, NS, US, N0, U0$)
 ($M', N, U, None, NE, UE, NS, US, N0, U0$)⟩
if ⟨Propagated $L C \notin$ set M ⟩ |
cdcl-forget-subsumed:
 ⟨*cdcl-forget* ($M, N, U, None, NE, UE, NS, add-mset C US, N0, U0$)
 ($M', N, U, None, NE, UE, NS, US, N0, U0$)⟩

inductive *pcdcl-core* :: ⟨'v prag-st \Rightarrow 'v prag-st \Rightarrow bool⟩ **for** $S T$:: ⟨'v prag-st⟩ **where**
 ⟨*cdcl-conflict* $S T \Longrightarrow$ *pcdcl-core* $S T$ ⟩ |
 ⟨*cdcl-propagate* $S T \Longrightarrow$ *pcdcl-core* $S T$ ⟩ |
 ⟨*cdcl-decide* $S T \Longrightarrow$ *pcdcl-core* $S T$ ⟩ |
 ⟨*cdcl-skip* $S T \Longrightarrow$ *pcdcl-core* $S T$ ⟩ |
 ⟨*cdcl-resolve* $S T \Longrightarrow$ *pcdcl-core* $S T$ ⟩ |
 ⟨*cdcl-backtrack* $S T \Longrightarrow$ *pcdcl-core* $S T$ ⟩

inductive *pcdcl-core-stgy* :: ⟨'v prag-st \Rightarrow 'v prag-st \Rightarrow bool⟩ **for** $S T$:: ⟨'v prag-st⟩ **where**
 ⟨*cdcl-conflict* $S T \Longrightarrow$ *pcdcl-core-stgy* $S T$ ⟩ |
 ⟨*cdcl-propagate* $S T \Longrightarrow$ *pcdcl-core-stgy* $S T$ ⟩ |
 ⟨no-step *cdcl-conflict* $S \Longrightarrow$ no-step *cdcl-propagate* $S \Longrightarrow$ *cdcl-decide* $S T \Longrightarrow$ *pcdcl-core-stgy* $S T$ ⟩ |
 ⟨*cdcl-skip* $S T \Longrightarrow$ *pcdcl-core-stgy* $S T$ ⟩ |
 ⟨*cdcl-resolve* $S T \Longrightarrow$ *pcdcl-core-stgy* $S T$ ⟩ |
 ⟨*cdcl-backtrack* $S T \Longrightarrow$ *pcdcl-core-stgy* $S T$ ⟩

5.3 The new rules

Now the different part

inductive *cdcl-subsumed* :: ⟨'v prag-st \Rightarrow 'v prag-st \Rightarrow bool⟩ **where**
subsumed-II:
 ⟨*cdcl-subsumed* ($M, N + \{\#C, C'\#\}, U, D, NE, UE, NS, US, N0, U0$)
 ($M, add-mset C N, U, D, NE, UE, add-mset C' NS, US, N0, U0$)⟩
if ⟨ $C \subseteq\# C'$ ⟩ |
subsumed-LL:
 ⟨*cdcl-subsumed* ($M, N, U + \{\#C, C'\#\}, D, NE, UE, NS, US, N0, U0$)
 ($M, N, add-mset C U, D, NE, UE, NS, add-mset C' US, N0, U0$)⟩
if ⟨ $C \subseteq\# C'$ ⟩ |
subsumed-IL:

$\langle \text{cdcl-subsumed } (M, \text{add-mset } C \ N, \text{add-mset } C' \ U, D, NE, UE, NS, US, N0, U0)$
 $(M, \text{add-mset } C \ N, U, D, NE, UE, NS, \text{add-mset } C' \ US, N0, U0) \rangle$
if $\langle C \subseteq \# \ C' \rangle$

lemma *state-of-cdcl-subsumed*:

$\langle \text{cdcl-subsumed } S \ T \implies \text{state-of } S = \text{state-of } T \rangle$
 $\langle \text{proof} \rangle$

Resolution requires to restart (or a very careful thinking where the clause can be used, so for now, we require level 0). The names 'I' and 'L' refers to 'irredundant' and 'learnt'.

We need the assumption $\neg \text{tautology } (C + C')$ because learned clauses cannot be tautologies.

inductive *cdcl-resolution* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**

resolution-II:

$\langle \text{cdcl-resolution } (M, N + \{\# \text{add-mset } L \ C, \text{add-mset } (-L) \ C'\#\}, U, D, NE, UE, NS, US, N0, U0)$
 $(M, N + \{\# \text{add-mset } L \ C, \text{add-mset } (-L) \ C', \text{remdups-mset } (C + C')\#\}, U, D, NE, UE, NS,$
 $US, N0, U0) \rangle$

if $\langle \text{count-decided } M = 0 \rangle$ |

resolution-LL:

$\langle \text{cdcl-resolution } (M, N, U + \{\# \text{add-mset } L \ C, \text{add-mset } (-L) \ C'\#\}, D, NE, UE, NS, US, N0, U0)$
 $(M, N, U + \{\# \text{add-mset } L \ C, \text{add-mset } (-L) \ C', \text{remdups-mset } (C + C')\#\}, D, NE, UE, NS,$
 $US, N0, U0) \rangle$

if $\langle \text{count-decided } M = 0 \rangle$ **and** $\langle \neg \text{tautology } (C + C') \rangle$ |

resolution-IL:

$\langle \text{cdcl-resolution } (M, N + \{\# \text{add-mset } L \ C\#\}, U + \{\# \text{add-mset } (-L) \ C'\#\}, D, NE, UE, NS, US,$
 $N0, U0)$
 $(M, N + \{\# \text{add-mset } L \ C\#\}, U + \{\# \text{add-mset } (-L) \ C', \text{remdups-mset } (C + C')\#\}, D, NE, UE,$
 $NS, US, N0, U0) \rangle$

if $\langle \text{count-decided } M = 0 \rangle$ **and** $\langle \neg \text{tautology } (C + C') \rangle$

lemma *cdcl-resolution-still-entailed*:

$\langle \text{cdcl-resolution } S \ T \implies \text{consistent-interp } I \implies I \models_m \text{pget-all-init-clss } S \implies I \models_m \text{pget-all-init-clss}$
 $T \rangle$

$\langle \text{proof} \rangle$

Tautologies are always entailed by the clause set, but not necessarily entailed by a non-total model of the clauses. Therefore, we do not learn tautologies.

E.g.: $(A \vee B) \wedge (A \vee C)$ entails the clause $\neg B \vee B$, but the model containing only the literal A does not entail the latter. This is also why this predicate is different from the previous one: in *cdcl-resolution* we can learn a tautology because we do not destroy models. This is not the case in this predicate.

This function has nothing to with CDCL's learn: any clause can be learned by this function, including the empty clause.

TODO: for clauses in U, drop entailment and level test!

inductive *cdcl-learn-clause* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**

learn-clause:

$\langle \text{cdcl-learn-clause } (M, N, U, D, NE, UE, NS, US, N0, U0)$
 $(M, \text{add-mset } C \ N, U, D, NE, UE, NS, US, N0, U0) \rangle$

if $\langle \text{atms-of } C \subseteq \text{atms-of-mm } (N + NE + NS + N0) \rangle$ **and**

$\langle N + NE + NS + N0 \models_{pm} C \rangle$ **and**

$\langle \neg \text{tautology } C \rangle$ **and**

$\langle \text{count-decided } M = 0 \rangle$ **and**

$\langle \text{distinct-mset } C \rangle$ |

learn-clause-R:

$\langle \text{cdcl-learn-clause } (M, N, U, D, NE, UE, NS, US, N0, U0)$
 $(M, N, \text{add-mset } C U, D, NE, UE, NS, US, N0, U0) \rangle$
if $\langle \text{atms-of } C \subseteq \text{atms-of-mm } (N + NE + NS + N0) \rangle$ **and**
 $\langle N + NE + NS + N0 \models_{pm} C \rangle$ **and**
 $\langle \neg \text{tautology } C \rangle$ **and**
 $\langle \text{count-decided } M = 0 \rangle$ **and**
 $\langle \text{distinct-mset } C \rangle$ |
promote-clause:
 $\langle \text{cdcl-learn-clause } (M, N, \text{add-mset } C U, D, NE, UE, NS, US, N0, U0)$
 $(M, \text{add-mset } C N, U, D, NE, UE, NS, US, N0, U0) \rangle$
if $\langle \neg \text{tautology } C \rangle$

lemma *cdcl-learn-clause-still-entailed:*

$\langle \text{cdcl-learn-clause } S T \implies \text{consistent-interp } I \implies$
 $\text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init } (\text{state-of } S) \implies$
 $I \models_m \text{pget-all-init-clss } S \implies I \models_m \text{pget-all-init-clss } T \rangle$
 $\langle \text{proof} \rangle$

Detection and removal of pure literals.

inductive *cdcl-pure-literal-remove* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**
cdcl-pure-literal-remove:

$\langle \text{cdcl-pure-literal-remove } (M, N, U, \text{None}, NE, UE, NS, US, N0, U0)$
 $(\text{Propagated } L \{ \#L\# \} \# M, N, U, \text{None}, \text{add-mset } \{ \#L\# \} NE, UE, NS, US, N0, U0) \rangle$
if $\langle -L \notin \bigcup (\text{set-mset } ' (\text{set-mset } N)) \rangle$
 $\langle \text{atm-of } L \in \text{atms-of-mm } (N + NE + NS + N0) \rangle$
 $\langle \text{undefined-lit } M L \rangle$
 $\langle \text{count-decided } M = 0 \rangle$

lemma *pure-literal-can-be-removed:*

assumes
 $\langle -L \notin \bigcup (\text{set-mset } ' (\text{set-mset } N)) \rangle$ **and**
 $\langle I \models_m N \rangle$
shows $\langle \text{insert } L (I - \{-L\}) \models_m N \rangle$
 $\langle \text{proof} \rangle$

Inprocessing version of propagate and conflict.

The rules are very liberal to be used as freely as possible. They are similar to their CDCL counterpart, but do not cover exactly the same use-cases.

inductive *cdcl-inp-propagate* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**
propagate:

$\langle \text{cdcl-inp-propagate } (M, N, U, \text{None}, NE, UE, NS, US, N0, U0)$
 $(\text{Propagated } L' D \# M, N, U, \text{None}, NE, UE, NS, US, N0, U0) \rangle$
if
 $\langle L' \in \# D \rangle$ **and** $\langle M \models_{as} \text{CNot } (\text{remove1-mset } L' D) \rangle$ **and**
 $\langle \text{undefined-lit } M L' \rangle$ $\langle D \in \# N + U + NE + UE + NS + US \rangle$ **and**
 $\langle \text{count-decided } M = 0 \rangle$

inductive *cdcl-inp-conflict* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**
conflict:

$\langle \text{cdcl-inp-conflict } (M, N, U, \text{None}, NE, UE, NS, US, N0, U0)$
 $(M, N, U, \text{Some } \{ \# \}, NE, UE, NS, US, N0, U0) \rangle$
if
 $\langle N + NE + NS + N0 \models_{pm} \{ \# \} \rangle$ **and**
 $\langle \text{count-decided } M = 0 \rangle$

inductive *cdcl-flush-unit* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**

learn-unit-clause-R:

$\langle \text{cdcl-flush-unit } (M, \text{add-mset } \{\#L\# \} N, U, D, NE, UE, NS, US, N0, U0)$
 $(M, N, U, D, \text{add-mset } \{\#L\# \} NE, UE, NS, US, N0, U0) \rangle$

if $\langle \text{get-level } M \ L = 0 \rangle \langle L \in \text{lits-of-l } M \rangle |$

learn-unit-clause-I:

$\langle \text{cdcl-flush-unit } (M, N, \text{add-mset } \{\#L\# \} U, D, NE, UE, NS, US, N0, U0)$
 $(M, N, U, D, NE, \text{add-mset } \{\#L\# \} UE, NS, US, N0, U0) \rangle$

if $\langle \text{get-level } M \ L = 0 \rangle \langle L \in \text{lits-of-l } M \rangle$

inductive *cdcl-unitres-true* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**

$\langle \text{cdcl-unitres-true } (M, \text{add-mset } (\text{add-mset } L \ C) N, U, D, NE, UE, NS, US, N0, U0)$
 $(M, N, U, D, \text{add-mset } (\text{add-mset } L \ C) NE, UE, NS, US, N0, U0) \rangle$

if $\langle \text{get-level } M \ L = 0 \rangle \langle L \in \text{lits-of-l } M \rangle |$

$\langle \text{cdcl-unitres-true } (M, N, \text{add-mset } (\text{add-mset } L \ C) U, D, NE, UE, NS, US, N0, U0)$
 $(M, N, U, D, NE, \text{add-mset } (\text{add-mset } L \ C) UE, NS, US, N0, U0) \rangle$

if $\langle \text{get-level } M \ L = 0 \rangle \langle L \in \text{lits-of-l } M \rangle$

Even if we don't generated tautologies in the learnt clauses, we still keep the possibility to remove them later

inductive *cdcl-promote-false* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**

$\langle \text{cdcl-promote-false } (M, \text{add-mset } \{\#\} N, U, D, NE, UE, NS, US, N0, U0)$
 $(M, N, U, \text{Some } \{\#\}, NE, UE, NS, US, \text{add-mset } \{\#\} N0, U0) \rangle |$

$\langle \text{cdcl-promote-false } (M, N, \text{add-mset } \{\#\} U, D, NE, UE, NS, US, N0, U0)$
 $(M, N, U, \text{Some } \{\#\}, NE, UE, NS, US, N0, \text{add-mset } \{\#\} U0) \rangle$

inductive *pcdcl* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**

$\langle \text{pcdcl-core } S \ T \Longrightarrow \text{pcdcl } S \ T \rangle |$
 $\langle \text{cdcl-learn-clause } S \ T \Longrightarrow \text{pcdcl } S \ T \rangle |$
 $\langle \text{cdcl-resolution } S \ T \Longrightarrow \text{pcdcl } S \ T \rangle |$
 $\langle \text{cdcl-subsumed } S \ T \Longrightarrow \text{pcdcl } S \ T \rangle |$
 $\langle \text{cdcl-flush-unit } S \ T \Longrightarrow \text{pcdcl } S \ T \rangle |$
 $\langle \text{cdcl-inp-propagate } S \ T \Longrightarrow \text{pcdcl } S \ T \rangle |$
 $\langle \text{cdcl-inp-conflict } S \ T \Longrightarrow \text{pcdcl } S \ T \rangle |$
 $\langle \text{cdcl-unitres-true } S \ T \Longrightarrow \text{pcdcl } S \ T \rangle |$
 $\langle \text{cdcl-promote-false } S \ T \Longrightarrow \text{pcdcl } S \ T \rangle |$
 $\langle \text{cdcl-pure-literal-remove } S \ T \Longrightarrow \text{pcdcl } S \ T \rangle$

inductive *pcdcl-stgy* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **for** $S \ T :: \langle 'v \text{ prag-st} \rangle$ **where**

$\langle \text{pcdcl-core-stgy } S \ T \Longrightarrow \text{pcdcl-stgy } S \ T \rangle |$
 $\langle \text{cdcl-learn-clause } S \ T \Longrightarrow \text{pcdcl-stgy } S \ T \rangle |$
 $\langle \text{cdcl-resolution } S \ T \Longrightarrow \text{pcdcl-stgy } S \ T \rangle |$
 $\langle \text{cdcl-subsumed } S \ T \Longrightarrow \text{pcdcl-stgy } S \ T \rangle |$
 $\langle \text{cdcl-flush-unit } S \ T \Longrightarrow \text{pcdcl-stgy } S \ T \rangle |$
 $\langle \text{cdcl-inp-propagate } S \ T \Longrightarrow \text{pcdcl-stgy } S \ T \rangle |$
 $\langle \text{cdcl-inp-conflict } S \ T \Longrightarrow \text{pcdcl-stgy } S \ T \rangle |$
 $\langle \text{cdcl-unitres-true } S \ T \Longrightarrow \text{pcdcl-stgy } S \ T \rangle$

lemma *pcdcl-core-stgy-pcdcl*: $\langle \text{pcdcl-core-stgy } S \ T \Longrightarrow \text{pcdcl-core } S \ T \rangle$

$\langle \text{proof} \rangle$

lemma *pcdcl-stgy-pcdcl*: $\langle \text{pcdcl-stgy } S \ T \Longrightarrow \text{pcdcl } S \ T \rangle$

$\langle \text{proof} \rangle$

lemma *rtranclp-pcdcl-stgy-pcdcl*: $\langle \text{pcdcl-stgy}^{**} S \ T \Longrightarrow \text{pcdcl}^{**} S \ T \rangle$

$\langle \text{proof} \rangle$

5.4 Invariants

To avoid adding a new component, we store tautologies in the subsumed components, even though we could also store them separately. Finally, we really want to get rid of tautologies from the clause set. They require special cases in the arena module.

definition *psubsumed-invs* :: $\langle 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{psubsumed-invs } S \longleftrightarrow$
 $(\forall C \in \# \text{psubsumed-init-clauses } S. \text{tautology } C \vee$
 $(\exists C' \in \# \text{pget-init-clauses } S + \text{punit-init-clauses } S + \text{pinit-clauses0 } S. C' \subseteq \# C)) \wedge$
 $(\forall C \in \# \text{psubsumed-learned-clauses } S. \text{tautology } C \vee$
 $(\exists C' \in \# \text{pget-clauses } S + \text{punit-clauses } S + \text{pclasses0 } S. C' \subseteq \# C)) \rangle$

In the TWL version, we also had $\forall C \in \# NE + UE. \exists L. L \in \# C \wedge (D = \text{None} \vee 0 < \text{count-decided } M \longrightarrow \text{get-level } M L = 0 \wedge L \in \text{lits-of-l } M)$ as definition. This make is possible to express the conflict analysis at level 0. However, it makes the code easier to not do this analysis, because we already know that we will derive the empty clause (but do not know what the trail will be).

We could simplify the invariant to $\forall C \in \# NE + UE. \exists L. L \in \# C \wedge \text{get-level } M L = 0 \wedge L \in \text{lits-of-l } M$ at the price of an uglier correctness theorem.

Final remark, we could simplify the invariant in another way: We could have $D = \{\#L\# \}$. This, however, requires that we either remove all literals set at level 0, including in *reasons*, which we would rather avoid, or add the new clause $\{\#L\# \}$ each time we propagate a clause at level 0 and change the reason to this new clause. In either cases, we could use the subsumed components to store the clauses.

fun *entailed-clss-inv* :: $\langle 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{entailed-clss-inv } (M, N, U, D, NE, UE, NS, US) \longleftrightarrow$
 $(\forall C \in \# NE + UE.$
 $(\exists L. L \in \# C \wedge ((D = \text{None} \vee \text{count-decided } M > 0) \longrightarrow (\text{get-level } M L = 0 \wedge L \in \text{lits-of-l } M)))) \rangle$

lemmas *entailed-clss-inv-def* = *entailed-clss-inv.simps*

lemmas [*simp del*] = *entailed-clss-inv.simps*

fun *clauses0-inv* :: $\langle 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{clauses0-inv } (M, N, U, D, NE, UE, NS, US, N0, U0) \longleftrightarrow (\forall C \in \# N0 + U0. C = \{\#\}) \wedge$
 $(\{\#\} \in \# N0 + U0 \longrightarrow D = \text{Some } \{\#\}) \rangle$

lemma *clauses0-inv-def*:
 $\langle \text{clauses0-inv } (M, N, U, D, NE, UE, NS, US, N0, U0) \longleftrightarrow (\forall C \in \# N0 + U0. C = \{\#\}) \wedge$
 $(N0 + U0 \neq \{\#\} \longrightarrow D = \text{Some } \{\#\}) \rangle$
 $\langle \text{proof} \rangle$

lemmas [*simp del*] = *clauses0-inv.simps*

5.5 Relation to CDCL

lemma *cdcl-decide-is-decide*:
 $\langle \text{cdcl-decide } S T \Longrightarrow \text{cdcl}_W\text{-restart-mset.decide } (\text{state-of } S) (\text{state-of } T) \rangle$
 $\langle \text{proof} \rangle$

lemma *decide-is-cdcl-decide*:
 $\langle \text{cdcl}_W\text{-restart-mset.decide } (\text{state-of } S) T \Longrightarrow \text{Ex}(\text{cdcl-decide } S) \rangle$

⟨proof⟩

lemma *cdcl-skip-is-skip*:

⟨*cdcl-skip* $S T \implies \text{cdcl}_W\text{-restart-mset.skip (state-of } S) \text{ (state-of } T)$ ⟩

⟨proof⟩

lemma *skip-is-cdcl-skip*:

⟨*cdcl*_W-restart-mset.skip (state-of S) $T \implies \text{Ex}(\text{cdcl-skip } S)$ ⟩

⟨proof⟩

lemma *cdcl-resolve-is-resolve*:

⟨*cdcl-resolve* $S T \implies \text{cdcl}_W\text{-restart-mset.resolve (state-of } S) \text{ (state-of } T)$ ⟩

⟨proof⟩

lemma *resolve-is-cdcl-resolve*:

⟨*cdcl*_W-restart-mset.resolve (state-of S) $T \implies \text{Ex}(\text{cdcl-resolve } S)$ ⟩

⟨proof⟩

lemma *cdcl-backtrack-is-backtrack*:

⟨*cdcl-backtrack* $S T \implies \text{cdcl}_W\text{-restart-mset.backtrack (state-of } S) \text{ (state-of } T)$ ⟩

⟨proof⟩

lemma *backtrack-is-cdcl-backtrack*:

⟨*cdcl*_W-restart-mset.backtrack (state-of S) $T \implies \text{Ex}(\text{cdcl-backtrack } S)$ ⟩

⟨proof⟩

lemma *cdcl-conflict-is-conflict*:

⟨*cdcl-conflict* $S T \implies \text{cdcl}_W\text{-restart-mset.conflict (state-of } S) \text{ (state-of } T)$ ⟩

⟨proof⟩

lemma *conflict-is-cdcl-conflictD*:

assumes

confl: ⟨*cdcl*_W-restart-mset.conflict (state-of S) T ⟩ **and**

sub: ⟨*psubsumed-invs* S ⟩ **and**

ent: ⟨*entailed-clss-inv* S ⟩ **and**

invs: ⟨*cdcl*_W-restart-mset.cdcl_W-all-struct-inv (state-of S)⟩ **and**

clss0: ⟨*clauses0-inv* S ⟩

shows ⟨ $\text{Ex}(\text{cdcl-conflict } S)$ ⟩

⟨proof⟩

lemma *cdcl-propagate-is-propagate*:

⟨*cdcl-propagate* $S T \implies \text{cdcl}_W\text{-restart-mset.propagate (state-of } S) \text{ (state-of } T)$ ⟩

⟨proof⟩

lemma *propagate-is-cdcl-propagateD*:

assumes

confl: ⟨*cdcl*_W-restart-mset.propagate (state-of S) T ⟩ **and**

sub: ⟨*psubsumed-invs* S ⟩ **and**

ent: ⟨*entailed-clss-inv* S ⟩ **and**

invs: ⟨*cdcl*_W-restart-mset.cdcl_W-all-struct-inv (state-of S)⟩ **and**

clss0: ⟨*clauses0-inv* S ⟩

shows ⟨ $\text{Ex}(\text{cdcl-propagate } S) \vee \text{Ex}(\text{cdcl-conflict } S)$ ⟩

⟨proof⟩

lemma *pcdcl-core-is-cdcl*:

$\langle \text{pcdcl-core } S \ T \implies \text{cdcl}_W\text{-restart-mset.cdcl}_W \text{ (state-of } S \text{) (state-of } T \text{)} \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-pcdcl-core-is-cdcl*:

$\langle \text{pcdcl-core}^{**} \ S \ T \implies \text{cdcl}_W\text{-restart-mset.cdcl}_W^{**} \text{ (state-of } S \text{) (state-of } T \text{)} \rangle$
 $\langle \text{proof} \rangle$

lemma *pcdcl-core-stgy-is-cdcl-stgy*:

assumes

conf1: $\langle \text{pcdcl-core-stgy } S \ T \rangle$ **and**
sub: $\langle \text{psubsumed-invs } S \rangle$ **and**
ent: $\langle \text{entailed-clss-inv } S \rangle$ **and**
invs: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv (state-of } S \text{)} \rangle$ **and**
clss0: $\langle \text{clauses0-inv } S \rangle$

shows $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy (state-of } S \text{) (state-of } T \text{)} \rangle$
 $\langle \text{proof} \rangle$

lemma *no-step-pcdcl-core-stgy-is-cdcl-stgy*:

assumes

conf1: $\langle \text{no-step pcdcl-core-stgy } S \rangle$ **and**
sub: $\langle \text{psubsumed-invs } S \rangle$ **and**
ent: $\langle \text{entailed-clss-inv } S \rangle$ **and**
clss0: $\langle \text{clauses0-inv } S \rangle$ **and**
invs: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv (state-of } S \text{)} \rangle$

shows $\langle \text{no-step cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy (state-of } S \text{)} \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-resolution-psubsumed-invs*:

$\langle \text{cdcl-resolution } S \ T \implies \text{psubsumed-invs } S \implies \text{psubsumed-invs } T \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-resolution-entailed-clss-inv*:

$\langle \text{cdcl-resolution } S \ T \implies \text{entailed-clss-inv } S \implies \text{entailed-clss-inv } T \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-pure-literal-remove-psubsumed-invs*:

$\langle \text{cdcl-pure-literal-remove } S \ T \implies \text{psubsumed-invs } S \implies \text{psubsumed-invs } T \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-pure-literal-remove-entailed-clss-inv*:

$\langle \text{cdcl-pure-literal-remove } S \ T \implies \text{entailed-clss-inv } S \implies \text{entailed-clss-inv } T \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-subsumed-psubsumed-invs*:

$\langle \text{cdcl-subsumed } S \ T \implies \text{psubsumed-invs } S \implies \text{psubsumed-invs } T \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-subsumed-entailed-clss-inv*:

$\langle \text{cdcl-subsumed } S \ T \implies \text{entailed-clss-inv } S \implies \text{entailed-clss-inv } T \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-learn-clause-psubsumed-invs*:

$\langle \text{cdcl-learn-clause } S \ T \implies \text{psubsumed-invs } S \implies \text{psubsumed-invs } T \rangle$

⟨proof⟩

lemma *cdcl-learn-clause-entailed-clss-inv*:

⟨*cdcl-learn-clause* $S T \implies \text{entailed-clss-inv } S \implies \text{entailed-clss-inv } T$ ⟩

⟨proof⟩

lemma *cdcl-learn-clause-all-struct-inv*:

assumes

⟨*cdcl-learn-clause* $S T$ ⟩ **and**

⟨*cdcl_W-restart-mset.cdcl_W-all-struct-inv* (*state-of* S)⟩

shows

⟨*cdcl_W-restart-mset.cdcl_W-all-struct-inv* (*state-of* T)⟩

⟨proof⟩

lemma *cdcl-subsumed-all-struct-inv*:

assumes

⟨*cdcl-subsumed* $S T$ ⟩ **and**

⟨*cdcl_W-restart-mset.cdcl_W-all-struct-inv* (*state-of* S)⟩

shows

⟨*cdcl_W-restart-mset.cdcl_W-all-struct-inv* (*state-of* T)⟩

⟨proof⟩

lemma *all-decomposition-implies-monoI*:

⟨*all-decomposition-implies* $N M \implies N \subseteq N' \implies \text{all-decomposition-implies } N' M$ ⟩

⟨proof⟩

lemma *cdcl-resolution-all-struct-inv*:

assumes

⟨*cdcl-resolution* $S T$ ⟩ **and**

⟨*cdcl_W-restart-mset.cdcl_W-all-struct-inv* (*state-of* S)⟩

shows

⟨*cdcl_W-restart-mset.cdcl_W-all-struct-inv* (*state-of* T)⟩

⟨proof⟩

lemma *cdcl-pure-literal-remove-all-struct-inv*:

assumes

⟨*cdcl-pure-literal-remove* $S T$ ⟩ **and**

⟨*cdcl_W-restart-mset.cdcl_W-all-struct-inv* (*state-of* S)⟩

shows

⟨*cdcl_W-restart-mset.cdcl_W-all-struct-inv* (*state-of* T)⟩

⟨proof⟩

lemma *cdcl-flush-unit-unchanged*:

⟨*cdcl-flush-unit* $S T \implies \text{state-of } S = \text{state-of } T$ ⟩

⟨proof⟩

lemma *cdcl-inp-conflict-all-struct-inv*:

⟨*cdcl-inp-conflict* $S T \implies$

cdcl_W-restart-mset.cdcl_W-all-struct-inv (*state-of* S) \implies

cdcl_W-restart-mset.cdcl_W-all-struct-inv (*state-of* T)⟩

⟨proof⟩

lemma *cdcl-inp-propagate-is-propagate*: ⟨*cdcl-inp-propagate* $S T \implies$

cdcl_W-restart-mset.propagate (*state-of* S) (*state-of* T)⟩

⟨proof⟩

lemma *cdcl-inp-propagate-all-struct-inv*:

⟨*cdcl-inp-propagate* $S T \implies$
 cdcl_W-restart-mset.cdcl_W-all-struct-inv (state-of S) \implies
 cdcl_W-restart-mset.cdcl_W-all-struct-inv (state-of T)⟩
⟨*proof*⟩

lemma *cdcl-unitres-true-same*:

⟨*cdcl-unitres-true* $S T \implies$ *state-of $S =$ state-of T* ⟩
⟨*proof*⟩

lemma *cdcl-promote-false-all-struct-inv*:

⟨*cdcl-promote-false* $S T \implies$
 cdcl_W-restart-mset.cdcl_W-all-struct-inv (state-of S) \implies
 cdcl_W-restart-mset.cdcl_W-all-struct-inv (state-of T)⟩
⟨*proof*⟩

lemma *pcdcl-all-struct-inv*:

⟨*pcdcl* $S T \implies$
 cdcl_W-restart-mset.cdcl_W-all-struct-inv (state-of S) \implies
 cdcl_W-restart-mset.cdcl_W-all-struct-inv (state-of T)⟩
⟨*proof*⟩

lemma *cdcl-unitres-true-entailed-clss-inv*:

⟨*cdcl-unitres-true* $S T \implies$ *entailed-clss-inv $S \implies$ entailed-clss-inv T* ⟩
⟨*proof*⟩

lemma *cdcl-unitres-true-psubsumed-invs*:

⟨*cdcl-unitres-true* $S T \implies$ *psubsumed-invs $S \implies$ psubsumed-invs T* ⟩
⟨*proof*⟩

definition *pcdcl-all-struct-invs* :: $\langle \rightarrow \rangle$ **where**

⟨*pcdcl-all-struct-invs* $S \iff$
 cdcl_W-restart-mset.cdcl_W-all-struct-inv (state-of S) \wedge
 entailed-clss-inv S \wedge
 psubsumed-invs S \wedge
 clauses0-inv S ⟩

lemma *entailed-clss-inv-Propagated*:

assumes ⟨*entailed-clss-inv (M, N, U, None, NE, UE, NS, US)*⟩ **and**
 undef: ⟨*undefined-lit M L'*⟩
shows ⟨*entailed-clss-inv (Propagated L' D # M, N, U, None, NE, UE, NS, US)*⟩
⟨*proof*⟩

lemma *entailed-clss-inv-skip*:

assumes ⟨*entailed-clss-inv (Propagated L' D'' # M, N, U, Some D, NE, UE, NS, US)*⟩
shows ⟨*entailed-clss-inv (M, N, U, Some D', NE, UE, NS, US)*⟩
⟨*proof*⟩

lemma *entailed-clss-inv-ConflictD*: ⟨*entailed-clss-inv (M, N, U, None, NE, UE, NS, US)* \implies

entailed-clss-inv (M, N, U, Some D, NE, UE, NS, US)⟩
⟨*proof*⟩

lemma *entailed-clss-inv-Decided*:

assumes ⟨*entailed-clss-inv (M, N, U, None, NE, UE, NS, US)*⟩ **and**

undef: $\langle \text{undefined-lit } M \ L' \rangle$

shows $\langle \text{entailed-clss-inv } (\text{Decided } L' \# M, N, U, \text{None}, NE, UE, NS, US) \rangle$
 $\langle \text{proof} \rangle$

lemma *get-all-ann-decomposition-lvl0-still*:

$\langle (\text{Decided } K \# M1, M2) \in \text{set } (\text{get-all-ann-decomposition } M) \implies L \in \text{lits-of-l } M \implies$
 $\text{get-level } M \ L = 0 \implies L \in \text{lits-of-l } M1 \wedge \text{get-level } M1 \ L = 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-backtrack-entailed-clss-inv*:

$\langle \text{cdcl-backtrack } S \ T \implies \text{entailed-clss-inv } S \implies \text{entailed-clss-inv } T \rangle$ **for** $S \ T :: \langle 'v \text{ prag-st} \rangle$
 $\langle \text{proof} \rangle$

lemma *pcdcl-core-entails-clss-inv*:

$\langle \text{pcdcl-core } S \ T \implies \text{entailed-clss-inv } S \implies \text{entailed-clss-inv } T \rangle$
 $\langle \text{proof} \rangle$

lemma *pcdcl-core-clauses0-inv*:

$\langle \text{pcdcl-core } S \ T \implies \text{clauses0-inv } S \implies \text{clauses0-inv } T \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-flush-unit-invs*:

$\langle \text{cdcl-flush-unit } S \ T \implies \text{psubsumed-invs } S \implies \text{psubsumed-invs } T \rangle$
 $\langle \text{cdcl-flush-unit } S \ T \implies \text{entailed-clss-inv } S \implies \text{entailed-clss-inv } T \rangle$
 $\langle \text{cdcl-flush-unit } S \ T \implies \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant } (\text{state-of } S) \implies$
 $\text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant } (\text{state-of } T) \rangle$
 $\langle \text{cdcl-flush-unit } S \ T \implies \text{clauses0-inv } S \implies \text{clauses0-inv } T \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-inp-propagate-invs*:

$\langle \text{cdcl-inp-propagate } S \ T \implies \text{psubsumed-invs } S \implies \text{psubsumed-invs } T \rangle$
 $\langle \text{cdcl-inp-propagate } S \ T \implies \text{entailed-clss-inv } S \implies \text{entailed-clss-inv } T \rangle$
 $\langle \text{cdcl-inp-propagate } S \ T \implies \text{clauses0-inv } S \implies \text{clauses0-inv } T \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-inp-conflict-invs*:

$\langle \text{cdcl-inp-conflict } S \ T \implies \text{psubsumed-invs } S \implies \text{psubsumed-invs } T \rangle$
 $\langle \text{cdcl-inp-conflict } S \ T \implies \text{entailed-clss-inv } S \implies \text{entailed-clss-inv } T \rangle$
 $\langle \text{cdcl-inp-conflict } S \ T \implies \text{clauses0-inv } S \implies \text{clauses0-inv } T \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-promote-false-invs*:

$\langle \text{cdcl-promote-false } S \ T \implies \text{psubsumed-invs } S \implies \text{psubsumed-invs } T \rangle$
 $\langle \text{cdcl-promote-false } S \ T \implies \text{entailed-clss-inv } S \implies \text{entailed-clss-inv } T \rangle$
 $\langle \text{cdcl-promote-false } S \ T \implies \text{clauses0-inv } S \implies \text{clauses0-inv } T \rangle$
 $\langle \text{proof} \rangle$

lemma *pcdcl-entails-clss-inv*:

$\langle \text{pcdcl } S \ T \implies \text{entailed-clss-inv } S \implies \text{entailed-clss-inv } T \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-pcdcl-entails-clss-inv*:

$\langle \text{pcdcl}^{**} \ S \ T \implies \text{entailed-clss-inv } S \implies \text{entailed-clss-inv } T \rangle$
 $\langle \text{proof} \rangle$

lemma *pcdcl-core-psubsumed-invs*:

$\langle \text{pcdcl-core } S \ T \implies \text{psubsumed-invs } S \implies \text{psubsumed-invs } T \rangle$
 $\langle \text{proof} \rangle$

lemma *pcdcl-psubsumed-invs*:

$\langle \text{pcdcl } S \ T \implies \text{psubsumed-invs } S \implies \text{psubsumed-invs } T \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-pcdcl-psubsumed-invs*:

$\langle \text{pcdcl}^{**} \ S \ T \implies \text{psubsumed-invs } S \implies \text{psubsumed-invs } T \rangle$
 $\langle \text{proof} \rangle$

lemma *pcdcl-clauses0-inv*:

$\langle \text{pcdcl } S \ T \implies \text{clauses0-inv } S \implies \text{clauses0-inv } T \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-pcdcl-clauses0-inv*:

$\langle \text{pcdcl}^{**} \ S \ T \implies \text{clauses0-inv } S \implies \text{clauses0-inv } T \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-pcdcl-core-stgy-pcdcl*: $\langle \text{pcdcl-core-stgy}^{**} \ S \ T \implies \text{pcdcl}^{**} \ S \ T \rangle$

$\langle \text{proof} \rangle$

lemma *rtranclp-pcdcl-core-stgy-is-cdcl-stgy*:

assumes

conft: $\langle \text{pcdcl-core-stgy}^{**} \ S \ T \rangle$ **and**

sub: $\langle \text{psubsumed-invs } S \rangle$ **and**

ent: $\langle \text{entailed-clss-inv } S \rangle$ **and**

invs: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv } (\text{state-of } S) \rangle$ **and**

$\langle \text{clauses0-inv } S \rangle$

shows $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy}^{**} \ (\text{state-of } S) \ (\text{state-of } T) \rangle$

$\langle \text{proof} \rangle$

lemma *pcdcl-all-struct-invs*:

$\langle \text{pcdcl } S \ T \implies$
 $\text{pcdcl-all-struct-invs } S \implies$
 $\text{pcdcl-all-struct-invs } T \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-pcdcl-all-struct-invs*:

$\langle \text{pcdcl}^{**} \ S \ T \implies \text{pcdcl-all-struct-invs } S \implies \text{pcdcl-all-struct-invs } T \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-resolution-entailed-by-init*:

assumes $\langle \text{cdcl-resolution } S \ T \rangle$ **and**

$\langle \text{pcdcl-all-struct-invs } S \rangle$ **and**

$\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init } (\text{state-of } S) \rangle$

shows $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init } (\text{state-of } T) \rangle$

$\langle \text{proof} \rangle$

lemma *cdcl-pure-literal-remove-entailed-by-init*:

assumes $\langle \text{cdcl-pure-literal-remove } S \ T \rangle$ **and**

$\langle \text{pcdcl-all-struct-invs } S \rangle$ **and**

$\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init } (\text{state-of } S) \rangle$

shows $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init } (\text{state-of } T) \rangle$

$\langle \text{proof} \rangle$

lemma *cdcl-subsumed-entailed-by-init*:

assumes $\langle \text{cdcl-subsumed } S \ T \rangle$ **and**

$\langle \text{pcdcl-all-struct-invs } S \rangle$ **and**

$\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state-of } S \rangle$

shows $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state-of } T \rangle$

$\langle \text{proof} \rangle$

lemma *cdcl-learn-clause-entailed-by-init*:

assumes $\langle \text{cdcl-learn-clause } S \ T \rangle$ **and**

$\langle \text{pcdcl-all-struct-invs } S \rangle$ **and**

$\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state-of } S \rangle$

shows $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state-of } T \rangle$

$\langle \text{proof} \rangle$

lemma *cdcl-inp-propagate-entailed-by-init*:

assumes $\langle \text{cdcl-inp-propagate } S \ T \rangle$ **and**

$\langle \text{pcdcl-all-struct-invs } S \rangle$ **and**

$\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state-of } S \rangle$

shows $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state-of } T \rangle$

$\langle \text{proof} \rangle$

lemma *cdcl-inp-conflict-entailed-by-init*:

assumes $\langle \text{cdcl-inp-conflict } S \ T \rangle$ **and**

$\langle \text{pcdcl-all-struct-invs } S \rangle$ **and**

$\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state-of } S \rangle$

shows $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state-of } T \rangle$

$\langle \text{proof} \rangle$

lemma *cdcl-promote-false-entailed-by-init*:

assumes $\langle \text{cdcl-promote-false } S \ T \rangle$ **and**

$\langle \text{pcdcl-all-struct-invs } S \rangle$ **and**

$\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state-of } S \rangle$

shows $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state-of } T \rangle$

$\langle \text{proof} \rangle$

lemma *pcdcl-entailed-by-init*:

assumes $\langle \text{pcdcl } S \ T \rangle$ **and**

$\langle \text{pcdcl-all-struct-invs } S \rangle$ **and**

$\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state-of } S \rangle$

shows $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state-of } T \rangle$

$\langle \text{proof} \rangle$

lemma *rtranclp-pcdcl-entailed-by-init*:

assumes $\langle \text{pcdcl}^{**} S \ T \rangle$ **and**

$\langle \text{pcdcl-all-struct-invs } S \rangle$ **and**

$\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state-of } S \rangle$

shows $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state-of } T \rangle$

$\langle \text{proof} \rangle$

lemma *pcdcl-core-stgy-stgy-invs*:

assumes

confl: $\langle \text{pcdcl-core-stgy } S \ T \rangle$ **and**

sub: $\langle \text{psubsumed-invs } S \rangle$ **and**

ent: $\langle \text{entailed-clss-inv } S \rangle$ **and**

invs: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv (state-of } S) \rangle$ **and**
 $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } S) \rangle$ **and**
clss0: $\langle \text{clauses0-inv } S \rangle$
shows $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } T) \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-subsumed-stgy-stgy-invs*:

assumes
confl: $\langle \text{cdcl-subsumed } S T \rangle$ **and**
sub: $\langle \text{psubsumed-invs } S \rangle$ **and**
ent: $\langle \text{entailed-clss-inv } S \rangle$ **and**
invs: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv (state-of } S) \rangle$ **and**
 $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } S) \rangle$
shows $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } T) \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-resolution-stgy-stgy-invs*:

assumes
confl: $\langle \text{cdcl-resolution } S T \rangle$ **and**
sub: $\langle \text{psubsumed-invs } S \rangle$ **and**
ent: $\langle \text{entailed-clss-inv } S \rangle$ **and**
invs: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv (state-of } S) \rangle$ **and**
 $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } S) \rangle$
shows $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } T) \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-pure-literal-remove-stgy-stgy-invs*:

assumes
confl: $\langle \text{cdcl-pure-literal-remove } S T \rangle$ **and**
sub: $\langle \text{psubsumed-invs } S \rangle$ **and**
ent: $\langle \text{entailed-clss-inv } S \rangle$ **and**
invs: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv (state-of } S) \rangle$ **and**
 $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } S) \rangle$
shows $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } T) \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-learn-clause-stgy-stgy-invs*:

assumes
confl: $\langle \text{cdcl-learn-clause } S T \rangle$ **and**
sub: $\langle \text{psubsumed-invs } S \rangle$ **and**
ent: $\langle \text{entailed-clss-inv } S \rangle$ **and**
invs: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv (state-of } S) \rangle$ **and**
 $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } S) \rangle$
shows $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } T) \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-inp-conflict-stgy-stgy-invs*:

assumes
confl: $\langle \text{cdcl-inp-conflict } S T \rangle$
shows $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } T) \rangle$
 $\langle \text{proof} \rangle$

lemma *pcdcl-stgy-stgy-invs*:

assumes
confl: $\langle \text{pcdcl-stgy } S T \rangle$ **and**

sub: $\langle \text{psubsumed-inv } S \rangle$ **and**
ent: $\langle \text{entailed-clss-inv } S \rangle$ **and**
invs: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv (state-of } S) \rangle$ **and**
 $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } S) \rangle$ **and**
clss0: $\langle \text{clauses0-inv } S \rangle$
shows $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } T) \rangle$
 $\langle \text{proof} \rangle$

5.6 Higher-level rules

5.6.1 Simplify unit clauses

Initially, I wanted to remove the literals one at a time, but this has two disadvantages:

- more subsumed clauses are generated.
- it makes it possible to remove all literals making it much easier to express that the clause is correctly watched in the next refinement step. Undefined literals are correctly watched. Mixed situation are strange (we could watch two false clauses during the simplification only).

As we only make the rule strictly more powerful and that the proof is not different, I changed the rule instead of doing any other solution.

inductive *cdcl-unitres* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**

cdcl-unitresI:

$\langle \text{cdcl-unitres } (M, N + \{\#C+C'\# \}, U, \text{None}, NE, UE, NS, US, N0, U0)$
 $(M, N + \{\#C\# \}, U, \text{None}, NE, UE, \text{add-mset } (C+C') \text{ NS, US, N0, U0}) \rangle$
if $\langle \text{count-decided } M = 0 \rangle$ **and** $\langle \text{add-mset } (C+C') (N + NE + NS + N0) \models_{\text{psm}} \text{mset-set } (C\text{Not } C') \rangle$
 $\langle \neg \text{tautology } C \rangle \langle \text{distinct-mset } C \rangle \mid$

cdcl-unitresI-unit:

$\langle \text{cdcl-unitres } (M, N + \{\#C+C'\# \}, U, \text{None}, NE, UE, NS, US, N0, U0)$
 $(\text{Propagated } K \ C \ \# \ M, N, U, \text{None}, NE + \{\#C\# \}, UE, \text{add-mset } (C+C') \text{ NS, US, N0, U0}) \rangle$
if $\langle \text{count-decided } M = 0 \rangle$ **and** $\langle \text{add-mset } (C+C') (N + NE + NS + N0) \models_{\text{psm}} \text{mset-set } (C\text{Not } C') \rangle$
 $\langle \neg \text{tautology } C \rangle \langle \text{distinct-mset } C \rangle \langle C = \{\#K\# \} \rangle \langle \text{undefined-lit } M \ K \rangle \mid$

cdcl-unitresR:

$\langle \text{cdcl-unitres } (M, N, U + \{\#C+C'\# \}, \text{None}, NE, UE, NS, US, N0, U0)$
 $(M, N, U + \{\#C\# \}, \text{None}, NE, UE, NS, \text{add-mset } (C+C') \text{ US, N0, U0}) \rangle$
if $\langle \text{count-decided } M = 0 \rangle$ **and** $\langle (N + NE + NS + N0) \models_{\text{psm}} \text{mset-set } (C\text{Not } C') \rangle \langle \neg \text{tautology } C \rangle$
 $\langle \text{distinct-mset } C \rangle \langle \text{atms-of } C \subseteq \text{atms-of-mm } (N+NE+NS+N0) \rangle \mid$

cdcl-unitresR-unit:

$\langle \text{cdcl-unitres } (M, N, U + \{\#C+C'\# \}, \text{None}, NE, UE, NS, US, N0, U0)$
 $(\text{Propagated } K \ C \ \# \ M, N, U, \text{None}, NE, UE + \{\#C\# \}, NS, \text{add-mset } (C+C') \text{ US, N0, U0}) \rangle$
if $\langle \text{count-decided } M = 0 \rangle$ **and** $\langle (N + NE + NS + N0) \models_{\text{psm}} \text{mset-set } (C\text{Not } C') \rangle \langle \neg \text{tautology } C \rangle$
 $\langle \text{distinct-mset } C \rangle \langle \text{atms-of } C \subseteq \text{atms-of-mm } (N+NE+NS+N0) \rangle \langle C = \{\#K\# \} \rangle \langle \text{undefined-lit } M$
 $K \rangle \mid$

cdcl-unitresR-empty:

$\langle \text{cdcl-unitres } (M, N, U + \{\#C'\# \}, \text{None}, NE, UE, NS, US, N0, U0)$
 $(M, N, U, \text{Some } \{\#\}, NE, UE, NS, \text{add-mset } (C') \text{ US, N0, add-mset } \{\#\} \text{ U0}) \rangle$
if $\langle \text{count-decided } M = 0 \rangle$ **and** $\langle (N + NE + NS + N0) \models_{\text{psm}} \text{mset-set } (C\text{Not } C') \rangle \mid$

cdcl-unitresI-empty:

$\langle \text{cdcl-unitres } (M, N + \{\#C'\# \}, U, \text{None}, NE, UE, NS, US, N0, U0)$
 $(M, N, U, \text{Some } \{\#\}, NE, UE, \text{add-mset } (C') \text{ NS, US, add-mset } \{\#\} \text{ N0, U0}) \rangle$
if $\langle \text{count-decided } M = 0 \rangle$ **and** $\langle (\text{add-mset } C' \ N + NE + NS + N0) \models_{\text{psm}} \text{mset-set } (C\text{Not } C') \rangle$

lemma *true-clss-clb-or-true-clss-clb-or-not-true-clss-clb-or-generalise'*:
 $\langle N \models_{ps} C \text{Not } C' \implies N \models_p C + C' \implies C'' \subseteq_{\#} C' \implies N \models_p (C + (C' - C'')) \rangle$
 $\langle \text{proof} \rangle$

lemmas *true-clss-clb-or-true-clss-clb-or-not-true-clss-clb-or-generalise = true-clss-clb-or-true-clss-clb-or-not-true-clss-clb-or-generalise'*[of $N C' C C'$ for $C C' N$, simplified]

lemma *cdcl-unitres-learn-subsumeE*:
assumes $\langle \text{cdcl-unitres } S X \rangle$
 $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state-of } S) \rangle$
obtains $T U V W$ **where** $\langle \text{cdcl-learn-clause } S T \rangle \langle \text{cdcl-subsumed } T U \rangle$
 $\langle \text{cdcl-propagate}^{**} U V \rangle$
 $\langle \text{cdcl-flush-unit}^{**} V W \rangle$
 $\langle \text{cdcl-promote-false}^{**} W X \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-unitres-learn-subsume*:
assumes $\langle \text{cdcl-unitres } S U \rangle \langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state-of } S) \rangle$
shows $\langle \text{pcdcl}^{**} S U \rangle$
 $\langle \text{proof} \rangle$

lemma *get-all-ann-decomposition-count-decided0*:
 $\langle \text{count-decided } M = 0 \implies \text{get-all-ann-decomposition } M = [([], M)] \rangle$
 $\langle \text{proof} \rangle$

The following two lemmas gives the nicer introduction rule that are what anyone expects from removing false literals.

lemma *cdcl-unitresI1*:
assumes
invs: $\langle \text{pcdcl-all-struct-invs } (M, N, U + \{\#C+C'\#\}, \text{None}, NE, UE, NS, US, N0, U0) \rangle$ **and**
L: $\langle \forall L. L \in_{\#} C' \longrightarrow -L \in \text{lits-of-l } M \rangle$ **and**
[simp]: $\langle \text{count-decided } M = 0 \rangle$ **and**
 $\langle \neg \text{tautology } C \rangle$ **and**
ent-init: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state-of } (M, N, U + \{\#C+C'\#\}, \text{None}, NE, UE, NS, US, N0, U0)) \rangle$
shows $\langle \text{cdcl-unitres } (M, N, U + \{\#C+C'\#\}, \text{None}, NE, UE, NS, US, N0, U0) (M, N, U + \{\#C\#\}, \text{None}, NE, UE, NS, \text{add-mset } (C+C') US, N0, U0) \rangle$ (**is** $\langle \text{cdcl-unitres } ?S ?T \rangle$)
 $\langle \text{proof} \rangle$

lemma *cdcl-unitresI1-unit*:
fixes $K :: \langle 'v \text{ literal} \rangle$
defines $\langle C \equiv \{\#K\#\} \rangle$
assumes
invs: $\langle \text{pcdcl-all-struct-invs } (M, N, U + \{\#C+C'\#\}, \text{None}, NE, UE, NS, US, N0, U0) \rangle$ **and**
L: $\langle \forall L. L \in_{\#} C' \longrightarrow -L \in \text{lits-of-l } M \rangle$ **and**
[simp]: $\langle \text{count-decided } M = 0 \rangle$ **and**
 $\langle \neg \text{tautology } C \rangle$ **and**
undef: $\langle \text{undefined-lit } M K \rangle$ **and**
ent-init: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state-of } (M, N, U + \{\#C+C'\#\}, \text{None}, NE, UE, NS, US, N0, U0)) \rangle$
shows $\langle \text{cdcl-unitres } (M, N, U + \{\#C+C'\#\}, \text{None}, NE, UE, NS, US, N0, U0) (\text{Propagated } K C \# M, N, U, \text{None}, NE, UE + \{\#C\#\}, NS, \text{add-mset } (C+C') US, N0, U0) \rangle$ (**is** $\langle \text{cdcl-unitres } ?S ?T \rangle$)
 $\langle \text{proof} \rangle$

lemma *cdcl-unitiesI2*:

assumes

invs: $\langle \text{pcdcl-all-struct-invs } (M, N + \{\#C+C'\#\}, U, \text{None}, NE, UE, NS, US, N0, U0) \rangle$ **and**

L: $\langle \forall L. L \in \# C' \longrightarrow -L \in \text{lits-of-l } M \rangle$ **and**

[*simp*]: $\langle \text{count-decided } M = 0 \rangle$ **and**

$\langle \neg \text{tautology } C \rangle$ **and**

ent-init: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init}$

$(\text{state-of } (M, N + \{\#C+C'\#\}, U, \text{None}, NE, UE, NS, US, N0, U0)) \rangle$

shows $\langle \text{cdcl-unities } (M, N + \{\#C+C'\#\}, U, \text{None}, NE, UE, NS, US, N0, U0)$

$(M, N + \{\#C\#\}, U, \text{None}, NE, UE, \text{add-mset } (C+C') \text{ NS, US, N0, U0}) \rangle$ (**is** $\langle \text{cdcl-unities } ?S$

$?T \rangle$)

$\langle \text{proof} \rangle$

lemma *cdcl-unitiesI2-unit*:

fixes *K* :: $\langle 'v \text{ literal} \rangle$

defines $\langle C \equiv \{\#K\#\} \rangle$

assumes

invs: $\langle \text{pcdcl-all-struct-invs } (M, N + \{\#C+C'\#\}, U, \text{None}, NE, UE, NS, US, N0, U0) \rangle$ **and**

L: $\langle \forall L. L \in \# C' \longrightarrow -L \in \text{lits-of-l } M \rangle$ **and**

[*simp*]: $\langle \text{count-decided } M = 0 \rangle$ **and**

$\langle \neg \text{tautology } C \rangle$ **and**

undef: $\langle \text{undefined-lit } M K \rangle$ **and**

ent-init: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init}$

$(\text{state-of } (M, N + \{\#C+C'\#\}, U, \text{None}, NE, UE, NS, US, N0, U0)) \rangle$

shows $\langle \text{cdcl-unities } (M, N + \{\#C+C'\#\}, U, \text{None}, NE, UE, NS, US, N0, U0)$

$(\text{Propagated } K C \# M, N, U, \text{None}, NE + \{\#C\#\}, UE, \text{add-mset } (C+C') \text{ NS, US, N0, U0}) \rangle$ (**is**

$\langle \text{cdcl-unities } ?S ?T \rangle$)

$\langle \text{proof} \rangle$

5.6.2 Subsumption resolution

Subsumption-Resolution rules are the composition of resolution, subsumption, learning of a clause, and potentially forget. However, we have decided to not model the forget, because we would like to map the calculus to a version without restarts.

inductive *cdcl-subresolution* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**

subresolution-II:

$\langle \text{cdcl-subresolution } (M, N + \{\#\text{add-mset } L C, \text{add-mset } (-L) C'\#\}, U, D, NE, UE, NS, US, N0, U0)$

$(M, N + \{\#\text{add-mset } L C, \text{remdups-mset } C'\#\}, U, D, NE, UE, \text{add-mset } (\text{add-mset } (-L) C') \text{ NS, US, N0, U0}) \rangle$

if $\langle \text{count-decided } M = 0 \rangle \langle C \subseteq \# C' \rangle$

subresolution-LL:

$\langle \text{cdcl-subresolution } (M, N, U + \{\#\text{add-mset } L C, \text{add-mset } (-L) C'\#\}, D, NE, UE, NS, US, N0, U0)$

$(M, N, U + \{\#\text{add-mset } L C, \text{remdups-mset } (C')\#\}, D, NE, UE, NS, \text{add-mset } (\text{add-mset } (-L) C') \text{ US, N0, U0}) \rangle$

if $\langle \text{count-decided } M = 0 \rangle$ **and** $\langle \neg \text{tautology } (C + C') \rangle$ **and** $\langle C \subseteq \# C' \rangle$

subresolution-LI:

$\langle \text{cdcl-subresolution } (M, N + \{\#\text{add-mset } L C\#\}, U + \{\#\text{add-mset } (-L) C'\#\}, D, NE, UE, NS, US, N0, U0)$

$(M, N + \{\#\text{add-mset } L C\#\}, U + \{\#\text{remdups-mset } (C')\#\}, D, NE, UE, NS, \text{add-mset } (\text{add-mset } (-L) C') \text{ US, N0, U0}) \rangle$

if $\langle \text{count-decided } M = 0 \rangle$ **and** $\langle \neg \text{tautology } (C + C') \rangle$ **and** $\langle C \subseteq \# C' \rangle$

subresolution-IL:

$\langle \text{cdcl-subresolution } (M, N + \{\#\text{add-mset } L C\#\}, U + \{\#\text{add-mset } (-L) C'\#\}, D, NE, UE, NS,$

$US, N0, U0$
 $(M, N + \{\#remdups-mset\ C\# \}, U + \{\#add-mset\ (-L)\ C'\# \}, D, NE, UE,$
 $add-mset\ (add-mset\ L\ C)\ NS, US, N0, U0)$
if $\langle count-decided\ M = 0 \rangle$ **and** $\langle \neg tautology\ (C + C') \rangle$ **and** $\langle C' \subseteq\# C \rangle$

lemma *cdcl-subresolution*:

assumes $\langle cdcl-subresolution\ S\ T \rangle$
shows $\langle pcdcl^{**}\ S\ T \rangle$
 $\langle proof \rangle$

5.7 Variable elimination

This is a very first attempt to definit Variable elimination in a very general way. However, it is not clear how to handle tautologies (because the variable is not elimination in this case).

definition *elim-var* :: $\langle 'v \Rightarrow 'v\ clauses \Rightarrow 'v\ clauses \rangle$ **where**

$\langle elim-var\ L\ N =$
 $\{\#C \in\# N. L \notin\ atms-of\ C\#\} +$
 $(\lambda(C, D). removeAll-mset\ (Pos\ L)\ (removeAll-mset\ (Neg\ L)\ (C + D)))\ \#\$
 $((filter-mset\ (\lambda C. Pos\ L \in\# C)\ N) \times\# (filter-mset\ (\lambda C. Neg\ L \in\# C)\ N)) \rangle$

lemma

$\langle L \notin\ atms-of-mm\ (elim-var\ L\ N) \rangle$
 $\langle proof \rangle$

5.8 Bactrack for unit clause

This is the specific case where we learn a new unit clause and directly add it to the right place.

inductive *cdcl-backtrack-unit* :: $\langle 'v\ prag-st \Rightarrow 'v\ prag-st \Rightarrow bool \rangle$ **where**

$\langle cdcl-backtrack-unit\ (M, N, U, Some\ (add-mset\ L\ D), NE, UE, NS, US, N0, U0)$
 $(Propagated\ L\ \{\#L\#\}\ \#\ M1, N, U, None, NE, add-mset\ \{\#L\#\}\ UE, NS, US, N0, U0) \rangle$
if
 $\langle (Decided\ K\ \#\ M1, M2) \in\ set\ (get-all-ann-decomposition\ M) \rangle$ **and**
 $\langle get-level\ M\ L = count-decided\ M \rangle$ **and**
 $\langle get-level\ M\ L = get-maximum-level\ M\ \{\#L\#\} \rangle$ **and**
 $\langle get-level\ M\ K = 1 \rangle$ **and**
 $\langle N + U + NE + UE + NS + US + N0 + U0 \models pm\ \{\#L\#\} \rangle$

lemma *cdcl-backtrack-unit-is-backtrack*:

assumes $\langle cdcl-backtrack-unit\ S\ U \rangle$
obtains T **where**
 $\langle cdcl-backtrack\ S\ T \rangle$ **and**
 $\langle cdcl-flush-unit\ T\ U \rangle$
 $\langle proof \rangle$

lemma *cdcl-backtrack-unit-pcdcl-all-struct-invs*:

$\langle cdcl-backtrack-unit\ S\ T \Longrightarrow pcdcl-all-struct-invs\ S \Longrightarrow pcdcl-all-struct-invs\ T \rangle$
 $\langle proof \rangle$

lemma *cdcl-backtrack-unit-is-CDCL-backtrack*:

$\langle cdcl-backtrack-unit\ S\ T \Longrightarrow cdcl_W-restart-mset.backtrack\ (state-of\ S)\ (state-of\ T) \rangle$
 $\langle proof \rangle$

5.9 Subsume and promote

inductive *cdcl-subsumed-RI* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**
cdcl-subsumed-RI:
 $\langle \text{cdcl-subsumed-RI } (M, \text{add-mset } C' N, \text{add-mset } C U, D, NE, UE, NS, US, N0, U0)$
 $(M, \text{add-mset } C N, U, D, NE, UE, NS + \{\#C'\#\}, US, N0, U0) \rangle$
if $\langle C \subseteq \# C' \rangle \langle \neg \text{tautology } C \rangle \langle \text{distinct-mset } C \rangle$

lemma *cdcl-subsumed-RID*:

assumes
 $\langle \text{cdcl-subsumed-RI } S W \rangle$
obtains *T* **where**
 $\langle \text{cdcl-learn-clause } S T \rangle$ **and**
 $\langle \text{cdcl-subsumed } T W \rangle$
 $\langle \text{proof} \rangle$

lemma *cdcl-subsumed-RI-pcdcl*:

assumes
 $\langle \text{cdcl-subsumed-RI } S W \rangle$
shows
 $\langle \text{pcdcl}^{**} S W \rangle$
 $\langle \text{proof} \rangle$

5.10 Termination

We here define the terminating fragment of our pragmatic CDCL. Basically, there is a design decision to take here. Either we decide to move *cdcl-backtrack-unit* into the *pcdcl-core* or we decide to define a larger terminating fragment. The first option is easier (even if we need a larger core), but we want to be able to add subsumption test after backtrack and this requires an extension of the core anyway.

inductive *pcdcl-tcore* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **for** *S T* :: $\langle 'v \text{ prag-st} \rangle$ **where**
 $\langle \text{pcdcl-core } S T \Longrightarrow \text{pcdcl-tcore } S T \rangle$ |
 $\langle \text{cdcl-subsumed } S T \Longrightarrow \text{pcdcl-tcore } S T \rangle$ |
 $\langle \text{cdcl-flush-unit } S T \Longrightarrow \text{pcdcl-tcore } S T \rangle$ |
 $\langle \text{cdcl-backtrack-unit } S T \Longrightarrow \text{pcdcl-tcore } S T \rangle$

inductive *pcdcl-tcore-stgy* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **for** *S T* :: $\langle 'v \text{ prag-st} \rangle$ **where**
 $\langle \text{pcdcl-core-stgy } S T \Longrightarrow \text{pcdcl-tcore-stgy } S T \rangle$ |
 $\langle \text{cdcl-subsumed } S T \Longrightarrow \text{pcdcl-tcore-stgy } S T \rangle$ |
 $\langle \text{cdcl-flush-unit } S T \Longrightarrow \text{pcdcl-tcore-stgy } S T \rangle$ |
 $\langle \text{cdcl-backtrack-unit } S T \Longrightarrow \text{pcdcl-tcore-stgy } S T \rangle$

lemma *pcdcl-tcore-stgy-pcdcl-tcoreD*: $\langle \text{pcdcl-tcore-stgy } S T \Longrightarrow \text{pcdcl-tcore } S T \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-pcdcl-tcore-stgy-pcdcl-tcoreD*:
 $\langle \text{pcdcl-tcore-stgy}^{**} S T \Longrightarrow \text{pcdcl-tcore}^{**} S T \rangle$
 $\langle \text{proof} \rangle$

definition *pcdcl-core-measure* :: $\langle \rightarrow \rangle$ **where**

$\langle \text{pcdcl-core-measure} = \{(T, S). \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv } (\text{state-of } S) \wedge$
 $\text{cdcl}_W\text{-restart-mset.cdcl}_W (\text{state-of } S) (\text{state-of } T)\} \cup$
 $\{(T, S). \text{state-of } S = \text{state-of } T \wedge \text{size } (\text{pget-clauses } S) > \text{size } (\text{pget-clauses } T)\} \rangle$

lemma *wf-pcdcl-core-measure*:

⟨wf pcdcl-core-measure⟩

⟨proof⟩

lemma *pcdcl-tcore-in-pcdcl-core-measure*:

⟨{(T, S). pcdcl-all-struct-invs S ∧ pcdcl-tcore S T} ⊆ pcdcl-core-measure⟩

⟨proof⟩

lemmas *wf-pcdcl-tcore = wf-subset[OF wf-pcdcl-core-measure pcdcl-tcore-in-pcdcl-core-measure]*

lemma *wf-pcdcl-tcore-stgy*: ⟨wf {(T, S). pcdcl-all-struct-invs S ∧ pcdcl-tcore-stgy S T}⟩

⟨proof⟩

lemma *pcdcl-tcore-pcdcl*: ⟨pcdcl-tcore S T ⇒ pcdcl** S T⟩

⟨proof⟩

lemma *pcdcl-tcore-pcdcl-all-struct-invs*:

⟨pcdcl-tcore S T ⇒ pcdcl-all-struct-invs S ⇒ pcdcl-all-struct-invs T⟩

⟨proof⟩

lemma *pcdcl-core-stgy-no-smaller-propa*:

⟨pcdcl-core-stgy S T ⇒ pcdcl-all-struct-invs S ⇒ cdcl_W-restart-mset.no-smaller-propa (state-of S)⟩

⇒

cdcl_W-restart-mset.no-smaller-propa (state-of T)⟩

⟨proof⟩

lemma *pcdcl-stgy-no-smaller-propa*:

⟨pcdcl-stgy S T ⇒ pcdcl-all-struct-invs S ⇒ cdcl_W-restart-mset.no-smaller-propa (state-of S)⟩

cdcl_W-restart-mset.no-smaller-propa (state-of T)⟩

⟨proof⟩

lemma *rtranclp-pcdcl-stgy-no-smaller-propa*:

⟨pcdcl-stgy** S T ⇒ pcdcl-all-struct-invs S ⇒ cdcl_W-restart-mset.no-smaller-propa (state-of S)⟩

⇒

cdcl_W-restart-mset.no-smaller-propa (state-of T)⟩

⟨proof⟩

lemma *pcdcl-tcore-stgy-pcdcl-stgy*: ⟨pcdcl-tcore-stgy S T ⇒ pcdcl-stgy⁺⁺ S T⟩

⟨proof⟩

lemma *pcdcl-tcore-stgy-no-smaller-propa*:

⟨pcdcl-tcore-stgy S T ⇒ pcdcl-all-struct-invs S ⇒ cdcl_W-restart-mset.no-smaller-propa (state-of S)⟩

⇒

cdcl_W-restart-mset.no-smaller-propa (state-of T)⟩

⟨proof⟩

lemma *rtranclp-pcdcl-tcore-stgy-no-smaller-propa*:

⟨pcdcl-tcore-stgy** S T ⇒ pcdcl-all-struct-invs S ⇒ cdcl_W-restart-mset.no-smaller-propa (state-of S)⟩

⇒

cdcl_W-restart-mset.no-smaller-propa (state-of T)⟩

⟨proof⟩

lemma *pcdcl-stgy-stgy-invariant*:

$\langle \text{pcdcl-stgy } S \ T \implies \text{pcdcl-all-struct-invs } S \implies \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } S) \implies$
 $\text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } T) \rangle$
 (proof)

lemma *rtranclp-pcdcl-stgy-stgy-invariant*:
 $\langle \text{pcdcl-stgy}^{**} S \ T \implies \text{pcdcl-all-struct-invs } S \implies \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } S) \implies$
 $\text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } T) \rangle$
 (proof)

lemma *pcdcl-tcore-nocp-pcdcl-tcore-stgy*:
 $\langle \text{pcdcl-tcore } S \ T \implies \text{no-step cdcl-propagate } S \implies \text{no-step cdcl-conflict } S \implies \text{pcdcl-tcore-stgy } S \ T \rangle$
 (proof)

lemma *pcdcl-tcore-stgy-pcdcl-stgy'*: $\langle \text{pcdcl-tcore-stgy } S \ T \implies \text{pcdcl-stgy}^{**} S \ T \rangle$
 (proof)

lemma *rtranclp-pcdcl-tcore-stgy-pcdcl-stgy'*: $\langle \text{pcdcl-tcore-stgy}^{**} S \ T \implies \text{pcdcl-stgy}^{**} S \ T \rangle$
 (proof)

lemma *pcdcl-core-stgy-conflict-non-zero-unless-level-0*:
 $\langle \text{pcdcl-core-stgy } S \ T \implies \text{pcdcl-all-struct-invs } S \implies$
 $\text{cdcl}_W\text{-restart-mset.no-false-clause (state-of } S) \implies$
 $\text{cdcl}_W\text{-restart-mset.conflict-non-zero-unless-level-0 (state-of } S) \implies$
 $\text{cdcl}_W\text{-restart-mset.conflict-non-zero-unless-level-0 (state-of } T) \rangle$
 (proof)

lemma *pcdcl-stgy-conflict-non-zero-unless-level-0*:
 $\langle \text{pcdcl-stgy } S \ T \implies \text{pcdcl-all-struct-invs } S \implies$
 $\text{cdcl}_W\text{-restart-mset.no-false-clause (state-of } S) \implies$
 $\text{cdcl}_W\text{-restart-mset.conflict-non-zero-unless-level-0 (state-of } S) \implies$
 $\text{cdcl}_W\text{-restart-mset.conflict-non-zero-unless-level-0 (state-of } T) \rangle$
 (proof)

TODO: rename to full_t or something along that line.

definition *full2* :: $\langle ('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{full2 transf transf2} = (\lambda S \ S'. \text{rtranclp transf } S \ S' \wedge \text{no-step transf2 } S') \rangle$

end

theory *Pragmatic-CDCL-Restart*

imports *Pragmatic-CDCL*

begin

5.11 Restarts

While refactoring the code (or more precisely, creating the PCDCL version with restarts), I realised that the restarts as specified in my thesis are not exactly as I want to have them and not how they are implemented in SAT solvers. The problem is that I introduced both restart and garbage collection at the same time, but they have a different termination criterion:

- Restarts are always applicable as long you have learned at least one clause since the last

restart.

- GC must be applied after longer and longer time intervals.

In the version from the thesis, I use the second criterion to justify termination for both criteria. Due to the implementation, it did not really make a difference for small number of conflicts, but limited the ability to restart after many conflicts have been generated. I don't know if performance will change, as I already observed that changing the restart heuristic can have dramatic effects, but fixing it is always better, because it only gives more freedom to the implementation (including to not change anything).

The first criterion does not allow deleting clauses (including the useless US component as I did previously). The termination in both cases comes from non-relearning of clauses.

The proofs changed dramatically (and become much more messy) but that was expected, because the base calculus has changed a lot (new clauses can be learned).

Another difference is that after restarting, I allow anything following a CDCL run to happen. Proving that this terminates is delayed to the concrete implementation (e.g., vivification or HBR) and does not matter of the overall termination proof (but is obviously important in when generating code!).

inductive *pcdcl-restart* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**
restart-trail:

$\langle \text{pcdcl-restart } (M, N, U, \text{None}, NE, UE, NS, US, N0, U0)$
 $(M', N', U', \text{None}, NE + NE', UE'', NS, \{\#\}, N0, \{\#\}) \rangle$

if

$\langle (\text{Decided } K \# M', M2) \in \text{set } (\text{get-all-ann-decomposition } M) \rangle$ **and**
 $\langle U' + UE' \subseteq\# U \rangle$ **and**
 $\langle N = N' + NE' \rangle$ **and**
 $\langle \forall E \in \#NE' + UE'. \exists L \in \#E. L \in \text{lits-of-l } M' \wedge \text{get-level } M' L = 0 \rangle$
 $\langle \forall L E. \text{Propagated } L E \in \text{set } M' \longrightarrow E \in \#(N + U') + NE + UE'' \rangle$
 $\langle UE'' \subseteq\# UE + UE' \rangle$

restart-clauses:

$\langle \text{pcdcl-restart } (M, N, U, \text{None}, NE, UE, NS, US, N0, U0)$
 $(M, N', U', \text{None}, NE + NE', UE'', NS, US', N0, \{\#\}) \rangle$

if

$\langle U' + UE' \subseteq\# U \rangle$ **and**
 $\langle N = N' + NE' \rangle$ **and**
 $\langle \forall E \in \#NE' + UE'. \exists L \in \#E. L \in \text{lits-of-l } M \wedge \text{get-level } M L = 0 \rangle$
 $\langle \forall L E. \text{Propagated } L E \in \text{set } M \longrightarrow E \in \#(N + U') + NE + UE'' \rangle$
 $\langle UE'' \subseteq\# UE + UE' \rangle$
 $\langle US' = \{\#\} \rangle$

inductive *pcdcl-restart-only* :: $\langle 'v \text{ prag-st} \Rightarrow 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**
restart-trail:

$\langle \text{pcdcl-restart-only } (M, N, U, \text{None}, NE, UE, NS, US, N0, U0)$
 $(M', N, U, \text{None}, NE, UE, NS, US, N0, U0) \rangle$

if

$\langle (\text{Decided } K \# M', M2) \in \text{set } (\text{get-all-ann-decomposition } M) \vee M = M' \rangle$

lemma *mset-le-incr-right1*:

$a \subseteq\#(b :: 'a \text{ multiset}) \Longrightarrow a \subseteq\#b+c$ *(proof)*

lemma *pcdcl-restart-cdcl_W-stgy*:

fixes $S V :: \langle 'v \text{ prag-st} \rangle$

assumes

$\langle \text{pcdcl-restart } S V \rangle$ **and**

$\langle \text{pcdcl-all-struct-invs } S \rangle$ **and**

$\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } S) \rangle$ **and**

$\langle \text{cdcl}_W\text{-restart-mset.no-smaller-propa (state-of } S) \rangle$

shows

$\langle \exists T. \text{cdcl}_W\text{-restart-mset.restart (state-of } S) T \wedge \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy}^{**} T \text{ (state-of } V) \wedge$
 $\text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-restart}^{**} \text{ (state-of } S) \text{ (state-of } V) \rangle$

$\langle \text{proof} \rangle$

lemma *pcdcl-restart-cdcl_W*:

assumes

$\langle \text{pcdcl-restart } S V \rangle$ **and**

$\langle \text{pcdcl-all-struct-invs } S \rangle$

shows

$\langle \exists T. \text{cdcl}_W\text{-restart-mset.restart (state-of } S) T \wedge \text{cdcl}_W\text{-restart-mset.cdcl}_W^{**} T \text{ (state-of } V) \rangle$

$\langle \text{proof} \rangle$

lemma *mset-sum-eq-add-msetD*: $A + B = \text{add-mset } C D \implies C \in\# A \vee C \in\# B$

$\langle \text{proof} \rangle$

lemma (**in** $-$) *pcdcl-restart-pcdcl-all-struct-invs*:

fixes $S V :: \langle 'v \text{ prag-st} \rangle$

assumes

$\langle \text{pcdcl-restart } S V \rangle$ **and**

$\langle \text{pcdcl-all-struct-invs } S \rangle$

shows

$\langle \text{pcdcl-all-struct-invs } V \rangle$

$\langle \text{proof} \rangle$

lemma (**in** *conflict-driven-clause-learning-with-adding-init-clause_W*) *restart-no-smaller-propa*:

$\langle \text{restart } S T \implies \text{no-smaller-propa } S \implies \text{no-smaller-propa } T \rangle$

$\langle \text{proof} \rangle$

The next theorem does not use the existence of the decomposition to prove a much stronger version.

lemma (**in** $-$) *pcdcl-restart-no-smaller-propa*:

fixes $S V :: \langle 'v \text{ prag-st} \rangle$

assumes

$\langle \text{pcdcl-restart } S V \rangle$ **and**

$\langle \text{pcdcl-all-struct-invs } S \rangle$ **and**

$\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } S) \rangle$ **and**

$\langle \text{cdcl}_W\text{-restart-mset.no-smaller-propa (state-of } S) \rangle$

shows

$\langle \text{cdcl}_W\text{-restart-mset.no-smaller-propa (state-of } V) \rangle$

$\langle \text{proof} \rangle$

lemma *pcdcl-restart-no-smaller-propa'*:

fixes $S V :: \langle 'v \text{ prag-st} \rangle$

assumes

$\langle \text{pcdcl-restart } S V \rangle$ **and**

$\langle cdcl_W\text{-restart-mset.no-smaller-propa (state-of } S \rangle$
shows
 $\langle cdcl_W\text{-restart-mset.no-smaller-propa (state-of } V \rangle$
 $\langle \text{proof} \rangle$

lemma *pcdcl-restart-only-cdcl_W-stgy*:

fixes $S V :: \langle 'v \text{ prag-st} \rangle$

assumes

$\langle pcdcl\text{-restart-only } S V \rangle$ **and**

$\langle pcdcl\text{-all-struct-invs } S \rangle$ **and**

$\langle cdcl_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant (state-of } S \rangle$ **and**

$\langle cdcl_W\text{-restart-mset.no-smaller-propa (state-of } S \rangle$

shows

$\langle \exists T. cdcl_W\text{-restart-mset.restart (state-of } S) T \wedge cdcl_W\text{-restart-mset.cdcl}_W\text{-stgy}^{**} T (state-of } V) \wedge$
 $cdcl_W\text{-restart-mset.cdcl}_W\text{-restart}^{**} (state-of } S) (state-of } V) \rangle$

$\langle \text{proof} \rangle$

lemma *pcdcl-restart-only-cdcl_W*:

assumes

$\langle pcdcl\text{-restart-only } S V \rangle$ **and**

$\langle pcdcl\text{-all-struct-invs } S \rangle$

shows

$\langle \exists T. cdcl_W\text{-restart-mset.restart (state-of } S) T \wedge cdcl_W\text{-restart-mset.cdcl}_W^{**} T (state-of } V) \rangle$

$\langle \text{proof} \rangle$

lemma *pcdcl-restart-only-pcdcl-all-struct-invs*:

fixes $S V :: \langle 'v \text{ prag-st} \rangle$

assumes

$\langle pcdcl\text{-restart-only } S V \rangle$ **and**

$\langle pcdcl\text{-all-struct-invs } S \rangle$

shows

$\langle pcdcl\text{-all-struct-invs } V \rangle$

$\langle \text{proof} \rangle$

definition *pcdcl-final-state* :: $\langle 'v \text{ prag-st} \Rightarrow \text{bool} \rangle$ **where**

$\langle pcdcl\text{-final-state } S \longleftrightarrow \text{no-step pcdcl-core } S \vee$

$(\text{count-decided (pget-trail } S) = 0 \wedge \text{pget-conflict } S \neq \text{None}) \rangle$

context *twl-restart-ops*

begin

The following definition diverges from our previous attempt... mostly because we barely used it anyway. The problem was that we need to stop in final states which was not covered in the previous form.

The main issue is the termination of the calculus. Between two restarts we allow very abstract inprocessing that makes it possible to add clauses. However, this means that we can add the same clause over and over and that have reached the bound (or subsume these clauses).

TODO: add a forget rule in *pcdcl-stgy* instead of having it in restart.

The state is defined as an accumulator: The first component is the state we had after the last full restart (or the beginning of the search). We tried to make do without it, but the problem is managing to express the condition $f n < \text{size (pget-all-learned-clss } T) - \text{size (pget-all-learned-clss } R)$ without it. In a fast attempt, we completely oversaw that issue and had (in the current notations) $f n < \text{size (pget-all-learned-clss } T) - \text{size (pget-all-learned-clss } S)$. This has, however, a very different semantics and allows much fewer restarts.

One minor drawback is that we compare the number of clauses to the number of clauses after inprocessing instead of before inprocessing. The problem is that inprocessing could add the clause several times. I am not certain how to avoid that problem. An obvious solution is to ensure that no already-present clause is added (or at least that all duplicates have been removed) but it is not clear how to implement that inprocessing is not done until fixpoint.

type-synonym (in $-$) $'v$ prag-st-restart = $\langle 'v$ prag-st \times $'v$ prag-st \times $'v$ prag-st \times nat \times nat \times bool \rangle

abbreviation (in $-$) $current$ -state :: $\langle 'v$ prag-st-restart \Rightarrow $'v$ prag-st \rangle **where**
 $\langle current$ -state $S \equiv fst$ (snd (snd S)) \rangle

abbreviation (in $-$) $current$ -number :: $\langle 'v$ prag-st-restart \Rightarrow nat \rangle **where**
 $\langle current$ -number $S \equiv fst$ (snd (snd (snd (snd S)))) \rangle

abbreviation (in $-$) $current$ -restart-count :: $\langle 'v$ prag-st-restart \Rightarrow nat \rangle **where**
 $\langle current$ -restart-count $S \equiv fst$ (snd (snd (snd S))) \rangle

abbreviation (in $-$) $last$ -GC-state :: $\langle 'v$ prag-st-restart \Rightarrow $'v$ prag-st \rangle **where**
 $\langle last$ -GC-state $S \equiv fst$ S \rangle

abbreviation (in $-$) $last$ -restart-state :: $\langle 'v$ prag-st-restart \Rightarrow $'v$ prag-st \rangle **where**
 $\langle last$ -restart-state $S \equiv fst$ (snd S) \rangle

abbreviation (in $-$) $restart$ -continue :: $\langle 'v$ prag-st-restart \Rightarrow bool \rangle **where**
 $\langle restart$ -continue $S \equiv snd$ (snd (snd (snd (snd S)))) \rangle

As inprocessing, we allow a slightly different set of rules, including restart. Remember that the termination below takes the inprocessing as a granted (terminated) process. And without restrictions, the inprocessing does not terminate (it can restart).

On limitation of the following system is that *pcdcl-inprocessing* is not allowed to derive the empty clause. We tried to lift this limitation, but finally decided against it. The main problem is that the regularity and the termination gets lost due to that rule.

The right place to handle the special case is later when reaching the final state in our or wherever the predicate is used.

inductive (in $-$) $pcdcl$ -inprocessing :: $\langle 'v$ prag-st \Rightarrow $'v$ prag-st \Rightarrow bool \rangle
where

$\langle pcdcl$ S $T \Longrightarrow pcdcl$ -inprocessing S T \rangle |
 $\langle pcdcl$ -restart S $T \Longrightarrow pcdcl$ -inprocessing S T \rangle

inductive $pcdcl$ -stgy-restart
:: $\langle 'v$ prag-st-restart \Rightarrow $'v$ prag-st-restart \Rightarrow bool \rangle

where

step:

$\langle pcdcl$ -stgy-restart $(R, S, T, m, n, True)$ $(R, S, T', m, n, True)$ \rangle

if

$\langle pcdcl$ -tcore-stgy T T' \rangle |

restart-step:

$\langle pcdcl$ -stgy-restart $(R, S, T, m, n, True)$ $(W, W, W, m, Suc$ $n, True)$ \rangle

if

$\langle size$ (pget-all-learned-cls T) $-$ $size$ (pget-all-learned-cls R) $>$ f n \rangle **and**

$\langle pcdcl$ -inprocessing** T V \rangle

$\langle cdcl_W$ -restart-mset.no-smaller-propa (state-of V) \rangle **and**

$\langle pcdcl$ -stgy** V W \rangle |

restart-noGC-step:
 ⟨pcdcl-stgy-restart (R, S, T, m, n, True) (R, U, U, Suc m, n, True)⟩
if
 ⟨size (pget-all-learned-clss T) > size (pget-all-learned-clss S)⟩ **and**
 ⟨pcdcl-restart-only T U⟩ |
 restart-full:
 ⟨pcdcl-stgy-restart (R, S, T, m, n, True) (R, S, T, m, n, False)⟩
if
 ⟨pcdcl-final-state T⟩

end

lemma (in $-$) pcdcl-tcore-conflict-final-state-still:
assumes
 ⟨pcdcl-tcore S T⟩ **and**
 ⟨count-decided (pget-trail S) = 0 ∧ pget-conflict S ≠ None⟩
shows ⟨count-decided (pget-trail T) = 0 ∧ pget-conflict T ≠ None ∧
 pget-all-learned-clss S = pget-all-learned-clss T⟩
 ⟨proof⟩

lemma (in $-$) rtranclp-pcdcl-tcore-conflict-final-state-still:
assumes
 ⟨pcdcl-tcore** S T⟩ **and**
 ⟨count-decided (pget-trail S) = 0 ∧ pget-conflict S ≠ None⟩
shows
 ⟨count-decided (pget-trail T) = 0 ∧ pget-conflict T ≠ None ∧
 pget-all-learned-clss S = pget-all-learned-clss T⟩
 ⟨proof⟩

lemma pcdcl-tcore-no-core-no-learned:
assumes ⟨pcdcl-tcore S T⟩ **and**
 ⟨no-step pcdcl-core S⟩
shows ⟨pget-all-learned-clss S = pget-all-learned-clss T⟩
 ⟨proof⟩

lemma (in $-$) pcdcl-tcore-no-step-final-state-still:
assumes
 ⟨pcdcl-tcore S T⟩ **and**
 ⟨no-step pcdcl-core S⟩
shows ⟨no-step pcdcl-core T⟩
 ⟨proof⟩

lemma (in $-$) rtranclp-pcdcl-tcore-no-step-final-state-still:
assumes
 ⟨pcdcl-tcore** S T⟩ **and**
 ⟨no-step pcdcl-core S⟩
shows ⟨no-step pcdcl-core T⟩
 ⟨proof⟩

lemma rtranclp-pcdcl-tcore-no-core-no-learned:
assumes ⟨pcdcl-tcore** S T⟩ **and**
 ⟨no-step pcdcl-core S⟩
shows ⟨pget-all-learned-clss S = pget-all-learned-clss T⟩
 ⟨proof⟩

lemma *no-step-pcdcl-core-stgy-pcdcl-coreD*:
 $\langle \text{no-step pcdcl-core-stgy } S \implies \text{no-step pcdcl-core } S \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-pcdcl-tcore-stgy-no-core-no-learned*:
assumes $\langle \text{pcdcl-tcore-stgy}^{**} S T \rangle$ **and**
 $\langle \text{no-step pcdcl-core } S \rangle$
shows $\langle \text{pget-all-learned-clss } S = \text{pget-all-learned-clss } T \rangle$
 $\langle \text{proof} \rangle$

inductive *pcdcl-stgy-only-restart* **for** S **where**
restart-noGC-step:
 $\langle \text{pcdcl-stgy-only-restart } (S) (U) \rangle$
if
 $\langle \text{pcdcl-tcore-stgy}^{++} S T \rangle$ **and**
 $\langle \text{size } (\text{pget-all-learned-clss } T) > \text{size } (\text{pget-all-learned-clss } S) \rangle$ **and**
 $\langle \text{pcdcl-restart-only } T U \rangle$

lemma *restart-noGC-stepI*:
 $\langle \text{pcdcl-stgy-only-restart } (S) (U) \rangle$
if
 $\langle \text{pcdcl-tcore-stgy}^{**} S T \rangle$ **and**
 $\langle \text{size } (\text{pget-all-learned-clss } T) > \text{size } (\text{pget-all-learned-clss } S) \rangle$ **and**
 $\langle \text{pcdcl-restart-only } T U \rangle$
 $\langle \text{proof} \rangle$

lemma *pcdcl-tcore-stgy-step-or-unchanged*:
 $\langle \text{pcdcl-tcore-stgy } S T \implies \text{pcdcl-core-stgy}^{**} S T \vee \text{state-of } S = \text{state-of } T \vee$
 $(\exists T'. \text{cdcl-backtrack } S T' \wedge \text{state-of } T' = \text{state-of } T) \rangle$
 $\langle \text{proof} \rangle$

lemma *pcdcl-tcore-stgy-cdcl_W-stgy*:
 $\langle \text{pcdcl-tcore-stgy } S T \implies \text{pcdcl-all-struct-invs } S \implies$
 $\text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy}^{**} (\text{state-of } S) (\text{state-of } T) \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-pcdcl-tcore-stgy-cdcl_W-stgy*:
 $\langle \text{pcdcl-tcore-stgy}^{**} S T \implies \text{pcdcl-all-struct-invs } S \implies$
 $\text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy}^{**} (\text{state-of } S) (\text{state-of } T) \rangle$
 $\langle \text{proof} \rangle$

lemma [*simp*]: $\langle \text{learned-clss } (\text{state-of } S) = \text{pget-all-learned-clss } S \rangle$
 $\langle \text{proof} \rangle$

lemma
pcdcl-tcore-stgy-init-learned:
 $\langle \text{pcdcl-tcore-stgy } S T \implies \text{pget-init-learned-clss } S \subseteq\# \text{pget-init-learned-clss } T \rangle$ **and**
pcdcl-tcore-stgy-psubsumed-learned-clauses:
 $\langle \text{pcdcl-tcore-stgy } S T \implies \text{psubsumed-learned-clauses } S \subseteq\# \text{psubsumed-learned-clauses } T \rangle$ **and**
pcdcl-tcore-stgy-plearned-clauses0:
 $\langle \text{pcdcl-tcore-stgy } S T \implies \text{plearned-clauses0 } S \subseteq\# \text{plearned-clauses0 } T \rangle$
 $\langle \text{proof} \rangle$

lemma

rtranclp-pcdcl-tcore-stgy-init-learned:
 $\langle \text{pcdcl-tcore-stgy}^{**} S T \implies \text{pget-init-learned-clss } S \subseteq\# \text{pget-init-learned-clss } T \rangle$ **and**
rtranclp-pcdcl-tcore-stgy-psubsumed-learned-clauses:
 $\langle \text{pcdcl-tcore-stgy}^{**} S T \implies \text{psubsumed-learned-clauses } S \subseteq\# \text{psubsumed-learned-clauses } T \rangle$ **and**
rtranclp-pcdcl-tcore-stgy-plearned-clauses0:
 $\langle \text{pcdcl-tcore-stgy}^{**} S T \implies \text{plearned-clauses0 } S \subseteq\# \text{plearned-clauses0 } T \rangle$
 ⟨proof⟩

lemma

pcdcl-stgy-only-restart-init-learned:
 $\langle \text{pcdcl-stgy-only-restart } S T \implies \text{pget-init-learned-clss } S \subseteq\# \text{pget-init-learned-clss } T \rangle$ **and**
pcdcl-stgy-only-restart-psubsumed-learned-clauses:
 $\langle \text{pcdcl-stgy-only-restart } S T \implies \text{psubsumed-learned-clauses } S \subseteq\# \text{psubsumed-learned-clauses } T \rangle$ **and**
pcdcl-stgy-only-restart-plearned-clauses0:
 $\langle \text{pcdcl-stgy-only-restart } S T \implies \text{plearned-clauses0 } S \subseteq\# \text{plearned-clauses0 } T \rangle$
 ⟨proof⟩

lemma *cdcl-tw1-stgy-restart-new:*

assumes
 $\langle \text{pcdcl-stgy-only-restart } S U \rangle$ **and**
invs: $\langle \text{pcdcl-all-struct-invs } S \rangle$ **and**
propa: $\langle \text{cdcl}_W\text{-restart-mset.no-smaller-propa (state-of } S) \rangle$ **and**
dist: $\langle \text{distinct-mset (pget-learned-clss } S - A) \rangle$
shows $\langle \text{distinct-mset (pget-learned-clss } U - A) \rangle$
 ⟨proof⟩

lemma *cdcl-tw1-stgy-restart-new':*

assumes
 $\langle \text{pcdcl-stgy-only-restart } S U \rangle$ **and**
invs: $\langle \text{pcdcl-all-struct-invs } S \rangle$ **and**
propa: $\langle \text{cdcl}_W\text{-restart-mset.no-smaller-propa (state-of } S) \rangle$ **and**
dist: $\langle \text{distinct-mset (pget-all-learned-clss } S - A) \rangle$
shows $\langle \text{distinct-mset (pget-all-learned-clss } U - A) \rangle$
 ⟨proof⟩

lemma *pcdcl-stgy-only-restart-all-struct-invs:*

$\langle \text{pcdcl-stgy-only-restart } S T \implies \text{pcdcl-all-struct-invs } S \implies \text{pcdcl-all-struct-invs } T \rangle$
 ⟨proof⟩

lemma *rtranclp-pcdcl-stgy-only-restart-all-struct-invs:*

$\langle \text{pcdcl-stgy-only-restart}^{**} S T \implies \text{pcdcl-all-struct-invs } S \implies \text{pcdcl-all-struct-invs } T \rangle$
 ⟨proof⟩

lemma *pcdcl-tcore-stgy-all-struct-invs:*

$\langle \text{pcdcl-tcore-stgy } S T \implies \text{pcdcl-all-struct-invs } S \implies \text{pcdcl-all-struct-invs } T \rangle$
 ⟨proof⟩

lemma *rtranclp-pcdcl-tcore-stgy-all-struct-invs:*

$\langle \text{pcdcl-tcore-stgy}^{**} S T \implies \text{pcdcl-all-struct-invs } S \implies \text{pcdcl-all-struct-invs } T \rangle$
 ⟨proof⟩

lemma *pcdcl-restart-only-no-smaller-propa:*

$\langle \text{pcdcl-restart-only } S T \implies \text{pcdcl-all-struct-invs } S \implies$
 $\text{cdcl}_W\text{-restart-mset.no-smaller-propa (state-of } S) \implies$
 $\text{cdcl}_W\text{-restart-mset.no-smaller-propa (state-of } T) \rangle$
 $\langle \text{proof} \rangle$

lemma *pcdcl-stgy-only-restart-no-smaller-propa*:
 $\langle \text{pcdcl-stgy-only-restart } S T \implies \text{pcdcl-all-struct-invs } S \implies$
 $\text{cdcl}_W\text{-restart-mset.no-smaller-propa (state-of } S) \implies$
 $\text{cdcl}_W\text{-restart-mset.no-smaller-propa (state-of } T) \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-pcdcl-stgy-only-restart-no-smaller-propa*:
 $\langle \text{pcdcl-stgy-only-restart}^{**} S T \implies \text{pcdcl-all-struct-invs } S \implies \text{cdcl}_W\text{-restart-mset.no-smaller-propa}$
 $(\text{state-of } S) \implies$
 $\text{cdcl}_W\text{-restart-mset.no-smaller-propa (state-of } T) \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-cdcl-twl-stgy-restart-new-abs*:
assumes
 $\langle \text{pcdcl-stgy-only-restart}^{**} S T \rangle$ **and**
 $\text{invs: } \langle \text{pcdcl-all-struct-invs } S \rangle$ **and**
 $\text{propa: } \langle \text{cdcl}_W\text{-restart-mset.no-smaller-propa (state-of } S) \rangle$ **and**
 $\text{dist: } \langle \text{distinct-mset (pget-learned-clss } S - A) \rangle$
shows $\langle \text{distinct-mset (pget-learned-clss } T - A) \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-cdcl-twl-stgy-restart-new-abs'*:
assumes
 $\langle \text{pcdcl-stgy-only-restart}^{**} S T \rangle$ **and**
 $\text{invs: } \langle \text{pcdcl-all-struct-invs } S \rangle$ **and**
 $\text{propa: } \langle \text{cdcl}_W\text{-restart-mset.no-smaller-propa (state-of } S) \rangle$ **and**
 $\text{dist: } \langle \text{distinct-mset (pget-all-learned-clss } S - A) \rangle$
shows $\langle \text{distinct-mset (pget-all-learned-clss } T - A) \rangle$
 $\langle \text{proof} \rangle$

lemma *pcdcl-tcore-stgy-pget-all-learned-clss-mono*:
 $\langle \text{pcdcl-tcore-stgy } S T \implies \text{size (pget-all-learned-clss } S) \leq \text{size (pget-all-learned-clss } T) \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-pcdcl-tcore-stgy-pget-all-learned-clss-mono*:
 $\langle \text{pcdcl-tcore-stgy}^{**} S T \implies \text{size (pget-all-learned-clss } S) \leq \text{size (pget-all-learned-clss } T) \rangle$
 $\langle \text{proof} \rangle$

lemma *pcdcl-stgy-only-restart-pget-all-learned-clss-mono*:
 $\langle \text{pcdcl-stgy-only-restart } S T \implies \text{size (pget-all-learned-clss } S) \leq \text{size (pget-all-learned-clss } T) \rangle$
 $\langle \text{proof} \rangle$

lemma [*simp*]: $\langle \text{init-clss (state-of } S) = \text{pget-all-init-clss } S \rangle$
 $\langle \text{proof} \rangle$

lemma *pcdcl-tcore-stgy-pget-all-init-clss*:
 $\langle \text{pcdcl-tcore-stgy } S T \implies \text{pget-all-init-clss } S = \text{pget-all-init-clss } T \rangle$
 $\langle \text{proof} \rangle$

lemma *rtranclp-pcdcl-tcore-stgy-pget-all-init-clss*:
 $\langle \text{pcdcl-tcore-stgy}^{**} S T \implies \text{pget-all-init-clss } S = \text{pget-all-init-clss } T \rangle$

⟨proof⟩

TODO: Move

lemma *card-simple-cls-with-tautology*:

assumes ⟨finite N ⟩

shows ⟨finite $\{C. \text{atms-of } C \subseteq N \wedge \text{distinct-mset } C\}$ ⟩ **and**
⟨card $\{C. \text{atms-of } C \subseteq N \wedge \text{distinct-mset } C\} \leq 4 \wedge \text{card } N$ ⟩

⟨proof⟩

lemma *pcdcl-stgy-only-restart-pget-all-init-cls*:

⟨pcdcl-stgy-only-restart $S T \implies \text{pget-all-init-cls } S = \text{pget-all-init-cls } T$ ⟩

⟨proof⟩

lemma *rtranclp-pcdcl-stgy-only-restart-pget-all-init-cls*:

⟨pcdcl-stgy-only-restart** $S T \implies \text{pget-all-init-cls } S = \text{pget-all-init-cls } T$ ⟩

⟨proof⟩

lemma

assumes ⟨pcdcl-stgy-only-restart** $S T$ ⟩ **and**

⟨pcdcl-all-struct-invs S ⟩ **and**

⟨cdcl_W-restart-mset.no-smaller-propa (state-of S)⟩

shows

rtranclp-pcdcl-stgy-only-restart-distinct-mset:

⟨distinct-mset (pget-all-learned-cls $T - \text{pget-all-learned-cls } S$)⟩ **and**

rtranclp-pcdcl-stgy-only-restart-bound:

⟨card (set-mset (pget-all-learned-cls $T - \text{pget-all-learned-cls } S$))

$\leq 4 \wedge (\text{card } (\text{atms-of-mm } (\text{pget-all-init-cls } S)))$ ⟩ **and**

rtranclp-pcdcl-stgy-only-restart-bound-size:

⟨size (pget-all-learned-cls $T - \text{pget-all-learned-cls } S$)

$\leq 4 \wedge (\text{card } (\text{atms-of-mm } (\text{pget-all-init-cls } S)))$ ⟩

⟨proof⟩

lemma *wf-pcdcl-stgy-only-restart*:

⟨wf $\{(T, S :: 'v \text{ prag-st}). \text{pcdcl-all-struct-invs } S \wedge$

$\text{cdcl}_W\text{-restart-mset.no-smaller-propa (state-of } S) \wedge \text{pcdcl-stgy-only-restart } S T\}$ ⟩

⟨proof⟩

lemma *pcdcl-core-stgy-pget-all-init-cls*:

⟨pcdcl-core-stgy $S T \implies \text{atms-of-mm } (\text{pget-all-init-cls } S) =$
 $\text{atms-of-mm } (\text{pget-all-init-cls } T)$ ⟩

⟨proof⟩

lemma *cdcl-unitres-true-all-init-cls*:

⟨cdcl-unitres-true $S T \implies (\text{pget-all-init-cls } S) = (\text{pget-all-init-cls } T)$ ⟩

⟨proof⟩

lemma *pcdcl-stgy-pget-all-init-cls*:

⟨pcdcl-stgy $S T \implies \text{pcdcl-all-struct-invs } S \implies \text{atms-of-mm } (\text{pget-all-init-cls } S) =$
 $\text{atms-of-mm } (\text{pget-all-init-cls } T)$ ⟩

⟨proof⟩

lemma *rtranclp-pcdcl-stgy-pget-all-init-cls*:

⟨pcdcl-stgy** $S T \implies \text{pcdcl-all-struct-invs } S \implies$

$\text{atms-of-mm } (\text{pget-all-init-cls } S) = \text{atms-of-mm } (\text{pget-all-init-cls } T)$ ⟩

⟨proof⟩

lemma *pcdcl-tcore-pget-all-init-clss:*

⟨ $pcdcl-tcore\ S\ T \implies pget-all-init-clss\ S = pget-all-init-clss\ T$ ⟩

⟨proof⟩

lemma *rtranclp-pcdcl-tcore-pget-all-init-clss:*

⟨ $pcdcl-tcore^{**}\ S\ T \implies pget-all-init-clss\ S = pget-all-init-clss\ T$ ⟩

⟨proof⟩

lemma *pcdcl-core-pget-all-init-clss:*

⟨ $pcdcl-core\ S\ T \implies pget-all-init-clss\ S = pget-all-init-clss\ T$ ⟩

⟨proof⟩

lemma *rtranclp-pcdcl-core-pget-all-init-clss:*

⟨ $pcdcl-core^{**}\ S\ T \implies pget-all-init-clss\ S = pget-all-init-clss\ T$ ⟩

⟨proof⟩

lemma *pcdcl-pget-all-init-clss:*

⟨ $pcdcl\ S\ T \implies pcdcl-all-struct-invs\ S \implies atms-of-mm\ (pget-all-init-clss\ S) =$
 $atms-of-mm\ (pget-all-init-clss\ T)$ ⟩

⟨proof⟩

lemma *rtranclp-pcdcl-pget-all-init-clss:*

⟨ $pcdcl^{**}\ S\ T \implies pcdcl-all-struct-invs\ S \implies atms-of-mm\ (pget-all-init-clss\ S) =$
 $atms-of-mm\ (pget-all-init-clss\ T)$ ⟩

⟨proof⟩

lemma *pcdcl-inprocessing-pget-all-init-clss:*

⟨ $pcdcl-inprocessing\ S\ T \implies pcdcl-all-struct-invs\ S \implies$
 $atms-of-mm\ (pget-all-init-clss\ T) = atms-of-mm\ (pget-all-init-clss\ S)$ ⟩

⟨proof⟩

lemma *pcdcl-inprocessing-pcdcl-all-struct-invs:*

⟨ $pcdcl-inprocessing\ S\ T \implies pcdcl-all-struct-invs\ S \implies pcdcl-all-struct-invs\ T$ ⟩

⟨proof⟩

lemma *rtranclp-pcdcl-inprocessing-pcdcl-all-struct-invs:*

⟨ $pcdcl-inprocessing^{**}\ S\ T \implies pcdcl-all-struct-invs\ S \implies pcdcl-all-struct-invs\ T$ ⟩

⟨proof⟩

lemma *rtranclp-pcdcl-inprocessing-pget-all-init-clss:*

⟨ $pcdcl-inprocessing^{**}\ S\ T \implies pcdcl-all-struct-invs\ S \implies atms-of-mm\ (pget-all-init-clss\ T) = atms-of-mm$
 $(pget-all-init-clss\ S)$ ⟩

⟨proof⟩

context *twl-restart-ops*

begin

lemma *pcdcl-stgy-restart-pget-all-init-clss:*

⟨ $pcdcl-stgy-restart\ S\ T \implies pcdcl-all-struct-invs\ (current-state\ S) \implies$
 $atms-of-mm\ (pget-all-init-clss\ (last-restart-state\ S)) = atms-of-mm\ (pget-all-init-clss\ (current-state$
 $S)) \implies$
 $atms-of-mm\ (pget-all-init-clss\ (last-GC-state\ S)) = atms-of-mm\ (pget-all-init-clss\ (current-state\ S))$ ⟩

⟨proof⟩

⟨ $atms-of-mm\ (pget-all-init-clss\ (current-state\ T)) = atms-of-mm\ (pget-all-init-clss\ (current-state\ S)) \wedge$ ⟩

$atms-of-mm (pget-all-init-cls (last-restart-state T)) = atms-of-mm (pget-all-init-cls (current-state S)) \wedge$
 $atms-of-mm (pget-all-init-cls (last-GC-state T)) = atms-of-mm (pget-all-init-cls (current-state S)) \rangle$
 ⟨proof⟩

definition $pcdcl-stgy-restart-inv :: \langle 'v prag-st-restart \Rightarrow bool \rangle$ **where**
 $\langle pcdcl-stgy-restart-inv = (\lambda(Q, R, S, m, n). pcdcl-all-struct-invs Q \wedge$
 $cdcl_W-restart-mset.no-smaller-propa (state-of Q) \wedge$
 $pcdcl-stgy-only-restart^{**} Q R \wedge pcdcl-tcore-stgy^{**} R S) \rangle$

lemma $pcdcl-stgy-restart-inv-alt-def$:
 $\langle pcdcl-stgy-restart-inv = (\lambda(Q, R, S, m, n). pcdcl-all-struct-invs Q \wedge$
 $cdcl_W-restart-mset.no-smaller-propa (state-of Q) \wedge pcdcl-all-struct-invs R \wedge$
 $cdcl_W-restart-mset.no-smaller-propa (state-of R) \wedge pcdcl-all-struct-invs S \wedge$
 $cdcl_W-restart-mset.no-smaller-propa (state-of S) \wedge$
 $pcdcl-stgy-only-restart^{**} Q R \wedge pcdcl-tcore-stgy^{**} R S) \rangle$
 (is $\langle - = ?P \rangle$)
 ⟨proof⟩

lemma $no-smaller-propa-lvl0$:
 $\langle count-decided (pget-trail U) = 0 \Longrightarrow cdcl_W-restart-mset.no-smaller-propa (state-of U) \rangle$
 ⟨proof⟩

lemma $pcdcl-stgy-restart-pcdcl-stgy-restart-inv$:
assumes $\langle pcdcl-stgy-restart S T \rangle \langle pcdcl-stgy-restart-inv S \rangle$
shows $\langle pcdcl-stgy-restart-inv T \rangle$
 ⟨proof⟩

lemma $rtranclp-pcdcl-stgy-restart-pcdcl-stgy-restart-inv$:
assumes $\langle pcdcl-stgy-restart^{**} S T \rangle \langle pcdcl-stgy-restart-inv S \rangle$
shows $\langle pcdcl-stgy-restart-inv T \rangle$
 ⟨proof⟩

lemma $pcdcl-stgy-restart-current-number$:
 $\langle pcdcl-stgy-restart S T \Longrightarrow current-number S \leq current-number T \rangle$
 ⟨proof⟩

lemma $rtranclp-pcdcl-stgy-restart-current-number$:
 $\langle pcdcl-stgy-restart^{**} S T \Longrightarrow current-number S \leq current-number T \rangle$
 ⟨proof⟩

lemma $pcdcl-stgy-restart-no-GC-either$:
 $\langle pcdcl-stgy-restart S T \Longrightarrow pcdcl-stgy-restart-inv S \Longrightarrow$
 $current-number T = current-number S \Longrightarrow$
 $(last-restart-state T = current-state T \wedge pcdcl-stgy-only-restart (last-restart-state S) (current-state T))$
 \vee
 $(last-restart-state S = last-restart-state T \wedge pcdcl-tcore-stgy (current-state S) (current-state T)) \vee$
 $(last-restart-state S = last-restart-state T \wedge (current-state S) = (current-state T) \wedge \neg restart-continue$
 $T) \rangle$
 ⟨proof⟩

lemma $rtranclp-pcdcl-stgy-restart-decomp$:
 $\langle pcdcl-stgy-restart^{**} S T \Longrightarrow pcdcl-stgy-restart-inv S \Longrightarrow$
 $current-number T = current-number S \Longrightarrow pcdcl-stgy-only-restart^{**} (current-state S) (last-restart-state$
 $S) \Longrightarrow$

$\langle \text{pcdcl-stgy-only-restart}^{**} (\text{current-state } S) (\text{last-restart-state } T) \wedge \text{pcdcl-tcore-stgy}^{**} (\text{last-restart-state } T) (\text{current-state } T) \rangle$
 $\langle \text{proof} \rangle$

lemma

assumes $\langle \text{pcdcl-stgy-restart}^{**} S T \rangle$ **and**
 $\langle \text{inv: pcdcl-stgy-restart-inv } S \rangle$ **and**
 $\langle \text{current-number } T = \text{current-number } S \rangle$ **and**
 $\langle \text{pcdcl-stgy-only-restart}^{**} (\text{current-state } S) (\text{last-restart-state } S) \rangle$ **and**
 $\langle \text{distinct-mset } (\text{pget-all-learned-clss } (\text{current-state } S) - A) \rangle$

shows

$\text{rtranclp-pcdcl-stgy-restart-distinct-mset:}$
 $\langle \text{distinct-mset } (\text{pget-all-learned-clss } (\text{current-state } T) - A) \rangle$ **(is ?dist) and**
 $\text{rtranclp-pcdcl-stgy-restart-bound:}$
 $\langle \text{card } (\text{set-mset } (\text{pget-all-learned-clss } (\text{current-state } T) - A))$
 $\leq 4 \wedge (\text{card } (\text{atms-of-mm } (\text{pget-all-init-clss } (\text{current-state } S)))) \rangle$ **(is ?A) and**
 $\text{rtranclp-pcdcl-stgy-restart-bound-size:}$
 $\langle \text{size } (\text{pget-all-learned-clss } (\text{current-state } T) - A)$
 $\leq 4 \wedge (\text{card } (\text{atms-of-mm } (\text{pget-all-init-clss } (\text{current-state } S)))) \rangle$ **(is ?B)**

$\langle \text{proof} \rangle$

lemma $\text{pcdcl-stgy-restart-mono:}$

$\langle \text{pcdcl-stgy-restart } S T \implies \text{current-number } S = \text{current-number } T \implies$
 $\text{pget-all-learned-clss } (\text{current-state } S) \subseteq\# \text{pget-all-learned-clss } (\text{current-state } T) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{rtranclp-pcdcl-stgy-restart-mono:}$

$\langle \text{pcdcl-stgy-restart}^{**} S T \implies \text{current-number } S = \text{current-number } T \implies$
 $\text{pget-all-learned-clss } (\text{current-state } S) \subseteq\# \text{pget-all-learned-clss } (\text{current-state } T) \rangle$
 $\langle \text{proof} \rangle$

end

The termination proof contains the usual boilerplate that hide the real argument. The idea of the proof is to consider an infinite chain of $\text{pcdcl-stgy-restart}$. From it:

- We know that have eventually to do a restart, as pcdcl-tcore-stgy terminates.
- Then, if there are no GC, as we will eventually restart we can extract a sequence of $\text{pcdcl-stgy-only-restart}$. That terminates, so we have to eventually GC.
- Finally, as f is unbounded, at some we have learned more clauses that admissible.

The proof is make more complicated by the extraction of the subsequences: we derive the subsequence of indices f' , then by induction we prove properties on the subsequence.

context twl-restart

begin

theorem $\text{wf-pcdcl-twl-stgy-restart:}$

$\langle \text{wf } \{(T, S :: 'v \text{ prag-st-restart}). \text{pcdcl-stgy-restart-inv } S \wedge \text{pcdcl-stgy-restart } S T\} \rangle$
 $\langle \text{proof} \rangle$

end

end