Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

March 23, 2016

Contents

theory *Wellfounded-More*
imports *Main*

begin

1 Transitions

This theory contains more facts about closure, the definition of full transformations, and well-foundedness.

1.1 More theorems about Closures

This is the equivalent of $?r \leq ?s \Longrightarrow ?r^{**} \leq ?s^{**}$ for *tranclp*

lemma *tranclp-mono-explicit*:
$r^{++}\ a\ b \Longrightarrow r \leq s \Longrightarrow s^{++}\ a\ b$
⟨*proof*⟩

lemma *tranclp-mono*:
assumes *mono*: $r \leq s$
shows $r^{++} \leq s^{++}$
⟨*proof*⟩

lemma *tranclp-idemp-rel*:
$R^{++++}\ a\ b \longleftrightarrow R^{++}\ a\ b$
⟨*proof*⟩

Equivalent of $?r^{****} = ?r^{**}$

lemma *trancl-idemp*: $(r^+)^+ = r^+$
⟨*proof*⟩

lemmas *tranclp-idemp*[*simp*] = *trancl-idemp*[*to-pred*]

This theorem already exists as $?r^{**}\ ?a\ ?b \equiv ?a = ?b \lor ?r^{++}\ ?a\ ?b$ (and sledgehammer uses it), but it makes sense to duplicate it, because it is unclear how stable the lemmas in Nitpick are.

lemma *rtranclp-unfold*: $rtranclp\ r\ a\ b \longleftrightarrow (a = b \lor tranclp\ r\ a\ b)$
⟨*proof*⟩

lemma *tranclp-unfold-end*: $tranclp\ r\ a\ b \longleftrightarrow (\exists\, a'.\ rtranclp\ r\ a\ a' \land r\ a'\ b)$
⟨*proof*⟩

lemma *tranclp-unfold-begin*: $tranclp\ r\ a\ b \longleftrightarrow (\exists\, a'.\ r\ a\ a' \land rtranclp\ r\ a'\ b)$
⟨*proof*⟩

lemma *trancl-set-tranclp*: $(a,\ b) \in \{(b,a).\ P\ a\ b\}^+ \longleftrightarrow P^{++}\ b\ a$
⟨*proof*⟩

lemma *tranclp-rtranclp-rtranclp-rel*: $R^{++**}\ a\ b \longleftrightarrow R^{**}\ a\ b$
⟨*proof*⟩

lemma *tranclp-rtranclp-rtranclp*[*simp*]: $R^{++**} = R^{**}$
⟨*proof*⟩

lemma *rtranclp-exists-last-with-prop*:
assumes $R\ x\ z$
and $R^{**}\ z\ z'$ **and** $P\ x\ z$
shows $\exists\, y\ y'.\ R^{**}\ x\ y \land R\ y\ y' \land P\ y\ y' \land (\lambda a\ b.\ R\ a\ b \land \neg P\ a\ b)^{**}\ y'\ z'$
⟨*proof*⟩

lemma *rtranclp-and-rtranclp-left*: $(\lambda\ a\ b.\ P\ a\ b \land Q\ a\ b)^{**}\ S\ T \Longrightarrow P^{**}\ S\ T$
⟨*proof*⟩

1.2 Full Transitions

We define here properties to define properties after all possible transitions.

abbreviation *no-step step S* $\equiv (\forall\, S'.\ \neg step\ S\ S')$

definition *full1* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ **where**
$full1\ transf = (\lambda S\ S'.\ tranclp\ transf\ S\ S' \land (\forall\, S''.\ \neg\ transf\ S'\ S''))$

definition *full*:: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ **where**
$full\ transf = (\lambda S\ S'.\ rtranclp\ transf\ S\ S' \land (\forall\, S''.\ \neg\ transf\ S'\ S''))$

lemma *rtranclp-full1I*:
$R^{**}\ a\ b \Longrightarrow full1\ R\ b\ c \Longrightarrow full1\ R\ a\ c$
⟨*proof*⟩

lemma *tranclp-full1I*:
$R^{++}\ a\ b \Longrightarrow full1\ R\ b\ c \Longrightarrow full1\ R\ a\ c$
⟨*proof*⟩

lemma *rtranclp-fullI*:
$R^{**}\ a\ b \Longrightarrow \mathit{full}\ R\ b\ c \Longrightarrow \mathit{full}\ R\ a\ c$
⟨*proof*⟩

lemma *tranclp-full-full1I*:
$R^{++}\ a\ b \Longrightarrow \mathit{full}\ R\ b\ c \Longrightarrow \mathit{full1}\ R\ a\ c$
⟨*proof*⟩

lemma *full-fullI*:
$R\ a\ b \Longrightarrow \mathit{full}\ R\ b\ c \Longrightarrow \mathit{full1}\ R\ a\ c$
⟨*proof*⟩

lemma *full-unfold*:
$\mathit{full}\ r\ S\ S' \longleftrightarrow ((S = S' \land \mathit{no\text{-}step}\ r\ S') \lor \mathit{full1}\ r\ S\ S')$
⟨*proof*⟩

lemma *full1-is-full*[*intro*]: $\mathit{full1}\ R\ S\ T \Longrightarrow \mathit{full}\ R\ S\ T$
⟨*proof*⟩

lemma *not-full1-rtranclp-relation*: $\neg\mathit{full1}\ R^{**}\ a\ b$
⟨*proof*⟩

lemma *not-full-rtranclp-relation*: $\neg\mathit{full}\ R^{**}\ a\ b$
⟨*proof*⟩

lemma *full1-tranclp-relation-full*:
$\mathit{full1}\ R^{++}\ a\ b \longleftrightarrow \mathit{full1}\ R\ a\ b$
⟨*proof*⟩

lemma *full-tranclp-relation-full*:
$\mathit{full}\ R^{++}\ a\ b \longleftrightarrow \mathit{full}\ R\ a\ b$
⟨*proof*⟩

lemma *rtranclp-full1-eq-or-full1*:
$(\mathit{full1}\ R)^{**}\ a\ b \longleftrightarrow (a = b \lor \mathit{full1}\ R\ a\ b)$
⟨*proof*⟩

lemma *tranclp-full1-full1*:
$(\mathit{full1}\ R)^{++}\ a\ b \longleftrightarrow \mathit{full1}\ R\ a\ b$
⟨*proof*⟩

1.3 Well-Foundedness and Full Transitions

lemma *wf-exists-normal-form*:
assumes $\mathit{wf}{:}\mathit{wf}\ \{(x, y).\ R\ y\ x\}$
shows $\exists\, b.\ R^{**}\ a\ b \land \mathit{no\text{-}step}\ R\ b$
⟨*proof*⟩

lemma *wf-exists-normal-form-full*:
assumes $\mathit{wf}{:}\mathit{wf}\ \{(x, y).\ R\ y\ x\}$
shows $\exists\, b.\ \mathit{full}\ R\ a\ b$
⟨*proof*⟩

1.4 More Well-Foundedness

A little list of theorems that could be useful, but are hidden:

- link between *wf* and infinite chains: $wf\ ?r = (\nexists f.\ \forall i.\ (f\ (Suc\ i),\ f\ i) \in\ ?r)$, $[\![wf\ ?r;\ \bigwedge k.\ (?f\ (Suc\ k),\ ?f\ k) \notin\ ?r \Longrightarrow\ ?thesis]\!] \Longrightarrow\ ?thesis$

lemma *wf-if-measure-in-wf*:
$wf\ R \Longrightarrow (\bigwedge a\ b.\ (a,\ b) \in S \Longrightarrow (\nu\ a,\ \nu\ b)\in R) \Longrightarrow wf\ S$
⟨*proof*⟩

lemma *wfP-if-measure*: **fixes** $f :: 'a \Rightarrow nat$
shows $(\bigwedge x\ y.\ P\ x \Longrightarrow g\ x\ y\ \Longrightarrow f\ y < f\ x) \Longrightarrow wf\ \{(y,x).\ P\ x \wedge g\ x\ y\}$
⟨*proof*⟩

lemma *wf-if-measure-f*:
assumes $wf\ r$
shows $wf\ \{(b,\ a).\ (f\ b,\ f\ a) \in r\}$
⟨*proof*⟩

lemma *wf-wf-if-measure'*:
assumes $wf\ r$ **and** H: $(\bigwedge x\ y.\ P\ x \Longrightarrow g\ x\ y \Longrightarrow (f\ y,\ f\ x) \in r)$
shows $wf\ \{(y,x).\ P\ x \wedge g\ x\ y\}$
⟨*proof*⟩

lemma *wf-lex-less*: $wf\ (lex\ \{(a,\ b).\ (a::nat) < b\})$
⟨*proof*⟩

lemma *wfP-if-measure2*: **fixes** $f :: 'a \Rightarrow nat$
shows $(\bigwedge x\ y.\ P\ x\ y \Longrightarrow g\ x\ y\ \Longrightarrow f\ x < f\ y) \Longrightarrow wf\ \{(x,y).\ P\ x\ y \wedge g\ x\ y\}$
⟨*proof*⟩

lemma *lexord-on-finite-set-is-wf*:
assumes
P-finite: $\bigwedge U.\ P\ U \longrightarrow U \in A$ **and**
finite: $finite\ A$ **and**
wf: $wf\ R$ **and**
trans: $trans\ R$
shows $wf\ \{(T,\ S).\ (P\ S \wedge P\ T) \wedge (T,\ S) \in lexord\ R\}$
⟨*proof*⟩

lemma *wf-fst-wf-pair*:
assumes $wf\ \{(M',\ M).\ R\ M'\ M\}$
shows $wf\ \{((M',\ N'),\ (M,\ N)).\ R\ M'\ M\}$
⟨*proof*⟩

lemma *wf-snd-wf-pair*:
assumes $wf\ \{(M',\ M).\ R\ M'\ M\}$
shows $wf\ \{((M',\ N'),\ (M,\ N)).\ R\ N'\ N\}$
⟨*proof*⟩

lemma *wf-if-measure-f-notation2*:
assumes $wf\ r$
shows $wf\ \{(b,\ h\ a)|b\ a.\ (f\ b,\ f\ (h\ a)) \in r\}$

⟨*proof*⟩

lemma *wf-wf-if-measure′-notation2*:
assumes *wf r* **and** *H*: $(\bigwedge x\ y.\ P\ x \Longrightarrow g\ x\ y \Longrightarrow (f\ y,\ f\ (h\ x)) \in r)$
shows *wf* $\{(y,h\ x)|\ y\ x.\ P\ x \land g\ x\ y\}$
⟨*proof*⟩

end
theory *List-More*
imports *Main ../lib/Multiset-More*
begin

Sledgehammer parameters

sledgehammer-params[*debug*]

2 Various Lemmas

Close to $(\bigwedge n.\ \forall m{<}n.\ ?P\ m \Longrightarrow ?P\ n) \Longrightarrow ?P\ ?n$, but with a separation between zero and non-zero, and case names.

thm *nat-less-induct*
lemma *nat-less-induct-case*[*case-names 0 Suc*]:
  assumes
    P 0 **and**
    $\bigwedge n.\ (\forall m < Suc\ n.\ P\ m) \Longrightarrow P\ (Suc\ n)$
  shows *P n*
  ⟨*proof*⟩

This is only proved in simple cases by auto. In assumptions, nothing happens, and $?P\ (\mathit{if}\ ?Q\ \mathit{then}\ ?x\ \mathit{else}\ ?y) = (\neg\ (?Q \land \neg\ ?P\ ?x \lor \neg\ ?Q \land \neg\ ?P\ ?y))$ can blow up goals (because of other if expression).

lemma *if-0-1-ge-0*[*simp*]:
  $0 < (\mathit{if}\ P\ \mathit{then}\ a\ \mathit{else}\ (0{::}nat)) \longleftrightarrow P \land 0 < a$
  ⟨*proof*⟩

Bounded function have not been defined in Isabelle.

definition *bounded* **where**
$bounded\ f \longleftrightarrow (\exists b.\ \forall n.\ f\ n \le b)$

abbreviation *unbounded* :: $('a \Rightarrow 'b{::}ord) \Rightarrow bool$ **where**
$unbounded\ f \equiv \neg\ bounded\ f$

lemma *not-bounded-nat-exists-larger*:
  fixes $f :: nat \Rightarrow nat$
  assumes *unbound*: *unbounded f*
  shows $\exists n.\ f\ n > m \land n > n_0$
⟨*proof*⟩

lemma *bounded-const-product*:
  fixes $k :: nat$ **and** $f :: nat \Rightarrow nat$
  assumes $k > 0$
  shows $bounded\ f \longleftrightarrow bounded\ (\lambda i.\ k * f\ i)$
  ⟨*proof*⟩

This lemma is not used, but here to show that a property that can be expected from *bounded* holds.

lemma *bounded-finite-linorder*:
fixes $f :: {'a} \Rightarrow {'a} ::\{finite,\ linorder\}$
shows *bounded f*
⟨*proof*⟩

3 More List

3.1 *upt*

The simplification rules are not very handy, because $[?i..<Suc\ ?j] = (if\ ?i \leq ?j\ then\ [?i..<?j]\ @\ [?j]\ else\ [])$ leads to a case distinction, that we do not want if the condition is not in the context.

lemma *upt-Suc-le-append*: $\neg i \leq j \Longrightarrow [i..<Suc\ j] = []$
⟨*proof*⟩

lemmas *upt-simps*[*simp*] = *upt-Suc-append upt-Suc-le-append*

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

lemma
assumes $i \leq n - m$
shows $take\ i\ [m..<n] = [m..<m+i]$
⟨*proof*⟩

The counterpart for this lemma when $n - m < i$ is $length\ ?xs \leq ?n \Longrightarrow take\ ?n\ ?xs = ?xs$. It is close to $?i + ?m \leq ?n \Longrightarrow take\ ?m\ [?i..<?n] = [?i..<?i + ?m]$, but seems more general.

lemma *take-upt-bound-minus*[*simp*]:
assumes $i \leq n - m$
shows $take\ i\ [m..<n] = [m\ ..<m+i]$
⟨*proof*⟩

lemma *append-cons-eq-upt*:
assumes $A\ @\ B = [m..<n]$
shows $A = [m\ ..<m+length\ A]$ **and** $B = [m + length\ A..<n]$
⟨*proof*⟩

lemma *length-list-Suc-0*:
$length\ W = Suc\ 0 \longleftrightarrow (\exists L.\ W = [L])$
⟨*proof*⟩

lemma *length-list-2*: $length\ S = 2 \longleftrightarrow (\exists a\ b.\ S = [a,\ b])$
⟨*proof*⟩

The converse of $?A\ @\ ?B = [?m..<?n] \Longrightarrow ?A = [?m..<?m + length\ ?A]$
$?A\ @\ ?B = [?m..<?n] \Longrightarrow ?B = [?m + length\ ?A..<?n]$ does not hold, for example if B is empty and A is $[0::'a]$:

lemma $A\ @\ B = [m..<\ n] \longleftrightarrow A = [m\ ..<m+length\ A] \wedge B = [m + length\ A..<n]$

⟨*proof*⟩

A more restrictive version holds:

lemma $B \neq [] \Longrightarrow A @ B = [m..< n] \longleftrightarrow A = [m ..<m+length\ A] \wedge B = [m + length\ A..<n]$
(**is** $?P \Longrightarrow ?A = ?B$)
⟨*proof*⟩

lemma *append-cons-eq-upt-length-i*:
assumes $A @ i \# B = [m..<n]$
shows $A = [m ..<i]$
⟨*proof*⟩

lemma *append-cons-eq-upt-length*:
assumes $A @ i \# B = [m..<n]$
shows $length\ A = i - m$
⟨*proof*⟩

lemma *append-cons-eq-upt-length-i-end*:
assumes $A @ i \# B = [m..<n]$
shows $B = [Suc\ i ..<n]$
⟨*proof*⟩

lemma *Max-n-upt*: $Max\ (insert\ 0\ \{Suc\ 0..<n\}) = n - Suc\ 0$
⟨*proof*⟩

lemma *upt-decomp-lt*:
assumes H: $xs @ i \# ys @ j \# zs = [m ..< n]$
shows $i < j$
⟨*proof*⟩

3.2 Lexicographic Ordering

lemma *lexn-Suc*:
$(x \# xs, y \# ys) \in lexn\ r\ (Suc\ n) \longleftrightarrow$
$(length\ xs = n \wedge length\ ys = n) \wedge ((x, y) \in r \vee (x = y \wedge (xs, ys) \in lexn\ r\ n))$
⟨*proof*⟩

lemma *lexn-n*:
$n > 0 \Longrightarrow (x \# xs, y \# ys) \in lexn\ r\ n \longleftrightarrow$
$(length\ xs = n-1 \wedge length\ ys = n-1) \wedge ((x, y) \in r \vee (x = y \wedge (xs, ys) \in lexn\ r\ (n - 1)))$
⟨*proof*⟩

There is some subtle point in the proof here. *1* is converted to *Suc 0*, but *2* is not: meaning that *1* is automatically simplified by default using the default simplification rule $lexn\ ?r\ 0 = \{\}$
$lexn\ ?r\ (Suc\ ?n) = map\text{-}prod\ (\lambda(x, xs).\ x \# xs)\ (\lambda(x, xs).\ x \# xs)$ ‘ $(?r <*lex*> lexn\ ?r\ ?n) \cap \{(xs, ys).\ length\ xs = Suc\ ?n \wedge length\ ys = Suc\ ?n\}$. However, the latter needs additional simplification rule.

lemma *lexn2-conv*:
$([a, b], [c, d]) \in lexn\ r\ 2 \longleftrightarrow (a, c) \in r \vee (a = c \wedge (b, d) \in r)$
⟨*proof*⟩

lemma *lexn3-conv*:
$([a, b, c], [a', b', c']) \in lexn\ r\ 3 \longleftrightarrow$
$(a, a') \in r \vee (a = a' \wedge (b, b') \in r) \vee (a = a' \wedge b = b' \wedge (c, c') \in r)$
⟨*proof*⟩

3.3 Remove

3.3.1 More lemmas about remove

lemma *remove1-nil*:
  $remove1\ (-\ L)\ W = [] \longleftrightarrow (W = [] \lor W = [-L])$
  ⟨*proof*⟩

lemma *remove1-mset-single-add*:
  $a \neq b \Longrightarrow remove1\text{-}mset\ a\ (\{\#b\#\} + C) = \{\#b\#\} + remove1\text{-}mset\ a\ C$
  $remove1\text{-}mset\ a\ (\{\#a\#\} + C) = C$
  ⟨*proof*⟩

3.3.2 Remove under condition

This function removes the first element when the condition f holds. It generalises *remove1*.

fun *remove1-cond* **where**
$remove1\text{-}cond\ f\ [] = []\ |$
$remove1\text{-}cond\ f\ (C' \# L) = (if\ f\ \ C'\ then\ L\ else\ C' \#\ remove1\text{-}cond\ f\ L)$

lemma $remove1\ x\ xs = remove1\text{-}cond\ ((op =)\ x)\ xs$
  ⟨*proof*⟩

lemma *mset-map-mset-remove1-cond*:
  $mset\ (map\ mset\ (remove1\text{-}cond\ (\lambda L.\ mset\ L = mset\ a)\ C)) =$
    $remove1\text{-}mset\ (mset\ a)\ (mset\ (map\ mset\ C))$
  ⟨*proof*⟩

We can also generalise *removeAll*, which is close to *filter*:

fun *removeAll-cond* **where**
$removeAll\text{-}cond\ f\ [] = []\ |$
$removeAll\text{-}cond\ f\ (C' \# L) =$
  $(if\ f\ C'\ then\ removeAll\text{-}cond\ f\ L\ else\ C' \#\ removeAll\text{-}cond\ f\ L)$

lemma $removeAll\ x\ xs = removeAll\text{-}cond\ ((op =)\ x)\ xs$
  ⟨*proof*⟩

lemma $removeAll\text{-}cond\ P\ xs = filter\ (\lambda x.\ \neg P\ x)\ xs$
  ⟨*proof*⟩

lemma *mset-map-mset-removeAll-cond*:
  $mset\ (map\ mset\ (removeAll\text{-}cond\ (\lambda b.\ mset\ b = mset\ a)\ C))$
    $= removeAll\text{-}mset\ (mset\ a)\ (mset\ (map\ mset\ C))$
  ⟨*proof*⟩

Take from `../lib/Multiset_More.thy`, but named:

abbreviation *union-mset-list* **where**
$union\text{-}mset\text{-}list\ xs\ ys \equiv case\text{-}prod\ append\ (fold\ (\lambda x\ (ys,\ zs).\ (remove1\ x\ ys,\ x \#\ zs))\ xs\ (ys,\ []))$

lemma *union-mset-list*:
  $mset\ xs\ \#\cup\ mset\ ys = mset\ (union\text{-}mset\text{-}list\ xs\ ys)$
⟨*proof*⟩

end
theory *Prop-Logic*

imports *Main*

begin

4 Logics

In this section we define the syntax of the formula and an abstraction over it to have simpler proofs. After that we define some properties like subformula and rewriting.

4.1 Definition and abstraction

The propositional logic is defined inductively. The type parameter is the type of the variables.

datatype $'v\ propo =$
$FT \mid FF \mid FVar\ 'v \mid FNot\ 'v\ propo \mid FAnd\ \ 'v\ propo\ \ 'v\ propo \mid FOr\ \ 'v\ propo\ \ 'v\ propo$
$\mid FImp\ \ 'v\ propo\ \ 'v\ propo \mid FEq\ \ 'v\ propo\ \ 'v\ propo$

We do not define any notation for the formula, to distinguish properly between the formulas and Isabelle's logic.

To ease the proofs, we will write the the formula on a homogeneous manner, namely a connecting argument and a list of arguments.

datatype $'v\ connective = CT \mid CF \mid CVar\ 'v \mid CNot \mid CAnd \mid COr \mid CImp \mid CEq$

abbreviation $nullary\text{-}connective \equiv \{CF\} \cup \{CT\} \cup \{CVar\ x \mid x.\ True\}$
definition $binary\text{-}connectives \equiv \{CAnd,\ COr,\ CImp,\ CEq\}$

We define our own induction principal: instead of distinguishing every constructor, we group them by arity.

lemma *propo-induct-arity*[*case-names nullary unary binary*]:
fixes $\varphi\ \psi :: \ 'v\ propo$
assumes *nullary*: $(\bigwedge\varphi\ x.\ \varphi = FF \lor \varphi = FT \lor \varphi = FVar\ x \Longrightarrow P\ \varphi)$
and *unary*: $(\bigwedge\psi.\ P\ \psi \Longrightarrow P\ (FNot\ \psi))$
and *binary*: $(\bigwedge\varphi\ \psi 1\ \psi 2.\ P\ \psi 1 \Longrightarrow P\ \psi 2 \Longrightarrow \varphi = FAnd\ \psi 1\ \psi 2 \lor \varphi = FOr\ \psi 1\ \psi 2 \lor \varphi = FImp$
$\psi 1\ \psi 2$
$\lor\ \varphi = FEq\ \psi 1\ \psi 2 \Longrightarrow P\ \varphi)$
shows $P\ \psi$
$\langle proof \rangle$

The function *conn* is the interpretation of our representation (connective and list of arguments). We define any thing that has no sense to be false

fun $conn :: \ 'v\ connective \Rightarrow \ 'v\ propo\ list \Rightarrow \ 'v\ propo$ **where**
$conn\ CT\ [] = FT \mid$
$conn\ CF\ [] = FF \mid$
$conn\ (CVar\ v)\ [] = FVar\ v \mid$
$conn\ CNot\ [\varphi] = FNot\ \varphi \mid$
$conn\ CAnd\ (\varphi\ \#\ [\psi]) = FAnd\ \varphi\ \psi \mid$
$conn\ COr\ (\varphi\ \#\ [\psi]) = FOr\ \varphi\ \psi \mid$
$conn\ CImp\ (\varphi\ \#\ [\psi]) = FImp\ \varphi\ \psi \mid$
$conn\ CEq\ (\varphi\ \#\ [\psi]) = FEq\ \varphi\ \psi \mid$
$conn\ \text{-}\ \text{-} = FF$

We will often use case distinction, based on the arity of the *'v connective*, thus we define our own splitting principle.

lemma *connective-cases-arity*[*case-names nullary binary unary*]:
  assumes *nullary*: $\bigwedge x.\ c = CT \vee c = CF \vee c = CVar\ x \Longrightarrow P$
  and *binary*: $c \in$ *binary-connectives* $\Longrightarrow P$
  and *unary*: $c = CNot \ \Longrightarrow P$
  shows P
  ⟨*proof*⟩

lemma *connective-cases-arity-2*[*case-names nullary unary binary*]:
  assumes *nullary*: $c \in$ *nullary-connective* $\Longrightarrow P$
  and *unary*: $c = CNot \ \Longrightarrow P$
  and *binary*: $c \in$ *binary-connectives* $\Longrightarrow P$
  shows P
  ⟨*proof*⟩

Our previous definition is not necessary correct (connective and list of arguments) , so we define an inductive predicate.

inductive *wf-conn* :: *'v connective* ⇒ *'v propo list* ⇒ *bool* **for** *c* :: *'v connective* **where**
wf-conn-nullary[*simp*]: $(c = CT \vee c = CF \vee c\ = CVar\ v) \Longrightarrow$ *wf-conn* c [] |
wf-conn-unary[*simp*]: $c = CNot \Longrightarrow$ *wf-conn* c [ψ] |
wf-conn-binary[*simp*]: $c \in$ *binary-connectives* $\Longrightarrow$ *wf-conn* c (ψ # ψ' # [])
thm *wf-conn.induct*
lemma *wf-conn-induct*[*consumes 1*, *case-names CT CF CVar CNot COr CAnd CImp CEq*]:
  assumes *wf-conn c x* **and**
    $(\bigwedge v.\ c = CT \Longrightarrow P\ [])$ **and**
    $(\bigwedge v.\ c = CF \Longrightarrow P\ [])$ **and**
    $(\bigwedge v.\ c = CVar\ v \Longrightarrow P\ [])$ **and**
    $(\bigwedge \psi.\ c = CNot \Longrightarrow P\ [\psi])$ **and**
    $(\bigwedge \psi\ \psi'.\ c = COr \Longrightarrow P\ [\psi, \psi'])$ **and**
    $(\bigwedge \psi\ \psi'.\ c = CAnd \Longrightarrow P\ [\psi, \psi'])$ **and**
    $(\bigwedge \psi\ \psi'.\ c = CImp \Longrightarrow P\ [\psi, \psi'])$ **and**
    $(\bigwedge \psi\ \psi'.\ c = CEq \Longrightarrow P\ [\psi, \psi'])$
  shows $P\ x$
  ⟨*proof*⟩

4.2 properties of the abstraction

First we can define simplification rules.

lemma *wf-conn-conn*[*simp*]:
  wf-conn $CT\ l \Longrightarrow$ *conn* $CT\ l = FT$
  wf-conn $CF\ l \Longrightarrow$ *conn* $CF\ l = FF$
  wf-conn $(CVar\ x)\ l \Longrightarrow$ *conn* $(CVar\ x)\ l = FVar\ x$
  ⟨*proof*⟩

lemma *wf-conn-list-decomp*[*simp*]:
  wf-conn $CT\ l \longleftrightarrow l = []$
  wf-conn $CF\ l \longleftrightarrow l = []$
  wf-conn $(CVar\ x)\ l \longleftrightarrow l = []$
  wf-conn $CNot\ (\xi\ @\ \varphi\ \#\ \xi') \longleftrightarrow \xi = [] \wedge \xi' = []$
  ⟨*proof*⟩

lemma *wf-conn-list*:
  $wf\text{-}conn\ c\ l \Longrightarrow conn\ c\ l = FT \longleftrightarrow (c = CT \wedge l = [])$
  $wf\text{-}conn\ c\ l \Longrightarrow conn\ c\ l = FF \longleftrightarrow (c = CF \wedge l = [])$
  $wf\text{-}conn\ c\ l \Longrightarrow conn\ c\ l = FVar\ x \longleftrightarrow (c = CVar\ x \wedge l = [])$
  $wf\text{-}conn\ c\ l \Longrightarrow conn\ c\ l = FAnd\ a\ b \longleftrightarrow (c = CAnd \wedge l = a\ \#\ b\ \#\ [])$
  $wf\text{-}conn\ c\ l \Longrightarrow conn\ c\ l = FOr\ a\ b \longleftrightarrow (c = COr \wedge l = a\ \#\ b\ \#\ [])$
  $wf\text{-}conn\ c\ l \Longrightarrow conn\ c\ l = FEq\ a\ b \longleftrightarrow (c = CEq \wedge l = a\ \#\ b\ \#\ [])$
  $wf\text{-}conn\ c\ l \Longrightarrow conn\ c\ l = FImp\ a\ b \longleftrightarrow (c = CImp \wedge l = a\ \#\ b\ \#\ [])$
  $wf\text{-}conn\ c\ l \Longrightarrow conn\ c\ l = FNot\ a \longleftrightarrow (c = CNot \wedge l = a\ \#\ [])$
  ⟨*proof*⟩

In the binary connective cases, we will often decompose the list of arguments (of length 2) into two elements.

lemma *list-length2-decomp*: $length\ l = 2 \Longrightarrow (\exists\ a\ b.\ l = a\ \#\ b\ \#\ [])$
  ⟨*proof*⟩

wf-conn for binary operators means that there are two arguments.

lemma *wf-conn-bin-list-length*:
  fixes $l :: 'v\ propo\ list$
  assumes *conn*: $c \in binary\text{-}connectives$
  shows $length\ l = 2 \longleftrightarrow wf\text{-}conn\ c\ l$
⟨*proof*⟩

lemma *wf-conn-not-list-length*[*iff*]:
  fixes $l :: 'v\ propo\ list$
  shows $wf\text{-}conn\ CNot\ l \longleftrightarrow length\ l = 1$
  ⟨*proof*⟩

Decomposing the Not into an element is moreover very useful.

lemma *wf-conn-Not-decomp*:
  fixes $l :: 'v\ propo\ list$ **and** $a :: 'v$
  assumes *corr*: $wf\text{-}conn\ CNot\ l$
  shows $\exists\ a.\ l = [a]$
  ⟨*proof*⟩

The *wf-conn* remains correct if the length of list does not change. This lemma is very useful when we do one rewriting step

lemma *wf-conn-no-arity-change*:
  $length\ l = length\ l' \Longrightarrow wf\text{-}conn\ c\ l \longleftrightarrow wf\text{-}conn\ c\ l'$
⟨*proof*⟩

lemma *wf-conn-no-arity-change-helper*:
  $length\ (\xi\ @\ \varphi\ \#\ \xi') = length\ (\xi\ @\ \varphi'\ \#\ \xi')$
  ⟨*proof*⟩

The injectivity of *conn* is useful to prove equality of the connectives and the lists.

lemma *conn-inj-not*:
  assumes *correct*: $wf\text{-}conn\ c\ l$
  and *conn*: $conn\ c\ l = FNot\ \psi$
  shows $c = CNot$ **and** $l = [\psi]$
  ⟨*proof*⟩

lemma *conn-inj*:
  fixes $c\ ca :: 'v\ connective$ **and** $l\ \psi s :: 'v\ propo\ list$
  assumes *corr*: $wf\text{-}conn\ ca\ l$
  and *corr'*: $wf\text{-}conn\ c\ \psi s$
  and *eq*: $conn\ ca\ l = conn\ c\ \psi s$
  shows $ca = c \wedge \psi s = l$
  ⟨*proof*⟩

4.3 Subformulas and properties

A characterization using sub-formulas is interesting for rewriting: we will define our relation on the sub-term level, and then lift the rewriting on the term-level. So the rewriting takes place on a subformula.

inductive *subformula* $:: 'v\ propo \Rightarrow 'v\ propo \Rightarrow bool$ (**infix** $\preceq$ *45*) **for** φ **where**
subformula-refl[*simp*]: $\varphi \preceq \varphi$ |
subformula-into-subformula: $\psi \in set\ l \Longrightarrow wf\text{-}conn\ c\ l \Longrightarrow \varphi \preceq \psi \Longrightarrow \varphi \preceq conn\ c\ l$

On the *subformula-into-subformula*, we can see why we use our *conn* representation: one case is enough to express the subformulas property instead of listing all the cases.

This is an example of a property related to subformulas.

lemma *subformula-in-subformula-not*:
shows *b*: $FNot\ \varphi \preceq \psi \Longrightarrow \varphi \preceq \psi$
  ⟨*proof*⟩

lemma *subformula-in-binary-conn*:
  assumes *conn*: $c \in binary\text{-}connectives$
  shows $f \preceq conn\ c\ [f, g]$
  and $g \preceq conn\ c\ [f, g]$
⟨*proof*⟩

lemma *subformula-trans*:
 $\psi \preceq \psi' \Longrightarrow \varphi \preceq \psi \Longrightarrow \varphi \preceq \psi'$
  ⟨*proof*⟩

lemma *subformula-leaf*:
  fixes $\varphi\ \psi :: 'v\ propo$
  assumes *incl*: $\varphi \preceq \psi$
  and *simple*: $\psi = FT \vee \psi = FF \vee \psi = FVar\ x$
  shows $\varphi = \psi$
  ⟨*proof*⟩

lemma *subfurmula-not-incl-eq*:
  assumes $\varphi \preceq conn\ c\ l$
  and $wf\text{-}conn\ c\ l$
  and $\forall \psi.\ \psi \in set\ l \longrightarrow \neg\ \varphi \preceq \psi$
  shows $\varphi = conn\ c\ l$
  ⟨*proof*⟩

lemma *wf-subformula-conn-cases*:
  $wf\text{-}conn\ c\ l \Longrightarrow \varphi \preceq conn\ c\ l \longleftrightarrow (\varphi = conn\ c\ l \vee (\exists \psi.\ \psi \in set\ l \wedge \varphi \preceq \psi))$
  ⟨*proof*⟩

lemma *subformula-decomp-explicit*[*simp*]:
  $\varphi \preceq FAnd\ \psi\ \psi' \longleftrightarrow (\varphi = FAnd\ \psi\ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi')$ (**is** *?P FAnd*)

$\varphi \preceq FOr\ \psi\ \psi' \longleftrightarrow (\varphi = FOr\ \psi\ \psi' \lor \varphi \preceq \psi \lor \varphi \preceq \psi')$
$\varphi \preceq FEq\ \psi\ \psi' \longleftrightarrow (\varphi = FEq\ \psi\ \psi' \lor \varphi \preceq \psi \lor \varphi \preceq \psi')$
$\varphi \preceq FImp\ \psi\ \psi' \longleftrightarrow (\varphi = FImp\ \psi\ \psi' \lor \varphi \preceq \psi \lor \varphi \preceq \psi')$
$\langle proof \rangle$

lemma *wf-conn-helper-facts*[*iff*]:
wf-conn CNot [φ]
wf-conn CT []
wf-conn CF []
wf-conn (*CVar x*) []
wf-conn CAnd [φ, ψ]
wf-conn COr [φ, ψ]
wf-conn CImp [φ, ψ]
wf-conn CEq [φ, ψ]
$\langle proof \rangle$

lemma *exists-c-conn*: $\exists\ c\ l.\ \varphi = conn\ c\ l \land wf\text{-}conn\ c\ l$
$\langle proof \rangle$

lemma *subformula-conn-decomp*[*simp*]:
assumes *wf*: *wf-conn c l*
shows $\varphi \preceq conn\ c\ l \longleftrightarrow (\varphi = conn\ c\ l \lor (\exists\ \psi \in set\ l.\ \varphi \preceq \psi))$ (**is** *?A* $\longleftrightarrow$ *?B*)
$\langle proof \rangle$

lemma *subformula-leaf-explicit*[*simp*]:
$\varphi \preceq FT \longleftrightarrow \varphi = FT$
$\varphi \preceq FF \longleftrightarrow \varphi = FF$
$\varphi \preceq FVar\ x \longleftrightarrow \varphi = FVar\ x$
$\langle proof \rangle$

The variables inside the formula gives precisely the variables that are needed for the formula.

primrec *vars-of-prop*:: *'v propo* $\Rightarrow$ *'v set* **where**
vars-of-prop FT = {} |
vars-of-prop FF = {} |
vars-of-prop (*FVar x*) = {*x*} |
vars-of-prop (*FNot* φ) = *vars-of-prop* φ |
vars-of-prop (*FAnd* φ ψ) = *vars-of-prop* φ $\cup$ *vars-of-prop* ψ |
vars-of-prop (*FOr* φ ψ) = *vars-of-prop* φ $\cup$ *vars-of-prop* ψ |
vars-of-prop (*FImp* φ ψ) = *vars-of-prop* φ $\cup$ *vars-of-prop* ψ |
vars-of-prop (*FEq* φ ψ) = *vars-of-prop* φ $\cup$ *vars-of-prop* ψ

lemma *vars-of-prop-incl-conn*:
fixes ξ ξ' :: *'v propo list* **and** ψ :: *'v propo* **and** *c* :: *'v connective*
assumes *corr*: *wf-conn c l* **and** *incl*: $\psi \in set\ l$
shows *vars-of-prop* ψ $\subseteq$ *vars-of-prop* (*conn c l*)
$\langle proof \rangle$

The set of variables is compatible with the subformula order.

lemma *subformula-vars-of-prop*:
$\varphi \preceq \psi \Longrightarrow vars\text{-}of\text{-}prop\ \varphi \subseteq vars\text{-}of\text{-}prop\ \psi$
$\langle proof \rangle$

4.4 Positions

Instead of 1 or 2 we use L or R

datatype *sign* = *L* | *R*

We use *nil* instead of ε.

fun $pos :: 'v\ propo \Rightarrow sign\ list\ set$ **where**
$pos\ FF = \{[]\}$ |
$pos\ FT = \{[]\}$ |
$pos\ (FVar\ x) = \{[]\}$ |
$pos\ (FAnd\ \varphi\ \psi) = \{[]\} \cup \{\ L\ \#\ p \mid p.\ p\in pos\ \varphi\} \cup \{\ R\ \#\ p \mid p.\ p\in pos\ \psi\}$ |
$pos\ (FOr\ \varphi\ \psi) = \{[]\} \cup \{\ L\ \#\ p \mid p.\ p\in pos\ \varphi\} \cup \{\ R\ \#\ p \mid p.\ p\in pos\ \psi\}$ |
$pos\ (FEq\ \varphi\ \psi) = \{[]\} \cup \{\ L\ \#\ p \mid p.\ p\in pos\ \varphi\} \cup \{\ R\ \#\ p \mid p.\ p\in pos\ \psi\}$ |
$pos\ (FImp\ \varphi\ \psi) = \{[]\} \cup \{\ L\ \#\ p \mid p.\ p\in pos\ \varphi\} \cup \{\ R\ \#\ p \mid p.\ p\in pos\ \psi\}$ |
$pos\ (FNot\ \varphi) = \{[]\} \cup \{\ L\ \#\ p \mid p.\ p\in pos\ \varphi\}$

lemma *finite-pos*: $finite\ (pos\ \varphi)$
  ⟨*proof*⟩

lemma *finite-inj-comp-set*:
  fixes $s :: 'v\ set$
  assumes *finite*: *finite s*
  and *inj*: *inj f*
  shows $card\ (\{f\ p \mid p.\ p \in s\}) = card\ s$
  ⟨*proof*⟩

lemma *cons-inject*:
  $inj\ (op\ \#\ s)$
  ⟨*proof*⟩

lemma *finite-insert-nil-cons*:
  $finite\ s \Longrightarrow card\ (insert\ []\ \{L\ \#\ p \mid p.\ p \in s\}) = 1 + card\ \{L\ \#\ p \mid p.\ p \in s\}$
  ⟨*proof*⟩

lemma *cord-not*[*simp*]:
  $card\ (pos\ (FNot\ \varphi)) = 1 + card\ (pos\ \varphi)$
⟨*proof*⟩

lemma *card-seperate*:
  assumes *finite s1* **and** *finite s2*
  shows $card\ (\{L\ \#\ p \mid p.\ p \in s1\} \cup \{R\ \#\ p \mid p.\ p \in s2\}) = card\ (\{L\ \#\ p \mid p.\ p \in s1\})$
    $+\ card(\{R\ \#\ p \mid p.\ p \in s2\})$ (**is** $card\ (?L\cup?R) = card\ ?L + card\ ?R$)
⟨*proof*⟩

definition *prop-size* **where** $prop\text{-}size\ \varphi = card\ (pos\ \varphi)$

lemma *prop-size-vars-of-prop*:
  fixes $\varphi :: 'v\ propo$
  shows $card\ (vars\text{-}of\text{-}prop\ \varphi) \le prop\text{-}size\ \varphi$

  ⟨*proof*⟩

value $pos\ (FImp\ (FAnd\ (FVar\ P)\ (FVar\ Q))\ (FOr\ (FVar\ P)\ (FVar\ Q)))$

inductive $path\text{-}to :: sign\ list \Rightarrow 'v\ propo \Rightarrow 'v\ propo \Rightarrow bool$ **where**
path-to-refl[*intro*]: $path\text{-}to\ []\ \varphi\ \varphi$ |
path-to-l: $c\in binary\text{-}connectives \lor c = CNot \Longrightarrow wf\text{-}conn\ c\ (\varphi\#l) \Longrightarrow path\text{-}to\ p\ \varphi\ \varphi' \Longrightarrow$

$path\text{-}to\ (L\#p)\ (conn\ c\ (\varphi\#l))\ \varphi'$ |
$path\text{-}to\text{-}r$: $c \in binary\text{-}connectives \Longrightarrow wf\text{-}conn\ c\ (\psi\#\varphi\#[]) \Longrightarrow path\text{-}to\ p\ \varphi\ \varphi' \Longrightarrow$
$path\text{-}to\ (R\#p)\ (conn\ c\ (\psi\#\varphi\#[]))\ \varphi'$

There is a deep link between subformulas and pathes: a (correct) path leads to a subformula and a subformula is associated to a given path.

lemma *path-to-subformula*:
$path\text{-}to\ p\ \varphi\ \varphi' \Longrightarrow \varphi' \preceq \varphi$
⟨*proof*⟩

lemma *subformula-path-exists*:
fixes $\varphi\ \varphi'$:: $'v\ propo$
shows $\varphi' \preceq \varphi \Longrightarrow \exists\, p.\ path\text{-}to\ p\ \varphi\ \varphi'$
⟨*proof*⟩

fun *replace-at* :: $sign\ list \Rightarrow\ 'v\ propo \Rightarrow\ 'v\ propo \Rightarrow\ 'v\ propo$ **where**
$replace\text{-}at\ []\ \text{-}\ \psi = \psi$ |
$replace\text{-}at\ (L\ \#\ l)\ (FAnd\ \varphi\ \varphi')\ \psi = FAnd\ (replace\text{-}at\ l\ \varphi\ \psi)\ \varphi'$|
$replace\text{-}at\ (R\ \#\ l)\ (FAnd\ \varphi\ \varphi')\ \psi = FAnd\ \varphi\ (replace\text{-}at\ l\ \varphi'\ \psi)$ |
$replace\text{-}at\ (L\ \#\ l)\ (FOr\ \varphi\ \varphi')\ \psi = FOr\ (replace\text{-}at\ l\ \varphi\ \psi)\ \varphi'$ |
$replace\text{-}at\ (R\ \#\ l)\ (FOr\ \varphi\ \varphi')\ \psi = FOr\ \varphi\ (replace\text{-}at\ l\ \varphi'\ \psi)$ |
$replace\text{-}at\ (L\ \#\ l)\ (FEq\ \varphi\ \varphi')\ \psi = FEq\ (replace\text{-}at\ l\ \varphi\ \psi)\ \varphi'$|
$replace\text{-}at\ (R\ \#\ l)\ (FEq\ \varphi\ \varphi')\ \psi = FEq\ \varphi\ (replace\text{-}at\ l\ \varphi'\ \psi)$ |
$replace\text{-}at\ (L\ \#\ l)\ (FImp\ \varphi\ \varphi')\ \psi = FImp\ (replace\text{-}at\ l\ \varphi\ \psi)\ \varphi'$|
$replace\text{-}at\ (R\ \#\ l)\ (FImp\ \varphi\ \varphi')\ \psi = FImp\ \varphi\ (replace\text{-}at\ l\ \varphi'\ \psi)$ |
$replace\text{-}at\ (L\ \#\ l)\ (FNot\ \varphi)\ \psi = FNot\ (replace\text{-}at\ l\ \varphi\ \psi)$

5 Semantics over the syntax

Given the syntax defined above, we define a semantics, by defining an evaluation function *eval*. This function is the bridge between the logic as we define it here and the built-in logic of Isabelle.

fun *eval* :: $('v \Rightarrow bool) \Rightarrow\ 'v\ propo \Rightarrow bool$ (**infix** $\models$ *50*) **where**
$\mathcal{A} \models FT = True$ |
$\mathcal{A} \models FF = False$ |
$\mathcal{A} \models FVar\ v = (\mathcal{A}\ v)$ |
$\mathcal{A} \models FNot\ \varphi = (\neg(\mathcal{A} \models \varphi))$ |
$\mathcal{A} \models FAnd\ \varphi_1\ \varphi_2 = (\mathcal{A} \models \varphi_1 \wedge \mathcal{A} \models \varphi_2)$ |
$\mathcal{A} \models FOr\ \varphi_1\ \varphi_2 = (\mathcal{A} \models \varphi_1 \vee \mathcal{A} \models \varphi_2)$ |
$\mathcal{A} \models FImp\ \varphi_1\ \varphi_2 = (\mathcal{A} \models \varphi_1 \longrightarrow \mathcal{A} \models \varphi_2)$ |
$\mathcal{A} \models FEq\ \varphi_1\ \varphi_2 = (\mathcal{A} \models \varphi_1 \longleftrightarrow \mathcal{A} \models \varphi_2)$

definition *evalf* (**infix** $\models f$ *50*) **where**
$evalf\ \varphi\ \psi = (\forall\, A.\ A \models \varphi \longrightarrow A \models \psi)$

The deduction rule is in the book. And the proof looks like to the one of the book.

theorem *deduction-theorem*:
$(\varphi \models f\ \psi) \longleftrightarrow (\forall\, A.\ (A \models FImp\ \varphi\ \psi))$
⟨*proof*⟩

A shorter proof:

lemma $\varphi \models f\ \psi \longleftrightarrow (\forall\, A.\ A \models FImp\ \varphi\ \psi)$
⟨*proof*⟩

definition *same-over-set*:: $('v \Rightarrow bool) \Rightarrow('v \Rightarrow bool) \Rightarrow 'v\ set \Rightarrow bool$ **where**
same-over-set $A\ B\ S = (\forall\, c{\in}S.\ A\ c = B\ c)$

If two mapping A and B have the same value over the variables, then the same formula are satisfiable.

lemma *same-over-set-eval*:
  assumes *same-over-set* $A\ B$ (*vars-of-prop* φ)
  shows $A \models \varphi \longleftrightarrow B \models \varphi$
  ⟨*proof*⟩

end
theory *Prop-Abstract-Transformation*
imports *Main Prop-Logic Wellfounded-More*

begin

This file is devoted to abstract properties of the transformations, like consistency preservation and lifting from terms to proposition.

6 Rewrite systems and properties

6.1 Lifting of rewrite rules

We can lift a rewrite relation r over a full1 formula: the relation r works on terms, while *propo-rew-step* works on formulas.

inductive *propo-rew-step* :: $('v\ propo \Rightarrow 'v\ propo \Rightarrow bool) \Rightarrow 'v\ propo \Rightarrow 'v\ propo \Rightarrow bool$
  for r :: $'v\ propo \Rightarrow 'v\ propo \Rightarrow bool$ **where**
global-rel: $r\ \varphi\ \psi \Longrightarrow$ *propo-rew-step* $r\ \varphi\ \psi$ |
propo-rew-one-step-lift: *propo-rew-step* $r\ \varphi\ \varphi' \Longrightarrow$ *wf-conn* $c\ (\psi s\ @\ \varphi\ \#\ \psi s')$
  $\Longrightarrow$ *propo-rew-step* r (*conn* $c\ (\psi s\ @\ \varphi\ \#\ \psi s')$) (*conn* $c\ (\psi s\ @\ \varphi'\#\ \psi s')$)

Here is a more precise link between the lifting and the subformulas: if a rewriting takes place between φ and φ', then there are two subformulas ψ in φ and ψ' in φ', ψ' is the result of the rewriting of r on ψ.

This lemma is only a health condition:

lemma *propo-rew-step-subformula-imp*:
shows *propo-rew-step* $r\ \varphi\ \varphi' \Longrightarrow \exists\ \psi\ \psi'.\ \psi \preceq \varphi \wedge \psi' \preceq \varphi' \wedge r\ \psi\ \psi'$
  ⟨*proof*⟩

The converse is moreover true: if there is a ψ and ψ', then every formula φ containing ψ, can be rewritten into a formula φ', such that it contains φ'.

lemma *propo-rew-step-subformula-rec*:
  fixes $\psi\ \psi'\ \varphi$ :: $'v\ propo$
  shows $\psi \preceq \varphi \Longrightarrow r\ \psi\ \psi' \Longrightarrow (\exists\varphi'.\ \psi' \preceq \varphi' \wedge$ *propo-rew-step* $r\ \varphi\ \varphi')$
⟨*proof*⟩

lemma *propo-rew-step-subformula*:
  $(\exists\psi\ \psi'.\ \psi \preceq \varphi \wedge r\ \psi\ \psi') \longleftrightarrow (\exists\varphi'.$ *propo-rew-step* $r\ \varphi\ \varphi')$
  ⟨*proof*⟩

lemma *consistency-decompose-into-list*:
  assumes *wf*: *wf-conn* $c\ l$ **and** *wf'*: *wf-conn* $c\ l'$

and *same*: $\forall n.\ (A \models l\ !\ n \longleftrightarrow (A \models l'\ !\ n))$
shows $(A \models conn\ c\ l) = (A \models conn\ c\ l')$
$\langle proof \rangle$

Relation between *propo-rew-step* and the rewriting we have seen before: *propo-rew-step* $r\ \varphi\ \varphi'$ means that we rewrite ψ inside φ (ie at a path p) into ψ'.

lemma *propo-rew-step-rewrite*:
fixes $\varphi\ \varphi' :: 'v\ propo$ **and** $r :: 'v\ propo \Rightarrow 'v\ propo \Rightarrow bool$
assumes *propo-rew-step* $r\ \varphi\ \varphi'$
shows $\exists \psi\ \psi'\ p.\ r\ \psi\ \psi' \wedge$ *path-to* $p\ \varphi\ \psi \wedge$ *replace-at* $p\ \varphi\ \psi' = \varphi'$
$\langle proof \rangle$

6.2 Consistency preservation

We define *preserves-un-sat*: it means that a relation preserves consistency.

definition *preserves-un-sat* **where**
preserves-un-sat $r \longleftrightarrow (\forall \varphi\ \psi.\ r\ \varphi\ \psi \longrightarrow (\forall A.\ A \models \varphi \longleftrightarrow A \models \psi))$

lemma *propo-rew-step-preservers-val-explicit*:
propo-rew-step $r\ \varphi\ \psi \Longrightarrow$ *preserves-un-sat* $r \Longrightarrow$ *propo-rew-step* $r\ \varphi\ \psi \Longrightarrow (\forall A.\ A \models \varphi \longleftrightarrow A \models \psi)$
$\langle proof \rangle$

lemma *propo-rew-step-preservers-val'*:
assumes *preserves-un-sat* r
shows *preserves-un-sat* (*propo-rew-step* r)
$\langle proof \rangle$

lemma *preserves-un-sat-OO*[*intro*]:
preserves-un-sat $f \Longrightarrow$ *preserves-un-sat* $g \Longrightarrow$ *preserves-un-sat* (f *OO* g)
$\langle proof \rangle$

lemma *star-consistency-preservation-explicit*:
assumes (*propo-rew-step* r)$\hat{}{**}$ $\varphi\ \psi$ **and** *preserves-un-sat* r
shows $\forall A.\ A \models \varphi \longleftrightarrow A \models \psi$
$\langle proof \rangle$

lemma *star-consistency-preservation*:
preserves-un-sat $r \Longrightarrow$ *preserves-un-sat* (*propo-rew-step* r)$\hat{}{**}$
$\langle proof \rangle$

6.3 Full Lifting

In the previous a relation was lifted to a formula, now we define the relation such it is applied as long as possible. The definition is thus simply: it can be derived and nothing more can be derived.

lemma *full-ropo-rew-step-preservers-val*[*simp*]:
preserves-un-sat $r \Longrightarrow$ *preserves-un-sat* (*full* (*propo-rew-step* r))
$\langle proof \rangle$

lemma *full-propo-rew-step-subformula*:

full (*propo-rew-step r*) φ' φ $\Longrightarrow \neg(\exists\ \psi\ \psi'.\ \psi \preceq \varphi \wedge r\ \psi\ \psi')$
⟨*proof*⟩

7 Transformation testing

7.1 Definition and first properties

To prove correctness of our transformation, we create a *all-subformula-st* predicate. It tests recursively all subformulas. At each step, the actual formula is tested. The aim of this *test-symb* function is to test locally some properties of the formulas (i.e. at the level of the connective or at first level). This allows a clause description between the rewrite relation and the *test-symb*

definition *all-subformula-st* :: (*'a propo* ⇒ *bool*) ⇒ *'a propo* ⇒ *bool* **where**
all-subformula-st test-symb $\varphi \equiv \forall\psi.\ \psi \preceq \varphi \longrightarrow$ *test-symb* ψ

lemma *test-symb-imp-all-subformula-st*[*simp*]:
test-symb FT $\Longrightarrow$ *all-subformula-st test-symb FT*
test-symb FF $\Longrightarrow$ *all-subformula-st test-symb FF*
test-symb (*FVar x*) $\Longrightarrow$ *all-subformula-st test-symb* (*FVar x*)
⟨*proof*⟩

lemma *all-subformula-st-test-symb-true-phi*:
all-subformula-st test-symb $\varphi \Longrightarrow$ *test-symb* φ
⟨*proof*⟩

lemma *all-subformula-st-decomp-imp*:
wf-conn c l $\Longrightarrow$ (*test-symb* (*conn c l*) $\wedge$ ($\forall\varphi\in$ *set l. all-subformula-st test-symb* φ))
$\Longrightarrow$ *all-subformula-st test-symb* (*conn c l*)
⟨*proof*⟩

To ease the finding of proofs, we give some explicit theorem about the decomposition.

lemma *all-subformula-st-decomp-rec*:
all-subformula-st test-symb (*conn c l*) $\Longrightarrow$ *wf-conn c l*
$\Longrightarrow$ (*test-symb* (*conn c l*) $\wedge$ ($\forall\varphi\in$ *set l. all-subformula-st test-symb* φ))
⟨*proof*⟩

lemma *all-subformula-st-decomp*:
fixes *c* :: *'v connective* **and** *l* :: *'v propo list*
assumes *wf-conn c l*
shows *all-subformula-st test-symb* (*conn c l*)
$\longleftrightarrow$ (*test-symb* (*conn c l*) $\wedge$ ($\forall\varphi\in$ *set l. all-subformula-st test-symb* φ))
⟨*proof*⟩

lemma *helper-fact*: $c \in$ *binary-connectives* $\longleftrightarrow$ (c = *COr* $\vee$ c = *CAnd* $\vee$ c = *CEq* $\vee$ c = *CImp*)
⟨*proof*⟩
lemma *all-subformula-st-decomp-explicit*[*simp*]:
fixes φ ψ :: *'v propo*
shows *all-subformula-st test-symb* (*FAnd* φ ψ)
$\longleftrightarrow$ (*test-symb* (*FAnd* φ ψ) $\wedge$ *all-subformula-st test-symb* φ $\wedge$ *all-subformula-st test-symb* ψ)
and *all-subformula-st test-symb* (*FOr* φ ψ)
$\longleftrightarrow$ (*test-symb* (*FOr* φ ψ) $\wedge$ *all-subformula-st test-symb* φ $\wedge$ *all-subformula-st test-symb* ψ)
and *all-subformula-st test-symb* (*FNot* φ)
$\longleftrightarrow$ (*test-symb* (*FNot* φ) $\wedge$ *all-subformula-st test-symb* φ)

and *all-subformula-st test-symb* $(FEq\ \varphi\ \psi)$
$\longleftrightarrow$ $(test\text{-}symb\ (FEq\ \varphi\ \psi) \wedge\ all\text{-}subformula\text{-}st\ test\text{-}symb\ \varphi \wedge all\text{-}subformula\text{-}st\ test\text{-}symb\ \psi)$
and *all-subformula-st test-symb* $(FImp\ \varphi\ \psi)$
$\longleftrightarrow$ $(test\text{-}symb\ (FImp\ \varphi\ \psi) \wedge all\text{-}subformula\text{-}st\ test\text{-}symb\ \varphi \wedge all\text{-}subformula\text{-}st\ test\text{-}symb\ \psi)$
$\langle proof \rangle$

As *all-subformula-st* tests recursively, the function is true on every subformula.

lemma *subformula-all-subformula-st*:
$\psi \preceq \varphi \Longrightarrow all\text{-}subformula\text{-}st\ test\text{-}symb\ \varphi \Longrightarrow all\text{-}subformula\text{-}st\ test\text{-}symb\ \psi$
$\langle proof \rangle$

The following theorem *no-test-symb-step-exists* shows the link between the *test-symb* function and the corresponding rewrite relation r: if we assume that if every time *test-symb* is true, then a r can be applied, finally as long as $\neg$ *all-subformula-st test-symb* φ, then something can be rewritten in φ.

lemma *no-test-symb-step-exists*:
fixes r:: $'v\ propo \Rightarrow\ 'v\ propo \Rightarrow bool$ **and** *test-symb*:: $'v\ propo \Rightarrow bool$ **and** x :: $'v$
and φ :: $'v\ propo$
assumes *test-symb-false-nullary*: $\forall x.\ test\text{-}symb\ FF \wedge test\text{-}symb\ FT \wedge test\text{-}symb\ (FVar\ x)$
and $\forall \varphi'.\ \varphi' \preceq \varphi \longrightarrow (\neg test\text{-}symb\ \varphi') \longrightarrow (\exists\ \psi.\ r\ \varphi'\ \psi)$ **and**
$\neg$ *all-subformula-st test-symb* φ
shows $(\exists \psi\ \psi'.\ \psi \preceq \varphi \wedge r\ \psi\ \psi')$
$\langle proof \rangle$

7.2 Invariant conservation

If two rewrite relation are independant (or at least independant enough), then the property characterizing the first relation *all-subformula-st test-symb* remains true. The next show the same property, with changes in the assumptions.

The assumption $\forall \varphi'\ \psi.\ \varphi' \preceq \Phi \longrightarrow r\ \varphi'\ \psi \longrightarrow all\text{-}subformula\text{-}st\ test\text{-}symb\ \varphi' \longrightarrow all\text{-}subformula\text{-}st\ test\text{-}symb\ \psi$ means that rewriting with r does not mess up the property we want to preserve locally.

The previous assumption is not enough to go from r to *propo-rew-step* r: we have to add the assumption that rewriting inside does not mess up the term: $\forall c\ \xi\ \varphi\ \xi'\ \varphi'.\ \varphi \preceq \Phi \longrightarrow propo\text{-}rew\text{-}step\ r\ \varphi\ \varphi' \longrightarrow wf\text{-}conn\ c\ (\xi\ @\ \varphi\ \#\ \xi') \longrightarrow test\text{-}symb\ (conn\ c\ (\xi\ @\ \varphi\ \#\ \xi')) \longrightarrow test\text{-}symb\ \varphi' \longrightarrow test\text{-}symb\ (conn\ c\ (\xi\ @\ \varphi'\ \#\ \xi'))$

7.2.1 Invariant while lifting of the rewriting relation

The condition $\varphi \preceq \Phi$ (that will by used with $\Phi = \varphi$ most of the time) is here to ensure that the recursive conditions on Φ will moreover hold for the subterm we are rewriting. For example if there is no equivalence symbol in Φ, we do not have to care about equivalence symbols in the two previous assumptions.

lemma *propo-rew-step-inv-stay'*:
fixes r:: $'v\ propo \Rightarrow\ 'v\ propo \Rightarrow bool$ **and** *test-symb*:: $'v\ propo \Rightarrow bool$ **and** x :: $'v$
and $\varphi\ \psi\ \Phi$:: $'v\ propo$
assumes H: $\forall \varphi'\ \psi.\ \varphi' \preceq \Phi \longrightarrow r\ \varphi'\ \psi \longrightarrow all\text{-}subformula\text{-}st\ test\text{-}symb\ \varphi'$
$\longrightarrow all\text{-}subformula\text{-}st\ test\text{-}symb\ \psi$
and H': $\forall (c{::}\ 'v\ connective)\ \xi\ \varphi\ \xi'\ \varphi'.\ \varphi \preceq \Phi \longrightarrow propo\text{-}rew\text{-}step\ r\ \varphi\ \varphi'$
$\longrightarrow wf\text{-}conn\ c\ (\xi\ @\ \varphi\ \#\ \xi') \longrightarrow test\text{-}symb\ (conn\ c\ (\xi\ @\ \varphi\ \#\ \xi')) \longrightarrow test\text{-}symb\ \varphi'$

$\longrightarrow$ *test-symb* $(conn\ c\ (\xi\ @\ \varphi'\ \#\ \xi'))$ **and**
propo-rew-step $r\ \varphi\ \psi$ **and**
$\varphi \preceq \Phi$ **and**
all-subformula-st test-symb φ
shows *all-subformula-st test-symb* ψ
$\langle proof \rangle$

The need for $\varphi \preceq \Phi$ is not always necessary, hence we moreover have a version without inclusion.

lemma *propo-rew-step-inv-stay*:
fixes r:: $'v$ *propo* $\Rightarrow$ $'v$ *propo* $\Rightarrow$ *bool* **and** *test-symb*:: $'v$ *propo* $\Rightarrow$ *bool* **and** x :: $'v$
and $\varphi\ \psi$ :: $'v$ *propo*
assumes
H: $\forall \varphi'\ \psi.\ r\ \varphi'\ \psi \longrightarrow$ *all-subformula-st test-symb* $\varphi' \longrightarrow$ *all-subformula-st test-symb* ψ **and**
H': $\forall (c$:: $'v$ *connective*$)\ \xi\ \varphi\ \xi'\ \varphi'.$ *wf-conn* $c\ (\xi\ @\ \varphi\ \#\ \xi') \longrightarrow$ *test-symb* $(conn\ c\ (\xi\ @\ \varphi\ \#\ \xi'))$
$\longrightarrow$ *test-symb* $\varphi' \longrightarrow$ *test-symb* $(conn\ c\ (\xi\ @\ \varphi'\ \#\ \xi'))$ **and**
propo-rew-step $r\ \varphi\ \psi$ **and**
all-subformula-st test-symb φ
shows *all-subformula-st test-symb* ψ
$\langle proof \rangle$

The lemmas can be lifted to *full* $(propo\text{-}rew\text{-}step\ r)$ instead of *propo-rew-step*

7.2.2 Invariant after all rewriting

lemma *full-propo-rew-step-inv-stay-with-inc*:
fixes r:: $'v$ *propo* $\Rightarrow$ $'v$ *propo* $\Rightarrow$ *bool* **and** *test-symb*:: $'v$ *propo* $\Rightarrow$ *bool* **and** x :: $'v$
and $\varphi\ \psi$ :: $'v$ *propo*
assumes
H: $\forall\ \varphi\ \psi.$ *propo-rew-step* $r\ \varphi\ \psi \longrightarrow$ *all-subformula-st test-symb* φ
$\longrightarrow$ *all-subformula-st test-symb* ψ **and**
H': $\forall (c$:: $'v$ *connective*$)\ \xi\ \varphi\ \xi'\ \varphi'.\ \varphi \preceq \Phi \longrightarrow$ *propo-rew-step* $r\ \varphi\ \varphi'$
$\longrightarrow$ *wf-conn* $c\ (\xi\ @\ \varphi\ \#\ \xi') \longrightarrow$ *test-symb* $(conn\ c\ (\xi\ @\ \varphi\ \#\ \xi')) \longrightarrow$ *test-symb* φ'
$\longrightarrow$ *test-symb* $(conn\ c\ (\xi\ @\ \varphi'\ \#\ \xi'))$ **and**
$\varphi \preceq \Phi$ **and**
full: *full* $(propo\text{-}rew\text{-}step\ r)\ \varphi\ \psi$ **and**
init: *all-subformula-st test-symb* φ
shows *all-subformula-st test-symb* ψ
$\langle proof \rangle$

lemma *full-propo-rew-step-inv-stay'*:
fixes r:: $'v$ *propo* $\Rightarrow$ $'v$ *propo* $\Rightarrow$ *bool* **and** *test-symb*:: $'v$ *propo* $\Rightarrow$ *bool* **and** x :: $'v$
and $\varphi\ \psi$ :: $'v$ *propo*
assumes
H: $\forall\ \varphi\ \psi.$ *propo-rew-step* $r\ \varphi\ \psi \longrightarrow$ *all-subformula-st test-symb* φ
$\longrightarrow$ *all-subformula-st test-symb* ψ **and**
H': $\forall (c$:: $'v$ *connective*$)\ \xi\ \varphi\ \xi'\ \varphi'.$ *propo-rew-step* $r\ \varphi\ \varphi' \longrightarrow$ *wf-conn* $c\ (\xi\ @\ \varphi\ \#\ \xi')$
$\longrightarrow$ *test-symb* $(conn\ c\ (\xi\ @\ \varphi\ \#\ \xi')) \longrightarrow$ *test-symb* $\varphi' \longrightarrow$ *test-symb* $(conn\ c\ (\xi\ @\ \varphi'\ \#\ \xi'))$ **and**
full: *full* $(propo\text{-}rew\text{-}step\ r)\ \varphi\ \psi$ **and**
init: *all-subformula-st test-symb* φ
shows *all-subformula-st test-symb* ψ
$\langle proof \rangle$

lemma *full-propo-rew-step-inv-stay*:
fixes r:: $'v$ *propo* $\Rightarrow$ $'v$ *propo* $\Rightarrow$ *bool* **and** *test-symb*:: $'v$ *propo* $\Rightarrow$ *bool* **and** x :: $'v$
and $\varphi\ \psi$ :: $'v$ *propo*

assumes
H: $\forall \varphi\ \psi.\ r\ \varphi\ \psi \longrightarrow$ *all-subformula-st test-symb* $\varphi \longrightarrow$ *all-subformula-st test-symb* ψ **and**
H': $\forall (c{::}\ 'v\ connective)\ \xi\ \varphi\ \xi'\ \varphi'.$ *wf-conn* $c\ (\xi\ @\ \varphi\ \#\ \xi') \longrightarrow$ *test-symb* $(conn\ c\ (\xi\ @\ \varphi\ \#\ \xi'))$
$\longrightarrow$ *test-symb* $\varphi' \longrightarrow$ *test-symb* $(conn\ c\ (\xi\ @\ \varphi'\ \#\ \xi'))$ **and**
full: *full* (*propo-rew-step* r) $\varphi\ \psi$ **and**
init: *all-subformula-st test-symb* φ
shows *all-subformula-st test-symb* ψ
⟨*proof*⟩

lemma *full-propo-rew-step-inv-stay-conn*:
fixes $r{::}\ 'v\ propo \Rightarrow\ 'v\ propo \Rightarrow bool$ **and** *test-symb*$::\ 'v\ propo \Rightarrow bool$ **and** $x :: 'v$
and $\varphi\ \psi :: 'v\ propo$
assumes
H: $\forall \varphi\ \psi.\ r\ \varphi\ \psi \longrightarrow$ *all-subformula-st test-symb* $\varphi \longrightarrow$ *all-subformula-st test-symb* ψ **and**
H': $\forall (c{::}\ 'v\ connective)\ l\ l'.$ *wf-conn* $c\ l \longrightarrow$ *wf-conn* $c\ l'$
$\longrightarrow$ (*test-symb* $(conn\ c\ l) \longleftrightarrow$ *test-symb* $(conn\ c\ l')$) **and**
full: *full* (*propo-rew-step* r) $\varphi\ \psi$ **and**
init: *all-subformula-st test-symb* φ
shows *all-subformula-st test-symb* ψ
⟨*proof*⟩

end
theory *Prop-Normalisation*
imports *Main Prop-Logic Prop-Abstract-Transformation ../lib/Multiset-More*
begin

Given the previous definition about abstract rewriting and theorem about them, we now have the detailed rule making the transformation into CNF/DNF.

8 Rewrite Rules

The idea of Christoph Weidenbach's book is to remove gradually the operators: first equivalencies, then implication, after that the unused true/false and finally the reorganizing the or/and. We will prove each transformation seperately.

8.1 Elimination of the equivalences

The first transformation consists in removing every equivalence symbol.

inductive *elim-equiv* $::\ 'v\ propo \Rightarrow\ 'v\ propo \Rightarrow bool$ **where**
elim-equiv[*simp*]: *elim-equiv* (*FEq* $\varphi\ \psi$) (*FAnd* (*FImp* $\varphi\ \psi$) (*FImp* $\psi\ \varphi$))

lemma *elim-equiv-transformation-consistent*:
$A \models FEq\ \varphi\ \psi \longleftrightarrow A \models FAnd\ (FImp\ \varphi\ \psi)\ (FImp\ \psi\ \varphi)$
⟨*proof*⟩

lemma *elim-equiv-explicit*: *elim-equiv* $\varphi\ \psi \Longrightarrow \forall A.\ A \models \varphi \longleftrightarrow A \models \psi$
⟨*proof*⟩

lemma *elim-equiv-consistent*: *preserves-un-sat elim-equiv*
⟨*proof*⟩

lemma *elimEquv-lifted-consistant*:

preserves-un-sat (*full* (*propo-rew-step elim-equiv*))
⟨*proof*⟩

This function ensures that there is no equivalencies left in the formula tested by *no-equiv-symb*.

fun *no-equiv-symb* :: $'v$ *propo* ⇒ *bool* **where**
no-equiv-symb (*FEq* - -) = *False* |
no-equiv-symb - = *True*

Given the definition of *no-equiv-symb*, it does not depend on the formula, but only on the connective used.

lemma *no-equiv-symb-conn-characterization*[*simp*]:
fixes *c* :: $'v$ *connective* **and** *l* :: $'v$ *propo list*
assumes *wf*: *wf-conn c l*
shows *no-equiv-symb* (*conn c l*) ⟷ *c* ≠ *CEq*
⟨*proof*⟩

definition *no-equiv* **where** *no-equiv* = *all-subformula-st no-equiv-symb*

lemma *no-equiv-eq*[*simp*]:
fixes φ ψ :: $'v$ *propo*
shows
¬*no-equiv* (*FEq* φ ψ)
no-equiv FT
no-equiv FF
⟨*proof*⟩

The following lemma helps to reconstruct *no-equiv* expressions: this representation is easier to use than the set definition.

lemma *all-subformula-st-decomp-explicit-no-equiv*[*iff*]:
fixes φ ψ :: $'v$ *propo*
shows
no-equiv (*FNot* φ) ⟷ *no-equiv* φ
no-equiv (*FAnd* φ ψ) ⟷ (*no-equiv* φ ∧ *no-equiv* ψ)
no-equiv (*FOr* φ ψ) ⟷ (*no-equiv* φ ∧ *no-equiv* ψ)
no-equiv (*FImp* φ ψ) ⟷ (*no-equiv* φ ∧ *no-equiv* ψ)
⟨*proof*⟩

A theorem to show the link between the rewrite relation *elim-equiv* and the function *no-equiv-symb*. This theorem is one of the assumption we need to characterize the transformation.

lemma *no-equiv-elim-equiv-step*:
fixes φ :: $'v$ *propo*
assumes *no-equiv*: ¬ *no-equiv* φ
shows $\exists \psi\ \psi'.\ \psi \preceq \varphi \wedge$ *elim-equiv* ψ ψ'
⟨*proof*⟩

Given all the previous theorem and the characterization, once we have rewritten everything, there is no equivalence symbol any more.

lemma *no-equiv-full-propo-rew-step-elim-equiv*:
full (*propo-rew-step elim-equiv*) φ ψ ⟹ *no-equiv* ψ
⟨*proof*⟩

8.2 Eliminate Implication

After that, we can eliminate the implication symbols.

inductive *elim-imp* :: $'v$ *propo* $\Rightarrow$ $'v$ *propo* $\Rightarrow$ *bool* **where**
[*simp*]: *elim-imp* (*FImp* φ ψ) (*FOr* (*FNot* φ) ψ)

lemma *elim-imp-transformation-consistent*:
  $A \models$ *FImp* φ $\psi \longleftrightarrow A \models$ *FOr* (*FNot* φ) ψ
  $\langle proof \rangle$

lemma *elim-imp-explicit*: *elim-imp* φ $\psi \Longrightarrow \forall A.\ A \models \varphi \longleftrightarrow A \models \psi$
  $\langle proof \rangle$

lemma *elim-imp-consistent*: *preserves-un-sat elim-imp*
  $\langle proof \rangle$

lemma *elim-imp-lifted-consistant*:
  preserves-un-sat (*full* (*propo-rew-step elim-imp*))
  $\langle proof \rangle$

fun *no-imp-symb* **where**
no-imp-symb (*FImp* - -) = *False* |
no-imp-symb - = *True*

lemma *no-imp-symb-conn-characterization*:
  wf-conn c l $\Longrightarrow$ *no-imp-symb* (*conn c l*) $\longleftrightarrow$ $c \neq$ *CImp*
  $\langle proof \rangle$

definition *no-imp* **where** *no-imp* $\equiv$ *all-subformula-st no-imp-symb*
declare *no-imp-def*[*simp*]

lemma *no-imp-Imp*[*simp*]:
  $\neg$*no-imp* (*FImp* φ ψ)
  no-imp FT
  no-imp FF
  $\langle proof \rangle$

lemma *all-subformula-st-decomp-explicit-imp*[*simp*]:
  fixes φ ψ :: $'v$ *propo*
  shows
    no-imp (*FNot* φ) $\longleftrightarrow$ *no-imp* φ
    no-imp (*FAnd* φ ψ) $\longleftrightarrow$ (*no-imp* φ $\wedge$ *no-imp* ψ)
    no-imp (*FOr* φ ψ) $\longleftrightarrow$ (*no-imp* φ $\wedge$ *no-imp* ψ)
  $\langle proof \rangle$

Invariant of the *elim-imp* transformation

lemma *elim-imp-no-equiv*:
  elim-imp φ $\psi \Longrightarrow$ *no-equiv* $\varphi \Longrightarrow$ *no-equiv* ψ
  $\langle proof \rangle$

lemma *elim-imp-inv*:
  fixes φ ψ :: $'v$ *propo*
  assumes *full* (*propo-rew-step elim-imp*) φ ψ **and** *no-equiv* φ
  shows *no-equiv* ψ
  $\langle proof \rangle$

lemma *no-no-imp-elim-imp-step-exists*:
  fixes φ :: $'v$ *propo*

assumes *no-equiv*: $\neg$ *no-imp* φ
shows $\exists\psi\ \psi'.\ \psi \preceq \varphi \wedge$ *elim-imp* $\psi\ \psi'$
⟨*proof*⟩

lemma *no-imp-full-propo-rew-step-elim-imp*: *full* (*propo-rew-step elim-imp*) $\varphi\ \psi \Longrightarrow$ *no-imp* ψ
⟨*proof*⟩

8.3 Eliminate all the True and False in the formula

Contrary to the book, we have to give the transformation and the "commutative" transformation. The latter is implicit in the book.

inductive *elimTB* **where**
ElimTB1: *elimTB* (*FAnd* φ *FT*) φ |
ElimTB1': *elimTB* (*FAnd FT* φ) φ |

ElimTB2: *elimTB* (*FAnd* φ *FF*) *FF* |
ElimTB2': *elimTB* (*FAnd FF* φ) *FF* |

ElimTB3: *elimTB* (*FOr* φ *FT*) *FT* |
ElimTB3': *elimTB* (*FOr FT* φ) *FT* |

ElimTB4: *elimTB* (*FOr* φ *FF*) φ |
ElimTB4': *elimTB* (*FOr FF* φ) φ |

ElimTB5: *elimTB* (*FNot FT*) *FF* |
ElimTB6: *elimTB* (*FNot FF*) *FT*

lemma *elimTB-consistent*: *preserves-un-sat elimTB*
⟨*proof*⟩

inductive *no-T-F-symb* :: *'v propo* $\Rightarrow$ *bool* **where**
no-T-F-symb-comp: $c \neq CF \Longrightarrow c \neq CT \Longrightarrow$ *wf-conn* $c\ l \Longrightarrow (\forall\varphi \in set\ l.\ \varphi \neq FT \wedge \varphi \neq FF)$
$\Longrightarrow$ *no-T-F-symb* (*conn* $c\ l$)

lemma *wf-conn-no-T-F-symb-iff*[*simp*]:
wf-conn $c\ \psi s \Longrightarrow$
no-T-F-symb (*conn* $c\ \psi s$) $\longleftrightarrow (c \neq CF \wedge c \neq CT \wedge (\forall\psi{\in}set\ \psi s.\ \psi \neq FF \wedge \psi \neq FT))$
⟨*proof*⟩

lemma *wf-conn-no-T-F-symb-iff-explicit*[*simp*]:
no-T-F-symb (*FAnd* $\varphi\ \psi$) $\longleftrightarrow (\forall\chi \in set\ [\varphi, \psi].\ \chi \neq FF \wedge \chi \neq FT)$
no-T-F-symb (*FOr* $\varphi\ \psi$) $\longleftrightarrow (\forall\chi \in set\ [\varphi, \psi].\ \chi \neq FF \wedge \chi \neq FT)$
no-T-F-symb (*FEq* $\varphi\ \psi$) $\longleftrightarrow (\forall\chi \in set\ [\varphi, \psi].\ \chi \neq FF \wedge \chi \neq FT)$
no-T-F-symb (*FImp* $\varphi\ \psi$) $\longleftrightarrow (\forall\chi \in set\ [\varphi, \psi].\ \chi \neq FF \wedge \chi \neq FT)$
⟨*proof*⟩

lemma *no-T-F-symb-false*[*simp*]:
fixes c :: *'v connective*
shows
$\neg$*no-T-F-symb* (*FT* :: *'v propo*)
$\neg$*no-T-F-symb* (*FF* :: *'v propo*)
⟨*proof*⟩

lemma *no-T-F-symb-bool*[*simp*]:
  fixes $x :: 'v$
  shows *no-T-F-symb* (*FVar* x)
  ⟨*proof*⟩

lemma *no-T-F-symb-fnot-imp*:
  $\neg$*no-T-F-symb* (*FNot* φ) $\Longrightarrow \varphi = FT \lor \varphi = FF$
⟨*proof*⟩

lemma *no-T-F-symb-fnot*[*simp*]:
  no-T-F-symb (*FNot* φ) $\longleftrightarrow \neg(\varphi = FT \lor \varphi = FF)$
  ⟨*proof*⟩

Actually it is not possible to remover every FT and FF: if the formula is equal to true or false, we can not remove it.

inductive *no-T-F-symb-except-toplevel* **where**
no-T-F-symb-except-toplevel-true[*simp*]: *no-T-F-symb-except-toplevel FT* |
no-T-F-symb-except-toplevel-false[*simp*]: *no-T-F-symb-except-toplevel FF* |
noTrue-no-T-F-symb-except-toplevel[*simp*]: *no-T-F-symb* $\varphi \Longrightarrow$ *no-T-F-symb-except-toplevel* φ

lemma *no-T-F-symb-except-toplevel-bool*:
  fixes $x :: 'v$
  shows *no-T-F-symb-except-toplevel* (*FVar* x)
  ⟨*proof*⟩

lemma *no-T-F-symb-except-toplevel-not-decom*:
  $\varphi \neq FT \Longrightarrow \varphi \neq FF \Longrightarrow$ *no-T-F-symb-except-toplevel* (*FNot* φ)
  ⟨*proof*⟩

lemma *no-T-F-symb-except-toplevel-bin-decom*:
  fixes $\varphi\ \psi :: 'v$ *propo*
  assumes $\varphi \neq FT$ **and** $\varphi \neq FF$ **and** $\psi \neq FT$ **and** $\psi \neq FF$
  and c: $c \in$ *binary-connectives*
  shows *no-T-F-symb-except-toplevel* (*conn* c [φ, ψ])
  ⟨*proof*⟩

lemma *no-T-F-symb-except-toplevel-if-is-a-true-false*:
  fixes $l :: 'v$ *propo list* **and** $c :: 'v$ *connective*
  assumes *corr*: *wf-conn c l*
  and $FT \in$ *set* $l \lor FF \in$ *set* l
  shows $\neg$*no-T-F-symb-except-toplevel* (*conn c l*)
  ⟨*proof*⟩

lemma *no-T-F-symb-except-top-level-false-example*[*simp*]:
  fixes $\varphi\ \psi :: 'v$ *propo*
  assumes $\varphi = FT \lor \psi = FT \lor \varphi = FF \lor \psi = FF$
  shows
    $\neg$ *no-T-F-symb-except-toplevel* (*FAnd* φ ψ)
    $\neg$ *no-T-F-symb-except-toplevel* (*FOr* φ ψ)
    $\neg$ *no-T-F-symb-except-toplevel* (*FImp* φ ψ)
    $\neg$ *no-T-F-symb-except-toplevel* (*FEq* φ ψ)
  ⟨*proof*⟩

lemma *no-T-F-symb-except-top-level-false-not*[*simp*]:
fixes φ ψ :: $'v$ *propo*
assumes $\varphi = FT \lor \varphi = FF$
shows
$\neg$ *no-T-F-symb-except-toplevel* (*FNot* φ)
⟨*proof*⟩

This is the local extension of *no-T-F-symb-except-toplevel*.

definition *no-T-F-except-top-level* **where**
no-T-F-except-top-level $\equiv$ *all-subformula-st no-T-F-symb-except-toplevel*

This is another property we will use. While this version might seem to be the one we want to prove, it is not since FT can not be reduced.

definition *no-T-F* **where**
no-T-F $\equiv$ *all-subformula-st no-T-F-symb*

lemma *no-T-F-except-top-level-false*:
fixes l :: $'v$ *propo list* **and** c :: $'v$ *connective*
assumes *wf-conn c l*
and $FT \in set\ l \lor FF \in set\ l$
shows $\neg$*no-T-F-except-top-level* (*conn c l*)
⟨*proof*⟩

lemma *no-T-F-except-top-level-false-example*[*simp*]:
fixes φ ψ :: $'v$ *propo*
assumes $\varphi = FT \lor \psi = FT \lor \varphi = FF \lor \psi = FF$
shows
$\neg$*no-T-F-except-top-level* (*FAnd* φ ψ)
$\neg$*no-T-F-except-top-level* (*FOr* φ ψ)
$\neg$*no-T-F-except-top-level* (*FEq* φ ψ)
$\neg$*no-T-F-except-top-level* (*FImp* φ ψ)
⟨*proof*⟩

lemma *no-T-F-symb-except-toplevel-no-T-F-symb*:
no-T-F-symb-except-toplevel $\varphi \Longrightarrow \varphi \neq FF \Longrightarrow \varphi \neq FT \Longrightarrow$ *no-T-F-symb* φ
⟨*proof*⟩

The two following lemmas give the precise link between the two definitions.

lemma *no-T-F-symb-except-toplevel-all-subformula-st-no-T-F-symb*:
no-T-F-except-top-level $\varphi \Longrightarrow \varphi \neq FF \Longrightarrow \varphi \neq FT \Longrightarrow$ *no-T-F* φ
⟨*proof*⟩

lemma *no-T-F-no-T-F-except-top-level*:
no-T-F $\varphi \Longrightarrow$ *no-T-F-except-top-level* φ
⟨*proof*⟩

lemma *no-T-F-except-top-level-simp*[*simp*]: *no-T-F-except-top-level FF no-T-F-except-top-level FT*
⟨*proof*⟩

lemma *no-T-F-no-T-F-except-top-level'*[*simp*]:
no-T-F-except-top-level $\varphi \longleftrightarrow (\varphi = FF \lor \varphi = FT \lor$ *no-T-F* $\varphi)$
⟨*proof*⟩

lemma *no-T-F-bin-decomp*[*simp*]:
  assumes *c*: $c \in$ *binary-connectives*
  shows *no-T-F* (*conn c* [φ, ψ]) $\longleftrightarrow$ (*no-T-F* φ $\wedge$ *no-T-F* ψ)
⟨*proof*⟩

lemma *no-T-F-bin-decomp-expanded*[*simp*]:
  assumes *c*: $c = CAnd \vee c = COr \vee c = CEq \vee c = CImp$
  shows *no-T-F* (*conn c* [φ, ψ]) $\longleftrightarrow$ (*no-T-F* φ $\wedge$ *no-T-F* ψ)
  ⟨*proof*⟩

lemma *no-T-F-comp-expanded-explicit*[*simp*]:
  fixes φ ψ :: *'v propo*
  shows
    no-T-F (*FAnd* φ ψ) $\longleftrightarrow$ (*no-T-F* φ $\wedge$ *no-T-F* ψ)
    no-T-F (*FOr* φ ψ) $\longleftrightarrow$ (*no-T-F* φ $\wedge$ *no-T-F* ψ)
    no-T-F (*FEq* φ ψ) $\longleftrightarrow$ (*no-T-F* φ $\wedge$ *no-T-F* ψ)
    no-T-F (*FImp* φ ψ) $\longleftrightarrow$ (*no-T-F* φ $\wedge$ *no-T-F* ψ)
  ⟨*proof*⟩

lemma *no-T-F-comp-not*[*simp*]:
  fixes φ ψ :: *'v propo*
  shows *no-T-F* (*FNot* φ) $\longleftrightarrow$ *no-T-F* φ
  ⟨*proof*⟩

lemma *no-T-F-decomp*:
  fixes φ ψ :: *'v propo*
  assumes φ: *no-T-F* (*FAnd* φ ψ) $\vee$ *no-T-F* (*FOr* φ ψ) $\vee$ *no-T-F* (*FEq* φ ψ) $\vee$ *no-T-F* (*FImp* φ ψ)
  shows *no-T-F* ψ **and** *no-T-F* φ
  ⟨*proof*⟩

lemma *no-T-F-decomp-not*:
  fixes φ :: *'v propo*
  assumes φ: *no-T-F* (*FNot* φ)
  shows *no-T-F* φ
  ⟨*proof*⟩

lemma *no-T-F-symb-except-toplevel-step-exists*:
  fixes φ ψ :: *'v propo*
  assumes *no-equiv* φ **and** *no-imp* φ
  shows $\psi \preceq \varphi \Longrightarrow \neg$ *no-T-F-symb-except-toplevel* $\psi \Longrightarrow \exists \psi'.$ *elimTB* ψ ψ'
⟨*proof*⟩

lemma *no-T-F-except-top-level-rew*:
  fixes φ :: *'v propo*
  assumes *noTB*: $\neg$ *no-T-F-except-top-level* φ **and** *no-equiv*: *no-equiv* φ **and** *no-imp*: *no-imp* φ
  shows $\exists \psi\ \psi'.\ \psi \preceq \varphi \wedge$ *elimTB* ψ ψ'
⟨*proof*⟩

lemma *elimTB-inv*:
  fixes φ ψ :: *'v propo*
  assumes *full* (*propo-rew-step elimTB*) φ ψ
  and *no-equiv* φ **and** *no-imp* φ
  shows *no-equiv* ψ **and** *no-imp* ψ
⟨*proof*⟩

lemma *elimTB-full-propo-rew-step*:
  fixes φ ψ :: $'v$ *propo*
  assumes *no-equiv* φ **and** *no-imp* φ **and** *full* (*propo-rew-step elimTB*) φ ψ
  shows *no-T-F-except-top-level* ψ
  ⟨*proof*⟩

8.4 PushNeg

Push the negation inside the formula, until the litteral.

inductive *pushNeg* **where**
PushNeg1[*simp*]: *pushNeg* (*FNot* (*FAnd* φ ψ)) (*FOr* (*FNot* φ) (*FNot* ψ)) |
PushNeg2[*simp*]: *pushNeg* (*FNot* (*FOr* φ ψ)) (*FAnd* (*FNot* φ) (*FNot* ψ)) |
PushNeg3[*simp*]: *pushNeg* (*FNot* (*FNot* φ)) φ

lemma *pushNeg-transformation-consistent*:
$A \models FNot\ (FAnd\ \varphi\ \psi) \longleftrightarrow A \models (FOr\ (FNot\ \varphi)\ (FNot\ \psi))$
$A \models FNot\ (FOr\ \varphi\ \psi) \longleftrightarrow A \models (FAnd\ (FNot\ \varphi)\ (FNot\ \psi))$
$A \models FNot\ (FNot\ \varphi) \longleftrightarrow A \models \varphi$
  ⟨*proof*⟩

lemma *pushNeg-explicit*: *pushNeg* φ $\psi \Longrightarrow \forall A.\ A \models \varphi \longleftrightarrow A \models \psi$
  ⟨*proof*⟩

lemma *pushNeg-consistent*: *preserves-un-sat pushNeg*
  ⟨*proof*⟩

lemma *pushNeg-lifted-consistant*:
preserves-un-sat (*full* (*propo-rew-step pushNeg*))
  ⟨*proof*⟩

fun *simple* **where**
simple FT = *True* |
simple FF = *True* |
simple (*FVar* -) = *True* |
simple - = *False*

lemma *simple-decomp*:
  simple $\varphi \longleftrightarrow (\varphi = FT \lor \varphi = FF \lor (\exists x.\ \varphi = FVar\ x))$
  ⟨*proof*⟩

lemma *subformula-conn-decomp-simple*:
  fixes φ ψ :: $'v$ *propo*
  assumes *s*: *simple* ψ
  shows $\varphi \preceq FNot\ \psi \longleftrightarrow (\varphi = FNot\ \psi \lor \varphi = \psi)$
⟨*proof*⟩

lemma *subformula-conn-decomp-explicit*[*simp*]:
  fixes φ :: $'v$ *propo* **and** x :: $'v$
  shows
    $\varphi \preceq FNot\ FT \longleftrightarrow (\varphi = FNot\ FT \lor \varphi = FT)$
    $\varphi \preceq FNot\ FF \longleftrightarrow (\varphi = FNot\ FF \lor \varphi = FF)$
    $\varphi \preceq FNot\ (FVar\ x) \longleftrightarrow (\varphi = FNot\ (FVar\ x) \lor \varphi = FVar\ x)$

⟨*proof*⟩

fun *simple-not-symb* **where**
simple-not-symb (*FNot* φ) = (*simple* φ) |
simple-not-symb - = *True*

definition *simple-not* **where**
simple-not = *all-subformula-st simple-not-symb*
declare *simple-not-def*[*simp*]

lemma *simple-not-Not*[*simp*]:
  ¬ *simple-not* (*FNot* (*FAnd* φ ψ))
  ¬ *simple-not* (*FNot* (*FOr* φ ψ))
  ⟨*proof*⟩

lemma *simple-not-step-exists*:
  fixes φ ψ :: *'v propo*
  assumes *no-equiv* φ **and** *no-imp* φ
  shows $\psi \preceq \varphi \Longrightarrow$ ¬ *simple-not-symb* $\psi \Longrightarrow \exists \psi'$. *pushNeg* ψ ψ'
  ⟨*proof*⟩

lemma *simple-not-rew*:
  fixes φ :: *'v propo*
  assumes *noTB*: ¬ *simple-not* φ **and** *no-equiv*: *no-equiv* φ **and** *no-imp*: *no-imp* φ
  shows $\exists \psi\ \psi'$. $\psi \preceq \varphi \wedge$ *pushNeg* ψ ψ'
⟨*proof*⟩

lemma *no-T-F-except-top-level-pushNeg1*:
  no-T-F-except-top-level (*FNot* (*FAnd* φ ψ)) $\Longrightarrow$ *no-T-F-except-top-level* (*FOr* (*FNot* φ) (*FNot* ψ))
  ⟨*proof*⟩

lemma *no-T-F-except-top-level-pushNeg2*:
  no-T-F-except-top-level (*FNot* (*FOr* φ ψ)) $\Longrightarrow$ *no-T-F-except-top-level* (*FAnd* (*FNot* φ) (*FNot* ψ))
  ⟨*proof*⟩

lemma *no-T-F-symb-pushNeg*:
  no-T-F-symb (*FOr* (*FNot* φ') (*FNot* ψ'))
  no-T-F-symb (*FAnd* (*FNot* φ') (*FNot* ψ'))
  no-T-F-symb (*FNot* (*FNot* φ'))
  ⟨*proof*⟩

lemma *propo-rew-step-pushNeg-no-T-F-symb*:
  propo-rew-step pushNeg φ $\psi \Longrightarrow$ *no-T-F-except-top-level* $\varphi \Longrightarrow$ *no-T-F-symb* $\varphi \Longrightarrow$ *no-T-F-symb* ψ
  ⟨*proof*⟩

lemma *propo-rew-step-pushNeg-no-T-F*:
  propo-rew-step pushNeg φ $\psi \Longrightarrow$ *no-T-F* $\varphi \Longrightarrow$ *no-T-F* ψ
⟨*proof*⟩

lemma *pushNeg-inv*:
  fixes φ ψ :: *'v propo*
  assumes *full* (*propo-rew-step pushNeg*) φ ψ
  and *no-equiv* φ **and** *no-imp* φ **and** *no-T-F-except-top-level* φ

shows *no-equiv* ψ **and** *no-imp* ψ **and** *no-T-F-except-top-level* ψ
⟨*proof*⟩

lemma *pushNeg-full-propo-rew-step*:
fixes φ ψ :: *'v propo*
assumes
no-equiv φ **and**
no-imp φ **and**
full (*propo-rew-step pushNeg*) φ ψ **and**
no-T-F-except-top-level φ
shows *simple-not* ψ
⟨*proof*⟩

8.5 Push inside

inductive *push-conn-inside* :: *'v connective* ⇒ *'v connective* ⇒ *'v propo* ⇒ *'v propo* ⇒ *bool*
for *c c'*:: *'v connective* **where**
push-conn-inside-l[*simp*]: $c = CAnd \lor c = COr \Longrightarrow c' = CAnd \lor c' = COr$
$\Longrightarrow$ *push-conn-inside c c'* (*conn c* [*conn c'* [$\varphi 1$, $\varphi 2$], ψ])
(*conn c'* [*conn c* [$\varphi 1$, ψ], *conn c* [$\varphi 2$, ψ]]) |
push-conn-inside-r[*simp*]: $c = CAnd \lor c = COr \Longrightarrow c' = CAnd \lor c' = COr$
$\Longrightarrow$ *push-conn-inside c c'* (*conn c* [ψ, *conn c'* [$\varphi 1$, $\varphi 2$]])
(*conn c'* [*conn c* [ψ, $\varphi 1$], *conn c* [ψ, $\varphi 2$]])

lemma *push-conn-inside-explicit*: *push-conn-inside c c'* φ $\psi \Longrightarrow \forall A.\ A \models \varphi \longleftrightarrow A \models \psi$
⟨*proof*⟩

lemma *push-conn-inside-consistent*: *preserves-un-sat* (*push-conn-inside c c'*)
⟨*proof*⟩

lemma *propo-rew-step-push-conn-inside*[*simp*]:
¬*propo-rew-step* (*push-conn-inside c c'*) *FT* ψ ¬*propo-rew-step* (*push-conn-inside c c'*) *FF* ψ
⟨*proof*⟩

inductive *not-c-in-c'-symb*:: *'v connective* ⇒ *'v connective* ⇒ *'v propo* ⇒ *bool* **for** *c c'* **where**
not-c-in-c'-symb-l[*simp*]: *wf-conn c* [*conn c'* [φ, φ'], ψ] $\Longrightarrow$ *wf-conn c'* [φ, φ']
$\Longrightarrow$ *not-c-in-c'-symb c c'* (*conn c* [*conn c'* [φ, φ'], ψ]) |
not-c-in-c'-symb-r[*simp*]: *wf-conn c* [ψ, *conn c'* [φ, φ']] $\Longrightarrow$ *wf-conn c'* [φ, φ']
$\Longrightarrow$ *not-c-in-c'-symb c c'* (*conn c* [ψ, *conn c'* [φ, φ']])

abbreviation *c-in-c'-symb c c'* φ ≡ ¬*not-c-in-c'-symb c c'* φ

lemma *c-in-c'-symb-simp*:
not-c-in-c'-symb c c' $\xi \Longrightarrow \xi = FF \lor \xi = FT \lor \xi = FVar\ x \lor \xi = FNot\ FF \lor \xi = FNot\ FT$
$\lor\ \xi = FNot\ (FVar\ x) \Longrightarrow False$
⟨*proof*⟩

lemma *c-in-c'-symb-simp'*[*simp*]:
¬*not-c-in-c'-symb c c' FF*
¬*not-c-in-c'-symb c c' FT*
¬*not-c-in-c'-symb c c'* (*FVar x*)
¬*not-c-in-c'-symb c c'* (*FNot FF*)

$\neg$*not-c-in-c'-symb c c'* (*FNot FT*)
$\neg$*not-c-in-c'-symb c c'* (*FNot* (*FVar x*))
⟨*proof*⟩

definition *c-in-c'-only* **where**
c-in-c'-only c c' $\equiv$ *all-subformula-st* (*c-in-c'-symb c c'*)

lemma *c-in-c'-only-simp*[*simp*]:
c-in-c'-only c c' FF
c-in-c'-only c c' FT
c-in-c'-only c c' (*FVar x*)
c-in-c'-only c c' (*FNot FF*)
c-in-c'-only c c' (*FNot FT*)
c-in-c'-only c c' (*FNot* (*FVar x*))
⟨*proof*⟩

lemma *not-c-in-c'-symb-commute*:
not-c-in-c'-symb c c' $\xi \Longrightarrow$ *wf-conn c* $[\varphi, \psi] \Longrightarrow \xi =$ *conn c* $[\varphi, \psi]$
$\Longrightarrow$ *not-c-in-c'-symb c c'* (*conn c* $[\psi, \varphi]$)
⟨*proof*⟩

lemma *not-c-in-c'-symb-commute'*:
wf-conn c $[\varphi, \psi] \Longrightarrow$ *c-in-c'-symb c c'* (*conn c* $[\varphi, \psi]$) $\longleftrightarrow$ *c-in-c'-symb c c'* (*conn c* $[\psi, \varphi]$)
⟨*proof*⟩

lemma *not-c-in-c'-comm*:
assumes *wf*: *wf-conn c* $[\varphi, \psi]$
shows *c-in-c'-only c c'* (*conn c* $[\varphi, \psi]$) $\longleftrightarrow$ *c-in-c'-only c c'* (*conn c* $[\psi, \varphi]$) (**is** *?A* $\longleftrightarrow$ *?B*)
⟨*proof*⟩

lemma *not-c-in-c'-simp*[*simp*]:
fixes $\varphi 1\ \varphi 2\ \psi$ :: *'v propo* **and** *x* :: *'v*
shows
c-in-c'-symb c c' FT
c-in-c'-symb c c' FF
c-in-c'-symb c c' (*FVar x*)
wf-conn c [*conn c'* $[\varphi 1, \varphi 2], \psi] \Longrightarrow$ *wf-conn c'* $[\varphi 1, \varphi 2]$
$\Longrightarrow \neg$ *c-in-c'-only c c'* (*conn c* [*conn c'* $[\varphi 1, \varphi 2], \psi]$)
⟨*proof*⟩

lemma *c-in-c'-symb-not*[*simp*]:
fixes *c c'* :: *'v connective* **and** ψ :: *'v propo*
shows *c-in-c'-symb c c'* (*FNot* ψ)
⟨*proof*⟩

lemma *c-in-c'-symb-step-exists*:
fixes φ :: *'v propo*
assumes *c*: *c* = *CAnd* $\vee$ *c* = *COr* **and** *c'*: *c'* = *CAnd* $\vee$ *c'* = *COr*
shows $\psi \preceq \varphi \Longrightarrow \neg$ *c-in-c'-symb c c'* $\psi \Longrightarrow \exists \psi'$. *push-conn-inside c c'* $\psi\ \psi'$
⟨*proof*⟩

lemma *c-in-c'-symb-rew*:
fixes φ :: *'v propo*

assumes *noTB*: $\neg$*c-in-c′-only* c c' φ
and c: $c = CAnd \lor c = COr$ **and** c': $c' = CAnd \lor c' = COr$
shows $\exists \psi\ \psi'.\ \psi \preceq \varphi \land$ *push-conn-inside* $c\ c'\ \psi\ \psi'$
⟨*proof*⟩

lemma *push-conn-insidec-in-c′-symb-no-T-F*:
fixes $\varphi\ \psi$ :: *′v propo*
shows *propo-rew-step* (*push-conn-inside* $c\ c'$) $\varphi\ \psi \Longrightarrow$ *no-T-F* $\varphi \Longrightarrow$ *no-T-F* ψ
⟨*proof*⟩

lemma *simple-propo-rew-step-push-conn-inside-inv*:
propo-rew-step (*push-conn-inside* $c\ c'$) $\varphi\ \psi \Longrightarrow$ *simple* $\varphi \Longrightarrow$ *simple* ψ
⟨*proof*⟩

lemma *simple-propo-rew-step-inv-push-conn-inside-simple-not*:
fixes $c\ c'$ :: *′v connective* **and** $\varphi\ \psi$ :: *′v propo*
shows *propo-rew-step* (*push-conn-inside* $c\ c'$) $\varphi\ \psi \Longrightarrow$ *simple-not* $\varphi \Longrightarrow$ *simple-not* ψ
⟨*proof*⟩

lemma *propo-rew-step-push-conn-inside-simple-not*:
fixes $\varphi\ \varphi'$ :: *′v propo* **and** $\xi\ \xi'$ :: *′v propo list* **and** c :: *′v connective*
assumes
propo-rew-step (*push-conn-inside* $c\ c'$) $\varphi\ \varphi'$ **and**
wf-conn c (ξ @ φ # ξ') **and**
simple-not-symb (*conn* c (ξ @ φ # ξ')) **and**
simple-not-symb φ'
shows *simple-not-symb* (*conn* c (ξ @ φ' # ξ'))
⟨*proof*⟩

lemma *push-conn-inside-not-true-false*:
push-conn-inside $c\ c'\ \varphi\ \psi \Longrightarrow \psi \neq FT \land \psi \neq FF$
⟨*proof*⟩

lemma *push-conn-inside-inv*:
fixes $\varphi\ \psi$ :: *′v propo*
assumes *full* (*propo-rew-step* (*push-conn-inside* $c\ c'$)) $\varphi\ \psi$
and *no-equiv* φ **and** *no-imp* φ **and** *no-T-F-except-top-level* φ **and** *simple-not* φ
shows *no-equiv* ψ **and** *no-imp* ψ **and** *no-T-F-except-top-level* ψ **and** *simple-not* ψ
⟨*proof*⟩

lemma *push-conn-inside-full-propo-rew-step*:
fixes $\varphi\ \psi$ :: *′v propo*
assumes
no-equiv φ **and**
no-imp φ **and**
full (*propo-rew-step* (*push-conn-inside* $c\ c'$)) $\varphi\ \psi$ **and**
no-T-F-except-top-level φ **and**
simple-not φ **and**
$c = CAnd \lor c = COr$ **and**
$c' = CAnd \lor c' = COr$
shows *c-in-c′-only* $c\ c'\ \psi$
⟨*proof*⟩

8.5.1 Only one type of connective in the formula (+ not)

inductive *only-c-inside-symb* :: $'v$ *connective* $\Rightarrow$ $'v$ *propo* $\Rightarrow$ *bool* **for** *c*:: $'v$ *connective* **where**
simple-only-c-inside[*simp*]: *simple* $\varphi \Longrightarrow$ *only-c-inside-symb c* φ |
simple-cnot-only-c-inside[*simp*]: *simple* $\varphi \Longrightarrow$ *only-c-inside-symb c* (*FNot* φ) |
only-c-inside-into-only-c-inside: *wf-conn c l* $\Longrightarrow$ *only-c-inside-symb c* (*conn c l*)

lemma *only-c-inside-symb-simp*[*simp*]:
only-c-inside-symb c FF only-c-inside-symb c FT only-c-inside-symb c (*FVar x*) ⟨*proof*⟩

definition *only-c-inside* **where** *only-c-inside c* = *all-subformula-st* (*only-c-inside-symb c*)

lemma *only-c-inside-symb-decomp*:
only-c-inside-symb c $\psi \longleftrightarrow$ (*simple* ψ
$\lor$ ($\exists$ φ'. ψ = *FNot* φ' $\land$ *simple* φ')
$\lor$ ($\exists$ *l*. ψ = *conn c l* $\land$ *wf-conn c l*))
⟨*proof*⟩

lemma *only-c-inside-symb-decomp-not*[*simp*]:
fixes *c* :: $'v$ *connective*
assumes *c*: $c \neq$ *CNot*
shows *only-c-inside-symb c* (*FNot* ψ) $\longleftrightarrow$ *simple* ψ
⟨*proof*⟩

lemma *only-c-inside-decomp-not*[*simp*]:
assumes *c*: $c \neq$ *CNot*
shows *only-c-inside c* (*FNot* ψ) $\longleftrightarrow$ *simple* ψ
⟨*proof*⟩

lemma *only-c-inside-decomp*:
only-c-inside c $\varphi \longleftrightarrow$
($\forall \psi$. $\psi \preceq \varphi \longrightarrow$ (*simple* ψ $\lor$ ($\exists$ φ'. ψ = *FNot* φ' $\land$ *simple* φ')
$\lor$ ($\exists$ *l*. ψ = *conn c l* $\land$ *wf-conn c l*)))
⟨*proof*⟩

lemma *only-c-inside-c-c'-false*:
fixes *c c'* :: $'v$ *connective* **and** *l* :: $'v$ *propo list* **and** φ :: $'v$ *propo*
assumes *cc'*: $c \neq c'$ **and** *c*: c = *CAnd* $\lor$ c = *COr* **and** *c'*: c' = *CAnd* $\lor$ c' = *COr*
and *only*: *only-c-inside c* φ **and** *incl*: *conn c' l* $\preceq \varphi$ **and** *wf*: *wf-conn c' l*
shows *False*
⟨*proof*⟩

lemma *only-c-inside-implies-c-in-c'-symb*:
assumes δ: $c \neq c'$ **and** *c*: c = *CAnd* $\lor$ c = *COr* **and** *c'*: c' = *CAnd* $\lor$ c' = *COr*
shows *only-c-inside c* $\varphi \Longrightarrow$ *c-in-c'-symb c c'* φ
⟨*proof*⟩

lemma *c-in-c'-symb-decomp-level1*:
fixes *l* :: $'v$ *propo list* **and** *c c' ca* :: $'v$ *connective*
shows *wf-conn ca l* $\Longrightarrow$ $ca \neq c \Longrightarrow$ *c-in-c'-symb c c'* (*conn ca l*)
⟨*proof*⟩

lemma *only-c-inside-implies-c-in-c'-only*:
  assumes δ: $c \neq c'$ **and** c: $c = CAnd \lor c = COr$ **and** c': $c' = CAnd \lor c' = COr$
  shows *only-c-inside* $c\ \varphi \Longrightarrow$ *c-in-c'-only* $c\ c'\ \varphi$
  ⟨*proof*⟩

lemma *c-in-c'-symb-c-implies-only-c-inside*:
  assumes δ: $c = CAnd \lor c = COr$ $c' = CAnd \lor c' = COr$ $c \neq c'$ **and** *wf*: *wf-conn* c $[\varphi, \psi]$
  and *inv*: *no-equiv* (*conn* c l) *no-imp* (*conn* c l) *simple-not* (*conn* c l)
  shows *wf-conn* $c\ l \Longrightarrow$ *c-in-c'-only* $c\ c'$ (*conn* c l) $\Longrightarrow$ $(\forall \psi \in$ *set* l. *only-c-inside* $c\ \psi)$
⟨*proof*⟩

8.5.2 Push Conjunction

definition *pushConj* **where** *pushConj* = *push-conn-inside* *CAnd* *COr*

lemma *pushConj-consistent*: *preserves-un-sat* *pushConj*
  ⟨*proof*⟩

definition *and-in-or-symb* **where** *and-in-or-symb* = *c-in-c'-symb* *CAnd* *COr*

definition *and-in-or-only* **where**
and-in-or-only = *all-subformula-st* (*c-in-c'-symb* *CAnd* *COr*)

lemma *pushConj-inv*:
  fixes $\varphi\ \psi$ :: *'v propo*
  assumes *full* (*propo-rew-step* *pushConj*) $\varphi\ \psi$
  and *no-equiv* φ **and** *no-imp* φ **and** *no-T-F-except-top-level* φ **and** *simple-not* φ
  shows *no-equiv* ψ **and** *no-imp* ψ **and** *no-T-F-except-top-level* ψ **and** *simple-not* ψ
  ⟨*proof*⟩

lemma *pushConj-full-propo-rew-step*:
  fixes $\varphi\ \psi$ :: *'v propo*
  assumes
    no-equiv φ **and**
    no-imp φ **and**
    full (*propo-rew-step* *pushConj*) $\varphi\ \psi$ **and**
    no-T-F-except-top-level φ **and**
    simple-not φ
  shows *and-in-or-only* ψ
  ⟨*proof*⟩

8.5.3 Push Disjunction

definition *pushDisj* **where** *pushDisj* = *push-conn-inside* *COr* *CAnd*

lemma *pushDisj-consistent*: *preserves-un-sat* *pushDisj*
  ⟨*proof*⟩

definition *or-in-and-symb* **where** *or-in-and-symb* = *c-in-c'-symb* *COr* *CAnd*

definition *or-in-and-only* **where**
or-in-and-only = *all-subformula-st* (*c-in-c'-symb* *COr* *CAnd*)

lemma *not-or-in-and-only-or-and*[*simp*]:
$\sim$*or-in-and-only* (*FOr* (*FAnd* $\psi 1$ $\psi 2$) φ')
⟨*proof*⟩

lemma *pushDisj-inv*:
fixes φ ψ :: $'v$ *propo*
assumes *full* (*propo-rew-step pushDisj*) φ ψ
and *no-equiv* φ **and** *no-imp* φ **and** *no-T-F-except-top-level* φ **and** *simple-not* φ
shows *no-equiv* ψ **and** *no-imp* ψ **and** *no-T-F-except-top-level* ψ **and** *simple-not* ψ
⟨*proof*⟩

lemma *pushDisj-full-propo-rew-step*:
fixes φ ψ :: $'v$ *propo*
assumes
no-equiv φ **and**
no-imp φ **and**
full (*propo-rew-step pushDisj*) φ ψ **and**
no-T-F-except-top-level φ **and**
simple-not φ
shows *or-in-and-only* ψ
⟨*proof*⟩

9 The full transformations

9.1 Abstract Property characterizing that only some connective are inside the others

9.1.1 Definition

The normal is a super group of groups

inductive *grouped-by* :: $'a$ *connective* $\Rightarrow$ $'a$ *propo* $\Rightarrow$ *bool* **for** c **where**
simple-is-grouped[*simp*]: *simple* $\varphi \Longrightarrow$ *grouped-by* c φ |
simple-not-is-grouped[*simp*]: *simple* $\varphi \Longrightarrow$ *grouped-by* c (*FNot* φ) |
connected-is-group[*simp*]: *grouped-by* c $\varphi \Longrightarrow$ *grouped-by* c $\psi \Longrightarrow$ *wf-conn* c [φ, ψ]
$\Longrightarrow$ *grouped-by* c (*conn* c [φ, ψ])

lemma *simple-clause*[*simp*]:
grouped-by c *FT*
grouped-by c *FF*
grouped-by c (*FVar* x)
grouped-by c (*FNot FT*)
grouped-by c (*FNot FF*)
grouped-by c (*FNot* (*FVar* x))
⟨*proof*⟩

lemma *only-c-inside-symb-c-eq-c'*:
only-c-inside-symb c (*conn* c' [$\varphi 1$, $\varphi 2$]) $\Longrightarrow$ $c' = CAnd \vee c' = COr \Longrightarrow$ *wf-conn* c' [$\varphi 1$, $\varphi 2$]
$\Longrightarrow c' = c$
⟨*proof*⟩

lemma *only-c-inside-c-eq-c'*:
only-c-inside c (*conn* c' [$\varphi 1$, $\varphi 2$]) $\Longrightarrow$ $c' = CAnd \vee c' = COr \Longrightarrow$ *wf-conn* c' [$\varphi 1$, $\varphi 2$] $\Longrightarrow c = c'$
⟨*proof*⟩

lemma *only-c-inside-imp-grouped-by*:
assumes c: $c \neq CNot$ **and** c': $c' = CAnd \vee c' = COr$
shows $only\text{-}c\text{-}inside\ c\ \varphi \Longrightarrow grouped\text{-}by\ c\ \varphi$ (**is** $?O\ \varphi \Longrightarrow ?G\ \varphi$)
⟨*proof*⟩

lemma *grouped-by-false*:
$grouped\text{-}by\ c\ (conn\ c'\ [\varphi, \psi]) \Longrightarrow c \neq c' \Longrightarrow wf\text{-}conn\ c'\ [\varphi, \psi] \Longrightarrow False$
⟨*proof*⟩

Then the CNF form is a conjunction of clauses: every clause is in CNF form and two formulas in CNF form can be related by an and.

inductive *super-grouped-by*:: $'a\ connective \Rightarrow\ 'a\ connective \Rightarrow\ 'a\ propo \Rightarrow bool$ **for** $c\ c'$ **where**
grouped-is-super-grouped[*simp*]: $grouped\text{-}by\ c\ \varphi \Longrightarrow super\text{-}grouped\text{-}by\ c\ c'\ \varphi$ |
connected-is-super-group: $super\text{-}grouped\text{-}by\ c\ c'\ \varphi \Longrightarrow super\text{-}grouped\text{-}by\ c\ c'\ \psi \Longrightarrow wf\text{-}conn\ c\ [\varphi, \psi]$
$\Longrightarrow super\text{-}grouped\text{-}by\ c\ c'\ (conn\ c'\ [\varphi, \psi])$

lemma *simple-cnf*[*simp*]:
$super\text{-}grouped\text{-}by\ c\ c'\ FT$
$super\text{-}grouped\text{-}by\ c\ c'\ FF$
$super\text{-}grouped\text{-}by\ c\ c'\ (FVar\ x)$
$super\text{-}grouped\text{-}by\ c\ c'\ (FNot\ FT)$
$super\text{-}grouped\text{-}by\ c\ c'\ (FNot\ FF)$
$super\text{-}grouped\text{-}by\ c\ c'\ (FNot\ (FVar\ x))$
⟨*proof*⟩

lemma *c-in-c'-only-super-grouped-by*:
assumes c: $c = CAnd \vee c = COr$ **and** c': $c' = CAnd \vee c' = COr$ **and** cc': $c \neq c'$
shows $no\text{-}equiv\ \varphi \Longrightarrow no\text{-}imp\ \varphi \Longrightarrow simple\text{-}not\ \varphi \Longrightarrow c\text{-}in\text{-}c'\text{-}only\ c\ c'\ \varphi$
$\Longrightarrow super\text{-}grouped\text{-}by\ c\ c'\ \varphi$
(**is** $?NE\ \varphi \Longrightarrow ?NI\ \varphi \Longrightarrow ?SN\ \varphi \Longrightarrow ?C\ \varphi \Longrightarrow ?S\ \varphi$)
⟨*proof*⟩

9.2 Conjunctive Normal Form

definition *is-conj-with-TF* **where** $is\text{-}conj\text{-}with\text{-}TF == super\text{-}grouped\text{-}by\ COr\ CAnd$

lemma *or-in-and-only-conjunction-in-disj*:
shows $no\text{-}equiv\ \varphi \Longrightarrow no\text{-}imp\ \varphi \Longrightarrow simple\text{-}not\ \varphi \Longrightarrow or\text{-}in\text{-}and\text{-}only\ \varphi \Longrightarrow is\text{-}conj\text{-}with\text{-}TF\ \varphi$
⟨*proof*⟩

definition *is-cnf* **where**
$is\text{-}cnf\ \varphi \equiv is\text{-}conj\text{-}with\text{-}TF\ \varphi \wedge no\text{-}T\text{-}F\text{-}except\text{-}top\text{-}level\ \varphi$

9.2.1 Full CNF transformation

The full1 CNF transformation consists simply in chaining all the transformation defined before.

definition *cnf-rew* **where** *cnf-rew* =
$(full\ (propo\text{-}rew\text{-}step\ elim\text{-}equiv))\ OO$
$(full\ (propo\text{-}rew\text{-}step\ elim\text{-}imp))\ OO$
$(full\ (propo\text{-}rew\text{-}step\ elimTB))\ OO$
$(full\ (propo\text{-}rew\text{-}step\ pushNeg))\ OO$
$(full\ (propo\text{-}rew\text{-}step\ pushDisj))$

lemma *cnf-rew-consistent*: $preserves\text{-}un\text{-}sat\ cnf\text{-}rew$

⟨*proof*⟩

lemma *cnf-rew-is-cnf*: *cnf-rew* φ φ' $\Longrightarrow$ *is-cnf* φ'
⟨*proof*⟩

9.3 Disjunctive Normal Form

definition *is-disj-with-TF* **where** *is-disj-with-TF* $\equiv$ *super-grouped-by CAnd COr*

lemma *and-in-or-only-conjunction-in-disj*:
shows *no-equiv* φ $\Longrightarrow$ *no-imp* φ $\Longrightarrow$ *simple-not* φ $\Longrightarrow$ *and-in-or-only* φ $\Longrightarrow$ *is-disj-with-TF* φ
⟨*proof*⟩

definition *is-dnf* :: *'a propo* $\Rightarrow$ *bool* **where**
is-dnf φ $\longleftrightarrow$ *is-disj-with-TF* φ $\wedge$ *no-T-F-except-top-level* φ

9.3.1 Full DNF transform

The full1 DNF transformation consists simply in chaining all the transformation defined before.

definition *dnf-rew* **where** *dnf-rew* $\equiv$
(*full* (*propo-rew-step elim-equiv*)) *OO*
(*full* (*propo-rew-step elim-imp*)) *OO*
(*full* (*propo-rew-step elimTB*)) *OO*
(*full* (*propo-rew-step pushNeg*)) *OO*
(*full* (*propo-rew-step pushConj*))

lemma *dnf-rew-consistent*: *preserves-un-sat dnf-rew*
⟨*proof*⟩

theorem *dnf-transformation-correction*:
dnf-rew φ φ' $\Longrightarrow$ *is-dnf* φ'
⟨*proof*⟩

10 More aggressive simplifications: Removing true and false at the beginning

10.1 Transformation

We should remove *FT* and *FF* at the beginning and not in the middle of the algorithm. To do this, we have to use more rules (one for each connective):

inductive *elimTBFull* **where**
ElimTBFull1 [*simp*]: *elimTBFull* (*FAnd* φ *FT*) φ |
ElimTBFull1' [*simp*]: *elimTBFull* (*FAnd FT* φ) φ |

ElimTBFull2 [*simp*]: *elimTBFull* (*FAnd* φ *FF*) *FF* |
ElimTBFull2' [*simp*]: *elimTBFull* (*FAnd FF* φ) *FF* |

ElimTBFull3 [*simp*]: *elimTBFull* (*FOr* φ *FT*) *FT* |
ElimTBFull3' [*simp*]: *elimTBFull* (*FOr FT* φ) *FT* |

ElimTBFull4 [*simp*]: *elimTBFull* (*FOr* φ *FF*) φ |
ElimTBFull4' [*simp*]: *elimTBFull* (*FOr FF* φ) φ |

ElimTBFull5 [*simp*]: *elimTBFull* (*FNot FT*) *FF* |
ElimTBFull5'[*simp*]: *elimTBFull* (*FNot FF*) *FT* |

ElimTBFull6-l[*simp*]: *elimTBFull* (*FImp FT* φ) φ |
ElimTBFull6-l'[*simp*]: *elimTBFull* (*FImp FF* φ) *FT* |
ElimTBFull6-r[*simp*]: *elimTBFull* (*FImp* φ *FT*) *FT* |
ElimTBFull6-r'[*simp*]: *elimTBFull* (*FImp* φ *FF*) (*FNot* φ) |

ElimTBFull7-l[*simp*]: *elimTBFull* (*FEq FT* φ) φ |
ElimTBFull7-l'[*simp*]: *elimTBFull* (*FEq FF* φ) (*FNot* φ) |
ElimTBFull7-r[*simp*]: *elimTBFull* (*FEq* φ *FT*) φ |
ElimTBFull7-r'[*simp*]: *elimTBFull* (*FEq* φ *FF*) (*FNot* φ)

The transformation is still consistent.

lemma *elimTBFull-consistent*: *preserves-un-sat elimTBFull*
⟨*proof*⟩

Contrary to the theorem ⟦*no-equiv* ?φ; *no-imp* ?φ; ?$\psi \preceq$?φ; $\neg$ *no-T-F-symb-except-toplevel* ?ψ⟧ $\Longrightarrow \exists \psi'$. *elimTB* ?$\psi$ ψ', we do not need the assumption *no-equiv* φ and *no-imp* φ, since our transformation is more general.

lemma *no-T-F-symb-except-toplevel-step-exists'*:
fixes φ :: *'v propo*
shows $\psi \preceq \varphi \Longrightarrow \neg$ *no-T-F-symb-except-toplevel* $\psi \Longrightarrow \exists \psi'$. *elimTBFull* ψ ψ'
⟨*proof*⟩

The same applies here. We do not need the assumption, but the deep link between $\neg$ *no-T-F-except-top-level* φ and the existence of a rewriting step, still exists.

lemma *no-T-F-except-top-level-rew'*:
fixes φ :: *'v propo*
assumes *noTB*: $\neg$ *no-T-F-except-top-level* φ
shows $\exists \psi\ \psi'$. $\psi \preceq \varphi \wedge$ *elimTBFull* ψ ψ'
⟨*proof*⟩

lemma *elimTBFull-full-propo-rew-step*:
fixes φ ψ :: *'v propo*
assumes *full* (*propo-rew-step elimTBFull*) φ ψ
shows *no-T-F-except-top-level* ψ
⟨*proof*⟩

10.2 More invariants

As the aim is to use the transformation as the first transformation, we have to show some more invariants for *elim-equiv* and *elim-imp*. For the other transformation, we have already proven it.

lemma *propo-rew-step-ElimEquiv-no-T-F*: *propo-rew-step elim-equiv* φ $\psi \Longrightarrow$ *no-T-F* $\varphi \Longrightarrow$ *no-T-F* ψ
⟨*proof*⟩

lemma *elim-equiv-inv'*:
fixes φ ψ :: *'v propo*
assumes *full* (*propo-rew-step elim-equiv*) φ ψ **and** *no-T-F-except-top-level* φ
shows *no-T-F-except-top-level* ψ

$\langle proof \rangle$

lemma *propo-rew-step-ElimImp-no-T-F*: *propo-rew-step elim-imp* φ ψ $\Longrightarrow$ *no-T-F* φ $\Longrightarrow$ *no-T-F* ψ
$\langle proof \rangle$

lemma *elim-imp-inv'*:
fixes φ ψ :: *'v propo*
assumes *full* (*propo-rew-step elim-imp*) φ ψ **and** *no-T-F-except-top-level* φ
shows*no-T-F-except-top-level* ψ
$\langle proof \rangle$

10.3 The new CNF and DNF transformation

The transformation is the same as before, but the order is not the same.

definition *dnf-rew'* :: *'a propo* $\Rightarrow$ *'a propo* $\Rightarrow$ *bool* **where**
dnf-rew' =
(*full* (*propo-rew-step elimTBFull*)) *OO*
(*full* (*propo-rew-step elim-equiv*)) *OO*
(*full* (*propo-rew-step elim-imp*)) *OO*
(*full* (*propo-rew-step pushNeg*)) *OO*
(*full* (*propo-rew-step pushConj*))

lemma *dnf-rew'-consistent*: *preserves-un-sat dnf-rew'*
$\langle proof \rangle$

theorem *cnf-transformation-correction*:
dnf-rew' φ φ' $\Longrightarrow$ *is-dnf* φ'
$\langle proof \rangle$

Given all the lemmas before the CNF transformation is easy to prove:

definition *cnf-rew'* :: *'a propo* $\Rightarrow$ *'a propo* $\Rightarrow$ *bool* **where**
cnf-rew' =
(*full* (*propo-rew-step elimTBFull*)) *OO*
(*full* (*propo-rew-step elim-equiv*)) *OO*
(*full* (*propo-rew-step elim-imp*)) *OO*
(*full* (*propo-rew-step pushNeg*)) *OO*
(*full* (*propo-rew-step pushDisj*))

lemma *cnf-rew'-consistent*: *preserves-un-sat cnf-rew'*
$\langle proof \rangle$

theorem *cnf'-transformation-correction*:
cnf-rew' φ φ' $\Longrightarrow$ *is-cnf* φ'
$\langle proof \rangle$

end

11 Partial Clausal Logic

theory *Partial-Clausal-Logic*
imports *../lib/Clausal-Logic List-More*
begin

11.1 Clauses

Clauses are (finite) multisets of literals.

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

11.2 Partial Interpretations

type-synonym *'a interp* = *'a literal set*

definition *true-lit* :: *'a interp* ⇒ *'a literal* ⇒ *bool* (**infix** $\models l$ *50*) **where**
$I \models l\ L \longleftrightarrow L \in I$

declare *true-lit-def*[*simp*]

11.2.1 Consistency

definition *consistent-interp* :: *'a literal set* ⇒ *bool* **where**
consistent-interp $I = (\forall L.\ \neg(L \in I \wedge -\ L \in I))$

lemma *consistent-interp-empty*[*simp*]:
consistent-interp {} ⟨*proof*⟩

lemma *consistent-interp-single*[*simp*]:
consistent-interp $\{L\}$ ⟨*proof*⟩

lemma *consistent-interp-subset*:
assumes
$A \subseteq B$ **and**
consistent-interp B
shows *consistent-interp* A
⟨*proof*⟩

lemma *consistent-interp-change-insert*:
$a \notin A \Longrightarrow -a \notin A \Longrightarrow$ *consistent-interp* (*insert* $(-a)$ A) $\longleftrightarrow$ *consistent-interp* (*insert* a A)
⟨*proof*⟩

lemma *consistent-interp-insert-pos*[*simp*]:
$a \notin A \Longrightarrow$ *consistent-interp* (*insert* a A) $\longleftrightarrow$ *consistent-interp* $A \wedge -a \notin A$
⟨*proof*⟩

lemma *consistent-interp-insert-not-in*:
consistent-interp $A \Longrightarrow a \notin A \Longrightarrow -a \notin A \Longrightarrow$ *consistent-interp* (*insert* a A)
⟨*proof*⟩

11.2.2 Atoms

definition *atms-of-ms* :: *'a literal multiset set* ⇒ *'a set* **where**
atms-of-ms $\psi s = \bigcup($*atms-of* ‘ $\psi s)$

lemma *atms-of-mmltiset*[*simp*]:
atms-of (*mset* a) = *atm-of* ‘ *set* a
⟨*proof*⟩

lemma *atms-of-ms-mset-unfold*:
atms-of-ms (*mset* ‘ b) = $(\bigcup x{\in}b.$ *atm-of* ‘ *set* $x)$

$\langle proof \rangle$

definition *atms-of-s* :: *'a literal set* $\Rightarrow$ *'a set* **where**
atms-of-s C = *atm-of* ' *C*

lemma *atms-of-ms-emtpy-set*[*simp*]:
atms-of-ms $\{\}$ = $\{\}$
$\langle proof \rangle$

lemma *atms-of-ms-memtpy*[*simp*]:
atms-of-ms $\{\{\#\}\}$ = $\{\}$
$\langle proof \rangle$

lemma *atms-of-ms-mono*:
$A \subseteq B \Longrightarrow$ *atms-of-ms* $A \subseteq$ *atms-of-ms* B
$\langle proof \rangle$

lemma *atms-of-ms-finite*[*simp*]:
finite $\psi s \Longrightarrow$ *finite* (*atms-of-ms* ψs)
$\langle proof \rangle$

lemma *atms-of-ms-union*[*simp*]:
atms-of-ms $(\psi s \cup \chi s)$ = *atms-of-ms* $\psi s \cup$ *atms-of-ms* χs
$\langle proof \rangle$

lemma *atms-of-ms-insert*[*simp*]:
atms-of-ms (*insert* ψs χs) = *atms-of* $\psi s \cup$ *atms-of-ms* χs
$\langle proof \rangle$

lemma *atms-of-ms-singleton*[*simp*]: *atms-of-ms* $\{L\}$ = *atms-of* L
$\langle proof \rangle$

lemma *atms-of-atms-of-ms-mono*[*simp*]:
$A \in \psi \Longrightarrow$ *atms-of* $A \subseteq$ *atms-of-ms* ψ
$\langle proof \rangle$

lemma *atms-of-ms-single-set-mset-atns-of*[*simp*]:
atms-of-ms (*single* ' *set-mset* B) = *atms-of* B
$\langle proof \rangle$

lemma *atms-of-ms-remove-incl*:
shows *atms-of-ms* (*Set.remove a* ψ) $\subseteq$ *atms-of-ms* ψ
$\langle proof \rangle$

lemma *atms-of-ms-remove-subset*:
atms-of-ms $(\varphi - \psi) \subseteq$ *atms-of-ms* φ
$\langle proof \rangle$

lemma *finite-atms-of-ms-remove-subset*[*simp*]:
finite (*atms-of-ms* A) $\Longrightarrow$ *finite* (*atms-of-ms* $(A - C)$)
$\langle proof \rangle$

lemma *atms-of-ms-empty-iff*:
atms-of-ms $A = \{\} \longleftrightarrow A = \{\{\#\}\} \vee A = \{\}$
$\langle proof \rangle$

lemma *in-implies-atm-of-on-atms-of-ms*:
  assumes $L \in\# C$ **and** $C \in N$
  shows *atm-of* $L \in$ *atms-of-ms* N
  ⟨*proof*⟩

lemma *in-plus-implies-atm-of-on-atms-of-ms*:
  assumes $C+\{\#L\#\} \in N$
  shows *atm-of* $L \in$ *atms-of-ms* N
  ⟨*proof*⟩

lemma *in-m-in-literals*:
  assumes $\{\#A\#\} + D \in \psi s$
  shows *atm-of* $A \in$ *atms-of-ms* ψs
  ⟨*proof*⟩

lemma *atms-of-s-union*[*simp*]:
  atms-of-s $(Ia \cup Ib) =$ *atms-of-s* $Ia \cup$ *atms-of-s* Ib
  ⟨*proof*⟩

lemma *atms-of-s-single*[*simp*]:
  atms-of-s $\{L\} = \{$*atm-of* $L\}$
  ⟨*proof*⟩

lemma *atms-of-s-insert*[*simp*]:
  atms-of-s (*insert* L Ib) $= \{$*atm-of* $L\} \cup$ *atms-of-s* Ib
  ⟨*proof*⟩

lemma *in-atms-of-s-decomp*[*iff*]:
  $P \in$ *atms-of-s* $I \longleftrightarrow$ (*Pos* $P \in I \lor$ *Neg* $P \in I$) (**is** *?P* $\longleftrightarrow$ *?Q*)
⟨*proof*⟩

lemma *atm-of-in-atm-of-set-in-uminus*:
  atm-of $L' \in$ *atm-of* ‘ $B \Longrightarrow L' \in B \lor - L' \in B$
  ⟨*proof*⟩

11.2.3 Totality

definition *total-over-set* :: *'a interp* ⇒ *'a set* ⇒ *bool* **where**
total-over-set I $S = (\forall l \in S.$ *Pos* $l \in I \lor$ *Neg* $l \in I)$

definition *total-over-m* :: *'a literal set* ⇒ *'a clause set* ⇒ *bool* **where**
total-over-m I $\psi s =$ *total-over-set* I (*atms-of-ms* ψs)

lemma *total-over-set-empty*[*simp*]:
  total-over-set I $\{\}$
  ⟨*proof*⟩

lemma *total-over-m-empty*[*simp*]:
  total-over-m I $\{\}$
  ⟨*proof*⟩

lemma *total-over-set-single*[*iff*]:
  total-over-set I $\{L\} \longleftrightarrow$ (*Pos* $L \in I \lor$ *Neg* $L \in I$)
  ⟨*proof*⟩

lemma *total-over-set-insert*[*iff*]:
$\quad$ *total-over-set* I (*insert* L *Ls*) $\longleftrightarrow$ ((*Pos* $L \in I \vee$ *Neg* $L \in I$) $\wedge$ *total-over-set* I *Ls*)
$\quad\langle proof\rangle$

lemma *total-over-set-union*[*iff*]:
$\quad$ *total-over-set* I (*Ls* $\cup$ *Ls'*) $\longleftrightarrow$ (*total-over-set* I *Ls* $\wedge$ *total-over-set* I *Ls'*)
$\quad\langle proof\rangle$

lemma *total-over-m-subset*:
$\quad A \subseteq B \Longrightarrow$ *total-over-m* I $B \Longrightarrow$ *total-over-m* I A
$\quad\langle proof\rangle$

lemma *total-over-m-sum*[*iff*]:
$\quad$ **shows** *total-over-m* I $\{C + D\}$ $\longleftrightarrow$ (*total-over-m* I $\{C\}$ $\wedge$ *total-over-m* I $\{D\}$)
$\quad\langle proof\rangle$

lemma *total-over-m-union*[*iff*]:
$\quad$ *total-over-m* I ($A \cup B$) $\longleftrightarrow$ (*total-over-m* I A $\wedge$ *total-over-m* I B)
$\quad\langle proof\rangle$

lemma *total-over-m-insert*[*iff*]:
$\quad$ *total-over-m* I (*insert* a A) $\longleftrightarrow$ (*total-over-set* I (*atms-of* a) $\wedge$ *total-over-m* I A)
$\quad\langle proof\rangle$

lemma *total-over-m-extension*:
$\quad$ **fixes** I :: *'v literal set* **and** A :: *'v clauses*
$\quad$ **assumes** *total*: *total-over-m* I A
$\quad$ **shows** $\exists I'$. *total-over-m* ($I \cup I'$) ($A \cup B$)
$\quad\quad \wedge$ ($\forall x \in I'$. *atm-of* $x \in$ *atms-of-ms* B $\wedge$ *atm-of* $x \notin$ *atms-of-ms* A)
$\langle proof\rangle$

lemma *total-over-m-consistent-extension*:
$\quad$ **fixes** I :: *'v literal set* **and** A :: *'v clauses*
$\quad$ **assumes** *total*: *total-over-m* I A
$\quad$ **and** *cons*: *consistent-interp* I
$\quad$ **shows** $\exists I'$. *total-over-m* ($I \cup I'$) ($A \cup B$)
$\quad\quad \wedge$ ($\forall x \in I'$. *atm-of* $x \in$ *atms-of-ms* B $\wedge$ *atm-of* $x \notin$ *atms-of-ms* A) $\wedge$ *consistent-interp* ($I \cup I'$)
$\langle proof\rangle$

lemma *total-over-set-atms-of-m*[*simp*]:
$\quad$ *total-over-set* *Ia* (*atms-of-s* *Ia*)
$\quad\langle proof\rangle$

lemma *total-over-set-literal-defined*:
$\quad$ **assumes** $\{\#A\#\} + D \in \psi s$
$\quad$ **and** *total-over-set* I (*atms-of-ms* ψs)
$\quad$ **shows** $A \in I \vee -A \in I$
$\quad\langle proof\rangle$

lemma *tot-over-m-remove*:
$\quad$ **assumes** *total-over-m* ($I \cup \{L\}$) $\{\psi\}$
$\quad$ **and** L: $\neg$ $L \in\# \psi$ $-L \notin\# \psi$
$\quad$ **shows** *total-over-m* I $\{\psi\}$
$\quad\langle proof\rangle$

lemma *total-union*:
  assumes *total-over-m* I ψ
  shows *total-over-m* $(I \cup I')$ ψ
  ⟨*proof*⟩

lemma *total-union-2*:
  assumes *total-over-m* I ψ
  and *total-over-m* I' ψ'
  shows *total-over-m* $(I \cup I')$ $(\psi \cup \psi')$
  ⟨*proof*⟩

11.2.4 Interpretations

definition *true-cls* :: $'a$ *interp* $\Rightarrow$ $'a$ *clause* $\Rightarrow$ *bool* (**infix** $\models$ *50*) **where**
  $I \models C \longleftrightarrow (\exists L \in\# C.\ I \models l\ L)$

lemma *true-cls-empty*[*iff*]: $\neg\ I \models \{\#\}$
  ⟨*proof*⟩

lemma *true-cls-singleton*[*iff*]: $I \models \{\#L\#\} \longleftrightarrow I \models l\ L$
  ⟨*proof*⟩

lemma *true-cls-union*[*iff*]: $I \models C + D \longleftrightarrow I \models C \lor I \models D$
  ⟨*proof*⟩

lemma *true-cls-mono-set-mset*: *set-mset* $C \subseteq$ *set-mset* $D \Longrightarrow I \models C \Longrightarrow I \models D$
  ⟨*proof*⟩

lemma *true-cls-mono-leD*[*dest*]: $A \subseteq\# B \Longrightarrow I \models A \Longrightarrow I \models B$
  ⟨*proof*⟩

lemma
  assumes $I \models \psi$
  shows *true-cls-union-increase*[*simp*]: $I \cup I' \models \psi$
  and *true-cls-union-increase'*[*simp*]: $I' \cup I \models \psi$
  ⟨*proof*⟩

lemma *true-cls-mono-set-mset-l*:
  assumes $A \models \psi$
  and $A \subseteq B$
  shows $B \models \psi$
  ⟨*proof*⟩

lemma *true-cls-replicate-mset*[*iff*]: $I \models$ *replicate-mset* n $L \longleftrightarrow n \neq 0 \land I \models l\ L$
  ⟨*proof*⟩

lemma *true-cls-empty-entails*[*iff*]: $\neg\ \{\} \models N$
  ⟨*proof*⟩

lemma *true-cls-not-in-remove*:
  assumes $L \notin\# \chi$
  and $I \cup \{L\} \models \chi$
  shows $I \models \chi$
  ⟨*proof*⟩

definition *true-clss* :: $'a$ *interp* $\Rightarrow$ $'a$ *clauses* $\Rightarrow$ *bool* (**infix** $\models s$ *50*) **where**

$I \models s\ CC \longleftrightarrow (\forall\, C \in CC.\ I \models C)$

lemma *true-clss-empty*[*simp*]: $I \models s\ \{\}$
⟨*proof*⟩

lemma *true-clss-singleton*[*iff*]: $I \models s\ \{C\} \longleftrightarrow I \models C$
⟨*proof*⟩

lemma *true-clss-empty-entails-empty*[*iff*]: $\{\} \models s\ N \longleftrightarrow N = \{\}$
⟨*proof*⟩

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

lemma *true-clss-union*[*iff*]: $I \models s\ CC \cup DD \longleftrightarrow I \models s\ CC \wedge I \models s\ DD$
⟨*proof*⟩

lemma *true-clss-insert*[*iff*]: $I \models s\ insert\ C\ DD \longleftrightarrow I \models C \wedge I \models s\ DD$
⟨*proof*⟩

lemma *true-clss-mono*: $DD \subseteq CC \Longrightarrow I \models s\ CC \Longrightarrow I \models s\ DD$
⟨*proof*⟩

lemma *true-clss-union-increase*[*simp*]:
assumes $I \models s\ \psi$
shows $I \cup I' \models s\ \psi$
⟨*proof*⟩

lemma *true-clss-union-increase'*[*simp*]:
assumes $I' \models s\ \psi$
shows $I \cup I' \models s\ \psi$
⟨*proof*⟩

lemma *true-clss-commute-l*:
$(I \cup I' \models s\ \psi) \longleftrightarrow (I' \cup I \models s\ \psi)$
⟨*proof*⟩

lemma *model-remove*[*simp*]: $I \models s\ N \Longrightarrow I \models s\ Set.remove\ a\ N$
⟨*proof*⟩

lemma *model-remove-minus*[*simp*]: $I \models s\ N \Longrightarrow I \models s\ N - A$
⟨*proof*⟩

lemma *notin-vars-union-true-cls-true-cls*:
assumes $\forall\, x \in I'.\ atm\text{-}of\ x \notin atms\text{-}of\text{-}ms\ A$
and $atms\text{-}of\ L \subseteq atms\text{-}of\text{-}ms\ A$
and $I \cup I' \models L$
shows $I \models L$
⟨*proof*⟩

lemma *notin-vars-union-true-clss-true-clss*:
assumes $\forall\, x \in I'.\ atm\text{-}of\ x \notin atms\text{-}of\text{-}ms\ A$
and $atms\text{-}of\text{-}ms\ L \subseteq atms\text{-}of\text{-}ms\ A$
and $I \cup I' \models s\ L$

shows $I \models s\ L$
⟨*proof*⟩

11.2.5 Satisfiability

definition *satisfiable* :: *'a clause set* ⇒ *bool* **where**
satisfiable $CC \equiv \exists I.\ (I \models s\ CC \wedge \textit{consistent-interp}\ I \wedge \textit{total-over-m}\ I\ CC)$

lemma *satisfiable-single*[*simp*]:
satisfiable $\{\{\#L\#\}\}$
⟨*proof*⟩

abbreviation *unsatisfiable* :: *'a clause set* ⇒ *bool* **where**
unsatisfiable $CC \equiv \neg$ *satisfiable* CC

lemma *satisfiable-decreasing*:
assumes *satisfiable* $(\psi \cup \psi')$
shows *satisfiable* ψ
⟨*proof*⟩

lemma *satisfiable-def-min*:
satisfiable CC
$\longleftrightarrow (\exists I.\ I \models s\ CC \wedge \textit{consistent-interp}\ I \wedge \textit{total-over-m}\ I\ CC \wedge \textit{atm-of}`I = \textit{atms-of-ms}\ CC)$
(**is** $?sat \longleftrightarrow ?B$)
⟨*proof*⟩

11.2.6 Entailment for Multisets of Clauses

definition *true-cls-mset* :: *'a interp* ⇒ *'a clause multiset* ⇒ *bool* (**infix** $\models m$ *50*) **where**
$I \models m\ CC \longleftrightarrow (\forall C \in\#\ CC.\ I \models C)$

lemma *true-cls-mset-empty*[*simp*]: $I \models m\ \{\#\}$
⟨*proof*⟩

lemma *true-cls-mset-singleton*[*iff*]: $I \models m\ \{\#C\#\} \longleftrightarrow I \models C$
⟨*proof*⟩

lemma *true-cls-mset-union*[*iff*]: $I \models m\ CC + DD \longleftrightarrow I \models m\ CC \wedge I \models m\ DD$
⟨*proof*⟩

lemma *true-cls-mset-image-mset*[*iff*]: $I \models m\ \textit{image-mset}\ f\ A \longleftrightarrow (\forall x \in\#\ A.\ I \models f\ x)$
⟨*proof*⟩

lemma *true-cls-mset-mono*: $\textit{set-mset}\ DD \subseteq \textit{set-mset}\ CC \Longrightarrow I \models m\ CC \Longrightarrow I \models m\ DD$
⟨*proof*⟩

lemma *true-clss-set-mset*[*iff*]: $I \models s\ \textit{set-mset}\ CC \longleftrightarrow I \models m\ CC$
⟨*proof*⟩

lemma *true-cls-mset-increasing-r*[*simp*]:
$I \models m\ CC \Longrightarrow I \cup J \models m\ CC$
⟨*proof*⟩

theorem *true-cls-remove-unused*:
assumes $I \models \psi$
shows $\{v \in I.\ \textit{atm-of}\ v \in \textit{atms-of}\ \psi\} \models \psi$

$\langle proof \rangle$

theorem *true-clss-remove-unused*:
assumes $I \models s\ \psi$
shows $\{v \in I.\ atm\text{-}of\ v \in atms\text{-}of\text{-}ms\ \psi\} \models s\ \psi$
$\langle proof \rangle$

A simple application of the previous theorem:

lemma *true-clss-union-decrease*:
assumes II': $I \cup I' \models \psi$
and H: $\forall v \in I'.\ atm\text{-}of\ v \notin atms\text{-}of\ \psi$
shows $I \models \psi$
$\langle proof \rangle$

lemma *multiset-not-empty*:
assumes $M \neq \{\#\}$
and $x \in\#\ M$
shows $\exists A.\ x = Pos\ A \vee x = Neg\ A$
$\langle proof \rangle$

lemma *atms-of-ms-empty*:
fixes $\psi :: 'v\ clauses$
assumes $atms\text{-}of\text{-}ms\ \psi = \{\}$
shows $\psi = \{\} \vee \psi = \{\{\#\}\}$
$\langle proof \rangle$

lemma *consistent-interp-disjoint*:
assumes *consI*: *consistent-interp I*
and *disj*: $atms\text{-}of\text{-}s\ A \cap atms\text{-}of\text{-}s\ I = \{\}$
and *consA*: *consistent-interp A*
shows $consistent\text{-}interp\ (A \cup I)$
$\langle proof \rangle$

lemma *total-remove-unused*:
assumes *total-over-m I* ψ
shows $total\text{-}over\text{-}m\ \{v \in I.\ atm\text{-}of\ v \in atms\text{-}of\text{-}ms\ \psi\}\ \psi$
$\langle proof \rangle$

lemma *true-cls-remove-hd-if-notin-vars*:
assumes $insert\ a\ M' \models D$
and $atm\text{-}of\ a \notin atms\text{-}of\ D$
shows $M' \models D$
$\langle proof \rangle$

lemma *total-over-set-atm-of*:
fixes $I :: 'v\ interp$ **and** $K :: 'v\ set$
shows $total\text{-}over\text{-}set\ I\ K \longleftrightarrow (\forall l \in K.\ l \in (atm\text{-}of\ `\ I))$
$\langle proof \rangle$

11.2.7 Tautologies

definition $tautology\ (\psi :: 'v\ clause) \equiv \forall I.\ total\text{-}over\text{-}set\ I\ (atms\text{-}of\ \psi) \longrightarrow I \models \psi$

lemma *tautology-Pos-Neg*[*intro*]:
assumes $Pos\ p \in\#\ A$ **and** $Neg\ p \in\#\ A$
shows *tautology A*

⟨*proof*⟩

lemma *tautology-minus*[*simp*]:
assumes $L \in\# A$ **and** $-L \in\# A$
shows *tautology* A
⟨*proof*⟩

lemma *tautology-exists-Pos-Neg*:
assumes *tautology* ψ
shows $\exists p.\ Pos\ p \in\# \psi \wedge Neg\ p \in\# \psi$
⟨*proof*⟩

lemma *tautology-decomp*:
tautology $\psi \longleftrightarrow (\exists p.\ Pos\ p \in\# \psi \wedge Neg\ p \in\# \psi)$
⟨*proof*⟩

lemma *tautology-false*[*simp*]: $\neg$*tautology* $\{\#\}$
⟨*proof*⟩

lemma *tautology-add-single*:
tautology $(\{\#a\#\} + L) \longleftrightarrow$ *tautology* $L \vee -a \in\# L$
⟨*proof*⟩

lemma *minus-interp-tautology*:
assumes $\{-L \mid L.\ L\in\# \chi\} \models \chi$
shows *tautology* χ
⟨*proof*⟩

lemma *remove-literal-in-model-tautology*:
assumes $I \cup \{Pos\ P\} \models \varphi$
and $I \cup \{Neg\ P\} \models \varphi$
shows $I \models \varphi \vee$ *tautology* φ
⟨*proof*⟩

lemma *tautology-imp-tautology*:
fixes $\chi\ \chi'$ :: *'v clause*
assumes $\forall I.$ *total-over-m* $I\ \{\chi\} \longrightarrow I \models \chi \longrightarrow I \models \chi'$ **and** *tautology* χ
shows *tautology* χ' ⟨*proof*⟩

11.2.8 Entailment for clauses and propositions

definition *true-cls-cls* :: *'a clause* $\Rightarrow$ *'a clause* $\Rightarrow$ *bool* (**infix** $\models f$ *49*) **where**
$\psi \models f\ \chi \longleftrightarrow (\forall I.$ *total-over-m* $I\ (\{\psi\} \cup \{\chi\}) \longrightarrow$ *consistent-interp* $I \longrightarrow I \models \psi \longrightarrow I \models \chi)$

definition *true-cls-clss* :: *'a clause* $\Rightarrow$ *'a clauses* $\Rightarrow$ *bool* (**infix** $\models fs$ *49*) **where**
$\psi \models fs\ \chi \longleftrightarrow (\forall I.$ *total-over-m* $I\ (\{\psi\} \cup \chi) \longrightarrow$ *consistent-interp* $I \longrightarrow I \models \psi \longrightarrow I \models s\ \chi)$

definition *true-clss-cls* :: *'a clauses* $\Rightarrow$ *'a clause* $\Rightarrow$ *bool* (**infix** $\models p$ *49*) **where**
$N \models p\ \chi \longleftrightarrow (\forall I.$ *total-over-m* $I\ (N \cup \{\chi\}) \longrightarrow$ *consistent-interp* $I \longrightarrow I \models s\ N \longrightarrow I \models \chi)$

definition *true-clss-clss* :: *'a clauses* $\Rightarrow$ *'a clauses* $\Rightarrow$ *bool* (**infix** $\models ps$ *49*) **where**
$N \models ps\ N' \longleftrightarrow (\forall I.$ *total-over-m* $I\ (N \cup N') \longrightarrow$ *consistent-interp* $I \longrightarrow I \models s\ N \longrightarrow I \models s\ N')$

lemma *true-cls-cls-refl*[*simp*]:
$A \models f\ A$
⟨*proof*⟩

lemma *true-cls-cls-insert-l*[*simp*]:
$a \models f\ C \Longrightarrow insert\ a\ A \models p\ C$
⟨*proof*⟩

lemma *true-cls-clss-empty*[*iff*]:
$N \models fs\ \{\}$
⟨*proof*⟩

lemma *true-prop-true-clause*[*iff*]:
$\{\varphi\} \models p\ \psi \longleftrightarrow \varphi \models f\ \psi$
⟨*proof*⟩

lemma *true-clss-clss-true-clss-cls*[*iff*]:
$N \models ps\ \{\psi\} \longleftrightarrow N \models p\ \psi$
⟨*proof*⟩

lemma *true-clss-clss-true-cls-clss*[*iff*]:
$\{\chi\} \models ps\ \psi \longleftrightarrow \chi \models fs\ \psi$
⟨*proof*⟩

lemma *true-clss-clss-empty*[*simp*]:
$N \models ps\ \{\}$
⟨*proof*⟩

lemma *true-clss-cls-subset*:
$A \subseteq B \Longrightarrow A \models p\ CC \Longrightarrow B \models p\ CC$
⟨*proof*⟩

lemma *true-clss-cs-mono-l*[*simp*]:
$A \models p\ CC \Longrightarrow A \cup B\ \models p\ CC$
⟨*proof*⟩

lemma *true-clss-cs-mono-l2*[*simp*]:
$B \models p\ CC \Longrightarrow A \cup B\ \models p\ CC$
⟨*proof*⟩

lemma *true-clss-cls-mono-r*[*simp*]:
$A \models p\ CC \Longrightarrow A \models p\ CC + CC'$
⟨*proof*⟩

lemma *true-clss-cls-mono-r'*[*simp*]:
$A \models p\ CC' \Longrightarrow A \models p\ CC + CC'$
⟨*proof*⟩

lemma *true-clss-clss-union-l*[*simp*]:
$A \models ps\ CC \Longrightarrow A \cup B\ \models ps\ CC$
⟨*proof*⟩

lemma *true-clss-clss-union-l-r*[*simp*]:
$B \models ps\ CC \Longrightarrow A \cup B \models ps\ CC$
⟨*proof*⟩

lemma *true-clss-cls-in*[*simp*]:
$CC \in A \Longrightarrow A \models p\ CC$

$\langle proof \rangle$

lemma *true-clss-cls-insert-l*[*simp*]:
$A \models p\ C \Longrightarrow insert\ a\ A \models p\ C$
$\langle proof \rangle$

lemma *true-clss-clss-insert-l*[*simp*]:
$A \models ps\ C \Longrightarrow insert\ a\ A \models ps\ C$
$\langle proof \rangle$

lemma *true-clss-clss-union-and*[*iff*]:
$A \models ps\ C \cup D \longleftrightarrow (A \models ps\ C \wedge A \models ps\ D)$
$\langle proof \rangle$

lemma *true-clss-clss-insert*[*iff*]:
$A \models ps\ insert\ L\ Ls \longleftrightarrow (A \models p\ L \wedge A \models ps\ Ls)$
$\langle proof \rangle$

lemma *true-clss-clss-subset*:
$A \subseteq B \Longrightarrow A \models ps\ CC \Longrightarrow B \models ps\ CC$
$\langle proof \rangle$

lemma *union-trus-clss-clss*[*simp*]: $A \cup B \models ps\ B$
$\langle proof \rangle$

lemma *true-clss-clss-remove*[*simp*]:
$A \models ps\ B \Longrightarrow A \models ps\ B - C$
$\langle proof \rangle$

lemma *true-clss-clss-subsetE*:
$N \models ps\ B \Longrightarrow A \subseteq B \Longrightarrow N \models ps\ A$
$\langle proof \rangle$

lemma *true-clss-clss-in-imp-true-clss-cls*:
assumes $N \models ps\ U$
and $A \in U$
shows $N \models p\ A$
$\langle proof \rangle$

lemma *all-in-true-clss-clss*: $\forall x \in B.\ x \in A \Longrightarrow A \models ps\ B$
$\langle proof \rangle$

lemma *true-clss-clss-left-right*:
assumes $A \models ps\ B$
and $A \cup B \models ps\ M$
shows $A \models ps\ M \cup B$
$\langle proof \rangle$

lemma *true-clss-clss-generalise-true-clss-clss*:
$A \cup C \models ps\ D \Longrightarrow B \models ps\ C \Longrightarrow A \cup B \models ps\ D$
$\langle proof \rangle$

lemma *true-clss-cls-or-true-clss-cls-or-not-true-clss-cls-or*:
assumes D: $N \models p\ D + \{\#- L\#\}$
and C: $N \models p\ C + \{\#L\#\}$

shows $N \models p\ D + C$
⟨*proof*⟩

lemma *true-cls-union-mset*[*iff*]: $I \models C \ \#\cup\ D \longleftrightarrow I \models C \lor I \models D$
⟨*proof*⟩

lemma *true-clss-cls-union-mset-true-clss-cls-or-not-true-clss-cls-or*:
assumes
D: $N \models p\ D + \{\#- L\#\}$ **and**
C: $N \models p\ C + \{\#L\#\}$
shows $N \models p\ D \ \#\cup\ C$
⟨*proof*⟩

lemma *satisfiable-carac*[*iff*]:
$(\exists I.\ \textit{consistent-interp}\ I \land I \models s\ \varphi) \longleftrightarrow \textit{satisfiable}\ \varphi$ (**is** $(\exists I.\ ?Q\ I) \longleftrightarrow ?S$)
⟨*proof*⟩

lemma *satisfiable-carac'*[*simp*]: $\textit{consistent-interp}\ I \Longrightarrow I \models s\ \varphi \Longrightarrow \textit{satisfiable}\ \varphi$
⟨*proof*⟩

11.3 Subsumptions

lemma *subsumption-total-over-m*:
assumes $A \subseteq\#\ B$
shows $\textit{total-over-m}\ I\ \{B\} \Longrightarrow \textit{total-over-m}\ I\ \{A\}$
⟨*proof*⟩

lemma *atms-of-replicate-mset-replicate-mset-uminus*[*simp*]:
$\textit{atms-of}\ (D - \textit{replicate-mset}\ (\textit{count}\ D\ L)\ L\ - \textit{replicate-mset}\ (\textit{count}\ D\ (-L))\ (-L))$
$= \textit{atms-of}\ D - \{\textit{atm-of}\ L\}$
⟨*proof*⟩

lemma *subsumption-chained*:
assumes
$\forall I.\ \textit{total-over-m}\ I\ \{D\} \longrightarrow I \models D \longrightarrow I \models \varphi$ **and**
$C \subseteq\#\ D$
shows $(\forall I.\ \textit{total-over-m}\ I\ \{C\} \longrightarrow I \models C \longrightarrow I \models \varphi) \lor \textit{tautology}\ \varphi$
⟨*proof*⟩

11.4 Removing Duplicates

lemma *tautology-remdups-mset*[*iff*]:
$\textit{tautology}\ (\textit{remdups-mset}\ C) \longleftrightarrow \textit{tautology}\ C$
⟨*proof*⟩

lemma *atms-of-remdups-mset*[*simp*]: $\textit{atms-of}\ (\textit{remdups-mset}\ C) = \textit{atms-of}\ C$
⟨*proof*⟩

lemma *true-cls-remdups-mset*[*iff*]: $I \models \textit{remdups-mset}\ C \longleftrightarrow I \models C$
⟨*proof*⟩

lemma *true-clss-cls-remdups-mset*[*iff*]: $A \models p\ \textit{remdups-mset}\ C \longleftrightarrow A \models p\ C$
⟨*proof*⟩

11.5 Set of all Simple Clauses

definition *simple-clss* :: $'v$ *set* $\Rightarrow$ $'v$ *clause set* **where**
simple-clss atms = $\{C.$ *atms-of* $C \subseteq$ *atms* $\wedge$ $\neg$*tautology* C $\wedge$ *distinct-mset* $C\}$

lemma *simple-clss-empty*[*simp*]:
simple-clss {} = {{#}}
⟨*proof*⟩

lemma *simple-clss-insert*:
assumes $l \notin$ *atms*
shows *simple-clss* (*insert l atms*) =
(*op* + {#*Pos l*#}) ‘ (*simple-clss atms*)
∪ (*op* + {#*Neg l*#}) ‘ (*simple-clss atms*)
∪ *simple-clss atms*(**is** *?I* = *?U*)
⟨*proof*⟩

lemma *simple-clss-finite*:
fixes *atms* :: $'v$ *set*
assumes *finite atms*
shows *finite* (*simple-clss atms*)
⟨*proof*⟩

lemma *simple-clssE*:
assumes
$x \in$ *simple-clss atms*
shows *atms-of* $x \subseteq$ *atms* $\wedge$ $\neg$*tautology* x $\wedge$ *distinct-mset* x
⟨*proof*⟩

lemma *cls-in-simple-clss*:
shows {#} $\in$ *simple-clss s*
⟨*proof*⟩

lemma *simple-clss-card*:
fixes *atms* :: $'v$ *set*
assumes *finite atms*
shows *card* (*simple-clss atms*) $\leq$ (*3*::*nat*) ^ (*card atms*)
⟨*proof*⟩

lemma *simple-clss-mono*:
assumes *incl*: *atms* $\subseteq$ *atms'*
shows *simple-clss atms* $\subseteq$ *simple-clss atms'*
⟨*proof*⟩

lemma *distinct-mset-not-tautology-implies-in-simple-clss*:
assumes *distinct-mset* χ **and** $\neg$*tautology* χ
shows $\chi \in$ *simple-clss* (*atms-of* χ)
⟨*proof*⟩

lemma *simplified-in-simple-clss*:
assumes *distinct-mset-set* ψ **and** $\forall \chi \in \psi.$ $\neg$*tautology* χ
shows $\psi \subseteq$ *simple-clss* (*atms-of-ms* ψ)
⟨*proof*⟩

11.6 Experiment: Expressing the Entailments as Locales

locale *entail* =
  fixes *entail* :: $'a\ set \Rightarrow 'b \Rightarrow bool$ (**infix** $\models e$ *50*)
  assumes *entail-insert*[*simp*]: $I \neq \{\} \Longrightarrow insert\ L\ I \models e\ x \longleftrightarrow \{L\} \models e\ x \lor I \models e\ x$
  assumes *entail-union*[*simp*]: $I \models e\ A \Longrightarrow I \cup I' \models e\ A$
begin

definition *entails* :: $'a\ set \Rightarrow 'b\ set \Rightarrow bool$ (**infix** $\models es$ *50*) **where**
  $I \models es\ A \longleftrightarrow (\forall a \in A.\ I \models e\ a)$

lemma *entails-empty*[*simp*]:
  $I \models es\ \{\}$
  ⟨*proof*⟩

lemma *entails-single*[*iff*]:
  $I \models es\ \{a\} \longleftrightarrow I \models e\ a$
  ⟨*proof*⟩

lemma *entails-insert-l*[*simp*]:
  $M \models es\ A \Longrightarrow insert\ L\ M \models es\ A$
  ⟨*proof*⟩

lemma *entails-union*[*iff*]: $I \models es\ CC \cup DD \longleftrightarrow I \models es\ CC \land I \models es\ DD$
  ⟨*proof*⟩

lemma *entails-insert*[*iff*]: $I \models es\ insert\ C\ DD \longleftrightarrow I \models e\ C \land I \models es\ DD$
  ⟨*proof*⟩

lemma *entails-insert-mono*: $DD \subseteq CC \Longrightarrow I \models es\ CC \Longrightarrow I \models es\ DD$
  ⟨*proof*⟩

lemma *entails-union-increase*[*simp*]:
 assumes $I \models es\ \psi$
 shows $I \cup I' \models es\ \psi$
 ⟨*proof*⟩

lemma *true-clss-commute-l*:
  $(I \cup I' \models es\ \psi) \longleftrightarrow (I' \cup I \models es\ \psi)$
  ⟨*proof*⟩

lemma *entails-remove*[*simp*]: $I \models es\ N \Longrightarrow I \models es\ Set.remove\ a\ N$
  ⟨*proof*⟩

lemma *entails-remove-minus*[*simp*]: $I \models es\ N \Longrightarrow I \models es\ N - A$
  ⟨*proof*⟩

end

interpretation *true-cls*: *entail true-cls*
  ⟨*proof*⟩

11.7 Entailment to be extended

definition *true-clss-ext* :: $'a\ literal\ set \Rightarrow 'a\ literal\ multiset\ set \Rightarrow bool$ (**infix** $\models sext$ *49*)
where

$I \models sext\ N \longleftrightarrow (\forall J.\ I \subseteq J \longrightarrow consistent\text{-}interp\ J \longrightarrow total\text{-}over\text{-}m\ J\ N \longrightarrow J \models s\ N)$

lemma *true-clss-imp-true-cls-ext*:
  $I \models s\ N \Longrightarrow I \models sext\ N$
  ⟨*proof*⟩

lemma *true-clss-ext-decrease-right-remove-r*:
  assumes $I \models sext\ N$
  shows $I \models sext\ N - \{C\}$
  ⟨*proof*⟩

lemma *consistent-true-clss-ext-satisfiable*:
  assumes *consistent-interp I* **and** $I \models sext\ A$
  shows *satisfiable A*
  ⟨*proof*⟩

lemma *not-consistent-true-clss-ext*:
  assumes $\neg$*consistent-interp I*
  shows $I \models sext\ A$
  ⟨*proof*⟩
end
theory *Prop-Logic-Multiset*
imports *../lib/Multiset-More Prop-Normalisation Partial-Clausal-Logic*
begin

12 Link with Multiset Version

12.1 Transformation to Multiset

fun *mset-of-conj* :: *'a propo ⇒ 'a literal multiset* **where**
$mset\text{-}of\text{-}conj\ (FOr\ \varphi\ \psi) = mset\text{-}of\text{-}conj\ \varphi + mset\text{-}of\text{-}conj\ \psi\ |$
$mset\text{-}of\text{-}conj\ (FVar\ v) = \{\#\ Pos\ v\ \#\}\ |$
$mset\text{-}of\text{-}conj\ (FNot\ (FVar\ v)) = \{\#\ Neg\ v\ \#\}\ |$
$mset\text{-}of\text{-}conj\ FF = \{\#\}$

fun *mset-of-formula* :: *'a propo ⇒ 'a literal multiset set* **where**
$mset\text{-}of\text{-}formula\ (FAnd\ \varphi\ \psi) = mset\text{-}of\text{-}formula\ \varphi \cup mset\text{-}of\text{-}formula\ \psi\ |$
$mset\text{-}of\text{-}formula\ (FOr\ \varphi\ \psi) = \{mset\text{-}of\text{-}conj\ (FOr\ \varphi\ \psi)\}\ |$
$mset\text{-}of\text{-}formula\ (FVar\ \psi) = \{mset\text{-}of\text{-}conj\ (FVar\ \psi)\}\ |$
$mset\text{-}of\text{-}formula\ (FNot\ \psi) = \{mset\text{-}of\text{-}conj\ (FNot\ \psi)\}\ |$
$mset\text{-}of\text{-}formula\ FF = \{\{\#\}\}\ |$
$mset\text{-}of\text{-}formula\ FT = \{\}$

12.2 Equisatisfiability of the two Version

lemma *is-conj-with-TF-FNot*:
  $is\text{-}conj\text{-}with\text{-}TF\ (FNot\ \varphi) \longleftrightarrow (\exists v.\ \varphi = FVar\ v \lor \varphi = FF \lor \varphi = FT)$
  ⟨*proof*⟩

lemma *grouped-by-COr-FNot*:
  $grouped\text{-}by\ COr\ (FNot\ \varphi) \longleftrightarrow (\exists v.\ \varphi = FVar\ v \lor \varphi = FF \lor \varphi = FT)$
  ⟨*proof*⟩

lemma
  shows *no-T-F-FF*[*simp*]: $\neg$*no-T-F FF* **and**

no-T-F-FT[*simp*]: ¬*no-T-F FT*
⟨*proof*⟩

lemma *grouped-by-CAnd-FAnd*:
grouped-by CAnd (*FAnd* φ*1* φ*2*) $\longleftrightarrow$ *grouped-by CAnd* φ*1* $\wedge$ *grouped-by CAnd* φ*2*
⟨*proof*⟩

lemma *grouped-by-COr-FOr*:
grouped-by COr (*FOr* φ*1* φ*2*) $\longleftrightarrow$ *grouped-by COr* φ*1* $\wedge$ *grouped-by COr* φ*2*
⟨*proof*⟩

lemma *grouped-by-COr-FAnd*[*simp*]: ¬ *grouped-by COr* (*FAnd* φ*1* φ*2*)
⟨*proof*⟩

lemma *grouped-by-COr-FEq*[*simp*]: ¬ *grouped-by COr* (*FEq* φ*1* φ*2*)
⟨*proof*⟩

lemma [*simp*]: ¬*grouped-by COr* (*FImp* φ ψ)
⟨*proof*⟩

lemma [*simp*]: ¬ *is-conj-with-TF* (*FImp* φ ψ)
⟨*proof*⟩

lemma [*simp*]: ¬*grouped-by COr* (*FEq* φ ψ)
⟨*proof*⟩

lemma [*simp*]: ¬ *is-conj-with-TF* (*FEq* φ ψ)
⟨*proof*⟩

lemma *is-conj-with-TF-Fand*:
is-conj-with-TF (*FAnd* φ*1* φ*2*) $\Longrightarrow$ *is-conj-with-TF* φ*1* $\wedge$ *is-conj-with-TF* φ*2*
⟨*proof*⟩

lemma *is-conj-with-TF-FOr*:
is-conj-with-TF (*FOr* φ*1* φ*2*) $\Longrightarrow$ *grouped-by COr* φ*1* $\wedge$ *grouped-by COr* φ*2*
⟨*proof*⟩

lemma *grouped-by-COr-mset-of-formula*:
grouped-by COr φ $\Longrightarrow$ *mset-of-formula* φ = (*if* φ = *FT then* {} *else* {*mset-of-conj* φ})
⟨*proof*⟩

When a formula is in CNF form, then there is equisatisfiability between the multiset version and the CNF form. Remark that the definition for the entailment are slightly different: *op* $\models$ uses a function assigning *True* or *False*, while *op* $\models$*s* uses a set where being in the list means entailment of a literal.

theorem
fixes φ :: *'v propo*
assumes *is-cnf* φ
shows *eval A* φ $\longleftrightarrow$ *Partial-Clausal-Logic.true-clss* ({*Pos v*|*v*. *A v*} ∪ {*Neg v*|*v*. ¬*A v*})
(*mset-of-formula* φ)
⟨*proof*⟩

end
theory *Prop-Resolution*

imports *Partial-Clausal-Logic List-More Wellfounded-More*

begin

13 Resolution

13.1 Simplification Rules

inductive *simplify* :: $'v$ *clauses* $\Rightarrow$ $'v$ *clauses* $\Rightarrow$ *bool* **for** N :: $'v$ *clause set* **where**
tautology-deletion:
$(A + \{\#Pos\ P\#\} + \{\#Neg\ P\#\}) \in N \Longrightarrow simplify\ \ N\ (N - \{A + \{\#Pos\ P\#\} + \{\#Neg\ P\#\}\})|$
condensation:
$(A + \{\#L\#\} + \{\#L\#\}) \in N \Longrightarrow simplify\ N\ (\ N - \{A + \{\#L\#\} + \{\#L\#\}\} \cup \{A + \{\#L\#\}\})\ |$
subsumption:
$A \in N \Longrightarrow A \subset\# \ B \Longrightarrow B \in N \Longrightarrow simplify\ N\ (N - \{B\})$

lemma *simplify-preserves-un-sat$'$*:
fixes $N\ N'$ :: $'v$ *clauses*
assumes *simplify* $N\ N'$
and *total-over-m* $I\ N$
shows $I \models s\ N' \longrightarrow I \models s\ N$
⟨*proof*⟩

lemma *simplify-preserves-un-sat*:
fixes $N\ N'$ :: $'v$ *clauses*
assumes *simplify* $N\ N'$
and *total-over-m* $I\ N$
shows $I \models s\ N \longrightarrow I \models s\ N'$
⟨*proof*⟩

lemma *simplify-preserves-un-sat$''$*:
fixes $N\ N'$ :: $'v$ *clauses*
assumes *simplify* $N\ N'$
and *total-over-m* $I\ N'$
shows $I \models s\ N \longrightarrow I \models s\ N'$
⟨*proof*⟩

lemma *simplify-preserves-un-sat-eq*:
fixes $N\ N'$ :: $'v$ *clauses*
assumes *simplify* $N\ N'$
and *total-over-m* $I\ N$
shows $I \models s\ N \longleftrightarrow I \models s\ N'$
⟨*proof*⟩

lemma *simplify-preserves-finite*:
assumes *simplify* $\psi\ \psi'$
shows *finite* $\psi \longleftrightarrow$ *finite* ψ'
⟨*proof*⟩

lemma *rtranclp-simplify-preserves-finite*:
assumes *rtranclp simplify* $\psi\ \psi'$
shows *finite* $\psi \longleftrightarrow$ *finite* ψ'
⟨*proof*⟩

lemma *simplify-atms-of-ms*:

assumes *simplify* ψ ψ'
shows *atms-of-ms* $\psi' \subseteq$ *atms-of-ms* ψ
$\langle proof \rangle$

lemma *rtranclp-simplify-atms-of-ms*:
assumes *rtranclp simplify* ψ ψ'
shows *atms-of-ms* $\psi' \subseteq$ *atms-of-ms* ψ
$\langle proof \rangle$

lemma *factoring-imp-simplify*:
assumes $\{\#L\#\} + \{\#L\#\} + C \in N$
shows $\exists N'.$ *simplify* N N'
$\langle proof \rangle$

13.2 Unconstrained Resolution

type-synonym *'v uncon-state* = *'v clauses*
inductive *uncon-res* :: *'v uncon-state* $\Rightarrow$ *'v uncon-state* $\Rightarrow$ *bool* **where**
resolution:
$\{\#Pos\ p\#\} + C \in N \Longrightarrow \{\#Neg\ p\#\} + D \in N \Longrightarrow (\{\#Pos\ p\#\} + C, \{\#Neg\ p\#\} + D) \notin$ *already-used*
$\Longrightarrow$ *uncon-res* (N) $(N \cup \{C + D\})$ |
factoring: $\{\#L\#\} + \{\#L\#\} + C \in N \Longrightarrow$ *uncon-res* N $(N \cup \{C + \{\#L\#\}\})$

lemma *uncon-res-increasing*:
assumes *uncon-res* S S' **and** $\psi \in S$
shows $\psi \in S'$
$\langle proof \rangle$

lemma *rtranclp-uncon-inference-increasing*:
assumes *rtranclp uncon-res* S S' **and** $\psi \in S$
shows $\psi \in S'$
$\langle proof \rangle$

13.2.1 Subsumption

definition *subsumes* :: *'a literal multiset* $\Rightarrow$ *'a literal multiset* $\Rightarrow$ *bool* **where**
subsumes χ $\chi' \longleftrightarrow$
$(\forall I.$ *total-over-m* I $\{\chi'\} \longrightarrow$ *total-over-m* I $\{\chi\})$
$\wedge$ $(\forall I.$ *total-over-m* I $\{\chi\} \longrightarrow I \models \chi \longrightarrow I \models \chi')$

lemma *subsumes-refl*[*simp*]:
subsumes χ χ
$\langle proof \rangle$

lemma *subsumes-subsumption*:
assumes *subsumes* D χ
and $C \subset\# D$ **and** $\neg$*tautology* χ
shows *subsumes* C χ $\langle proof \rangle$

lemma *subsumes-tautology*:
assumes *subsumes* $(C + \{\#Pos\ P\#\} + \{\#Neg\ P\#\})$ χ
shows *tautology* χ
$\langle proof \rangle$

13.3 Inference Rule

type-synonym *'v state* = *'v clauses* × (*'v clause* × *'v clause*) *set*
inductive *inference-clause* :: *'v state* ⇒ *'v clause* × (*'v clause* × *'v clause*) *set* ⇒ *bool*
(**infix** $\Rightarrow_{\text{Res}}$ *100*) **where**
resolution:
{#*Pos p*#} + *C* ∈ *N* ⟹ {#*Neg p*#} + *D* ∈ *N* ⟹ ({#*Pos p*#} + *C*, {#*Neg p*#} + *D*) ∉ *already-used*
⟹ *inference-clause* (*N*, *already-used*) (*C* + *D*, *already-used* ∪ {({#*Pos p*#} + *C*, {#*Neg p*#} + *D*)}) |
factoring: {#*L*#} + {#*L*#} + *C* ∈ *N* ⟹ *inference-clause* (*N*, *already-used*) (*C* + {#*L*#}, *already-used*)

inductive *inference* :: *'v state* ⇒ *'v state* ⇒ *bool* **where**
inference-step: *inference-clause S* (*clause*, *already-used*)
⟹ *inference S* (*fst S* ∪ {*clause*}, *already-used*)

abbreviation *already-used-inv*
:: *'a literal multiset set* × (*'a literal multiset* × *'a literal multiset*) *set* ⇒ *bool* **where**
already-used-inv state ≡
(∀ (*A*, *B*) ∈ *snd state*. ∃ *p*. *Pos p* ∈# *A* ∧ *Neg p* ∈# *B* ∧
((∃ χ ∈ *fst state*. *subsumes* χ ((*A* − {#*Pos p*#}) + (*B* − {#*Neg p*#})))
∨ *tautology* ((*A* − {#*Pos p*#}) + (*B* − {#*Neg p*#}))))

lemma *inference-clause-preserves-already-used-inv*:
assumes *inference-clause S S'*
and *already-used-inv S*
shows *already-used-inv* (*fst S* ∪ {*fst S'*}, *snd S'*)
⟨*proof*⟩

lemma *inference-preserves-already-used-inv*:
assumes *inference S S'*
and *already-used-inv S*
shows *already-used-inv S'*
⟨*proof*⟩

lemma *rtranclp-inference-preserves-already-used-inv*:
assumes *rtranclp inference S S'*
and *already-used-inv S*
shows *already-used-inv S'*
⟨*proof*⟩

lemma *subsumes-condensation*:
assumes *subsumes* (*C* + {#*L*#} + {#*L*#}) *D*
shows *subsumes* (*C* + {#*L*#}) *D*
⟨*proof*⟩

lemma *simplify-preserves-already-used-inv*:
assumes *simplify N N'*
and *already-used-inv* (*N*, *already-used*)
shows *already-used-inv* (*N'*, *already-used*)
⟨*proof*⟩

lemma
factoring-satisfiable: *I* ⊨ {#*L*#} + {#*L*#} + *C* ⟷ *I* ⊨ {#*L*#} + *C* **and**

resolution-satisfiable:
consistent-interp $I \Longrightarrow I \models \{\#Pos\ p\#\} + C \Longrightarrow I \models \{\#Neg\ p\#\} + D \Longrightarrow I \models C + D$ **and**
factoring-same-vars: *atms-of* $(\{\#L\#\} + \{\#L\#\} + C) =$ *atms-of* $(\{\#L\#\} + C)$
⟨*proof*⟩

lemma *inference-increasing*:
assumes *inference* $S\ S'$ **and** $\psi \in$ *fst* S
shows $\psi \in$ *fst* S'
⟨*proof*⟩

lemma *rtranclp-inference-increasing*:
assumes *rtranclp inference* $S\ S'$ **and** $\psi \in$ *fst* S
shows $\psi \in$ *fst* S'
⟨*proof*⟩

lemma *inference-clause-already-used-increasing*:
assumes *inference-clause* $S\ S'$
shows *snd* $S \subseteq$ *snd* S'
⟨*proof*⟩

lemma *inference-already-used-increasing*:
assumes *inference* $S\ S'$
shows *snd* $S \subseteq$ *snd* S'
⟨*proof*⟩

lemma *inference-clause-preserves-un-sat*:
fixes $N\ N'$ :: *'v clauses*
assumes *inference-clause* $T\ T'$
and *total-over-m* I (*fst* T)
and *consistent*: *consistent-interp* I
shows $I \models s$ *fst* $T \longleftrightarrow I \models s$ *fst* $T \cup \{$*fst* $T'\}$
⟨*proof*⟩

lemma *inference-preserves-un-sat*:
fixes $N\ N'$ :: *'v clauses*
assumes *inference* $T\ T'$
and *total-over-m* I (*fst* T)
and *consistent*: *consistent-interp* I
shows $I \models s$ *fst* $T \longleftrightarrow I \models s$ *fst* T'
⟨*proof*⟩

lemma *inference-clause-preserves-atms-of-ms*:
assumes *inference-clause* $S\ S'$
shows *atms-of-ms* (*fst* (*fst* $S \cup \{$*fst* $S'\}$, *snd* S')) $\subseteq$ *atms-of-ms* (*fst* S)
⟨*proof*⟩

lemma *inference-preserves-atms-of-ms*:
fixes $N\ N'$ :: *'v clauses*
assumes *inference* $T\ T'$
shows *atms-of-ms* (*fst* T') $\subseteq$ *atms-of-ms* (*fst* T)
⟨*proof*⟩

lemma *inference-preserves-total*:

fixes N N' :: $'v$ *clauses*
assumes *inference* $(N,$ *already-used*$)$ $(N',$ *already-used*$')$
shows *total-over-m* I $N \Longrightarrow$ *total-over-m* I N'
⟨*proof*⟩

lemma *rtranclp-inference-preserves-total*:
assumes *rtranclp inference* T T'
shows *total-over-m* I (*fst* T) $\Longrightarrow$ *total-over-m* I (*fst* T')
⟨*proof*⟩

lemma *rtranclp-inference-preserves-un-sat*:
assumes *rtranclp inference* N N'
and *total-over-m* I (*fst* N)
and *consistent*: *consistent-interp* I
shows $I \models s$ *fst* $N \longleftrightarrow I \models s$ *fst* N'
⟨*proof*⟩

lemma *inference-preserves-finite*:
assumes *inference* ψ ψ' **and** *finite* (*fst* ψ)
shows *finite* (*fst* ψ')
⟨*proof*⟩

lemma *inference-clause-preserves-finite-snd*:
assumes *inference-clause* ψ ψ' **and** *finite* (*snd* ψ)
shows *finite* (*snd* ψ')
⟨*proof*⟩

lemma *inference-preserves-finite-snd*:
assumes *inference* ψ ψ' **and** *finite* (*snd* ψ)
shows *finite* (*snd* ψ')
⟨*proof*⟩

lemma *rtranclp-inference-preserves-finite*:
assumes *rtranclp inference* ψ ψ' **and** *finite* (*fst* ψ)
shows *finite* (*fst* ψ')
⟨*proof*⟩

lemma *consistent-interp-insert*:
assumes *consistent-interp* I
and *atm-of* $P \notin$ *atm-of* ‘ I
shows *consistent-interp* (*insert* P I)
⟨*proof*⟩

lemma *simplify-clause-preserves-sat*:
assumes *simp*: *simplify* ψ ψ'
and *satisfiable* ψ'
shows *satisfiable* ψ
⟨*proof*⟩

lemma *simplify-preserves-unsat*:
assumes *inference* ψ ψ'

shows *satisfiable* $(fst\ \psi') \longrightarrow$ *satisfiable* $(fst\ \psi)$
⟨*proof*⟩

lemma *inference-preserves-unsat*:
assumes $inference^{**}\ S\ S'$
shows *satisfiable* $(fst\ S') \longrightarrow$ *satisfiable* $(fst\ S)$
⟨*proof*⟩

datatype $'v$ *sem-tree* = *Node* $'v$ $'v$ *sem-tree* $'v$ *sem-tree* | *Leaf*

fun *sem-tree-size* :: $'v$ *sem-tree* $\Rightarrow$ *nat* **where**
sem-tree-size Leaf = *0* |
sem-tree-size (*Node* - *ag ad*) = *1* + *sem-tree-size ag* + *sem-tree-size ad*

lemma *sem-tree-size*[*case-names bigger*]:
$(\bigwedge xs::\ 'v\ sem\text{-}tree.\ (\bigwedge ys::\ 'v\ sem\text{-}tree.\ sem\text{-}tree\text{-}size\ ys < sem\text{-}tree\text{-}size\ xs \Longrightarrow P\ ys) \Longrightarrow P\ xs)$
$\Longrightarrow P\ xs$
⟨*proof*⟩

fun *partial-interps* :: $'v$ *sem-tree* $\Rightarrow$ $'v$ *interp* $\Rightarrow$ $'v$ *clauses* $\Rightarrow$ *bool* **where**
partial-interps Leaf I $\psi = (\exists \chi.\ \neg\ I \models \chi \wedge \chi \in \psi \wedge$ *total-over-m I* $\{\chi\})$ |
partial-interps (*Node v ag ad*) *I* $\psi \longleftrightarrow$
(*partial-interps ag* $(I \cup \{Pos\ v\})\ \psi \wedge$ *partial-interps ad* $(I \cup \{Neg\ v\})\ \psi)$

lemma *simplify-preserve-partial-leaf*:
simplify $N\ N' \Longrightarrow$ *partial-interps Leaf I N* $\Longrightarrow$ *partial-interps Leaf I* N'
⟨*proof*⟩

lemma *simplify-preserve-partial-tree*:
assumes *simplify* $N\ N'$
and *partial-interps t I N*
shows *partial-interps t I* N'
⟨*proof*⟩

lemma *inference-preserve-partial-tree*:
assumes *inference* $S\ S'$
and *partial-interps t I* (*fst S*)
shows *partial-interps t I* $(fst\ S')$
⟨*proof*⟩

lemma *rtranclp-inference-preserve-partial-tree*:
assumes *rtranclp inference* $N\ N'$
and *partial-interps t I* (*fst N*)
shows *partial-interps t I* $(fst\ N')$
⟨*proof*⟩

function *build-sem-tree* :: $'v$ :: *linorder set* $\Rightarrow$ $'v$ *clauses* $\Rightarrow$ $'v$ *sem-tree* **where**
build-sem-tree atms ψ =
(*if atms* = $\{\} \vee \neg$ *finite atms*

then Leaf
else Node (*Min atms*) (*build-sem-tree* (*Set.remove* (*Min atms*) *atms*) ψ)
(*build-sem-tree* (*Set.remove* (*Min atms*) *atms*) ψ))
⟨*proof*⟩
termination
⟨*proof*⟩
declare *build-sem-tree.induct*[*case-names tree*]

lemma *unsatisfiable-empty*[*simp*]:
$\neg$*unsatisfiable* {}
⟨*proof*⟩

lemma *partial-interps-build-sem-tree-atms-general*:
fixes ψ :: *'v* :: *linorder clauses* **and** *p* :: *'v literal list*
assumes *unsat*: *unsatisfiable* ψ **and** *finite* ψ **and** *consistent-interp I*
and *finite atms*
and *atms-of-ms* ψ = *atms* $\cup$ *atms-of-s I* **and** *atms* $\cap$ *atms-of-s I* = {}
shows *partial-interps* (*build-sem-tree atms* ψ) *I* ψ
⟨*proof*⟩

lemma *partial-interps-build-sem-tree-atms*:
fixes ψ :: *'v* :: *linorder clauses* **and** *p* :: *'v literal list*
assumes *unsat*: *unsatisfiable* ψ **and** *finite*: *finite* ψ
shows *partial-interps* (*build-sem-tree* (*atms-of-ms* ψ) ψ) {} ψ
⟨*proof*⟩

lemma *can-decrease-count*:
fixes ψ'' :: *'v clauses* $\times$ (*'v clause* $\times$ *'v clause* $\times$ *'v*) *set*
assumes *count* χ *L* = *n*
and *L* $\in\#$ χ **and** $\chi \in$ *fst* ψ
shows $\exists\psi'\ \chi'$. *inference*** $\psi\ \psi' \wedge \chi' \in$ *fst* $\psi' \wedge (\forall L.\ L \in\# \chi \longleftrightarrow L \in\# \chi')$
$\wedge$ *count* χ' *L* = *1*
$\wedge$ $(\forall\varphi.\ \varphi \in$ *fst* $\psi \longrightarrow \varphi \in$ *fst* $\psi')$
$\wedge$ $(I \models \chi \longleftrightarrow I \models \chi')$
$\wedge$ $(\forall I'.$ *total-over-m* I' $\{\chi\} \longrightarrow$ *total-over-m* I' $\{\chi'\})$
⟨*proof*⟩

lemma *can-decrease-tree-size*:
fixes ψ :: *'v state* **and** *tree* :: *'v sem-tree*
assumes *finite* (*fst* ψ) **and** *already-used-inv* ψ
and *partial-interps tree I* (*fst* ψ)
shows $\exists$(*tree'*:: *'v sem-tree*) ψ'. *inference*** $\psi\ \psi' \wedge$ *partial-interps tree' I* (*fst* ψ')
$\wedge$ (*sem-tree-size tree'* < *sem-tree-size tree* $\vee$ *sem-tree-size tree* = *0*)
⟨*proof*⟩

lemma *inference-completeness-inv*:
fixes ψ :: *'v* ::*linorder state*
assumes
unsat: $\neg$*satisfiable* (*fst* ψ) **and**
finite: *finite* (*fst* ψ) **and**
a-u-v: *already-used-inv* ψ
shows $\exists\psi'$. (*inference*** $\psi\ \psi' \wedge \{\#\} \in$ *fst* ψ')
⟨*proof*⟩

lemma *inference-completeness*:
fixes ψ :: $'v$ ::*linorder state*
assumes *unsat*: $\neg$*satisfiable* (*fst* ψ)
and *finite*: *finite* (*fst* ψ)
and *snd* $\psi = \{\}$
shows $\exists \psi'$. (*rtranclp inference* ψ ψ' $\wedge$ $\{\#\} \in$ *fst* ψ')
⟨*proof*⟩

lemma *inference-soundness*:
fixes ψ :: $'v$ ::*linorder state*
assumes *rtranclp inference* ψ ψ' **and** $\{\#\} \in$ *fst* ψ'
shows *unsatisfiable* (*fst* ψ)
⟨*proof*⟩

lemma *inference-soundness-and-completeness*:
fixes ψ :: $'v$ ::*linorder state*
assumes *finite*: *finite* (*fst* ψ)
and *snd* $\psi = \{\}$
shows ($\exists \psi'$. (*inference*** ψ ψ' $\wedge$ $\{\#\} \in$ *fst* ψ')) $\longleftrightarrow$ *unsatisfiable* (*fst* ψ)
⟨*proof*⟩

13.4 Lemma about the simplified state

abbreviation *simplified* $\psi \equiv$ (*no-step simplify* ψ)

lemma *simplified-count*:
assumes *simp*: *simplified* ψ **and** χ: $\chi \in \psi$
shows *count* χ $L \leq 1$
⟨*proof*⟩

lemma *simplified-no-both*:
assumes *simp*: *simplified* ψ **and** χ: $\chi \in \psi$
shows $\neg$ ($L \in\# \chi \wedge -L \in\# \chi$)
⟨*proof*⟩

lemma *simplified-not-tautology*:
assumes *simplified* $\{\psi\}$
shows $^{\sim}$*tautology* ψ
⟨*proof*⟩

lemma *simplified-remove*:
assumes *simplified* $\{\psi\}$
shows *simplified* $\{\psi - \{\#l\#\}\}$
⟨*proof*⟩

lemma *in-simplified-simplified*:
assumes *simp*: *simplified* ψ **and** *incl*: $\psi' \subseteq \psi$
shows *simplified* ψ'
⟨*proof*⟩

lemma *simplified-in*:
assumes *simplified* ψ
and $N \in \psi$
shows *simplified* $\{N\}$
⟨*proof*⟩

lemma *subsumes-imp-formula*:
  assumes $\psi \leq\# \varphi$
  shows $\{\psi\} \models p\ \varphi$
  ⟨*proof*⟩

lemma *simplified-imp-distinct-mset-tauto*:
  assumes *simp*: *simplified* ψ'
  shows *distinct-mset-set* ψ' **and** $\forall \chi \in \psi'.\ \neg tautology\ \chi$
⟨*proof*⟩

lemma *simplified-no-more-full1-simplified*:
  assumes *simplified* ψ
  shows $\neg$*full1 simplify* $\psi\ \psi'$
  ⟨*proof*⟩

13.5 Resolution and Invariants

inductive *resolution* :: $'v\ state \Rightarrow 'v\ state \Rightarrow bool$ **where**
full1-simp: *full1 simplify* $N\ N' \Longrightarrow$ *resolution* (N, *already-used*) (N', *already-used*) |
inferring: *inference* (N, *already-used*) (N', *already-used'*) $\Longrightarrow$ *simplified* N
  $\Longrightarrow$ *full simplify* $N'\ N'' \Longrightarrow$ *resolution* (N, *already-used*) (N'', *already-used'*)

13.5.1 Invariants

lemma *resolution-finite*:
  assumes *resolution* $\psi\ \psi'$ **and** *finite* (*fst* ψ)
  shows *finite* (*fst* ψ')
  ⟨*proof*⟩

lemma *rtranclp-resolution-finite*:
  assumes $resolution^{**}\ \psi\ \psi'$ **and** *finite* (*fst* ψ)
  shows *finite* (*fst* ψ')
  ⟨*proof*⟩

lemma *resolution-finite-snd*:
  assumes *resolution* $\psi\ \psi'$ **and** *finite* (*snd* ψ)
  shows *finite* (*snd* ψ')
  ⟨*proof*⟩

lemma *rtranclp-resolution-finite-snd*:
  assumes $resolution^{**}\ \psi\ \psi'$ **and** *finite* (*snd* ψ)
  shows *finite* (*snd* ψ')
  ⟨*proof*⟩

lemma *resolution-always-simplified*:
 assumes *resolution* $\psi\ \psi'$
 shows *simplified* (*fst* ψ')
 ⟨*proof*⟩

lemma *tranclp-resolution-always-simplified*:
  assumes *tranclp resolution* $\psi\ \psi'$
  shows *simplified* (*fst* ψ')
  ⟨*proof*⟩

lemma *resolution-atms-of*:

assumes *resolution* ψ ψ' **and** *finite* (*fst* ψ)
shows *atms-of-ms* (*fst* ψ') $\subseteq$ *atms-of-ms* (*fst* ψ)
⟨*proof*⟩

lemma *rtranclp-resolution-atms-of*:
assumes *resolution*** ψ ψ' **and** *finite* (*fst* ψ)
shows *atms-of-ms* (*fst* ψ') $\subseteq$ *atms-of-ms* (*fst* ψ)
⟨*proof*⟩

lemma *resolution-include*:
assumes *res*: *resolution* ψ ψ' **and** *finite*: *finite* (*fst* ψ)
shows *fst* ψ' $\subseteq$ *simple-clss* (*atms-of-ms* (*fst* ψ))
⟨*proof*⟩

lemma *rtranclp-resolution-include*:
assumes *res*: *tranclp resolution* ψ ψ' **and** *finite*: *finite* (*fst* ψ)
shows *fst* ψ' $\subseteq$ *simple-clss* (*atms-of-ms* (*fst* ψ))
⟨*proof*⟩

abbreviation *already-used-all-simple*
:: (*'a literal multiset* $\times$ *'a literal multiset*) *set* $\Rightarrow$ *'a set* $\Rightarrow$ *bool* **where**
already-used-all-simple already-used vars $\equiv$
($\forall$ (A, B) $\in$ *already-used*. *simplified* $\{A\}$ $\wedge$ *simplified* $\{B\}$ $\wedge$ *atms-of* A $\subseteq$ *vars* $\wedge$ *atms-of* B $\subseteq$ *vars*)

lemma *already-used-all-simple-vars-incl*:
assumes *vars* $\subseteq$ *vars'*
shows *already-used-all-simple a vars* $\Longrightarrow$ *already-used-all-simple a vars'*
⟨*proof*⟩

lemma *inference-clause-preserves-already-used-all-simple*:
assumes *inference-clause* S S'
and *already-used-all-simple* (*snd* S) *vars*
and *simplified* (*fst* S)
and *atms-of-ms* (*fst* S) $\subseteq$ *vars*
shows *already-used-all-simple* (*snd* (*fst* S $\cup$ $\{$*fst* $S'\}$, *snd* S')) *vars*
⟨*proof*⟩

lemma *inference-preserves-already-used-all-simple*:
assumes *inference* S S'
and *already-used-all-simple* (*snd* S) *vars*
and *simplified* (*fst* S)
and *atms-of-ms* (*fst* S) $\subseteq$ *vars*
shows *already-used-all-simple* (*snd* S') *vars*
⟨*proof*⟩

lemma *already-used-all-simple-inv*:
assumes *resolution* S S'
and *already-used-all-simple* (*snd* S) *vars*
and *atms-of-ms* (*fst* S) $\subseteq$ *vars*
shows *already-used-all-simple* (*snd* S') *vars*
⟨*proof*⟩

lemma *rtranclp-already-used-all-simple-inv*:
assumes *resolution*** S S'
and *already-used-all-simple* (*snd* S) *vars*

and *atms-of-ms* (*fst* S) $\subseteq$ *vars*
and *finite* (*fst* S)
shows *already-used-all-simple* (*snd* S') *vars*
⟨*proof*⟩

lemma *inference-clause-simplified-already-used-subset*:
assumes *inference-clause* S S'
and *simplified* (*fst* S)
shows *snd* $S \subset$ *snd* S'
⟨*proof*⟩

lemma *inference-simplified-already-used-subset*:
assumes *inference* S S'
and *simplified* (*fst* S)
shows *snd* $S \subset$ *snd* S'
⟨*proof*⟩

lemma *resolution-simplified-already-used-subset*:
assumes *resolution* S S'
and *simplified* (*fst* S)
shows *snd* $S \subset$ *snd* S'
⟨*proof*⟩

lemma *tranclp-resolution-simplified-already-used-subset*:
assumes *tranclp resolution* S S'
and *simplified* (*fst* S)
shows *snd* $S \subset$ *snd* S'
⟨*proof*⟩

abbreviation *already-used-top vars* $\equiv$ *simple-clss vars* $\times$ *simple-clss vars*

lemma *already-used-all-simple-in-already-used-top*:
assumes *already-used-all-simple s vars* **and** *finite vars*
shows $s \subseteq$ *already-used-top vars*
⟨*proof*⟩

lemma *already-used-top-finite*:
assumes *finite vars*
shows *finite* (*already-used-top vars*)
⟨*proof*⟩

lemma *already-used-top-increasing*:
assumes $var \subseteq var'$ **and** *finite* var'
shows *already-used-top var* $\subseteq$ *already-used-top* var'
⟨*proof*⟩

lemma *already-used-all-simple-finite*:
fixes s :: (*'a literal multiset* $\times$ *'a literal multiset*) *set* **and** *vars* :: *'a set*
assumes *already-used-all-simple s vars* **and** *finite vars*
shows *finite s*
⟨*proof*⟩

abbreviation *card-simple vars* ψ $\equiv$ *card* (*already-used-top vars* $-$ ψ)

lemma *resolution-card-simple-decreasing*:

assumes *res*: *resolution* ψ ψ'
and *a-u-s*: *already-used-all-simple* (*snd* ψ) *vars*
and *finite-v*: *finite vars*
and *finite-fst*: *finite* (*fst* ψ)
and *finite-snd*: *finite* (*snd* ψ)
and *simp*: *simplified* (*fst* ψ)
and *atms-of-ms* (*fst* ψ) $\subseteq$ *vars*
shows *card-simple vars* (*snd* ψ') $<$ *card-simple vars* (*snd* ψ)
⟨*proof*⟩

lemma *tranclp-resolution-card-simple-decreasing*:
assumes *tranclp resolution* ψ ψ' **and** *finite-fst*: *finite* (*fst* ψ)
and *already-used-all-simple* (*snd* ψ) *vars*
and *atms-of-ms* (*fst* ψ) $\subseteq$ *vars*
and *finite-v*: *finite vars*
and *finite-snd*: *finite* (*snd* ψ)
and *simplified* (*fst* ψ)
shows *card-simple vars* (*snd* ψ') $<$ *card-simple vars* (*snd* ψ)
⟨*proof*⟩

lemma *tranclp-resolution-card-simple-decreasing-2*:
assumes *tranclp resolution* ψ ψ'
and *finite-fst*: *finite* (*fst* ψ)
and *empty-snd*: *snd* ψ = {}
and *simplified* (*fst* ψ)
shows *card-simple* (*atms-of-ms* (*fst* ψ)) (*snd* ψ') $<$ *card-simple* (*atms-of-ms* (*fst* ψ)) (*snd* ψ)
⟨*proof*⟩

13.5.2 well-foundness if the relation

lemma *wf-simplified-resolution*:
assumes *f-vars*: *finite vars*
shows *wf* {(*y*:: *'v*:: *linorder state*, *x*). (*atms-of-ms* (*fst x*) $\subseteq$ *vars* $\wedge$ *simplified* (*fst x*)
$\wedge$ *finite* (*snd x*) $\wedge$ *finite* (*fst x*) $\wedge$ *already-used-all-simple* (*snd x*) *vars*) $\wedge$ *resolution x y*}
⟨*proof*⟩

lemma *wf-simplified-resolution'*:
assumes *f-vars*: *finite vars*
shows *wf* {(*y*:: *'v*:: *linorder state*, *x*). (*atms-of-ms* (*fst x*) $\subseteq$ *vars* $\wedge$ $\neg$*simplified* (*fst x*)
$\wedge$ *finite* (*snd x*) $\wedge$ *finite* (*fst x*) $\wedge$ *already-used-all-simple* (*snd x*) *vars*) $\wedge$ *resolution x y*}
⟨*proof*⟩

lemma *wf-resolution*:
assumes *f-vars*: *finite vars*
shows *wf* ({(*y*:: *'v*:: *linorder state*, *x*). (*atms-of-ms* (*fst x*) $\subseteq$ *vars* $\wedge$ *simplified* (*fst x*)
$\wedge$ *finite* (*snd x*) $\wedge$ *finite* (*fst x*) $\wedge$ *already-used-all-simple* (*snd x*) *vars*) $\wedge$ *resolution x y*}
$\cup$ {(*y*, *x*). (*atms-of-ms* (*fst x*) $\subseteq$ *vars* $\wedge$ $\neg$ *simplified* (*fst x*) $\wedge$ *finite* (*snd x*) $\wedge$ *finite* (*fst x*)
$\wedge$ *already-used-all-simple* (*snd x*) *vars*) $\wedge$ *resolution x y*}) (**is** *wf* (*?R* $\cup$ *?S*))
⟨*proof*⟩

lemma *rtrancp-simplify-already-used-inv*:
assumes *simplify*** *S S'*
and *already-used-inv* (*S*, *N*)
shows *already-used-inv* (*S'*, *N*)

⟨*proof*⟩

lemma *full1-simplify-already-used-inv*:
assumes *full1 simplify* S S'
and *already-used-inv* (S, N)
shows *already-used-inv* (S', N)
⟨*proof*⟩

lemma *full-simplify-already-used-inv*:
assumes *full simplify* S S'
and *already-used-inv* (S, N)
shows *already-used-inv* (S', N)
⟨*proof*⟩
lemma *resolution-already-used-inv*:
assumes *resolution* S S'
and *already-used-inv* S
shows *already-used-inv* S'
⟨*proof*⟩

lemma *rtranclp-resolution-already-used-inv*:
assumes *resolution*** S S'
and *already-used-inv* S
shows *already-used-inv* S'
⟨*proof*⟩

lemma *rtanclp-simplify-preserves-unsat*:
assumes *simplify*** ψ ψ'
shows *satisfiable* $\psi' \longrightarrow$ *satisfiable* ψ
⟨*proof*⟩

lemma *full1-simplify-preserves-unsat*:
assumes *full1 simplify* ψ ψ'
shows *satisfiable* $\psi' \longrightarrow$ *satisfiable* ψ
⟨*proof*⟩

lemma *full-simplify-preserves-unsat*:
assumes *full simplify* ψ ψ'
shows *satisfiable* $\psi' \longrightarrow$ *satisfiable* ψ
⟨*proof*⟩

lemma *resolution-preserves-unsat*:
assumes *resolution* ψ ψ'
shows *satisfiable* (*fst* ψ') $\longrightarrow$ *satisfiable* (*fst* ψ)
⟨*proof*⟩

lemma *rtranclp-resolution-preserves-unsat*:
assumes *resolution*** ψ ψ'
shows *satisfiable* (*fst* ψ') $\longrightarrow$ *satisfiable* (*fst* ψ)
⟨*proof*⟩

lemma *rtranclp-simplify-preserve-partial-tree*:
assumes *simplify*** N N'
and *partial-interps* t I N
shows *partial-interps* t I N'
⟨*proof*⟩

lemma *full1-simplify-preserve-partial-tree*:
assumes *full1 simplify N N'*
and *partial-interps t I N*
shows *partial-interps t I N'*
⟨*proof*⟩

lemma *full-simplify-preserve-partial-tree*:
assumes *full simplify N N'*
and *partial-interps t I N*
shows *partial-interps t I N'*
⟨*proof*⟩

lemma *resolution-preserve-partial-tree*:
assumes *resolution S S'*
and *partial-interps t I* (*fst S*)
shows *partial-interps t I* (*fst S'*)
⟨*proof*⟩

lemma *rtranclp-resolution-preserve-partial-tree*:
assumes *resolution*** *S S'*
and *partial-interps t I* (*fst S*)
shows *partial-interps t I* (*fst S'*)
⟨*proof*⟩
thm *nat-less-induct nat.induct*

lemma *nat-ge-induct*[*case-names 0 Suc*]:
assumes *P 0*
and $(\bigwedge n.\ (\bigwedge m.\ m{<}Suc\ n \Longrightarrow P\ m) \Longrightarrow P\ (Suc\ n))$
shows *P n*
⟨*proof*⟩

lemma *wf-always-more-step-False*:
assumes *wf R*
shows $(\forall x.\ \exists z.\ (z,\ x){\in}R) \Longrightarrow False$
⟨*proof*⟩

lemma *finite-finite-mset-element-of-mset*[*simp*]:
assumes *finite N*
shows *finite* $\{f\ \varphi\ L\ |\varphi\ L.\ \varphi \in N \wedge L \in\#\ \varphi \wedge P\ \varphi\ L\}$
⟨*proof*⟩

value *card*
value *filter-mset*
value $\{\#count\ \varphi\ L\ |L \in\#\ \varphi.\ 2 \leq count\ \varphi\ L\#\}$
value $(\lambda\varphi.\ msetsum\ \{\#count\ \varphi\ L\ |L \in\#\ \varphi.\ 2 \leq count\ \varphi\ L\#\})$

syntax
-comprehension1'-mset :: *'a ⇒ 'b ⇒ 'b multiset ⇒ 'a multiset*
(({#-/. - : *setof* -#}))
translations
{#*e*. *x*: *setof M*#} == *CONST set-mset* (*CONST image-mset* (%*x*. *e*) *M*)
value {# *a*. *a* : *setof* {#*1*,*1*,*2*::*int*#}#} = {*1*,*2*}

definition *sum-count-ge-2* :: *'a multiset set* ⇒ *nat* (Ξ) **where**
$sum\text{-}count\text{-}ge\text{-}2 \equiv folding.F\ (\lambda\varphi.\ op\ +(msetsum\ \{\#count\ \varphi\ L\ |L \in\#\ \varphi.\ 2 \leq count\ \varphi\ L\#\}))\ 0$

interpretation *sum-count-ge-2*:
$folding\ (\lambda\varphi.\ op\ +(msetsum\ \{\#count\ \varphi\ L\ |L \in\#\ \varphi.\ 2 \leq count\ \varphi\ L\#\}))\ 0$
rewrites
$folding.F\ (\lambda\varphi.\ op\ +(msetsum\ \{\#count\ \varphi\ L\ |L \in\#\ \varphi.\ 2 \leq count\ \varphi\ L\#\}))\ 0 = sum\text{-}count\text{-}ge\text{-}2$
⟨*proof*⟩

lemma *finite-incl-le-setsum*:
$finite\ (B{::}'a\ multiset\ set) \Longrightarrow A \subseteq B \Longrightarrow \Xi\ A \leq \Xi\ B$
⟨*proof*⟩

lemma *simplify-finite-measure-decrease*:
$simplify\ N\ N' \Longrightarrow finite\ N \Longrightarrow card\ N' + \Xi\ N' < card\ N + \Xi\ N$
⟨*proof*⟩

lemma *simplify-terminates*:
$wf\ \{(N', N).\ finite\ N \wedge simplify\ N\ N'\}$
⟨*proof*⟩

lemma *wf-terminates*:
assumes *wf r*
shows $\exists N'.(N', N) \in r^* \ \wedge (\forall N''.\ (N'', N') \notin r)$
⟨*proof*⟩

lemma *rtranclp-simplify-terminates*:
assumes *fin*: *finite N*
shows $\exists N'.\ simplify^{**}\ N\ N' \wedge simplified\ N'$
⟨*proof*⟩

lemma *finite-simplified-full1-simp*:
assumes *finite N*
shows $simplified\ N \vee (\exists N'.\ full1\ simplify\ N\ N')$
⟨*proof*⟩

lemma *finite-simplified-full-simp*:
assumes *finite N*
shows $\exists N'.\ full\ simplify\ N\ N'$
⟨*proof*⟩

lemma *can-decrease-tree-size-resolution*:
fixes ψ :: *'v state* **and** *tree* :: *'v sem-tree*
assumes *finite* (*fst* ψ) **and** *already-used-inv* ψ
and *partial-interps tree I* (*fst* ψ)
and *simplified* (*fst* ψ)
shows $\exists (tree'{::}\ 'v\ sem\text{-}tree)\ \psi'.\ resolution^{**}\ \psi\ \psi' \wedge partial\text{-}interps\ tree'\ I\ (fst\ \psi')$
$\wedge\ (sem\text{-}tree\text{-}size\ tree' < sem\text{-}tree\text{-}size\ tree \vee sem\text{-}tree\text{-}size\ tree = 0)$
⟨*proof*⟩

lemma *resolution-completeness-inv*:
fixes ψ :: *'v ::linorder state*

assumes
unsat: $\neg$*satisfiable* (*fst* ψ) **and**
finite: *finite* (*fst* ψ) **and**
a-u-v: *already-used-inv* ψ
shows $\exists\,\psi'$. (*resolution*** ψ ψ' $\wedge$ $\{\#\} \in$ *fst* ψ')
$\langle$*proof*$\rangle$

lemma *resolution-preserves-already-used-inv*:
assumes *resolution* S S'
and *already-used-inv* S
shows *already-used-inv* S'
$\langle$*proof*$\rangle$

lemma *rtranclp-resolution-preserves-already-used-inv*:
assumes *resolution*** S S'
and *already-used-inv* S
shows *already-used-inv* S'
$\langle$*proof*$\rangle$

lemma *resolution-completeness*:
fixes ψ :: $'v$ ::*linorder state*
assumes *unsat*: $\neg$*satisfiable* (*fst* ψ)
and *finite*: *finite* (*fst* ψ)
and *snd* $\psi = \{\}$
shows $\exists\,\psi'$. (*resolution*** ψ ψ' $\wedge$ $\{\#\} \in$ *fst* ψ')
$\langle$*proof*$\rangle$

lemma *rtranclp-preserves-sat*:
assumes *simplify*** S S'
and *satisfiable* S
shows *satisfiable* S'
$\langle$*proof*$\rangle$

lemma *resolution-preserves-sat*:
assumes *resolution* S S'
and *satisfiable* (*fst* S)
shows *satisfiable* (*fst* S')
$\langle$*proof*$\rangle$

lemma *rtranclp-resolution-preserves-sat*:
assumes *resolution*** S S'
and *satisfiable* (*fst* S)
shows *satisfiable* (*fst* S')
$\langle$*proof*$\rangle$

lemma *resolution-soundness*:
fixes ψ :: $'v$ ::*linorder state*
assumes *resolution*** ψ ψ' **and** $\{\#\} \in$ *fst* ψ'
shows *unsatisfiable* (*fst* ψ)
$\langle$*proof*$\rangle$

lemma *resolution-soundness-and-completeness*:
fixes ψ :: $'v$ ::*linorder state*
assumes *finite*: *finite* (*fst* ψ)
and *snd*: *snd* $\psi = \{\}$

shows $(\exists\,\psi'.\ (\mathit{resolution}^{**}\ \psi\ \psi' \wedge \{\#\} \in \mathit{fst}\ \psi')) \longleftrightarrow \mathit{unsatisfiable}\ (\mathit{fst}\ \psi)$
  ⟨*proof*⟩

lemma *simplified-falsity*:
  assumes *simp*: *simplified* ψ
  and $\{\#\} \in \psi$
  shows $\psi = \{\{\#\}\}$
⟨*proof*⟩

lemma *simplify-falsity-in-preserved*:
  assumes *simplify* $\chi s\ \chi s'$
  and $\{\#\} \in \chi s$
  shows $\{\#\} \in \chi s'$
  ⟨*proof*⟩

lemma *rtranclp-simplify-falsity-in-preserved*:
  assumes $\mathit{simplify}^{**}\ \chi s\ \chi s'$
  and $\{\#\} \in \chi s$
  shows $\{\#\} \in \chi s'$
  ⟨*proof*⟩

lemma *resolution-falsity-get-falsity-alone*:
  assumes *finite* $(\mathit{fst}\ \psi)$
  shows $(\exists\,\psi'.\ (\mathit{resolution}^{**}\ \psi\ \psi' \wedge \{\#\} \in \mathit{fst}\ \psi')) \longleftrightarrow (\exists\,a\text{-}u\text{-}v.\ \mathit{resolution}^{**}\ \psi\ (\{\{\#\}\},\ a\text{-}u\text{-}v))$
    (**is** $?A \longleftrightarrow ?B$)
⟨*proof*⟩

lemma *resolution-soundness-and-completeness'*:
  fixes ψ :: $'v$ ::*linorder state*
  assumes
    finite: *finite* $(\mathit{fst}\ \psi)$**and**
    snd: *snd* $\psi = \{\}$
  shows $(\exists\,a\text{-}u\text{-}v.\ (\mathit{resolution}^{**}\ \psi\ (\{\{\#\}\},\ a\text{-}u\text{-}v))) \longleftrightarrow \mathit{unsatisfiable}\ (\mathit{fst}\ \psi)$
    ⟨*proof*⟩

end

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

begin

14 Partial Clausal Logic

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

14.1 Marked Literals

14.1.1 Definition

datatype ('*v*, '*lvl*, '*mark*) *marked-lit* =
  is-marked: *Marked* (*lit-of*: '*v literal*) (*level-of*: '*lvl*) |
  is-proped: *Propagated* (*lit-of*: '*v literal*) (*mark-of*: '*mark*)

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

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

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

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

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

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

lemma *lits-of-insert*[*simp*]:
lits-of (*insert L Ls*) = *insert* (*lit-of L*) (*lits-of Ls*)
⟨*proof*⟩

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

lemma *finite-lits-of-def*[*simp*]:
finite (*lits-of-l L*)
⟨*proof*⟩

abbreviation *unmark* **where**
unmark ≡ (λ*a*. {#*lit-of a*#})

abbreviation *unmark-s* **where**
unmark-s M ≡ *unmark* ‘ *M*

abbreviation *unmark-l* **where**
unmark-l M ≡ *unmark-s* (*set M*)

lemma *atms-of-ms-lambda-lit-of-is-atm-of-lit-of*[*simp*]:
atms-of-ms (*unmark-l M'*) = *atm-of* ‘ *lits-of-l M'*
⟨*proof*⟩

lemma *lits-of-l-empty-is-empty*[*iff*]:
lits-of-l M = {} ⟷ *M* = []
⟨*proof*⟩

14.1.2 Entailment

definition *true-annot* :: $('a, 'l, 'm)$ *marked-lits* ⇒ *'a clause* ⇒ *bool* (**infix** ⊨*a 49*) **where**

$I \models a\ C \longleftrightarrow (\textit{lits-of-l}\ I) \models C$

definition *true-annots* :: ('a, 'l, 'm) *marked-lits* ⇒ 'a *clauses* ⇒ *bool* (**infix** ⊨*as* *49*) **where**
$I \models as\ CC \longleftrightarrow (\forall\, C \in CC.\ I \models a\ C)$

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

lemma *true-annot-empty*[*simp*]:
$\neg I \models a\ \{\#\}$
⟨*proof*⟩

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

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

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

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

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

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

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

Link between ⊨*as* and ⊨*s*:

lemma *true-annots-true-cls*:
$I \models as\ CC \longleftrightarrow \textit{lits-of-l}\ I \models s\ CC$
⟨*proof*⟩

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

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

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

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

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

lemma *true-annots-marked-true-cls*[*iff*]:
$map\ (\lambda M.\ Marked\ M\ a)\ M \models as\ N \longleftrightarrow set\ M \models s\ N$
⟨*proof*⟩

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

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

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

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

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

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

14.1.3 Defined and undefined literals

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

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

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

lemma *atm-imp-marked-or-proped*:
assumes $x \in set\ I$
shows

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

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

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

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

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

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

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

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

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

14.2 Backtracking

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

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

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

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

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

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

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

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

14.3 Decomposition with respect to the marked literals

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

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

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

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

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

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

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

lemma *get-all-marked-decomposition-nil-backtrack-split-snd-nil*:
  $get\text{-}all\text{-}marked\text{-}decomposition\ S = [([], A)] \Longrightarrow snd\ (backtrack\text{-}split\ S) = []$
  ⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *Marked-cons-in-get-all-marked-decomposition-append-Marked-cons*:
$\exists\, M1\ M2.\ (Marked\ K\ i\ \#\ M1,\ M2) \in set\ (get\text{-}all\text{-}marked\text{-}decomposition\ (c\ @\ Marked\ K\ i\ \#\ c'))$
⟨*proof*⟩

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

lemma *all-decomposition-implies-empty*[*iff*]:
$all\text{-}decomposition\text{-}implies\ N\ []$ ⟨*proof*⟩

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

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

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

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

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

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

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

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

14.4 Negation of Clauses

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

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

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

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

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

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

lemma *Ball-CNot-Ball-mset*[*simp*] :

$(\forall x \in CNot\ D.\ P\ x) \longleftrightarrow (\forall L \in\#\ D.\ P\ \{\#-L\#\})$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

14.5 Other

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Other definition using *Union-mset*

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

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

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

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

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

14.6 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* **and**
    insert-cls :: *'v literal ⇒ 'cls ⇒ 'cls* **and**
    remove-lit :: *'v literal ⇒ 'cls ⇒ 'cls*
  assumes
    insert-cls[*simp*]: *mset-cls* (*insert-cls L C*) = *mset-cls C* + {#*L*#} **and**
    remove-lit[*simp*]: *mset-cls* (*remove-lit L C*) = *remove1-mset L* (*mset-cls C*)
begin
end

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

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

context
begin
  interpretation *list-cls*: *raw-cls mset*
    op # remove1
    ⟨*proof*⟩

  interpretation *cls-cls*: *raw-cls id*
    λL C. C + {#*L*#} *remove1-mset*

$\langle proof \rangle$

interpretation *list-cls*: *raw-ccls-union mset*
union-mset-list
op # remove1
$\langle proof \rangle$

interpretation *cls-cls*: *raw-ccls-union id*
op $\#\cup$ $\lambda L\ C.\ C + \{\#L\#\}$ *remove1-mset*
$\langle proof \rangle$
end

Over the abstract clauses, we have the following properties:

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

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

end

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

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

⟨*proof*⟩

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

interpretation *list-clss*: *raw-clss mset*
op # *remove1* $\lambda L.$ *mset* (*map mset L*) *op* @ $\lambda L\ C.\ L \in$ *set C op* #
remove-first
⟨*proof*⟩
end

end
theory *CDCL-WNOT-Measure*
imports *Main*
begin

15 Measure

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

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

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

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

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

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

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

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

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

lemma μ_C*-cons-non-empty-inf*:
assumes *M-ge-1*: $\forall i{\in}set\ M.\ i \geq 1$ **and** *M*: $M \neq []$

shows μ_C $s\ b\ M \geq b\ \hat{}\ (s - \mathit{length}\ M)$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

16 NOT's CDCL

16.1 Auxiliary Lemmas and Measure

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

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

lemma *atms-of-uminus-lit-atm-of-lit-of*:
atms-of {# −*lit-of x*. *x* ∈# *A*#} = *atm-of* ' (*lit-of* ' (*set-mset A*))
⟨*proof*⟩

lemma *atms-of-ms-single-image-atm-of-lit-of*:
atms-of-ms (*unmark-s A*) = *atm-of* ' (*lit-of* ' *A*)
⟨*proof*⟩

16.2 Initial definitions

16.2.1 The state

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

locale *dpll-state-ops* =
raw-clss mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss
for
mset-cls:: *'cls* ⇒ *'v clause* **and**
insert-cls :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
remove-lit :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
mset-clss:: *'clss* ⇒ *'v clauses* **and**
union-clss :: *'clss* ⇒ *'clss* ⇒ *'clss* **and**
in-clss :: *'cls* ⇒ *'clss* ⇒ *bool* **and**
insert-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
remove-from-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* +
fixes
trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *marked-lits* **and**
raw-clauses :: *'st* ⇒ *'clss* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *marked-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
$add\text{-}cls_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
$remove\text{-}cls_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st*
begin

notation *insert-cls* (**infix** !++ *50*)

notation *in-clss* (**infix** !∈! *50*)
notation *union-clss* (**infix** ⊕ *50*)
notation *insert-clss* (**infix** !++! *50*)

abbreviation $clauses_{NOT}$ **where**
$clauses_{NOT}$ *S* ≡ *mset-clss* (*raw-clauses S*)

end

locale *dpll-state* =
dpll-state-ops mset-cls insert-cls remove-lit — related to each clause
mset-clss union-clss in-clss insert-clss remove-from-clss — related to the clauses

trail raw-clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ — related to the state
for
mset-cls:: *'cls* ⇒ *'v clause* **and**
insert-cls :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
remove-lit :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
mset-clss:: *'clss* ⇒ *'v clauses* **and**

$union\text{-}clss :: 'clss \Rightarrow 'clss \Rightarrow 'clss$ **and**
$in\text{-}clss :: 'cls \Rightarrow 'clss \Rightarrow bool$ **and**
$insert\text{-}clss :: 'cls \Rightarrow 'clss \Rightarrow 'clss$ **and**
$remove\text{-}from\text{-}clss :: 'cls \Rightarrow 'clss \Rightarrow 'clss$ **and**
$trail :: 'st \Rightarrow ('v, unit, unit)\ marked\text{-}lits$ **and**
$raw\text{-}clauses :: 'st \Rightarrow 'clss$ **and**
$prepend\text{-}trail :: ('v, unit, unit)\ marked\text{-}lit \Rightarrow 'st \Rightarrow 'st$ **and**
$tl\text{-}trail :: 'st \Rightarrow 'st$ **and**
$add\text{-}cls_{NOT} :: 'cls \Rightarrow 'st \Rightarrow 'st$ **and**
$remove\text{-}cls_{NOT} :: 'cls \Rightarrow 'st \Rightarrow 'st$ +
assumes
trail-prepend-trail[*simp*]:
$\bigwedge st\ L.\ undefined\text{-}lit\ (trail\ st)\ (lit\text{-}of\ L) \Longrightarrow trail\ (prepend\text{-}trail\ L\ st) = L\ \#\ trail\ st$
and
tl-trail[*simp*]: $trail\ (tl\text{-}trail\ S) = tl\ (trail\ S)$ **and**
$trail\text{-}add\text{-}cls_{NOT}$[*simp*]: $\bigwedge st\ C.\ no\text{-}dup\ (trail\ st) \Longrightarrow trail\ (add\text{-}cls_{NOT}\ C\ st) = trail\ st$ **and**
$trail\text{-}remove\text{-}cls_{NOT}$[*simp*]: $\bigwedge st\ C.\ trail\ (remove\text{-}cls_{NOT}\ C\ st) = trail\ st$ **and**

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

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

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

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

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

lemma $trail\text{-}reduce\text{-}trail\text{-}to_{NOT}\text{-}nil$[*simp*]:
$trail\ (reduce\text{-}trail\text{-}to_{NOT}\ []\ S) = []$

$\langle proof \rangle$

lemma *clauses-reduce-trail-to*$_{NOT}$*-nil*:
  $clauses_{NOT}$ (*reduce-trail-to*$_{NOT}$ [] S) = $clauses_{NOT}$ S
  $\langle proof \rangle$

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

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

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

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

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

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

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

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

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

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

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

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

⟨*proof*⟩

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

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

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

end

16.2.2 Definition of the operation

locale *propagate-ops* =
dpll-state mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss
trail raw-clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$
for
mset-cls:: *'cls* ⇒ *'v clause* **and**
insert-cls :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
remove-lit :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
mset-clss:: *'clss* ⇒ *'v clauses* **and**
union-clss :: *'clss* ⇒ *'clss* ⇒ *'clss* **and**
in-clss :: *'cls* ⇒ *'clss* ⇒ *bool* **and**
insert-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
remove-from-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *marked-lits* **and**
raw-clauses :: *'st* ⇒ *'clss* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *marked-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
add-cls$_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
remove-cls$_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* +
fixes
propagate-cond :: (*'v*, *unit*, *unit*) *marked-lit* ⇒ *'st* ⇒ *bool*
begin
inductive *propagate*$_{NOT}$ :: *'st* ⇒ *'st* ⇒ *bool* **where**
propagate$_{NOT}$[*intro*]: *C* + {#*L*#} ∈# *clauses*$_{NOT}$ *S* $\Longrightarrow$ *trail S* ⊨*as CNot C*
$\Longrightarrow$ *undefined-lit* (*trail S*) *L*
$\Longrightarrow$ *propagate-cond* (*Propagated L* ()) *S*
$\Longrightarrow$ *T* ∼ *prepend-trail* (*Propagated L* ()) *S*
$\Longrightarrow$ *propagate*$_{NOT}$ *S T*
inductive-cases *propagate*$_{NOT}$*E*[*elim*]: *propagate*$_{NOT}$ *S T*

end

locale *decide-ops* =
dpll-state mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss
trail raw-clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$

for
  $mset\text{-}cls$:: $'cls \Rightarrow 'v\ clause$ **and**
  $insert\text{-}cls$ :: $'v\ literal \Rightarrow 'cls \Rightarrow 'cls$ **and**
  $remove\text{-}lit$ :: $'v\ literal \Rightarrow 'cls \Rightarrow 'cls$ **and**
  $mset\text{-}clss$:: $'clss \Rightarrow 'v\ clauses$ **and**
  $union\text{-}clss$ :: $'clss \Rightarrow 'clss \Rightarrow 'clss$ **and**
  $in\text{-}clss$ :: $'cls \Rightarrow 'clss \Rightarrow bool$ **and**
  $insert\text{-}clss$ :: $'cls \Rightarrow 'clss \Rightarrow 'clss$ **and**
  $remove\text{-}from\text{-}clss$ :: $'cls \Rightarrow 'clss \Rightarrow 'clss$ **and**
  $trail$ :: $'st \Rightarrow ('v,\ unit,\ unit)\ marked\text{-}lits$ **and**
  $raw\text{-}clauses$ :: $'st \Rightarrow 'clss$ **and**
  $prepend\text{-}trail$ :: $('v,\ unit,\ unit)\ marked\text{-}lit \Rightarrow 'st \Rightarrow 'st$ **and**
  $tl\text{-}trail$ :: $'st \Rightarrow 'st$ **and**
  $add\text{-}cls_{NOT}$ :: $'cls \Rightarrow 'st \Rightarrow 'st$ **and**
  $remove\text{-}cls_{NOT}$ :: $'cls \Rightarrow 'st \Rightarrow 'st$
begin
inductive $decide_{NOT}$ :: $'st \Rightarrow 'st \Rightarrow bool$ **where**
$decide_{NOT}[intro]$: $undefined\text{-}lit\ (trail\ S)\ L \Longrightarrow atm\text{-}of\ L \in atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S)$
  $\Longrightarrow T \sim prepend\text{-}trail\ (Marked\ L\ ())\ S$
  $\Longrightarrow decide_{NOT}\ S\ T$

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

locale $backjumping\text{-}ops$ =
  $dpll\text{-}state$ $mset\text{-}cls\ insert\text{-}cls\ remove\text{-}lit$
    $mset\text{-}clss\ union\text{-}clss\ in\text{-}clss\ insert\text{-}clss\ remove\text{-}from\text{-}clss$
    $trail\ raw\text{-}clauses\ prepend\text{-}trail\ tl\text{-}trail\ add\text{-}cls_{NOT}\ remove\text{-}cls_{NOT}$
  for
    $mset\text{-}cls$:: $'cls \Rightarrow 'v\ clause$ **and**
    $insert\text{-}cls$ :: $'v\ literal \Rightarrow 'cls \Rightarrow 'cls$ **and**
    $remove\text{-}lit$ :: $'v\ literal \Rightarrow 'cls \Rightarrow 'cls$ **and**
    $mset\text{-}clss$:: $'clss \Rightarrow 'v\ clauses$ **and**
    $union\text{-}clss$ :: $'clss \Rightarrow 'clss \Rightarrow 'clss$ **and**
    $in\text{-}clss$ :: $'cls \Rightarrow 'clss \Rightarrow bool$ **and**
    $insert\text{-}clss$ :: $'cls \Rightarrow 'clss \Rightarrow 'clss$ **and**
    $remove\text{-}from\text{-}clss$ :: $'cls \Rightarrow 'clss \Rightarrow 'clss$ **and**
    $trail$ :: $'st \Rightarrow ('v,\ unit,\ unit)\ marked\text{-}lits$ **and**
    $raw\text{-}clauses$ :: $'st \Rightarrow 'clss$ **and**
    $prepend\text{-}trail$ :: $('v,\ unit,\ unit)\ marked\text{-}lit \Rightarrow 'st \Rightarrow 'st$ **and**
    $tl\text{-}trail$ :: $'st \Rightarrow 'st$ **and**
    $add\text{-}cls_{NOT}$ :: $'cls \Rightarrow 'st \Rightarrow 'st$ **and**
    $remove\text{-}cls_{NOT}$ :: $'cls \Rightarrow 'st \Rightarrow 'st$ +
  fixes
    $backjump\text{-}conds$ :: $'v\ clause \Rightarrow 'v\ clause \Rightarrow 'v\ literal \Rightarrow 'st \Rightarrow 'st \Rightarrow bool$
begin

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

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

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

end

16.3 DPLL with backjumping

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

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

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

16.3.1 Definition

We define dpll with backjumping:

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

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

16.3.2 Basic properties

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

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

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

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

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

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

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

16.3.3 Termination

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

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

⟨*proof*⟩

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

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

Well-foundedness The bounds are the following:

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

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

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

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

16.3.4 Normal Forms

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

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

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

2. $\neg$ M $\models as$ N tells us that there is conflict.

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

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

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

theorem *dpll-backjump-final-state*:
  fixes A :: *'v literal multiset set* **and** S T :: *'st*
  assumes
    atms-of-mm (*clauses*$_{NOT}$ S) $\subseteq$ *atms-of-ms* A **and**
    atm-of ‘ *lits-of-l* (*trail* S) $\subseteq$ *atms-of-ms* A **and**
    no-dup (*trail* S) **and**
    finite A **and**
    inv: *inv* S **and**
    n-s: *no-step dpll-bj* S **and**
    decomp: *all-decomposition-implies-m* (*clauses*$_{NOT}$ S) (*get-all-marked-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

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

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

⟨proof⟩

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

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

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

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

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

lemma *rtranclp-dpll-bj-all-decomposition-implies-inv*:
assumes
$dpll\text{-}bj^{**}$ *S T* **and**
inv S
all-decomposition-implies-m ($clauses_{NOT}$ *S*) (*get-all-marked-decomposition* (*trail S*))
shows *all-decomposition-implies-m* ($clauses_{NOT}$ *T*) (*get-all-marked-decomposition* (*trail T*))
⟨proof⟩

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

lemma *wf-tranclp-dpll-bj*:
assumes *fin*: *finite A*

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

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

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

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

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

lemma *tranclp-dpll-bj-trail-mes-decreasing-prop*:
assumes *dpll*: *dpll-bj*$^{++}$ S T **and** *inv*: *inv* S **and**
N-A: *atms-of-mm* (*clauses*$_{NOT}$ S) $\subseteq$ *atms-of-ms* A **and**
M-A: *atm-of* ‘ *lits-of-l* (*trail* S) $\subseteq$ *atms-of-ms* A **and**
n-d: *no-dup* (*trail* S) **and**
fin-A: *finite* A
shows (*2+card* (*atms-of-ms* A)) ^ (*1+card* (*atms-of-ms* A))
$-\ \mu_C$ (*1+card* (*atms-of-ms* A)) (*2+card* (*atms-of-ms* A)) (*trail-weight* T)
$<$ (*2+card* (*atms-of-ms* A)) ^ (*1+card* (*atms-of-ms* A))
$-\ \mu_C$ (*1+card* (*atms-of-ms* A)) (*2+card* (*atms-of-ms* A)) (*trail-weight* S)
⟨*proof*⟩

end

16.4 CDCL

16.4.1 Learn and Forget

locale *learn-ops* =

dpll-state mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss
trail raw-clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$
for
mset-cls:: *'cls* ⇒ *'v clause* **and**
insert-cls :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
remove-lit :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
mset-clss:: *'clss* ⇒ *'v clauses* **and**
union-clss :: *'clss* ⇒ *'clss* ⇒ *'clss* **and**
in-clss :: *'cls* ⇒ *'clss* ⇒ *bool* **and**
insert-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
remove-from-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *marked-lits* **and**
raw-clauses :: *'st* ⇒ *'clss* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *marked-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
add-cls$_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
remove-cls$_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* +
fixes
learn-cond :: *'cls* ⇒ *'st* ⇒ *bool*
begin
inductive *learn* :: *'st* ⇒ *'st* ⇒ *bool* **where**
learn$_{NOT}$*-rule*: *clauses*$_{NOT}$ *S* ⊨*pm mset-cls C* ⟹
atms-of (*mset-cls C*) ⊆ *atms-of-mm* (*clauses*$_{NOT}$ *S*) ∪ *atm-of* ‘ (*lits-of-l* (*trail S*)) ⟹
learn-cond C S ⟹
T ∼ *add-cls*$_{NOT}$ *C S* ⟹
learn S T
inductive-cases *learn*$_{NOT}$*E*: *learn S T*

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

locale *forget-ops* =
dpll-state mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss
trail raw-clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$
for
mset-cls:: *'cls* ⇒ *'v clause* **and**
insert-cls :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
remove-lit :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
mset-clss:: *'clss* ⇒ *'v clauses* **and**
union-clss :: *'clss* ⇒ *'clss* ⇒ *'clss* **and**
in-clss :: *'cls* ⇒ *'clss* ⇒ *bool* **and**
insert-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
remove-from-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *marked-lits* **and**
raw-clauses :: *'st* ⇒ *'clss* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *marked-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
add-cls$_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
remove-cls$_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* +
fixes

forget-cond :: *'cls* ⇒ *'st* ⇒ *bool*
begin
inductive $forget_{NOT}$ :: *'st* ⇒ *'st* ⇒ *bool* **where**
$forget_{NOT}$:
removeAll-mset (*mset-cls* C)($clauses_{NOT}$ S) ⊨*pm* *mset-cls* C ⟹
forget-cond C S ⟹
C !∈! *raw-clauses* S ⟹
T ∼ $remove\text{-}cls_{NOT}$ C S ⟹
$forget_{NOT}$ S T
inductive-cases $forget_{NOT}E$: $forget_{NOT}$ S T

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

locale $learn\text{-}and\text{-}forget_{NOT}$ =
learn-ops mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss
trail raw-clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ *learn-cond* +
forget-ops mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss
trail raw-clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ *forget-cond*
for
mset-cls:: *'cls* ⇒ *'v clause* **and**
insert-cls :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
remove-lit :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
mset-clss:: *'clss* ⇒ *'v clauses* **and**
union-clss :: *'clss* ⇒ *'clss* ⇒ *'clss* **and**
in-clss :: *'cls* ⇒ *'clss* ⇒ *bool* **and**
insert-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
remove-from-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *marked-lits* **and**
raw-clauses :: *'st* ⇒ *'clss* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *marked-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
$add\text{-}cls_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
$remove\text{-}cls_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
learn-cond forget-cond :: *'cls* ⇒ *'st* ⇒ *bool*
begin
inductive $learn\text{-}and\text{-}forget_{NOT}$ :: *'st* ⇒ *'st* ⇒ *bool*
where
lf-learn: *learn* S T ⟹ $learn\text{-}and\text{-}forget_{NOT}$ S T |
lf-forget: $forget_{NOT}$ S T ⟹ $learn\text{-}and\text{-}forget_{NOT}$ S T
end

16.4.2 Definition of CDCL

locale *conflict-driven-clause-learning-ops* =
dpll-with-backjumping-ops mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss
trail raw-clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
inv backjump-conds propagate-conds +
$learn\text{-}and\text{-}forget_{NOT}$ *mset-cls insert-cls remove-lit*
mset-clss union-clss in-clss insert-clss remove-from-clss

trail raw-clauses prepend-trail tl-trail add-cls$_{NOT}$ remove-cls$_{NOT}$ learn-cond forget-cond
for
mset-cls:: *'cls* ⇒ *'v clause* **and**
insert-cls :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
remove-lit :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
mset-clss:: *'clss* ⇒ *'v clauses* **and**
union-clss :: *'clss* ⇒ *'clss* ⇒ *'clss* **and**
in-clss :: *'cls* ⇒ *'clss* ⇒ *bool* **and**
insert-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
remove-from-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *marked-lits* **and**
raw-clauses :: *'st* ⇒ *'clss* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *marked-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
add-cls$_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
remove-cls$_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
inv :: *'st* ⇒ *bool* **and**
backjump-conds :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool* **and**
propagate-conds :: (*'v*, *unit*, *unit*) *marked-lit* ⇒ *'st* ⇒ *bool* **and**
learn-cond forget-cond :: *'cls* ⇒ *'st* ⇒ *bool*
begin

inductive *cdcl$_{NOT}$* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
c-dpll-bj: *dpll-bj S S'* ⟹ *cdcl$_{NOT}$ S S'* |
c-learn: *learn S S'* ⟹ *cdcl$_{NOT}$ S S'* |
c-forget$_{NOT}$: *forget$_{NOT}$ S S'* ⟹ *cdcl$_{NOT}$ S S'*

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

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

Consistency of the trail **lemma** *cdcl$_{NOT}$-consistent*:
assumes
cdcl$_{NOT}$ S T **and**

inv S **and**
no-dup (*trail S*)
shows *consistent-interp* (*lits-of-l* (*trail T*))
⟨*proof*⟩

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

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

lemma $cdcl_{NOT}$*-atms-in-trail*:
assumes $cdcl_{NOT}$ *S T***and** *inv S* **and** *no-dup* (*trail S*)
and *atm-of* ‘ (*lits-of-l* (*trail S*)) ⊆ *atms-of-mm* ($clauses_{NOT}$ *S*)
shows *atm-of* ‘ (*lits-of-l* (*trail T*)) ⊆ *atms-of-mm* ($clauses_{NOT}$ *S*)
⟨*proof*⟩

lemma $cdcl_{NOT}$*-atms-in-trail-in-set*:
assumes
$cdcl_{NOT}$ *S T* **and** *inv S* **and** *no-dup* (*trail S*) **and**
atms-of-mm ($clauses_{NOT}$ *S*) ⊆ *A* **and**
atm-of ‘ (*lits-of-l* (*trail S*)) ⊆ *A*
shows *atm-of* ‘ (*lits-of-l* (*trail T*)) ⊆ *A*
⟨*proof*⟩

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

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

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

16.5 CDCL with invariant

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

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

lemma *rtranclp-*$cdcl_{NOT}$*-no-dup*:

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

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

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

lemma *rtranclp-*$cdcl_{NOT}$*-bj-sat-ext-iff*:
assumes $cdcl_{NOT}{}^{**}$ *S T***and** *inv S* **and** *no-dup* (*trail S*)
shows *I*⊨*sextm* $clauses_{NOT}$ *S* ⟷ *I*⊨*sextm* $clauses_{NOT}$ *T*
⟨*proof*⟩

definition $cdcl_{NOT}$*-NOT-all-inv* **where**
$cdcl_{NOT}$*-NOT-all-inv A S* ⟷ (*finite A* ∧ *inv S* ∧ *atms-of-mm* ($clauses_{NOT}$ *S*) ⊆ *atms-of-ms A*
∧ *atm-of* ' *lits-of-l* (*trail S*) ⊆ *atms-of-ms A* ∧ *no-dup* (*trail S*))

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

abbreviation *learn-or-forget* **where**
learn-or-forget S T ≡ *learn S T* ∨ $forget_{NOT}$ *S T*

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

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

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

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

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

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

lemma *cdcl$_{NOT}$-final-state*:
  assumes
    n-s: *no-step cdcl$_{NOT}$ S* **and**
    inv: *cdcl$_{NOT}$-NOT-all-inv A S* **and**
    decomp: *all-decomposition-implies-m* (*clauses$_{NOT}$ S*) (*get-all-marked-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*⟩

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

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

16.6 Termination

16.6.1 Restricting learn and forget

locale *conflict-driven-clause-learning-learning-before-backjump-only-distinct-learnt* =
  dpll-state mset-cls insert-cls remove-lit
    mset-clss union-clss in-clss insert-clss remove-from-clss
    trail raw-clauses prepend-trail tl-trail add-cls$_{NOT}$ remove-cls$_{NOT}$ +
  conflict-driven-clause-learning mset-cls insert-cls remove-lit

mset-clss union-clss in-clss insert-clss remove-from-clss
trail raw-clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
inv backjump-conds propagate-conds
$\lambda C\ S.$ *distinct-mset* (*mset-cls* C) $\wedge$ $\neg$*tautology* (*mset-cls* C) $\wedge$ *learn-restrictions* $C\ S$ $\wedge$
($\exists F\ K\ d\ F'\ C'\ L.$ *trail* $S = F'$ @ *Marked* K () # F $\wedge$ *mset-cls* $C = C' + \{\#L\#\}$ $\wedge$ $F \models as$ *CNot* C'
$\wedge$ $C' + \{\#L\#\} \notin\#$ $clauses_{NOT}$ S)
$\lambda C\ S.$ $\neg(\exists F'\ F\ K\ d\ L.$ *trail* $S = F'$ @ *Marked* K () # F $\wedge$ $F \models as$ *CNot* (*remove1-mset* L (*mset-cls* C)))
$\wedge$ *forget-restrictions* $C\ S$
for
mset-cls:: $'cls \Rightarrow$ $'v$ *clause* **and**
insert-cls :: $'v$ *literal* $\Rightarrow$ $'cls \Rightarrow$ $'cls$ **and**
remove-lit :: $'v$ *literal* $\Rightarrow$ $'cls \Rightarrow$ $'cls$ **and**
mset-clss:: $'clss \Rightarrow$ $'v$ *clauses* **and**
union-clss :: $'clss \Rightarrow$ $'clss \Rightarrow$ $'clss$ **and**
in-clss :: $'cls \Rightarrow$ $'clss \Rightarrow$ *bool* **and**
insert-clss :: $'cls \Rightarrow$ $'clss \Rightarrow$ $'clss$ **and**
remove-from-clss :: $'cls \Rightarrow$ $'clss \Rightarrow$ $'clss$ **and**
trail :: $'st \Rightarrow$ ($'v$, *unit*, *unit*) *marked-lits* **and**
raw-clauses :: $'st \Rightarrow$ $'clss$ **and**
prepend-trail :: ($'v$, *unit*, *unit*) *marked-lit* $\Rightarrow$ $'st \Rightarrow$ $'st$ **and**
tl-trail :: $'st \Rightarrow 'st$ **and**
$add\text{-}cls_{NOT}$ :: $'cls \Rightarrow$ $'st \Rightarrow$ $'st$ **and**
$remove\text{-}cls_{NOT}$ :: $'cls \Rightarrow$ $'st \Rightarrow$ $'st$ **and**
inv :: $'st \Rightarrow$ *bool* **and**
backjump-conds :: $'v$ *clause* $\Rightarrow$ $'v$ *clause* $\Rightarrow$ $'v$ *literal* $\Rightarrow$ $'st \Rightarrow$ $'st \Rightarrow$ *bool* **and**
propagate-conds :: ($'v$, *unit*, *unit*) *marked-lit* $\Rightarrow$ $'st \Rightarrow$ *bool* **and**
learn-restrictions forget-restrictions :: $'cls \Rightarrow$ $'st \Rightarrow$ *bool*
begin

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

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

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

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

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

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

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

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

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

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

lemma *learn-conflicting-increasing*:

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

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

abbreviation μ_L :: *nat* ⇒ *'st* ⇒ *nat* × *nat* **where**
μ_L b $S \equiv$ (*conflicting-bj-clss-yet* b S, *card* (*set-mset* ($clauses_{NOT}$ S)))

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

lemma *forget-*μ_L*-decrease*:
assumes $forget_{NOT}$: $forget_{NOT}$ S T
shows (μ_L b T, μ_L b S) ∈ *less-than* <∗*lex*∗> *less-than*
⟨*proof*⟩

lemma *set-condition-or-split*:
{*a*. (*a* = *b* ∨ *Q a*) ∧ *S a*} = (*if S b then* {*b*} *else* {}) ∪ {*a*. *Q a* ∧ *S a*}
⟨*proof*⟩

lemma *set-insert-neq*:
A ≠ *insert a A* ⟷ *a* ∉ *A*
⟨*proof*⟩

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

We have to assume the following:

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

definition μ_{CDCL} **where**
μ_{CDCL} A $T \equiv$ ((*2*+*card* (*atms-of-ms A*)) ^ (*1*+*card* (*atms-of-ms A*))
− μ_C (*1*+*card* (*atms-of-ms A*)) (*2*+*card* (*atms-of-ms A*)) (*trail-weight T*),
conflicting-bj-clss-yet (*card* (*atms-of-ms A*)) *T*, *card* (*set-mset* ($clauses_{NOT}$ T)))
lemma $cdcl_{NOT}$*-decreasing-measure*:
assumes
$cdcl_{NOT}$ S T **and**
inv: *inv S* **and**
atm-clss: *atms-of-mm* ($clauses_{NOT}$ S) ⊆ *atms-of-ms A* **and**
atm-lits: *atm-of* ‘ *lits-of-l* (*trail S*) ⊆ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**

fin-A: *finite A*
shows $(\mu_{CDCL}\ A\ T,\ \mu_{CDCL}\ A\ S)$
$\in$ *less-than* <*lex*> (*less-than* <*lex*> *less-than*)
⟨*proof*⟩

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

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

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

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

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

lemma *rtranclp-cdcl*$_{NOT}$*-clauses-bound*:
assumes
cdcl$_{NOT}$** *S T* **and**
inv S **and**
atms-of-mm (*clauses*$_{NOT}$ *S*) ⊆ *A* **and**
atm-of '(*lits-of-l* (*trail S*)) ⊆ *A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite A*
shows *set-mset* (*clauses*$_{NOT}$ *T*) ⊆ *set-mset* (*clauses*$_{NOT}$ *S*) ∪ *simple-clss A*

⟨*proof*⟩

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

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

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

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

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

lemma *rtranclp-cdcl*$_{NOT}$-μ_{CDCL}*'-bound-reduce-trail-to*$_{NOT}$:
assumes
cdcl$_{NOT}$** *S T* **and**
inv S **and**
atms-of-mm (*clauses*$_{NOT}$ *S*) $\subseteq$ *atms-of-ms A* **and**
atm-of '(*lits-of-l* (*trail S*)) $\subseteq$ *atms-of-ms A* **and**

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

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

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

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

16.7 CDCL with restarts

16.7.1 Definition

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

end

locale *conflict-driven-clause-learning-with-restarts* =
conflict-driven-clause-learning mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss
trail raw-clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$
inv backjump-conds propagate-conds learn-cond forget-cond
for
mset-cls:: *'cls* ⇒ *'v clause* **and**
insert-cls :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
remove-lit :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
mset-clss:: *'clss* ⇒ *'v clauses* **and**
union-clss :: *'clss* ⇒ *'clss* ⇒ *'clss* **and**

in-clss :: *'cls* ⇒ *'clss* ⇒ *bool* **and**
insert-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
remove-from-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *marked-lits* **and**
raw-clauses :: *'st* ⇒ *'clss* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *marked-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
$add\text{-}cls_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
$remove\text{-}cls_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
inv :: *'st* ⇒ *bool* **and**
backjump-conds :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool* **and**
propagate-conds :: (*'v*, *unit*, *unit*) *marked-lit* ⇒ *'st* ⇒ *bool* **and**
learn-cond forget-cond :: *'cls* ⇒ *'st* ⇒ *bool*
begin

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

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

16.7.2 Increasing restarts

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

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

locale $cdcl_{NOT}$*-increasing-restarts-ops* =
restart-ops $cdcl_{NOT}$ *restart* **for**
restart :: *'st* ⇒ *'st* ⇒ *bool* **and**
$cdcl_{NOT}$ :: *'st* ⇒ *'st* ⇒ *bool* +
fixes
f :: *nat* ⇒ *nat* **and**
bound-inv :: *'bound* ⇒ *'st* ⇒ *bool* **and**

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

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

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

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

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

lemma $cdcl_{NOT}$-*comp-n-le*:
assumes
$(cdcl_{NOT}{}^{\frown\frown}(Suc\ n))\ S\ T$ **and**
bound-inv A S

$cdcl_{NOT}$-*inv* S
shows $\mu\ A\ T < \mu\ A\ S - n$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

shows $cdcl_{NOT}$*-inv* (*fst* T)
⟨*proof*⟩

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

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

locale $cdcl_{NOT}$*-increasing-restarts* =
$cdcl_{NOT}$*-increasing-restarts-ops restart* $cdcl_{NOT}$ *f bound-inv* μ $cdcl_{NOT}$*-inv* μ*-bound* +
dpll-state mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss
trail raw-clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$
for
mset-cls:: *'cls* ⇒ *'v clause* **and**
insert-cls :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
remove-lit :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
mset-clss:: *'clss* ⇒ *'v clauses* **and**
union-clss :: *'clss* ⇒ *'clss* ⇒ *'clss* **and**
in-clss :: *'cls* ⇒ *'clss* ⇒ *bool* **and**
insert-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
remove-from-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *marked-lits* **and**
raw-clauses :: *'st* ⇒ *'clss* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *marked-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
add-cls$_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
remove-cls$_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
f :: *nat* ⇒ *nat* **and**
restart :: *'st* ⇒ *'st* ⇒ *bool* **and**
bound-inv :: *'bound* ⇒ *'st* ⇒ *bool* **and**
μ :: *'bound* ⇒ *'st* ⇒ *nat* **and**
$cdcl_{NOT}$ :: *'st* ⇒ *'st* ⇒ *bool* **and**
$cdcl_{NOT}$*-inv* :: *'st* ⇒ *bool* **and**
μ*-bound* :: *'bound* ⇒ *'st* ⇒ *nat* +
assumes
measure-bound: $\bigwedge A\ T\ V\ n.$ $cdcl_{NOT}$*-inv* T $\Longrightarrow$ *bound-inv* A T
$\Longrightarrow$ $cdcl_{NOT}$*-restart* (T, n) (V, *Suc* n) $\Longrightarrow$ μ A V $\le$ μ*-bound* A T **and**
$cdcl_{NOT}$*-raw-restart-*μ*-bound*:
$cdcl_{NOT}$*-restart* (T, a) (V, b) $\Longrightarrow$ $cdcl_{NOT}$*-inv* T $\Longrightarrow$ *bound-inv* A T
$\Longrightarrow$ μ*-bound* A V $\le$ μ*-bound* A T
begin

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

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

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

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

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

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

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

end

16.8 Merging backjump and learning

locale $cdcl_{NOT}$*-merge-bj-learn-ops* =
decide-ops mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss
trail raw-clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ +
forget-ops mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss
trail raw-clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ *forget-cond* +
propagate-ops mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss
trail raw-clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ *propagate-conds*
for
mset-cls:: $'cls \Rightarrow$ $'v$ *clause* **and**

insert-cls :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
remove-lit :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
mset-clss:: *'clss* ⇒ *'v clauses* **and**
union-clss :: *'clss* ⇒ *'clss* ⇒ *'clss* **and**
in-clss :: *'cls* ⇒ *'clss* ⇒ *bool* **and**
insert-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
remove-from-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *marked-lits* **and**
raw-clauses :: *'st* ⇒ *'clss* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *marked-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
$add\text{-}cls_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
$remove\text{-}cls_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
propagate-conds :: (*'v*, *unit*, *unit*) *marked-lit* ⇒ *'st* ⇒ *bool* **and**
forget-cond :: *'cls* ⇒ *'st* ⇒ *bool* +
fixes *backjump-l-cond* :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool*
begin
inductive *backjump-l* **where**
backjump-l: $trail\ S = F' \mathbin{@} Marked\ K\ ()\ \#\ F$
$\Longrightarrow no\text{-}dup\ (trail\ S)$
$\Longrightarrow T \sim prepend\text{-}trail\ (Propagated\ L\ ())\ (reduce\text{-}trail\text{-}to_{NOT}\ F\ (add\text{-}cls_{NOT}\ C''\ S))$
$\Longrightarrow C \in\#\ clauses_{NOT}\ S$
$\Longrightarrow trail\ S \models as\ CNot\ C$
$\Longrightarrow undefined\text{-}lit\ F\ L$
$\Longrightarrow atm\text{-}of\ L \in atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \cup atm\text{-}of\ `\ (lits\text{-}of\text{-}l\ (trail\ S))$
$\Longrightarrow clauses_{NOT}\ S \models pm\ C' + \{\#L\#\}$
$\Longrightarrow mset\text{-}cls\ C'' = C' + \{\#L\#\}$
$\Longrightarrow F \models as\ CNot\ C'$
$\Longrightarrow backjump\text{-}l\text{-}cond\ C\ C'\ L\ S\ T$
$\Longrightarrow backjump\text{-}l\ S\ T$

Avoid (meaningless) simplification:

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

inductive $cdcl_{NOT}\text{-}merged\text{-}bj\text{-}learn$ :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
$cdcl_{NOT}\text{-}merged\text{-}bj\text{-}learn\text{-}decide_{NOT}$: $decide_{NOT}\ S\ S' \Longrightarrow cdcl_{NOT}\text{-}merged\text{-}bj\text{-}learn\ S\ S'$ |
$cdcl_{NOT}\text{-}merged\text{-}bj\text{-}learn\text{-}propagate_{NOT}$: $propagate_{NOT}\ S\ S' \Longrightarrow cdcl_{NOT}\text{-}merged\text{-}bj\text{-}learn\ S\ S'$ |
$cdcl_{NOT}\text{-}merged\text{-}bj\text{-}learn\text{-}backjump\text{-}l$: $backjump\text{-}l\ S\ S' \Longrightarrow cdcl_{NOT}\text{-}merged\text{-}bj\text{-}learn\ S\ S'$ |
$cdcl_{NOT}\text{-}merged\text{-}bj\text{-}learn\text{-}forget_{NOT}$: $forget_{NOT}\ S\ S' \Longrightarrow cdcl_{NOT}\text{-}merged\text{-}bj\text{-}learn\ S\ S'$

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

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

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

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

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

sublocale *dpll-with-backjumping-ops mset-cls insert-cls remove-lit*
  mset-clss union-clss in-clss insert-clss remove-from-clss
  trail raw-clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ *inv*
  backjump-conds propagate-conds
⟨*proof*⟩

end

locale $cdcl_{NOT}$*-merge-bj-learn-proxy2* =
  $cdcl_{NOT}$*-merge-bj-learn-proxy mset-cls insert-cls remove-lit*
  mset-clss union-clss in-clss insert-clss remove-from-clss
  trail raw-clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
  propagate-conds forget-cond backjump-l-cond inv
for
  mset-cls:: $'cls \Rightarrow 'v$ *clause* **and**

insert-cls :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
remove-lit :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
mset-clss:: *'clss* ⇒ *'v clauses* **and**
union-clss :: *'clss* ⇒ *'clss* ⇒ *'clss* **and**
in-clss :: *'cls* ⇒ *'clss* ⇒ *bool* **and**
insert-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
remove-from-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *marked-lits* **and**
raw-clauses :: *'st* ⇒ *'clss* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *marked-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
add-cls$_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
remove-cls$_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
propagate-conds :: (*'v*, *unit*, *unit*) *marked-lit* ⇒ *'st* ⇒ *bool* **and**
forget-cond :: *'cls* ⇒ *'st* ⇒ *bool* **and**
backjump-l-cond :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool* **and**
inv :: *'st* ⇒ *bool*
begin

sublocale *conflict-driven-clause-learning-ops mset-cls insert-cls remove-lit*
mset-clss union-clss in-clss insert-clss remove-from-clss
trail raw-clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$
inv backjump-conds propagate-conds
λC -. *distinct-mset* (*mset-cls* C) ∧ ¬*tautology* (*mset-cls* C)
forget-cond
⟨*proof*⟩
end

locale *cdcl*$_{NOT}$*-merge-bj-learn* =
cdcl$_{NOT}$*-merge-bj-learn-proxy2* *mset-cls insert-cls remove-lit*
mset-clss union-clss in-clss insert-clss remove-from-clss
trail raw-clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$
propagate-conds forget-cond backjump-l-cond inv
for
mset-cls:: *'cls* ⇒ *'v clause* **and**
insert-cls :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
remove-lit :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
mset-clss:: *'clss* ⇒ *'v clauses* **and**
union-clss :: *'clss* ⇒ *'clss* ⇒ *'clss* **and**
in-clss :: *'cls* ⇒ *'clss* ⇒ *bool* **and**
insert-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
remove-from-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *marked-lits* **and**
raw-clauses :: *'st* ⇒ *'clss* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *marked-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
add-cls$_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
remove-cls$_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
backjump-l-cond :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool* **and**
propagate-conds :: (*'v*, *unit*, *unit*) *marked-lit* ⇒ *'st* ⇒ *bool* **and**
forget-cond :: *'cls* ⇒ *'st* ⇒ *bool* **and**
inv :: *'st* ⇒ *bool* +
assumes
dpll-merge-bj-inv: $\bigwedge S\ T.$ *dpll-bj* $S\ T \Longrightarrow$ *inv* $S \Longrightarrow$ *inv* T **and**
learn-inv: $\bigwedge S\ T.$ *learn* $S\ T \Longrightarrow$ *inv* $S \Longrightarrow$ *inv* T

begin

sublocale
conflict-driven-clause-learning mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss
trail raw-clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
inv backjump-conds propagate-conds
λC -. *distinct-mset* (*mset-cls* C) $\wedge$ $\neg$*tautology* (*mset-cls* C)
forget-cond
⟨*proof*⟩

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

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

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

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

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

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

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

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

lemma *wf-*$cdcl_{NOT}$*-merged-bj-learn*:

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

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

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

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

lemma *cdcl*$_{NOT}$*-merged-bj-learn-final-state*:
fixes *A* :: *'v literal multiset set* **and** *S T* :: *'st*
assumes
n-s: *no-step cdcl*$_{NOT}$*-merged-bj-learn S* **and**
atms-S: *atms-of-mm* $($*clauses*$_{NOT}$ *S*$) \subseteq$ *atms-of-ms A* **and**
atms-trail: *atm-of* ‘ *lits-of-l* $($*trail S*$) \subseteq$ *atms-of-ms A* **and**
n-d: *no-dup* $($*trail S*$)$ **and**
finite A **and**
inv: *inv S* **and**
decomp: *all-decomposition-implies-m* $($*clauses*$_{NOT}$ *S*$)$ $($*get-all-marked-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*⟩

lemma *full-cdcl*$_{NOT}$*-merged-bj-learn-final-state*:
fixes *A* :: *'v literal multiset set* **and** *S T* :: *'st*
assumes
full: *full cdcl*$_{NOT}$*-merged-bj-learn S T* **and**
atms-S: *atms-of-mm* $($*clauses*$_{NOT}$ *S*$) \subseteq$ *atms-of-ms A* **and**
atms-trail: *atm-of* ‘ *lits-of-l* $($*trail S*$) \subseteq$ *atms-of-ms A* **and**
n-d: *no-dup* $($*trail S*$)$ **and**

finite A **and**
inv: *inv* S **and**
decomp: *all-decomposition-implies-m* ($clauses_{NOT}$ S) (*get-all-marked-decomposition* (*trail* S))
shows *unsatisfiable* (*set-mset* ($clauses_{NOT}$ T))
$\vee$ (*trail* T $\models$*asm* $clauses_{NOT}$ T $\wedge$ *satisfiable* (*set-mset* ($clauses_{NOT}$ T)))
⟨*proof*⟩

end

16.8.1 Instantiations

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

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

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

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

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

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

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

lemma *rtranclp-*$cdcl_{NOT}$*-restart-all-decomposition-implies*:
assumes $cdcl_{NOT}$*-restart*** S T **and**

inv: *inv* (*fst S*) **and**
n-d: *no-dup* (*trail* (*fst S*)) **and**
decomp:
all-decomposition-implies-m ($clauses_{NOT}$ (*fst S*)) (*get-all-marked-decomposition* (*trail* (*fst S*)))
shows
all-decomposition-implies-m ($clauses_{NOT}$ (*fst T*)) (*get-all-marked-decomposition* (*trail* (*fst T*)))
⟨*proof*⟩

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

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

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

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

locale $cdcl_{NOT}$*-merge-bj-learn-with-backtrack-restarts* =
$cdcl_{NOT}$*-merge-bj-learn mset-cls insert-cls remove-lit*
mset-clss union-clss in-clss insert-clss remove-from-clss
trail raw-clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$
$\lambda C\ C'\ L'\ S\ T.$ *distinct-mset* (C' + {#L'#}) ∧ *backjump-l-cond* $C\ C'\ L'\ S\ T$
propagate-conds forget-conds inv
for
mset-cls:: *'cls* ⇒ *'v clause* **and**
insert-cls :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
remove-lit :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
mset-clss:: *'clss* ⇒ *'v clauses* **and**
union-clss :: *'clss* ⇒ *'clss* ⇒ *'clss* **and**
in-clss :: *'cls* ⇒ *'clss* ⇒ *bool* **and**
insert-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**

remove-from-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *marked-lits* **and**
raw-clauses :: *'st* ⇒ *'clss* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *marked-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
$add\text{-}cls_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
$remove\text{-}cls_{NOT}$ :: *'cls* ⇒ *'st* ⇒ *'st* **and**
propagate-conds :: (*'v*, *unit*, *unit*) *marked-lit* ⇒ *'st* ⇒ *bool* **and**
inv :: *'st* ⇒ *bool* **and**
forget-conds :: *'cls* ⇒ *'st* ⇒ *bool* **and**
backjump-l-cond :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool*
+
fixes *f* :: *nat* ⇒ *nat*
assumes
unbounded: *unbounded f* **and** *f-ge-1*: $\bigwedge n.\ n \geq 1 \Longrightarrow f\ n \geq 1$ **and**
inv-restart:$\bigwedge S\ T.\ inv\ S \Longrightarrow T \sim reduce\text{-}trail\text{-}to_{NOT}\ []\ S \Longrightarrow inv\ T$
begin

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

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

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

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

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

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

finite[*simp*]: *finite A*
shows *set-mset* (*clauses*$_{NOT}$ *T*) ⊆ *set-mset* (*not-simplified-cls* (*clauses*$_{NOT}$ *S*))
∪ *simple-clss* (*atms-of-ms A*)
⟨*proof*⟩

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

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

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

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

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

sublocale *cdcl*$_{NOT}$*-increasing-restarts* - - - - - - - - - - - - - - *f*
λ*S T*. *T* ∼ *reduce-trail-to*$_{NOT}$ ([]::*'a list*) *S*
λ*A S*. *atms-of-mm* (*clauses*$_{NOT}$ *S*) ⊆ *atms-of-ms A*
∧ *atm-of* ' *lits-of-l* (*trail S*) ⊆ *atms-of-ms A* ∧ *finite A*

μ_{CDCL}'*-merged* $cdcl_{NOT}$*-merged-bj-learn*
$\lambda S.$ *inv* $S \wedge$ *no-dup* (*trail* S)
$\lambda A\ T.$ ((*2*+*card* (*atms-of-ms* A)) ^ (*1*+*card* (*atms-of-ms* A))) $*$ *2*
+ *card* (*set-mset* (*not-simplified-cls*($clauses_{NOT}$ T)))
+ *3* ^ *card* (*atms-of-ms* A)
⟨*proof*⟩

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

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

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

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

lemma *full*-$cdcl_{NOT}$*-restart-normal-form*:
assumes
full: *full* $cdcl_{NOT}$*-restart* S T **and**
inv: *inv* (*fst* S) **and** *n-d*: *no-dup*(*trail* (*fst* S)) **and**
decomp: *all-decomposition-implies-m* ($clauses_{NOT}$ (*fst* S))
(*get-all-marked-decomposition* (*trail* (*fst* S))) **and**
atms-cls: *atms-of-mm* ($clauses_{NOT}$ (*fst* S)) ⊆ *atms-of-ms* A **and**
atms-trail: *atm-of* ‘ *lits-of-l* (*trail* (*fst* S)) ⊆ *atms-of-ms* A **and**
fin: *finite* A
shows *unsatisfiable* (*set-mset* ($clauses_{NOT}$ (*fst* S)))
∨ *lits-of-l* (*trail* (*fst* T)) ⊨*sextm* $clauses_{NOT}$ (*fst* S) ∧ *satisfiable* (*set-mset* ($clauses_{NOT}$ (*fst* S)))
⟨*proof*⟩

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

end

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

17 DPLL as an instance of NOT

17.1 DPLL with simple backtrack

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

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

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

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

⟨*proof*⟩

sublocale *dpll-state*
id $\lambda L\ C.\ C + \{\#L\#\}$ *remove1-mset*
id *op* + *op* $\in\#$ $\lambda L\ C.\ C + \{\#L\#\}$ *remove1-mset*
fst snd $\lambda L\ (M, N).\ (L\ \#\ M, N)$ $\lambda(M, N).\ (tl\ M, N)$
$\lambda C\ (M, N).\ (M, \{\#C\#\} + N)$ $\lambda C\ (M, N).\ (M,$ *removeAll-mset* $C\ N)$
⟨*proof*⟩

sublocale *backjumping-ops*
id $\lambda L\ C.\ C + \{\#L\#\}$ *remove1-mset*
id *op* + *op* $\in\#$ $\lambda L\ C.\ C + \{\#L\#\}$ *remove1-mset*
fst snd $\lambda L\ (M, N).\ (L\ \#\ M, N)$ $\lambda(M, N).\ (tl\ M, N)$
$\lambda C\ (M, N).\ (M, \{\#C\#\} + N)$ $\lambda C\ (M, N).\ (M,$ *removeAll-mset* $C\ N)$ λ- - - $S\ T.$ *backtrack* $S\ T$
⟨*proof*⟩

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

lemma *reduce-trail-to*$_{NOT}$:
reduce-trail-to$_{NOT}$ $F\ S$ =
(*if length* (*fst* S) $\geq$ *length* F
then drop (*length* (*fst* S) $-$ *length* F) (*fst* S)
else [],
snd S) (**is** *?R* = *?C*)
⟨*proof*⟩

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

lemma *can-do-bt-step*:
assumes
M: *fst* $S = F'$ @ *Marked* $K\ d\ \#\ F$ **and**
$C \in\#$ *snd* S **and**
C: *fst* $S \models$*as CNot* C
shows $\neg$ *no-step backtrack* S
⟨*proof*⟩

end

sublocale *dpll-with-backtrack* $\subseteq$ *dpll-with-backjumping-ops*
id $\lambda L\ C.\ C + \{\#L\#\}$ *remove1-mset*
id op + *op* $\in\#$ $\lambda L\ C.\ C + \{\#L\#\}$ *remove1-mset*
fst snd $\lambda L\ (M, N).\ (L\ \#\ M, N)$
$\lambda(M, N).\ (tl\ M, N)$ $\lambda C\ (M, N).\ (M, \{\#C\#\} + N)$ $\lambda C\ (M, N).\ (M,$ *removeAll-mset* $C\ N)$
$\lambda(M, N).$ *no-dup* $M \wedge$ *all-decomposition-implies-m* N (*get-all-marked-decomposition* M)
λ- - - $S\ T.$ *backtrack* $S\ T$
λ- -. *True*

$\langle proof \rangle$

sublocale *dpll-with-backtrack* $\subseteq$ *dpll-with-backjumping*
id $\lambda L\ C.\ C + \{\#L\#\}$ *remove1-mset*
id *op* $+$ *op* $\in\#$ $\lambda L\ C.\ C + \{\#L\#\}$ *remove1-mset*
fst snd $\lambda L\ (M, N).\ (L \# M, N)$
$\lambda(M, N).\ (tl\ M, N)\ \lambda C\ (M, N).\ (M, \{\#C\#\} + N)\ \lambda C\ (M, N).\ (M, removeAll\text{-}mset\ C\ N)$
$\lambda(M, N).$ *no-dup* $M \wedge$ *all-decomposition-implies-m* N (*get-all-marked-decomposition* M)
λ- - - $S\ T$. *backtrack* $S\ T$
λ- -. *True*
$\langle proof \rangle$

context *dpll-with-backtrack*
begin
term *learn*
end

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

corollary *full-dpll-final-state-conclusive*:
fixes $M\ M'$ :: $('v, unit, unit)$ *marked-lit list*
assumes
full: *full dpll-bj* $([], N)\ (M', N')$
shows *unsatisfiable* (*set-mset* N) $\vee$ ($M' \models asm\ N \wedge$ *satisfiable* (*set-mset* N))
$\langle proof \rangle$

corollary *full-dpll-normal-form-from-init-state*:
fixes $M\ M'$ :: $('v, unit, unit)$ *marked-lit list*
assumes
full: *full dpll-bj* $([], N)\ (M', N')$
shows $M' \models asm\ N \longleftrightarrow$ *satisfiable* (*set-mset* N)
$\langle proof \rangle$

interpretation *conflict-driven-clause-learning-ops*
id $\lambda L\ C.\ C + \{\#L\#\}$ *remove1-mset*
id *op* $+$ *op* $\in\#$ $\lambda L\ C.\ C + \{\#L\#\}$ *remove1-mset*
fst snd $\lambda L\ (M, N).\ (L \# M, N)$
$\lambda(M, N).\ (tl\ M, N)\ \lambda C\ (M, N).\ (M, \{\#C\#\} + N)\ \lambda C\ (M, N).\ (M, removeAll\text{-}mset\ C\ N)$
$\lambda(M, N).$ *no-dup* $M \wedge$ *all-decomposition-implies-m* N (*get-all-marked-decomposition* M)
λ- - - $S\ T$. *backtrack* $S\ T$
λ- -. *True* λ- -. *False* λ- -. *False*
$\langle proof \rangle$

interpretation *conflict-driven-clause-learning*
id $\lambda L\ C.\ C + \{\#L\#\}$ *remove1-mset*
id *op* $+$ *op* $\in\#$ $\lambda L\ C.\ C + \{\#L\#\}$ *remove1-mset*
fst snd $\lambda L\ (M, N).\ (L \# M, N)$
$\lambda(M, N).\ (tl\ M, N)\ \lambda C\ (M, N).\ (M, \{\#C\#\} + N)\ \lambda C\ (M, N).\ (M, removeAll\text{-}mset\ C\ N)$

$\lambda(M, N)$. *no-dup* $M \wedge$ *all-decomposition-implies-m* N (*get-all-marked-decomposition* M)
λ- - - S T. *backtrack* S T
λ- -. *True* λ- -. *False* λ- -. *False*
⟨*proof*⟩

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

Another proof of termination:

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

17.2 Adding restarts

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

locale *dpll-withbacktrack-and-restarts* =
dpll-with-backtrack +
fixes f :: *nat* $\Rightarrow$ *nat*
assumes *unbounded*: *unbounded* f **and** *f-ge-1*:$\bigwedge n.\ n \geq 1 \Longrightarrow f\ n \geq 1$
begin
sublocale $cdcl_{NOT}$*-increasing-restarts*
id λL C. C + $\{\#L\#\}$ *remove1-mset*
id *op* + *op* $\in\#$ λL C. C + $\{\#L\#\}$ *remove1-mset*
fst *snd* λL (M, N). $(L \# M, N)$ $\lambda(M, N)$. (*tl* M, N)
λC (M, N). $(M, \{\#C\#\} + N)$ λC (M, N). (M, *removeAll-mset* C N) f λ(-, N) S. S = ([], N)
λA (M, N). *atms-of-mm* $N \subseteq$ *atms-of-ms* $A \wedge$ *atm-of* ‘ *lits-of-l* $M \subseteq$ *atms-of-ms* $A \wedge$ *finite* A
$\wedge$ *all-decomposition-implies-m* N (*get-all-marked-decomposition* M)
λA T. (2+*card* (*atms-of-ms* A)) ^ (1+*card* (*atms-of-ms* A))
$- \mu_C$ (1+*card* (*atms-of-ms* A)) (2+*card* (*atms-of-ms* A)) (*trail-weight* T) *dpll-bj*
$\lambda(M, N)$. *no-dup* $M \wedge$ *all-decomposition-implies-m* N (*get-all-marked-decomposition* M)
λA -. (2+*card* (*atms-of-ms* A)) ^ (1+*card* (*atms-of-ms* A))
⟨*proof*⟩
end

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

18 DPLL

18.1 Rules

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

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

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

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

18.2 Invariants

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

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

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

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

Lemma theorem 2.8.2 page 71 of CW

lemma $dpll_W$*-propagate-is-conclusion*:
assumes $dpll_W\ S\ S'$
and *all-decomposition-implies-m* (*clauses* S) (*get-all-marked-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-marked-decomposition* (*trail* S'))
⟨*proof*⟩

Lemma theorem 2.8.3 page 72 of CW

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

Lemma theorem 2.8.4 page 72 of CW

lemma *only-propagated-vars-unsat*:
assumes *marked*: $\forall x \in$ *set* M. $\neg$ *is-marked* x
and *DN*: $D \in N$ **and** *D*: $M \models as$ *CNot* D
and *inv*: *all-decomposition-implies* N (*get-all-marked-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-marked-decomposition* (*trail S*))
and *atm-incl*: *atm-of* ‘ *lits-of-l* (*trail S*) $\subseteq$ *atms-of-mm* (*clauses S*)
and *consistent-interp* (*lits-of-l* (*trail S*))
and *no-dup* (*trail S*)
shows *all-decomposition-implies-m* (*clauses S'*) (*get-all-marked-decomposition* (*trail S'*))
and *atm-of* ‘ *lits-of-l* (*trail S'*) $\subseteq$ *atms-of-mm* (*clauses S'*)
and *clauses S* = *clauses S'*
and *consistent-interp* (*lits-of-l* (*trail S'*))
and *no-dup* (*trail S'*)
⟨*proof*⟩

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

lemma *dpll*$_W$*-all-inv-dest*[*dest*]:
assumes *dpll*$_W$*-all-inv S*
shows *all-decomposition-implies-m* (*clauses S*) (*get-all-marked-decomposition* (*trail S*))
and *atm-of* ‘ *lits-of-l* (*trail S*) $\subseteq$ *atms-of-mm* (*clauses S*)
and *consistent-interp* (*lits-of-l* (*trail S*)) $\wedge$ *no-dup* (*trail S*)
⟨*proof*⟩

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

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

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

lemma *dpll*$_W$*-can-do-step*:
assumes *consistent-interp* (*set M*)

and *distinct M*
and *atm-of* ‘ (*set M*) $\subseteq$ *atms-of-mm N*
shows *rtranclp* $dpll_W$ ([], *N*) (*map* (λM. *Marked M* ()) *M*, *N*)
⟨*proof*⟩

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

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

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

18.3 Termination

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

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

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

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

Proposition theorem 2.8.7 page 73 of CW

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

⟨*proof*⟩

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

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

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

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

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

18.4 Final States

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

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

18.5 Link with NOT's DPLL

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

declare $dpll_W$*-*$_{NOT}$*.state-simp*$_{NOT}$[*simp del*]
lemma *state-eq*$_{NOT}$*-iff-eq*[*iff*, *simp*]: $dpll_W$*-*$_{NOT}$*.state-eq*$_{NOT}$ *S* *T* ⟷ *S* = *T*
  ⟨*proof*⟩
lemma $dpll_W$*-*$dpll_W$*-bj*:
  assumes *inv*: $dpll_W$*-all-inv* *S* **and** *dpll*: $dpll_W$ *S* *T*
  shows $dpll_W$*-*$_{NOT}$*.dpll-bj* *S* *T*
  ⟨*proof*⟩

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

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

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

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

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

18.5.1 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.

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

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

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

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

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

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

lemma *get-rev-level-skip*[*simp*]:
  assumes *atm-of L* ∉ *atm-of* ‘ *lits-of-l M*
  shows *get-rev-level* (*M* @ *Marked K i* # *M*′) *n L* = *get-rev-level* (*Marked K i* # *M*′) *i L*
  ⟨*proof*⟩

lemma *get-rev-level-notin-end*[*simp*]:
  assumes *atm-of L* ∉ *atm-of* ‘ *lits-of-l M*′
  shows *get-rev-level* (*M* @ *M*′) *n L* = *get-rev-level M n L*
  ⟨*proof*⟩

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

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

assumes *atm-of L* ∈ *atm-of* ' *lits-of-l M*
shows *get-rev-level* (*M* @ *M'*) *n L* = *get-rev-level M n L*
⟨*proof*⟩

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

lemma *get-level-skip-beginning-not-marked-rev*:
assumes *atm-of L* ∉ *atm-of* ' *lit-of* '(*set S*)
and ∀ *s*∈*set S*. ¬*is-marked s*
shows *get-level* (*M* @ *rev S*) *L* = *get-level M L*
⟨*proof*⟩

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

lemma *get-rev-level-skip-beginning-not-marked*[*simp*]:
assumes *atm-of L* ∉ *atm-of* ' *lit-of* '(*set S*)
and ∀ *s*∈*set S*. ¬*is-marked s*
shows *get-rev-level* (*rev S* @ *rev M*) *0 L* = *get-level M L*
⟨*proof*⟩

lemma *get-level-skip-in-all-not-marked*:
fixes *M* :: (*'a*, *nat*, *'b*) *marked-lit list* **and** *L* :: *'a literal*
assumes ∀ *m*∈*set M*. ¬ *is-marked m*
and *atm-of L* ∈ *atm-of* ' *lit-of* ' (*set M*)
shows *get-rev-level M n L* = *n*
⟨*proof*⟩

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

abbreviation *MMax M* ≡ *Max* (*set-mset M*)

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

definition *get-maximum-level* :: (*'a*, *nat*, *'b*) *marked-lit list* ⇒ *'a literal multiset* ⇒ *nat*
where
get-maximum-level M D = *MMax* ({#*0*#} + *image-mset* (*get-level M*) *D*)

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

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

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

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

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

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

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

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

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

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

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

lemma *get-maximum-level-skip-un-marked-not-present*:
assumes $\forall L \in\# D.$ *atm-of L* $\in$ *atm-of* ‘ *lits-of-l aa* **and**
$\forall m \in$ *set M*. ¬ *is-marked m*
shows *get-maximum-level aa D* = *get-maximum-level* (*M* @ *aa*) *D*
⟨*proof*⟩

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

fun *get-maximum-possible-level*:: ('*b*, *nat*, '*c*) *marked-lit list* ⇒ *nat* **where**
get-maximum-possible-level [] = *0* |

get-maximum-possible-level (*Marked K i* # *l*) = *max i* (*get-maximum-possible-level l*) |
get-maximum-possible-level (*Propagated - -* # *l*) = *get-maximum-possible-level l*

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

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

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

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

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

fun *get-all-mark-of-propagated* **where**
get-all-mark-of-propagated [] = [] |
get-all-mark-of-propagated (*Marked - -* # *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*⟩

18.5.2 Properties about the levels

fun *get-all-levels-of-marked* :: (*'b*, *'a*, *'c*) *marked-lit list* ⇒ *'a list* **where**
get-all-levels-of-marked [] = [] |
get-all-levels-of-marked (*Marked l level* # *Ls*) = *level* # *get-all-levels-of-marked Ls* |
get-all-levels-of-marked (*Propagated - -* # *Ls*) = *get-all-levels-of-marked Ls*

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

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

lemma *get-all-levels-of-marked-append*[*simp*]:
get-all-levels-of-marked (*a* @ *b*) = *get-all-levels-of-marked a* @ *get-all-levels-of-marked b*
⟨*proof*⟩

lemma *in-get-all-levels-of-marked-iff-decomp*:
i ∈ *set* (*get-all-levels-of-marked M*) ⟷ (∃ *c K c'*. *M* = *c* @ *Marked K i* # *c'*) (**is** *?A* ⟷ *?B*)
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

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

begin

19 Weidenbach's CDCL

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

19.1 The State

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

locale $state_W$*-ops* =
raw-clss mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss
\+
raw-ccls-union mset-ccls union-ccls insert-ccls remove-clit
for
— Clause
mset-cls:: *'cls* ⇒ *'v clause* **and**
insert-cls :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
remove-lit :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**

— Multiset of Clauses
mset-clss:: *'clss* ⇒ *'v clauses* **and**
union-clss :: *'clss* ⇒ *'clss* ⇒ *'clss* **and**
in-clss :: *'cls* ⇒ *'clss* ⇒ *bool* **and**
insert-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
remove-from-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**

mset-ccls:: *'ccls* ⇒ *'v clause* **and**
union-ccls :: *'ccls* ⇒ *'ccls* ⇒ *'ccls* **and**
insert-ccls :: *'v literal* ⇒ *'ccls* ⇒ *'ccls* **and**
remove-clit :: *'v literal* ⇒ *'ccls* ⇒ *'ccls*
\+
fixes
ccls-of-cls :: *'cls* ⇒ *'ccls* **and**
cls-of-ccls :: *'ccls* ⇒ *'cls* **and**

trail :: *'st* ⇒ (*'v*, *nat*, *'v clause*) *marked-lits* **and**
hd-raw-trail :: *'st* ⇒ (*'v*, *nat*, *'cls*) *marked-lit* **and**
raw-init-clss :: *'st* ⇒ *'clss* **and**
raw-learned-clss :: *'st* ⇒ *'clss* **and**
backtrack-lvl :: *'st* ⇒ *nat* **and**
raw-conflicting :: *'st* ⇒ *'ccls option* **and**

cons-trail :: (*'v*, *nat*, *'cls*) *marked-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒ *'st* **and**
add-init-cls :: *'cls* ⇒ *'st* ⇒ *'st* **and**
add-learned-cls :: *'cls* ⇒ *'st* ⇒ *'st* **and**
remove-cls :: *'cls* ⇒ *'st* ⇒ *'st* **and**

update-backtrack-lvl :: *nat* ⇒ *'st* ⇒ *'st* **and**
update-conflicting :: *'ccls option* ⇒ *'st* ⇒ *'st* **and**

init-state :: *'clss* ⇒ *'st* **and**
restart-state :: *'st* ⇒ *'st*
assumes
mset-ccls-ccls-of-cls[*simp*]:
mset-ccls (*ccls-of-cls* C) = *mset-cls* C **and**
mset-cls-cls-of-ccls[*simp*]:
mset-cls (*cls-of-ccls* D) = *mset-ccls* D **and**
ex-mset-cls: ∃ *a*. *mset-cls a* = E
begin
fun *mmset-of-mlit* :: (*'a*, *'b*, *'cls*) *marked-lit* ⇒ (*'a*, *'b*, *'v clause*) *marked-lit*
where
mmset-of-mlit (*Propagated L C*) = *Propagated L* (*mset-cls C*) |
mmset-of-mlit (*Marked L i*) = *Marked L i*

lemma *lit-of-mmset-of-mlit*[*simp*]:
lit-of (*mmset-of-mlit a*) = *lit-of a*
⟨*proof*⟩

lemma *lit-of-mmset-of-mlit-set-lit-of-l*[*simp*]:
lit-of ‘ *mmset-of-mlit* ‘ *set M'* = *lits-of-l M'*
⟨*proof*⟩

lemma *map-mmset-of-mlit-true-annots-true-cls*[*simp*]:
map mmset-of-mlit M' ⊨*as C* ⟷ *M'* ⊨*as C*
⟨*proof*⟩

abbreviation *init-clss* ≡ λ*S*. *mset-clss* (*raw-init-clss S*)
abbreviation *learned-clss* ≡ λ*S*. *mset-clss* (*raw-learned-clss S*)
abbreviation *conflicting* ≡ λ*S*. *map-option mset-ccls* (*raw-conflicting S*)

notation *insert-cls* (**infix** !++ *50*)

notation *in-clss* (**infix** !∈! *50*)
notation *union-clss* (**infix** ⊕ *50*)
notation *insert-clss* (**infix** !++! *50*)

notation *union-ccls* (**infix** !∪ *50*)

definition *raw-clauses* :: *'st* ⇒ *'clss* **where**
raw-clauses S = *union-clss* (*raw-init-clss S*) (*raw-learned-clss S*)

abbreviation *clauses* :: *'st* ⇒ *'v clauses* **where**
clauses S ≡ *mset-clss* (*raw-clauses S*)

end

locale $state_W$ =
$state_W$*-ops*
— functions for clauses:
mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss

— functions for the conflicting clause:
mset-ccls union-ccls insert-ccls remove-clit

— Conversion between conflicting and non-conflicting
ccls-of-cls cls-of-ccls

— functions for the state:
— access functions:
trail hd-raw-trail raw-init-clss raw-learned-clss backtrack-lvl raw-conflicting
— changing state:
cons-trail tl-trail add-init-cls add-learned-cls remove-cls update-backtrack-lvl update-conflicting

— get state:
init-state
restart-state
for
mset-cls:: $'cls \Rightarrow 'v$ *clause* **and**
insert-cls :: $'v$ *literal* $\Rightarrow 'cls \Rightarrow 'cls$ **and**
remove-lit :: $'v$ *literal* $\Rightarrow 'cls \Rightarrow 'cls$ **and**

mset-clss:: $'clss \Rightarrow 'v$ *clauses* **and**
union-clss :: $'clss \Rightarrow 'clss \Rightarrow 'clss$ **and**
in-clss :: $'cls \Rightarrow 'clss \Rightarrow$ *bool* **and**
insert-clss :: $'cls \Rightarrow 'clss \Rightarrow 'clss$ **and**
remove-from-clss :: $'cls \Rightarrow 'clss \Rightarrow 'clss$ **and**

mset-ccls:: $'ccls \Rightarrow 'v$ *clause* **and**
union-ccls :: $'ccls \Rightarrow 'ccls \Rightarrow 'ccls$ **and**
insert-ccls :: $'v$ *literal* $\Rightarrow 'ccls \Rightarrow 'ccls$ **and**
remove-clit :: $'v$ *literal* $\Rightarrow 'ccls \Rightarrow 'ccls$ **and**

ccls-of-cls :: $'cls \Rightarrow 'ccls$ **and**
cls-of-ccls :: $'ccls \Rightarrow 'cls$ **and**

trail :: $'st \Rightarrow ('v, nat, 'v\ clause)$ *marked-lits* **and**
hd-raw-trail :: $'st \Rightarrow ('v, nat, 'cls)$ *marked-lit* **and**
raw-init-clss :: $'st \Rightarrow 'clss$ **and**
raw-learned-clss :: $'st \Rightarrow 'clss$ **and**
backtrack-lvl :: $'st \Rightarrow nat$ **and**
raw-conflicting :: $'st \Rightarrow 'ccls$ *option* **and**

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

init-state :: $'clss \Rightarrow 'st$ **and**
restart-state :: $'st \Rightarrow 'st$ +
assumes
hd-raw-trail: *trail* $S \neq [] \Longrightarrow$ *mmset-of-mlit* (*hd-raw-trail* S) = *hd* (*trail* S) **and**
trail-cons-trail[*simp*]:

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

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

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

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

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

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

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

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

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

clss-tl-trail[*simp*]: $clauses\ (tl\text{-}trail\ S) = clauses\ S$ **and**
clauses-add-learned-cls-unfolded:
$no\text{-}dup\ (trail\ S) \Longrightarrow clauses\ (add\text{-}learned\text{-}cls\ U\ S) =$
$\{\#mset\text{-}cls\ U\#\} + learned\text{-}clss\ S + init\text{-}clss\ S$
and
clauses-add-init-cls[*simp*]:
$no\text{-}dup\ (trail\ S) \Longrightarrow$
$clauses\ (add\text{-}init\text{-}cls\ N\ S) = \{\#mset\text{-}cls\ N\#\} + init\text{-}clss\ S + learned\text{-}clss\ S$ **and**
clauses-update-backtrack-lvl[*simp*]: $clauses\ (update\text{-}backtrack\text{-}lvl\ k\ S) = clauses\ S$ **and**
clauses-update-conflicting[*simp*]: $clauses\ (update\text{-}conflicting\ D\ S) = clauses\ S$ **and**
clauses-remove-cls[*simp*]:

$clauses$ $(remove\text{-}cls\ C\ S) = removeAll\text{-}mset\ (mset\text{-}cls\ C)\ (clauses\ S)$ **and**
$clauses\text{-}add\text{-}learned\text{-}cls[simp]$:
$no\text{-}dup\ (trail\ S) \Longrightarrow clauses\ (add\text{-}learned\text{-}cls\ C\ S) = \{\#mset\text{-}cls\ C\#\} + clauses\ S$ **and**
$clauses\text{-}restart[simp]$: $clauses\ (restart\text{-}state\ S) \subseteq\#\ clauses\ S$ **and**
$clauses\text{-}init\text{-}state[simp]$: $\bigwedge N.\ clauses\ (init\text{-}state\ N) = mset\text{-}clss\ N$
⟨*proof*⟩

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

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

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

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

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

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

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

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

lemmas $state\text{-}simp[simp] = state\text{-}eq\text{-}trail\ state\text{-}eq\text{-}init\text{-}clss\ state\text{-}eq\text{-}learned\text{-}clss\ state\text{-}eq\text{-}backtrack\text{-}lvl\ state\text{-}eq\text{-}conflicting\ state\text{-}eq\text{-}clauses\ state\text{-}eq\text{-}undefined\text{-}lit\ state\text{-}eq\text{-}raw\text{-}conflicting\text{-}None$

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

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

⟨*proof*⟩
termination
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *raw-conflicting-add-init-cls*[*simp*]:
  no-dup (*trail* S) $\Longrightarrow$
    raw-conflicting (*add-init-cls* C S) = *None* $\longleftrightarrow$ *raw-conflicting* S = *None*
  ⟨*proof*⟩

lemma *raw-conflicting-add-learned-cls*[*simp*]:
  no-dup (*trail* S) $\Longrightarrow$
    raw-conflicting (*add-learned-cls* C S) = *None* $\longleftrightarrow$ *raw-conflicting* S = *None*
  ⟨*proof*⟩

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

end — end of $state_W$ locale

19.2 CDCL Rules

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

locale $conflict\text{-}driven\text{-}clause\text{-}learning_W$ =
  $state_W$
    — functions for clauses:
    mset-cls insert-cls remove-lit
    mset-clss union-clss in-clss insert-clss remove-from-clss

    — functions for the conflicting clause:
    mset-ccls union-ccls insert-ccls remove-clit

    — conversion
    ccls-of-cls cls-of-ccls

    — functions for the state:
      — access functions:
    trail hd-raw-trail raw-init-clss raw-learned-clss backtrack-lvl raw-conflicting
      — changing state:
    cons-trail tl-trail add-init-cls add-learned-cls remove-cls update-backtrack-lvl
    update-conflicting

      — get state:
    init-state
    restart-state
  for
    mset-cls:: $'cls \Rightarrow 'v$ *clause* **and**

$insert\text{-}cls :: 'v\ literal \Rightarrow 'cls \Rightarrow 'cls$ **and**
$remove\text{-}lit :: 'v\ literal \Rightarrow 'cls \Rightarrow 'cls$ **and**

$mset\text{-}clss:: 'clss \Rightarrow 'v\ clauses$ **and**
$union\text{-}clss :: 'clss \Rightarrow 'clss \Rightarrow 'clss$ **and**
$in\text{-}clss :: 'cls \Rightarrow 'clss \Rightarrow bool$ **and**
$insert\text{-}clss :: 'cls \Rightarrow 'clss \Rightarrow 'clss$ **and**
$remove\text{-}from\text{-}clss :: 'cls \Rightarrow 'clss \Rightarrow 'clss$ **and**

$mset\text{-}ccls:: 'ccls \Rightarrow 'v\ clause$ **and**
$union\text{-}ccls :: 'ccls \Rightarrow 'ccls \Rightarrow 'ccls$ **and**
$insert\text{-}ccls :: 'v\ literal \Rightarrow 'ccls \Rightarrow 'ccls$ **and**
$remove\text{-}clit :: 'v\ literal \Rightarrow 'ccls \Rightarrow 'ccls$ **and**

$ccls\text{-}of\text{-}cls :: 'cls \Rightarrow 'ccls$ **and**
$cls\text{-}of\text{-}ccls :: 'ccls \Rightarrow 'cls$ **and**

$trail :: 'st \Rightarrow ('v, nat, 'v\ clause)\ marked\text{-}lits$ **and**
$hd\text{-}raw\text{-}trail :: 'st \Rightarrow ('v, nat, 'cls)\ marked\text{-}lit$ **and**
$raw\text{-}init\text{-}clss :: 'st \Rightarrow 'clss$ **and**
$raw\text{-}learned\text{-}clss :: 'st \Rightarrow 'clss$ **and**
$backtrack\text{-}lvl :: 'st \Rightarrow nat$ **and**
$raw\text{-}conflicting :: 'st \Rightarrow 'ccls\ option$ **and**

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

$init\text{-}state :: 'clss \Rightarrow 'st$ **and**
$restart\text{-}state :: 'st \Rightarrow 'st$
begin

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

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

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

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

inductive *backtrack* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
backtrack-rule:
  raw-conflicting S = Some D ⟹
  L ∈# *mset-ccls D* ⟹
  (*Marked K* (*i+1*) # *M1*, *M2*) ∈ *set* (*get-all-marked-decomposition* (*trail S*)) ⟹
  get-level (*trail S*) *L* = *backtrack-lvl S* ⟹
  get-level (*trail S*) *L* = *get-maximum-level* (*trail S*) (*mset-ccls D*) ⟹
  get-maximum-level (*trail S*) (*mset-ccls* (*remove-clit L D*)) ≡ *i* ⟹
  T ∼ *cons-trail* (*Propagated L* (*cls-of-ccls D*))
            (*reduce-trail-to M1*
              (*add-learned-cls* (*cls-of-ccls D*)
                (*update-backtrack-lvl i*
                  (*update-conflicting None S*)))) ⟹
  backtrack S T

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

inductive *decide* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
decide-rule:
  conflicting S = None ⟹
  undefined-lit (*trail S*) *L* ⟹
  atm-of L ∈ *atms-of-mm* (*init-clss S*) ⟹
  T ∼ *cons-trail* (*Marked L* (*backtrack-lvl S +1*)) (*incr-lvl 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* ⟹
   raw-conflicting S = Some E ⟹
   −*L* ∉# *mset-ccls E* ⟹
   mset-ccls 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-raw-trail S = *Propagated L E* ⟹
  L ∈# *mset-cls E* ⟹
  raw-conflicting S = Some D' ⟹
  −*L* ∈# *mset-ccls D'* ⟹
  get-maximum-level (*trail S*) (*mset-ccls* (*remove-clit* (−*L*) *D'*)) = *backtrack-lvl S* ⟹
  T ∼ *update-conflicting* (*Some* (*union-ccls* (*remove-clit* (−*L*) *D'*) (*ccls-of-cls* (*remove-lit L E*))))
   (*tl-trail S*) ⟹
  resolve S T

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

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

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

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

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

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

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

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

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

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

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

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

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

restart: $\bigwedge T.$ *restart* $S\ T \Longrightarrow P\ S\ T$ **and**
decide: $\bigwedge T.$ *decide* $S\ T \Longrightarrow P\ S\ T$ **and**
skip: $\bigwedge T.$ *skip* $S\ T \Longrightarrow P\ S\ T$ **and**
resolve: $\bigwedge T.$ *resolve* $S\ T \Longrightarrow P\ S\ T$ **and**
backtrack: $\bigwedge T.$ *backtrack* $S\ T \Longrightarrow P\ S\ T$
shows $P\ S\ S'$
⟨*proof*⟩

lemma $cdcl_W$*-all-induct*[*consumes 1, case-names propagate conflict forget restart decide skip resolve backtrack*]:
fixes S :: $'st$
assumes
$cdcl_W$: $cdcl_W\ S\ S'$ **and**
propagateH: $\bigwedge C\ L\ T.$ *conflicting* S = *None* $\Longrightarrow$
C !∈! *raw-clauses* $S \Longrightarrow$
$L \in\#$ *mset-cls* $C \Longrightarrow$
trail $S \models as$ *CNot* (*remove1-mset* L (*mset-cls* C)) $\Longrightarrow$
undefined-lit (*trail* S) $L \Longrightarrow$
$T \sim$ *cons-trail* (*Propagated* L C) $S \Longrightarrow$
$P\ S\ T$ **and**
conflictH: $\bigwedge D\ T.$ *conflicting* S = *None* $\Longrightarrow$
D !∈! *raw-clauses* $S \Longrightarrow$
trail $S \models as$ *CNot* (*mset-cls* D) $\Longrightarrow$
$T \sim$ *update-conflicting* (*Some* (*ccls-of-cls* D)) $S \Longrightarrow$
$P\ S\ T$ **and**
forgetH: $\bigwedge C\ U\ T.$ *conflicting* S = *None* $\Longrightarrow$
C !∈! *raw-learned-clss* $S \Longrightarrow$
$\neg$(*trail* S) $\models asm$ *clauses* $S \Longrightarrow$
mset-cls $C \notin$ *set* (*get-all-mark-of-propagated* (*trail* S)) $\Longrightarrow$
mset-cls $C \notin\#$ *init-clss* $S \Longrightarrow$
$T \sim$ *remove-cls* $C\ S \Longrightarrow$
$P\ S\ T$ **and**
restartH: $\bigwedge T.$ $\neg$*trail* $S \models asm$ *clauses* $S \Longrightarrow$
conflicting S = *None* $\Longrightarrow$
$T \sim$ *restart-state* $S \Longrightarrow$
$P\ S\ T$ **and**
decideH: $\bigwedge L\ T.$ *conflicting* S = *None* $\Longrightarrow$
undefined-lit (*trail* S) $L \Longrightarrow$
atm-of $L \in$ *atms-of-mm* (*init-clss* S) $\Longrightarrow$
$T \sim$ *cons-trail* (*Marked* L (*backtrack-lvl* S +*1*)) (*incr-lvl* S) $\Longrightarrow$
$P\ S\ T$ **and**
skipH: $\bigwedge L\ C'\ M\ E\ T.$
trail S = *Propagated* $L\ C'$ # $M \Longrightarrow$
raw-conflicting S = *Some* $E \Longrightarrow$
$-L \notin\#$ *mset-ccls* $E \Longrightarrow$ *mset-ccls* $E \neq \{\#\} \Longrightarrow$
$T \sim$ *tl-trail* $S \Longrightarrow$
$P\ S\ T$ **and**
resolveH: $\bigwedge L\ E\ M\ D\ T.$
trail S = *Propagated* L (*mset-cls* E) # $M \Longrightarrow$
$L \in\#$ *mset-cls* $E \Longrightarrow$
hd-raw-trail S = *Propagated* $L\ E \Longrightarrow$
raw-conflicting S = *Some* $D \Longrightarrow$
$-L \in\#$ *mset-ccls* $D \Longrightarrow$
get-maximum-level (*trail* S) (*mset-ccls* (*remove-clit* ($-L$) D)) = *backtrack-lvl* $S \Longrightarrow$
$T \sim$ *update-conflicting*

(*Some* (*union-ccls* (*remove-clit* (−*L*) *D*) (*ccls-of-cls* (*remove-lit* *L* *E*)))) (*tl-trail* *S*) $\Longrightarrow$
P S T **and**
backtrackH: $\bigwedge$*L D K i M1 M2 T*.
raw-conflicting *S* = *Some* *D* $\Longrightarrow$
L ∈# *mset-ccls* *D* $\Longrightarrow$
(*Marked* *K* (*i*+*1*) # *M1*, *M2*) ∈ *set* (*get-all-marked-decomposition* (*trail* *S*)) $\Longrightarrow$
get-level (*trail* *S*) *L* = *backtrack-lvl* *S* $\Longrightarrow$
get-level (*trail* *S*) *L* = *get-maximum-level* (*trail* *S*) (*mset-ccls* *D*) $\Longrightarrow$
get-maximum-level (*trail* *S*) (*remove1-mset* *L* (*mset-ccls* *D*)) ≡ *i* $\Longrightarrow$
T ∼ *cons-trail* (*Propagated* *L* (*cls-of-ccls* *D*))
(*reduce-trail-to* *M1*
(*add-learned-cls* (*cls-of-ccls* *D*)
(*update-backtrack-lvl* *i*
(*update-conflicting* *None* *S*)))) $\Longrightarrow$
P S T
shows *P S S′*
⟨*proof*⟩

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

⟨*proof*⟩

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

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

19.3 Invariants

19.3.1 Properties of the trail

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

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

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

lemma *cdcl*$_W$*-consistent-inv-2*:
  assumes
    cdcl$_W$ S S' **and**
    no-dup (*trail* S) **and**

backtrack-lvl S = *length* (*get-all-levels-of-marked* (*trail S*)) **and**
get-all-levels-of-marked (*trail S*) = *rev* [*1*..<*1*+*length* (*get-all-levels-of-marked* (*trail S*))]
shows *consistent-interp* (*lits-of-l* (*trail S'*))
⟨*proof*⟩

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

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

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

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

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

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

lemma $cdcl_W$*-M-level-inv-decomp*:

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

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

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

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

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

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

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

19.3.2 Better-Suited Induction Principle

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

lemma *backtrack-induction-lev*[*consumes 1*, *case-names M-devel-inv backtrack*]:
assumes
bt: *backtrack S T* **and**
inv: $cdcl_W$*-M-level-inv S* **and**
backtrackH: $\bigwedge K\ i\ M1\ M2\ L\ D\ T.$
raw-conflicting S = Some D $\Longrightarrow$

$L \in\# \ mset\text{-}ccls\ D \Longrightarrow$
$(Marked\ K\ (Suc\ i)\ \#\ M1,\ M2) \in set\ (get\text{-}all\text{-}marked\text{-}decomposition\ (trail\ S)) \Longrightarrow$
$get\text{-}level\ (trail\ S)\ L = backtrack\text{-}lvl\ S \Longrightarrow$
$get\text{-}level\ (trail\ S)\ L = get\text{-}maximum\text{-}level\ (trail\ S)\ (mset\text{-}ccls\ D) \Longrightarrow$
$get\text{-}maximum\text{-}level\ (trail\ S)\ (remove1\text{-}mset\ L\ (mset\text{-}ccls\ D)) \equiv i \Longrightarrow$
$undefined\text{-}lit\ M1\ L \Longrightarrow$
$T \sim cons\text{-}trail\ (Propagated\ L\ (cls\text{-}of\text{-}ccls\ D))$
$(reduce\text{-}trail\text{-}to\ M1$
$(add\text{-}learned\text{-}cls\ (cls\text{-}of\text{-}ccls\ D)$
$(update\text{-}backtrack\text{-}lvl\ i$
$(update\text{-}conflicting\ None\ S)))) \Longrightarrow$
$P\ S\ T$
shows $P\ S\ T$
$\langle proof \rangle$

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

lemma $cdcl_W$*-all-induct-lev-full*:
fixes S :: *'st*
assumes
$cdcl_W$: $cdcl_W\ S\ S'$ **and**
inv[*simp*]: $cdcl_W$*-M-level-inv* S **and**
propagateH: $\bigwedge C\ L\ T.\ conflicting\ S = None \Longrightarrow$
$C\ !\in!\ raw\text{-}clauses\ S \Longrightarrow$
$L \in\# \ mset\text{-}cls\ C \Longrightarrow$
$trail\ S \models as\ CNot\ (remove1\text{-}mset\ L\ (mset\text{-}cls\ C)) \Longrightarrow$
$undefined\text{-}lit\ (trail\ S)\ L \Longrightarrow$
$T \sim cons\text{-}trail\ (Propagated\ L\ C)\ S \Longrightarrow$
$P\ S\ T$ **and**
conflictH: $\bigwedge D\ T.\ conflicting\ S = None \Longrightarrow$
$D\ !\in!\ raw\text{-}clauses\ S \Longrightarrow$
$trail\ S \models as\ CNot\ (mset\text{-}cls\ D) \Longrightarrow$
$T \sim update\text{-}conflicting\ (Some\ (ccls\text{-}of\text{-}cls\ D))\ S \Longrightarrow$
$P\ S\ T$ **and**
forgetH: $\bigwedge C\ T.\ conflicting\ S = None \Longrightarrow$
$C\ !\in!\ raw\text{-}learned\text{-}clss\ S \Longrightarrow$
$\neg(trail\ S) \models asm\ clauses\ S \Longrightarrow$
$mset\text{-}cls\ C \notin set\ (get\text{-}all\text{-}mark\text{-}of\text{-}propagated\ (trail\ S)) \Longrightarrow$
$mset\text{-}cls\ C \notin\# \ init\text{-}clss\ S \Longrightarrow$
$T \sim remove\text{-}cls\ C\ S \Longrightarrow$
$P\ S\ T$ **and**
restartH: $\bigwedge T.\ \neg trail\ S \models asm\ clauses\ S \Longrightarrow$
$conflicting\ S = None \Longrightarrow$
$T \sim restart\text{-}state\ S \Longrightarrow$
$P\ S\ T$ **and**
decideH: $\bigwedge L\ T.\ conflicting\ S = None \Longrightarrow$
$undefined\text{-}lit\ (trail\ S)\ L \Longrightarrow$
$atm\text{-}of\ L \in atms\text{-}of\text{-}mm\ (init\text{-}clss\ S) \Longrightarrow$
$T \sim cons\text{-}trail\ (Marked\ L\ (backtrack\text{-}lvl\ S + 1))\ (incr\text{-}lvl\ S) \Longrightarrow$
$P\ S\ T$ **and**
skipH: $\bigwedge L\ C'\ M\ E\ T.$
$trail\ S = Propagated\ L\ C'\ \#\ M \Longrightarrow$
$raw\text{-}conflicting\ S = Some\ E \Longrightarrow$
$-L \notin\# \ mset\text{-}ccls\ E \Longrightarrow\ \ mset\text{-}ccls\ E \neq \{\#\} \Longrightarrow$
$T \sim tl\text{-}trail\ S \Longrightarrow$

$P\ S\ T$ **and**
resolveH: $\bigwedge L\ E\ M\ D\ T.$
trail S = *Propagated* L (*mset-cls* E) # $M \Longrightarrow$
$L \in\#$ *mset-cls* $E \Longrightarrow$
hd-raw-trail S = *Propagated* $L\ E \Longrightarrow$
raw-conflicting S = *Some* $D \Longrightarrow$
$-L \in\#$ *mset-ccls* $D \Longrightarrow$
get-maximum-level (*trail* S) (*mset-ccls* (*remove-clit* $(-L)$ D)) = *backtrack-lvl* $S \Longrightarrow$
$T \sim$ *update-conflicting*
(*Some* (*union-ccls* (*remove-clit* $(-L)$ D) (*ccls-of-cls* (*remove-lit* $L\ E$)))) (*tl-trail* S) $\Longrightarrow$
$P\ S\ T$ **and**
backtrackH: $\bigwedge K\ i\ M1\ M2\ L\ D\ T.$
raw-conflicting S = *Some* $D \Longrightarrow$
$L \in\#$ *mset-ccls* $D \Longrightarrow$
(*Marked* K (*Suc* i) # $M1$, $M2$) $\in$ *set* (*get-all-marked-decomposition* (*trail* S)) $\Longrightarrow$
get-level (*trail* S) L = *backtrack-lvl* $S \Longrightarrow$
get-level (*trail* S) L = *get-maximum-level* (*trail* S) (*mset-ccls* D) $\Longrightarrow$
get-maximum-level (*trail* S) (*remove1-mset* L (*mset-ccls* D)) $\equiv i \Longrightarrow$
undefined-lit $M1\ L \Longrightarrow$
$T \sim$ *cons-trail* (*Propagated* L (*cls-of-ccls* D))
(*reduce-trail-to* $M1$
(*add-learned-cls* (*cls-of-ccls* D)
(*update-backtrack-lvl* i
(*update-conflicting None* S)))) $\Longrightarrow$
$P\ S\ T$
shows $P\ S\ S'$
⟨*proof*⟩

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

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

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

raw-conflicting S = *Some* $D \Longrightarrow$
$-L \in\#$ *mset-ccls* $D \Longrightarrow$
get-maximum-level (*trail* S) (*mset-ccls* (*remove-clit* ($-L$) D)) = *backtrack-lvl* $S \Longrightarrow$
$T \sim$ *update-conflicting*
(*Some* (*union-ccls* (*remove-clit* ($-L$) D) (*ccls-of-cls* (*remove-lit* L E)))) (*tl-trail* S) $\Longrightarrow$
P S T **and**
backtrackH: $\bigwedge K\ i\ M1\ M2\ L\ D\ T.$
raw-conflicting S = *Some* $D \Longrightarrow$
$L \in\#$ *mset-ccls* $D \Longrightarrow$
(*Marked* K (*Suc* i) # *M1*, *M2*) $\in$ *set* (*get-all-marked-decomposition* (*trail* S)) $\Longrightarrow$
get-level (*trail* S) L = *backtrack-lvl* $S \Longrightarrow$
get-level (*trail* S) L = *get-maximum-level* (*trail* S) (*mset-ccls* D) $\Longrightarrow$
get-maximum-level (*trail* S) (*remove1-mset* L (*mset-ccls* D)) $\equiv i \Longrightarrow$
undefined-lit *M1* $L \Longrightarrow$
$T \sim$ *cons-trail* (*Propagated* L (*cls-of-ccls* D))
(*reduce-trail-to* *M1*
(*add-learned-cls* (*cls-of-ccls* D)
(*update-backtrack-lvl* i
(*update-conflicting* *None* S)))) $\Longrightarrow$
P S T
shows P S T
⟨*proof*⟩

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

19.3.3 Compatibility with *op* ∼

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

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

lemma *backtrack-levE*[*consumes 2*]:
backtrack S $S' \Longrightarrow$ $cdcl_W$*-M-level-inv* $S \Longrightarrow$
($\bigwedge K\ i\ M1\ M2\ L\ D.$
raw-conflicting S = *Some* $D \Longrightarrow$
$L \in\#$ *mset-ccls* $D \Longrightarrow$
(*Marked* K (*Suc* i) # *M1*, *M2*) $\in$ *set* (*get-all-marked-decomposition* (*trail* S)) $\Longrightarrow$
get-level (*trail* S) L = *backtrack-lvl* $S \Longrightarrow$
get-level (*trail* S) L = *get-maximum-level* (*trail* S) (*mset-ccls* D) $\Longrightarrow$
get-maximum-level (*trail* S) (*remove1-mset* L (*mset-ccls* D)) $\equiv i \Longrightarrow$
undefined-lit *M1* $L \Longrightarrow$
$S' \sim$ *cons-trail* (*Propagated* L (*cls-of-ccls* D))
(*reduce-trail-to* *M1*

(*add-learned-cls* (*cls-of-ccls* D)
(*update-backtrack-lvl* i
(*update-conflicting None* S)))) $\Longrightarrow P$) $\Longrightarrow$
P
⟨*proof*⟩
thm *allI*

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

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

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

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

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

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

shows $cdcl_W\ S'\ T'$
⟨*proof*⟩

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

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

19.3.4 Conservation of some Properties

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

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

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

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

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

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

19.3.5 Learned Clause

This invariant shows that:

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

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

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

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

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

19.3.6 No alien atom in the state

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

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

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

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

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

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

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

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

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

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

19.3.7 No duplicates all around

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

definition *distinct-*$cdcl_W$*-state* (S::$'st$)
$\longleftrightarrow$ (($\forall T$. *conflicting S* = *Some T* $\longrightarrow$ *distinct-mset T*)
$\wedge$ *distinct-mset-mset* (*learned-clss S*)
$\wedge$ *distinct-mset-mset* (*init-clss S*)

$\wedge$ ($\forall$ *L mark*. (*Propagated L mark* $\in$ *set* (*trail S*) $\longrightarrow$ *distinct-mset* (*mark*))))

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

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

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

lemma *distinct-cdcl*$_W$*-state-inv*:
assumes
cdcl$_W$ *S S'* **and**
lev-inv: *cdcl*$_W$*-M-level-inv S* **and**
distinct-cdcl$_W$*-state S*
shows *distinct-cdcl*$_W$*-state S'*
$\langle$*proof*$\rangle$

lemma *rtanclp-distinct-cdcl*$_W$*-state-inv*:
assumes
cdcl$_W$** *S S'* **and**
cdcl$_W$*-M-level-inv S* **and**
distinct-cdcl$_W$*-state S*
shows *distinct-cdcl*$_W$*-state S'*
$\langle$*proof*$\rangle$

19.3.8 Conflicts and co

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

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

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

lemma *backtrack-atms-of-D-in-M1*:
fixes *M1* :: ('*v*, *nat*, '*v clause*) *marked-lits*
assumes
inv: *cdcl*$_W$*-M-level-inv S* **and**
undef: *undefined-lit M1 L* **and**
i: *get-maximum-level* (*trail S*) (*mset-ccls* (*remove-clit L D*)) $\equiv$ *i* **and**
decomp: (*Marked K* (*Suc i*) # *M1*, *M2*)

$\in$ *set* (*get-all-marked-decomposition* (*trail S*)) **and**
S-lvl: *backtrack-lvl S* = *get-maximum-level* (*trail S*) (*mset-ccls D*) **and**
S-confl: *raw-conflicting S* = *Some D* **and**
undef: *undefined-lit M1 L* **and**
T: *T* $\sim$ *cons-trail* (*Propagated L* (*cls-of-ccls D*))
(*reduce-trail-to M1*
(*add-learned-cls* (*cls-of-ccls D*)
(*update-backtrack-lvl i*
(*update-conflicting None S*)))) **and**
confl: $\forall$ *T*. *conflicting S* = *Some T* $\longrightarrow$ *trail S* $\models$*as CNot T*
shows *atms-of* (*mset-ccls* (*remove-clit L D*)) $\subseteq$ *atm-of* ‘ *lits-of-l* (*tl* (*trail T*))
⟨*proof*⟩

lemma *distinct-atms-of-incl-not-in-other*:
assumes
a1: *no-dup* (*M* @ *M'*) **and**
a2: *atms-of D* $\subseteq$ *atm-of* ‘ *lits-of-l M'* **and**
a3: *x* $\in$ *atms-of D*
shows *x* $\notin$ *atm-of* ‘ *lits-of-l M*
⟨*proof*⟩

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

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

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

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

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

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

19.3.9 Putting all the invariants together

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

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

lemma *all-invariant-S0-cdcl*$_W$:
assumes *distinct-mset-mset* (*mset-clss N*)

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

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

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

lemma
assumes $\mathit{all\text{-}decomposition\text{-}implies\text{-}m}\ N\ (\mathit{get\text{-}all\text{-}marked\text{-}decomposition}\ M)$
and $\forall\, m \in \mathit{set}\ M.\ \neg \mathit{is\text{-}marked}\ m$
shows $\mathit{set\text{-}mset}\ N \models ps\ \mathit{unmark\text{-}l}\ M$
⟨*proof*⟩

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

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

19.3.10 No tautology is learned

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

lemma *learned-clss-are-not-tautologies*:

assumes
  $cdcl_W$ S S' **and**
  lev: $cdcl_W$*-M-level-inv* S **and**
  conflicting: $cdcl_W$*-conflicting* S **and**
  no-tauto: $\forall\, s \in\#$ *learned-clss* S. $\neg$*tautology* s
shows $\forall\, s \in\#$ *learned-clss* S'. $\neg$*tautology* s
$\langle proof\rangle$

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

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

19.4 CDCL Strong Completeness

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

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

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

lemma *image-set-mapi*:
f ‘ *set* (*mapi* g i M) = *set* (*mapi* ($\lambda x\ i.\ f\ (g\ x\ i)$) i M)
$\langle proof\rangle$

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

lemma *defined-lit-mapi*: *defined-lit* (*mapi Marked* i M) $L \longleftrightarrow$ *atm-of* $L \in$ *atm-of* ‘ *set* M
$\langle proof\rangle$

lemma $cdcl_W$*-can-do-step*:
assumes
  consistent-interp (*set* M) **and**
  distinct M **and**
  atm-of ‘ (*set* M) $\subseteq$ *atms-of-mm* (*mset-clss* N)
shows $\exists\, S$. *rtranclp* $cdcl_W$ (*init-state* N) S
  $\wedge$ *state* S = (*mapi Marked* (*length* M) M, *mset-clss* N, {#}, *length* M, *None*)
$\langle proof\rangle$

lemma $cdcl_W$*-strong-completeness*:
assumes
  MN: *set* M $\models$*sm* *mset-clss* N **and**
  cons: *consistent-interp* (*set* M) **and**
  dist: *distinct* M **and**
  atm: *atm-of* ‘ (*set* M) $\subseteq$ *atms-of-mm* (*mset-clss* N)

obtains S **where**
state $S = (mapi\ Marked\ (length\ M)\ M,\ mset\text{-}clss\ N,\ \{\#\},\ length\ M,\ None)$ **and**
rtranclp $cdcl_W$ $(init\text{-}state\ N)$ S **and**
final-$cdcl_W$*-state* S
⟨*proof*⟩

19.5 Higher level strategy

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

19.5.1 Definition

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

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

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

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

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

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

lemma *option-full-*$cdcl_W$*-cp*:
conflicting $S \neq None \Longrightarrow$ *full* $cdcl_W\text{-}cp\ S\ S$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

CDCL with the reasonable strategy: we fully propagate the conflict and propagate, then we apply any other possible rule *cdcl*$_W$*-o* S S' and re-apply conflict and propagate *full cdcl*$_W$*-cp* S' S''

inductive *cdcl*$_W$*-stgy* :: *'st* ⇒ *'st* ⇒ *bool* **for** S :: *'st* **where**
conflict': *full1 cdcl*$_W$*-cp* S S' $\Longrightarrow$ *cdcl*$_W$*-stgy* S S' |
other': *cdcl*$_W$*-o* S S' $\Longrightarrow$ *no-step cdcl*$_W$*-cp* S $\Longrightarrow$ *full cdcl*$_W$*-cp* S' S'' $\Longrightarrow$ *cdcl*$_W$*-stgy* S S''

19.5.2 Invariants

These are the same invariants as before, but lifted

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

lemma *rtranclp-cdcl*$_W$*-cp-learned-clause-inv*:
  assumes *cdcl*$_W$*-cp*** S S'

shows *learned-clss* S = *learned-clss* S'
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

shows *init-clss* S = *init-clss* S'
⟨*proof*⟩

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

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

lemma $cdcl_W$*-cp-dropWhile-trail*$'$:
assumes $cdcl_W$*-cp* S S'
obtains M **where** *trail* S' = M @ *trail* S **and** $(\forall l \in$ *set* M. $\neg$*is-marked* $l)$
⟨*proof*⟩

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

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

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

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

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

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

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

$\wedge$ *cdcl*$_W$*-cp a b*}
⟨*proof*⟩

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

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

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

19.5.3 Literal of highest level in conflicting clauses

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

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

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

lemma *not-conflict-not-any-negated-init-clss*:
assumes $\forall\ S'.\ \neg conflict\ S\ S'$
shows *no-clause-is-false S*
⟨*proof*⟩

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

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

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

$\langle proof \rangle$

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

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

lemma *no-chained-conflict*:
assumes *conflict S S'*
and *conflict S' S''*
shows *False*
$\langle proof \rangle$

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

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

19.5.4 Literal of highest level in marked literals

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

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

lemma *propagate-no-more-propagation-to-do*:
assumes *propagate*: *propagate S S'*
and *H*: *no-more-propagation-to-do S*
and *lev-inv*: *cdcl$_W$-M-level-inv S*
shows *no-more-propagation-to-do S'*
$\langle proof \rangle$

lemma *conflict-no-more-propagation-to-do*:
assumes
conflict: *conflict S S'* **and**

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

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

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

lemma *backtrack-no-decomp*:
assumes
S: *raw-conflicting* S = *Some* E **and**
LE: $L \in\#$ *mset-ccls* E **and**
L: *get-level* (*trail* S) L = *backtrack-lvl* S **and**
D: *get-maximum-level* (*trail* S) (*remove1-mset* L (*mset-ccls* E)) < *backtrack-lvl* S **and**
bt: *backtrack-lvl* S = *get-maximum-level* (*trail* S) (*mset-ccls* E) **and**
M-L: $cdcl_W$*-M-level-inv* S
shows $\exists S'.$ $cdcl_W$*-o* S S'
⟨*proof*⟩

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

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

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

lemma $cdcl_W$*-stgy-tranclp-*$cdcl_W$:

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

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

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

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

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

19.5.5 Strong completeness

lemma $cdcl_W$*-cp-propagate-confl*:
assumes $cdcl_W$*-cp* S T
shows *propagate*** S $T \vee (\exists\, S'.$ *propagate*** S $S' \wedge$ *conflict* S' $T)$
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-cp-propagate-confl*:
assumes $cdcl_W$*-cp*** S T
shows *propagate*** S $T \vee (\exists\, S'.$ *propagate*** S $S' \wedge$ *conflict* S' $T)$
⟨*proof*⟩

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

lemma $cdcl_W$*-cp-propagate-completeness*:
assumes *MN*: *set* $M \models\!s$ *set-mset* N **and**
cons: *consistent-interp* (*set* M) **and**
tot: *total-over-m* (*set* M) (*set-mset* N) **and**
lits-of-l (*trail* S) $\subseteq$ *set* M **and**
init-clss $S = N$ **and**
propagate** S S' **and**

learned-clss $S = \{\#\}$
shows *length* (*trail* S) $\leq$ *length* (*trail* S') $\wedge$ *lits-of-l* (*trail* S') $\subseteq$ *set* M
⟨*proof*⟩

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

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

See also $cdcl_W$*-cp*** *?S* *?S'* $\Longrightarrow \exists M$. *trail* *?S'* = M @ *trail* *?S* $\wedge$ ($\forall$ $l\in$*set* M. $\neg$ *is-marked* l)

lemma *rtranclp-propagate-is-trail-append*:
propagate** S T $\Longrightarrow \exists c$. *trail* T = c @ *trail* S
⟨*proof*⟩

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

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

lemma $cdcl_W$*-stgy-strong-completeness*:
assumes
MN: *set* M $\models$*s* *set-mset* (*mset-clss* N) **and**
cons: *consistent-interp* (*set* M) **and**
tot: *total-over-m* (*set* M) (*set-mset* (*mset-clss* N)) **and**

atm-incl: *atm-of* ‘ (*set M*) ⊆ *atms-of-mm* (*mset-clss N*) **and**
distM: *distinct M*
shows
∃ *M′ k S*.
lits-of-l M′ = *set M* ∧
state S = (*M′*, *mset-clss N*, {#}, *k*, *None*) ∧
$cdcl_W$*-stgy*** (*init-state N*) *S* ∧
final-$cdcl_W$*-state S*
⟨*proof*⟩

19.5.6 No conflict with only variables of level less than backtrack level

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *rtranclp-cdcl$_W$-stgy-no-smaller-confl-inv*:
assumes
cdcl$_W$-stgy** *S S'* **and**
n-l: *no-smaller-confl S* **and**
cls-false: *conflict-is-false-with-level S* **and**
lev: *cdcl$_W$-M-level-inv S* **and**
no-f: *no-clause-is-false S* **and**
dist: *distinct-cdcl$_W$-state S* **and**
conflicting: *cdcl$_W$-conflicting S* **and**
decomp: *all-decomposition-implies-m* (*init-clss S*) (*get-all-marked-decomposition* (*trail S*)) **and**
learned: *cdcl$_W$-learned-clause S* **and**
alien: *no-strange-atm S*
shows *no-smaller-confl S'* $\wedge$ *conflict-is-false-with-level S'*
$\langle$*proof*$\rangle$

19.5.7 Final States are Conclusive

lemma *full-cdcl$_W$-stgy-final-state-conclusive-non-false*:
fixes *S'* :: *'st*
assumes *full*: *full cdcl$_W$-stgy* (*init-state N*) *S'*
and *no-d*: *distinct-mset-mset* (*mset-clss N*)

and *no-empty*: $\forall D\in\#mset\text{-}clss\ N.\ D \neq \{\#\}$
shows $(conflicting\ S' = Some\ \{\#\} \wedge unsatisfiable\ (set\text{-}mset\ (init\text{-}clss\ S')))$
$\vee\ (conflicting\ S' = None \wedge trail\ S' \models asm\ init\text{-}clss\ S')$
⟨*proof*⟩

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

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

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

lemma *cdcl$_W$-stgy-fst-empty-conflicting-false*:
assumes *cdcl$_W$-stgy S S'*
and *trail S* = []
and *conflicting S* ≠ *None*
shows *False*
⟨*proof*⟩
thm *cdcl$_W$-cp.induct*[*split-format*(*complete*)]

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

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

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

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

lemma *rtranclp-cdcl$_W$-stgy-conflicting-is-false*:
$cdcl_W\text{-}stgy^{**}\ S\ S' \Longrightarrow conflicting\ S = Some\ \{\#\} \Longrightarrow S' = S$
⟨*proof*⟩

lemma *full-cdcl$_W$-init-clss-with-false-normal-form*:

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

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

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

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

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

19.6 Termination

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

The invariant contains all the structural invariants that holds,

definition $cdcl_W$*-all-struct-inv* **where**
$cdcl_W$*-all-struct-inv* $S \longleftrightarrow$
no-strange-atm S $\wedge$
$cdcl_W$*-M-level-inv* S $\wedge$

$(\forall s \in\# \; \textit{learned-clss } S. \; \neg \textit{tautology } s) \wedge$
$\textit{distinct-cdcl}_W\textit{-state } S \; \wedge$
$\textit{cdcl}_W\textit{-conflicting } S \wedge$
$\textit{all-decomposition-implies-m } (\textit{init-clss } S) \; (\textit{get-all-marked-decomposition } (\textit{trail } S)) \wedge$
$\textit{cdcl}_W\textit{-learned-clause } S$

lemma $\textit{cdcl}_W\textit{-all-struct-inv-inv}$:
assumes $\textit{cdcl}_W \; S \; S'$ **and** $\textit{cdcl}_W\textit{-all-struct-inv } S$
shows $\textit{cdcl}_W\textit{-all-struct-inv } S'$
⟨*proof*⟩

lemma $\textit{rtranclp-cdcl}_W\textit{-all-struct-inv-inv}$:
assumes $\textit{cdcl}_W{}^{**} \; S \; S'$ **and** $\textit{cdcl}_W\textit{-all-struct-inv } S$
shows $\textit{cdcl}_W\textit{-all-struct-inv } S'$
⟨*proof*⟩

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

lemma $\textit{rtranclp-cdcl}_W\textit{-stgy-cdcl}_W\textit{-all-struct-inv}$:
$\textit{cdcl}_W\textit{-stgy}^{**} \; S \; T \Longrightarrow \textit{cdcl}_W\textit{-all-struct-inv } S \Longrightarrow \textit{cdcl}_W\textit{-all-struct-inv } T$
⟨*proof*⟩

19.7 No Relearning of a clause

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

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

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

lemma *propagate-no-more-Marked-lit*:
assumes $\textit{propagate } S \; S'$
shows $\textit{Marked } K \; i \in \textit{set } (\textit{trail } S) \longleftrightarrow \textit{Marked } K \; i \in \textit{set } (\textit{trail } S')$
⟨*proof*⟩

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

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

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

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

lemma $cdcl_W$*-new-marked-at-beginning-is-decide*:
  assumes $cdcl_W$*-stgy S S'* **and**
  lev: $cdcl_W$*-M-level-inv S* **and**
  trail S' = *M'* @ *Marked L i* # *M* **and**
  trail S = *M*
  shows $\exists$ *T*. *decide S T* $\wedge$ *no-step* $cdcl_W$*-cp S*
  ⟨*proof*⟩

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

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

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

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

lemma *cdcl$_W$-stgy-trail-has-new-marked-is-decide-step*:
assumes *cdcl$_W$-stgy* S T
$\neg$ ($\exists c.$ *trail* $S = c$ @ *Marked* L i # H @ M) **and**
($\lambda a\ b.$ *cdcl$_W$-stgy* a b $\wedge$ ($\exists c.$ *trail* $a = c$ @ *Marked* L i # H @ M))** T U **and**
$\exists M'.$ *trail* $U = M'$ @ *Marked* L i # H @ M **and**
lev: *cdcl$_W$-M-level-inv* S
shows $\exists S'.$ *decide* S S' $\wedge$ *full cdcl$_W$-cp* S' T $\wedge$ *no-step cdcl$_W$-cp* S
$\langle proof \rangle$

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

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

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

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

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

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

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

lemma $cdcl_W$*-stgy-no-relearned-clause*:
assumes
invR: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ R$ **and**
st': $cdcl_W\text{-}stgy^{**}\ R\ S$ **and**
bt: $backtrack\ S\ T$ **and**
confl: $raw\text{-}conflicting\ S = Some\ E$ **and**
already-learned: $mset\text{-}ccls\ E \in\#\ clauses\ S$ **and**
R: $trail\ R = []$
shows *False*
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-stgy-distinct-mset-clauses*:
assumes
invR: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ R$ **and**
st: $cdcl_W\text{-}stgy^{**}\ R\ S$ **and**
dist: $distinct\text{-}mset\ (clauses\ R)$ **and**
R: $trail\ R = []$
shows $distinct\text{-}mset\ (clauses\ S)$
⟨*proof*⟩

lemma $cdcl_W$*-stgy-distinct-mset-clauses*:
assumes
st: $cdcl_W\text{-}stgy^{**}\ (init\text{-}state\ N)\ S$ **and**
no-duplicate-clause: $distinct\text{-}mset\ (mset\text{-}clss\ N)$ **and**
no-duplicate-in-clause: $distinct\text{-}mset\text{-}mset\ (mset\text{-}clss\ N)$
shows $distinct\text{-}mset\ (clauses\ S)$
⟨*proof*⟩

19.8 Decrease of a measure

fun $cdcl_W$*-measure* **where**
$cdcl_W\text{-}measure\ S =$
$[(3{::}nat)\ \widehat{}\ (card\ (atms\text{-}of\text{-}mm\ (init\text{-}clss\ S))) - card\ (set\text{-}mset\ (learned\text{-}clss\ S)),$
$if\ conflicting\ S = None\ then\ 1\ else\ 0,$
$if\ conflicting\ S = None\ then\ card\ (atms\text{-}of\text{-}mm\ (init\text{-}clss\ S)) - length\ (trail\ S)$
$else\ length\ (trail\ S)$
$]$

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

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

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

lemma $cdcl_W$*-measure-decreasing*:
  fixes *S* :: *'st*
  assumes
    $cdcl_W$ *S S'* **and**
    no-restart:
      ¬(*learned-clss S* ⊆# *learned-clss S'* ∧ [] = *trail S'* ∧ *conflicting S'* = *None*)
     and
    no-forget: *learned-clss S* ⊆# *learned-clss S'* **and**
    no-relearn: ⋀*S'*. *backtrack S S'* ⟹ ∀ *T*. *conflicting S* = *Some T* ⟶ *T* ∉# *learned-clss S*
      and
    alien: *no-strange-atm S* **and**
    M-level: $cdcl_W$*-M-level-inv S* **and**
    no-taut: ∀ *s* ∈# *learned-clss S*. ¬*tautology s* **and**
    no-dup: *distinct-cdcl*$_W$*-state S* **and**
    confl: $cdcl_W$*-conflicting S*
  shows ($cdcl_W$*-measure S'*, $cdcl_W$*-measure S*) ∈ *lexn* {(*a*, *b*). *a* < *b*} *3*
  ⟨*proof*⟩

lemma *propagate-measure-decreasing*:
  fixes *S* :: *'st*
  assumes *propagate S S'* **and** $cdcl_W$*-all-struct-inv S*
  shows ($cdcl_W$*-measure S'*, $cdcl_W$*-measure S*) ∈ *lexn* {(*a*, *b*). *a* < *b*} *3*
  ⟨*proof*⟩

lemma *conflict-measure-decreasing*:
  fixes *S* :: *'st*
  assumes *conflict S S'* **and** $cdcl_W$*-all-struct-inv S*
  shows ($cdcl_W$*-measure S'*, $cdcl_W$*-measure S*) ∈ *lexn* {(*a*, *b*). *a* < *b*} *3*
  ⟨*proof*⟩

lemma *decide-measure-decreasing*:
  fixes *S* :: *'st*
  assumes *decide S S'* **and** $cdcl_W$*-all-struct-inv S*
  shows ($cdcl_W$*-measure S'*, $cdcl_W$*-measure S*) ∈ *lexn* {(*a*, *b*). *a* < *b*} *3*
  ⟨*proof*⟩

lemma *trans-le*:
  $trans\ \{(a,\ (b::nat)).\ a < b\}$
  ⟨*proof*⟩

lemma $cdcl_W$*-cp-measure-decreasing*:
  fixes S :: $'st$
  assumes $cdcl_W\text{-}cp\ S\ S'$ **and** $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$
  shows $(cdcl_W\text{-}measure\ S',\ cdcl_W\text{-}measure\ S) \in lexn\ \{(a,\ b).\ a < b\}\ 3$
  ⟨*proof*⟩

lemma *tranclp-*$cdcl_W$*-cp-measure-decreasing*:
  fixes S :: $'st$
  assumes $cdcl_W\text{-}cp^{++}\ S\ S'$ **and** $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$
  shows $(cdcl_W\text{-}measure\ S',\ cdcl_W\text{-}measure\ S) \in lexn\ \{(a,\ b).\ a < b\}\ 3$
  ⟨*proof*⟩

lemma $cdcl_W$*-stgy-step-decreasing*:
  fixes $R\ S\ T$ :: $'st$
  assumes $cdcl_W\text{-}stgy\ S\ T$ **and**
  $cdcl_W\text{-}stgy^{**}\ R\ S$
  $trail\ R = []$ **and**
  $cdcl_W\text{-}all\text{-}struct\text{-}inv\ R$
  shows $(cdcl_W\text{-}measure\ T,\ cdcl_W\text{-}measure\ S) \in lexn\ \{(a,\ b).\ a < b\}\ 3$
⟨*proof*⟩

lemma *tranclp-*$cdcl_W$*-stgy-decreasing*:
  fixes $R\ S\ T$ :: $'st$
  assumes $cdcl_W\text{-}stgy^{++}\ R\ S$
  $trail\ R = []$ **and**
  $cdcl_W\text{-}all\text{-}struct\text{-}inv\ R$
  shows $(cdcl_W\text{-}measure\ S,\ cdcl_W\text{-}measure\ R) \in lexn\ \{(a,\ b).\ a < b\}\ 3$
  ⟨*proof*⟩

lemma *tranclp-*$cdcl_W$*-stgy-S0-decreasing*:
  fixes $R\ S\ T$ :: $'st$
  assumes
    pl: $cdcl_W\text{-}stgy^{++}\ (init\text{-}state\ N)\ S$ **and**
    no-dup: $distinct\text{-}mset\text{-}mset\ (mset\text{-}clss\ N)$
  shows $(cdcl_W\text{-}measure\ S,\ cdcl_W\text{-}measure\ (init\text{-}state\ N)) \in lexn\ \{(a,\ b).\ a < b\}\ 3$
⟨*proof*⟩

lemma *wf-tranclp-*$cdcl_W$*-stgy*:
  $wf\ \{(S::'st,\ init\text{-}state\ N)|$
    $S\ N.\ distinct\text{-}mset\text{-}mset\ (mset\text{-}clss\ N) \land cdcl_W\text{-}stgy^{++}\ (init\text{-}state\ N)\ S\}$
  ⟨*proof*⟩

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

end

end
theory *DPLL-CDCL-W-Implementation*

imports *Partial-Annotated-Clausal-Logic*
begin

20 Simple Implementation of the DPLL and CDCL

20.1 Common Rules

20.1.1 Propagation

The following theorem holds:

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

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

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

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

lemma *is-unit-clause-is-unit-clause-code*[*code*]:
 $is\text{-}unit\text{-}clause\ l\ M = is\text{-}unit\text{-}clause\text{-}code\ l\ M$
⟨*proof*⟩

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

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

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

lemma *is-unit-clause-nil*[*simp*]: $is\text{-}unit\text{-}clause\ []\ M = None$
 ⟨*proof*⟩

20.1.2 Unit propagation for all clauses

Finding the first clause to propagate

fun *find-first-unit-clause* :: $'a\ literal\ list\ list \Rightarrow ('a, 'b, 'c)\ marked\text{-}lit\ list$
 $\Rightarrow ('a\ literal \times 'a\ literal\ list)\ option$ **where**
$find\text{-}first\text{-}unit\text{-}clause\ (a\ \#\ l)\ M =$
 $(case\ is\text{-}unit\text{-}clause\ a\ M\ of$

$None \Rightarrow$ *find-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*)
$\Longrightarrow c \in set\ l \wedge\ M \models as\ CNot\ (mset\ c - \{\#a\#\}) \wedge undefined\text{-}lit\ M\ a \wedge a \in set\ c$
⟨*proof*⟩

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

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

20.1.3 Decide

fun *find-first-unused-var* :: *'a literal list list* $\Rightarrow$ *'a literal set* $\Rightarrow$ *'a literal option* **where**
find-first-unused-var (*a* # *l*) *M* =
(*case List.find* ($\lambda lit.\ lit \notin M \wedge -lit \notin M$) *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 ($\lambda lit.\ lit \notin M \wedge -lit \notin M$) $a = None \longleftrightarrow\ atm\text{-}of\ `\ set\ a \subseteq atm\text{-}of\ `\ M$
⟨*proof*⟩

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

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

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

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

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

end

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

20.2 Simple Implementation of DPLL

20.2.1 Combining the propagate and decide: a DPLL step

definition *DPLL-step* :: *int* $dpll_W$*-marked-lits* × *int literal list list*
  ⇒ *int* $dpll_W$*-marked-lits* × *int literal list list* **where**
DPLL-step = (λ(*Ms*, *N*).
  (*case find-first-unit-clause N Ms of*
    Some (*L*, -) ⇒ (*Propagated L* () # *Ms*, *N*)
  | - ⇒
    if ∃ *C* ∈ *set N*. (∀ *c* ∈ *set C*. −*c* ∈ *lits-of-l Ms*)
    then
      (*case backtrack-split Ms of*
        (-, *L* # *M*) ⇒ (*Propagated* (− (*lit-of L*)) () # *M*, *N*)
      | (-, -) ⇒ (*Ms*, *N*)
)
    else
    (*case find-first-unused-var N* (*lits-of-l Ms*) *of*
        Some a ⇒ (*Marked a* () # *Ms*, *N*)
      | *None* ⇒ (*Ms*, *N*))))

Example of propagation:

value *DPLL-step* ([*Marked* (*Neg 1*) ()], [[*Pos* (*1*::*int*), *Neg 2*]])

We define the conversion function between the states as defined in *Prop-DPLL* (with multisets) and here (with lists).

abbreviation *toS* ≡ λ(*Ms*::(*int*, *unit*, *unit*) *marked-lit list*)
                  (*N*:: *int literal list list*). (*Ms*, *mset* (*map mset N*))
abbreviation *toS'* ≡ λ(*Ms*::(*int*, *unit*, *unit*) *marked-lit list*,
                  N:: *int literal list list*). (*Ms*, *mset* (*map mset N*))

Proof of correctness of *DPLL-step*

lemma *DPLL-step-is-a-*$dpll_W$*-step*:
  assumes *step*: (*Ms'*, *N'*) = *DPLL-step* (*Ms*, *N*)
  and *neq*: (*Ms*, *N*) ≠ (*Ms'*, *N'*)
  shows $dpll_W$ (*toS Ms N*) (*toS Ms' N'*)
⟨*proof*⟩

lemma *DPLL-step-stuck-final-state*:
  assumes *step*: (*Ms*, *N*) = *DPLL-step* (*Ms*, *N*)
  shows *conclusive-*$dpll_W$*-state* (*toS Ms N*)
⟨*proof*⟩

20.2.2 Adding invariants

Invariant tested in the function **function** *DPLL-ci* :: *int* $dpll_W$*-marked-lits* ⇒ *int literal list list*
  ⇒ *int* $dpll_W$*-marked-lits* × *int literal list list* **where**
DPLL-ci Ms N =
  (*if* ¬$dpll_W$*-all-inv* (*Ms*, *mset* (*map mset N*))
  then (*Ms*, *N*)

else
let $(Ms', N') = DPLL\text{-}step\ (Ms, N)$ *in*
if $(Ms', N') = (Ms, N)$ *then* (Ms, N) *else* $DPLL\text{-}ci\ Ms'\ N)$
⟨*proof*⟩
termination
⟨*proof*⟩

No invariant tested **function** (*domintros*) *DPLL-part*:: *int* $dpll_W$*-marked-lits* ⇒ *int literal list list* ⇒
int $dpll_W$*-marked-lits* × *int literal list list* **where**
DPLL-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-implieS-2-eq3-and-dom*:
assumes $dpll_W$*-all-inv* (*Ms*, *mset* (*map mset N*))
shows *DPLL-ci Ms N* = *DPLL-part Ms N* ∧ *DPLL-part-dom* (*Ms*, *N*)
⟨*proof*⟩

lemma *DPLL-ci-*$dpll_W$*-rtranclp*:
assumes $DPLL\text{-}ci\ Ms\ N = (Ms', N')$
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** $(Ms', N) = DPLL\text{-}step\ (Ms, N)$
⟨*proof*⟩

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

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

lemma *DPLL-part-dpll$_W$-all-inv-final*:
fixes *M Ms'*:: (*int*, *unit*, *unit*) *marked-lit list* **and**
N :: *int literal list list*
assumes *inv*: *dpll$_W$-all-inv* (*Ms*, *mset* (*map mset N*))
and *MsN*: *DPLL-part Ms N* = (*Ms'*, *N*)
shows *conclusive-dpll$_W$-state* (*toS Ms' N*) ∧ *dpll$_W$*** (*toS Ms N*) (*toS Ms' N*)
⟨*proof*⟩

Embedding the invariant into the type

Defining the type **typedef** *dpll$_W$-state* =
{(*M*::(*int*, *unit*, *unit*) *marked-lit list*, *N*::*int literal list list*).
dpll$_W$-all-inv (*toS M N*)}
morphisms *rough-state-of state-of*
⟨*proof*⟩

lemma
DPLL-part-dom ([], *N*)
⟨*proof*⟩

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

DPLL **definition** *DPLL-step'* :: *dpll$_W$-state* ⇒ *dpll$_W$-state* **where**
DPLL-step' S = *state-of* (*DPLL-step* (*rough-state-of S*))

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

lemma *DPLL-step-dpll$_W$-conc-inv*:
DPLL-step (*rough-state-of S*) ∈ {(*M*, *N*). *dpll$_W$-all-inv* (*toS M N*)}
⟨*proof*⟩

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

function *DPLL-tot*:: *dpll$_W$-state* ⇒ *dpll$_W$-state* **where**
DPLL-tot S =
(*let S'* = *DPLL-step' S in*
if S' = *S then S else DPLL-tot S'*)
⟨*proof*⟩
termination
⟨*proof*⟩

lemma [*code*]:
DPLL-tot S =
(*let S'* = *DPLL-step' S in*
if S' = *S then S else DPLL-tot S'*) ⟨*proof*⟩

lemma *DPLL-tot-DPLL-step-DPLL-tot*[*simp*]: *DPLL-tot* (*DPLL-step′ S*) = *DPLL-tot S*
⟨*proof*⟩

lemma *DOPLL-step′-DPLL-tot*[*simp*]:
DPLL-step′ (*DPLL-tot S*) = *DPLL-tot S*
⟨*proof*⟩

lemma *DPLL-tot-final-state*:
assumes *DPLL-tot S* = *S*
shows *conclusive-dpll*$_W$*-state* (*toS′* (*rough-state-of S*))
⟨*proof*⟩

lemma *DPLL-tot-star*:
assumes *rough-state-of* (*DPLL-tot S*) = *S′*
shows $dpll_W{}^{**}$ (*toS′* (*rough-state-of S*)) (*toS′ S′*)
⟨*proof*⟩

lemma *rough-state-of-rough-state-of-nil*[*simp*]:
rough-state-of (*state-of* ([], *N*)) = ([], *N*)
⟨*proof*⟩

Theorem of correctness

lemma *DPLL-tot-correct*:
assumes *rough-state-of* (*DPLL-tot* (*state-of* (([], *N*)))) = (*M*, *N′*)
and (*M′*, *N′′*) = *toS′* (*M*, *N′*)
shows *M′* ⊨*asm N′′* ⟷ *satisfiable* (*set-mset N′′*)
⟨*proof*⟩

20.2.3 Code export

A conversion to *DPLL-W-Implementation.dpll*$_W$*-state* **definition** *Con* :: (*int*, *unit*, *unit*) *marked-lit list* × *int literal list list*
⇒ *dpll*$_W$*-state* **where**
Con xs = *state-of* (*if dpll*$_W$*-all-inv* (*toS* (*fst xs*) (*snd xs*)) *then xs else* ([], []))
lemma [*code abstype*]:
Con (*rough-state-of S*) = *S*
⟨*proof*⟩

declare *rough-state-of-DPLL-step′-DPLL-step*[*code abstract*]

lemma *Con-DPLL-step-rough-state-of-state-of*[*simp*]:
Con (*DPLL-step* (*rough-state-of s*)) = *state-of* (*DPLL-step* (*rough-state-of s*))
⟨*proof*⟩

A slightly different version of *DPLL-tot* where the returned boolean indicates the result.

definition *DPLL-tot-rep* **where**
DPLL-tot-rep S =
(*let* (*M*, *N*) = (*rough-state-of* (*DPLL-tot S*)) *in* (∀ *A* ∈ *set N*. (∃ *a*∈*set A*. *a* ∈ *lits-of-l* (*M*)), *M*))

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

- export *′a literal* from the SML Module *Clausal-Logic*;

- export the constructor *Con* from *DPLL-W-Implementation*;
- export the *int* constructor from *Arith*.

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

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

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

type-synonym $'a$ $cdcl_W$*-mark* = $'a$ *literal list*
type-synonym $cdcl_W$*-marked-level* = *nat*

type-synonym $'v$ $cdcl_W$*-marked-lit* = ($'v$, $cdcl_W$*-marked-level*, $'v$ $cdcl_W$*-mark*) *marked-lit*
type-synonym $'v$ $cdcl_W$*-marked-lits* = ($'v$, $cdcl_W$*-marked-level*, $'v$ $cdcl_W$*-mark*) *marked-lits*
type-synonym $'v$ $cdcl_W$*-state* =
$'v$ $cdcl_W$*-marked-lits* × $'v$ *literal list list* × $'v$ *literal list list* × *nat* ×
$'v$ *literal list option*

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

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

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

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

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

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

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

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

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

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

abbreviation *raw-remove-cls* **where**

raw-remove-cls $\equiv$ λC (M, N, U, S). $(M$, *removeAll-mset* C N, *removeAll-mset* C U, $S)$

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

abbreviation *raw-S0-cdcl$_W$* N $\equiv$ $(([], N, [], 0, None)$:: *'v cdcl$_W$-state-inv-st*)

fun *mmset-of-mlit'* :: (*'v, nat, 'v literal list*) *marked-lit* $\Rightarrow$ (*'v, nat, 'v clause*) *marked-lit*
where
mmset-of-mlit' (*Propagated L C*) = *Propagated L* (*mset C*) |
mmset-of-mlit' (*Marked L i*) = *Marked L i*

lemma *lit-of-mmset-of-mlit'*[*simp*]:
lit-of (*mmset-of-mlit' xa*) = *lit-of xa*
⟨*proof*⟩

abbreviation *trail* **where**
trail S $\equiv$ *map mmset-of-mlit'* (*raw-trail S*)

abbreviation *clauses-of-l* **where**
clauses-of-l $\equiv$ λL. *mset* (*map mset L*)

global-interpretation *state$_W$-ops*
mset::*'v literal list* $\Rightarrow$ *'v clause*
op # remove1

clauses-of-l op @ λL C. $L \in$ *set C op #* λC. *remove1-cond* (λL. *mset L* = *mset C*)

mset λ*xs ys*. *case-prod append* (*fold* (λx (*ys*, *zs*). (*remove1 x ys*, *x # zs*)) *xs* (*ys*, []))
op # remove1

id id

$\lambda(M, \text{-})$. *map mmset-of-mlit' M* $\lambda(M, \text{-})$. *hd M*
$\lambda(\text{-}, N, \text{-})$. N
$\lambda(\text{-}, \text{-}, U, \text{-})$. U
$\lambda(\text{-}, \text{-}, \text{-}, k, \text{-})$. k
$\lambda(\text{-}, \text{-}, \text{-}, \text{-}, C)$. C

λL (M, S). $(L \# M, S)$
$\lambda(M, S)$. (*tl M*, S)
λC (M, N, S). $(M, C \# N, S)$
λC (M, N, U, S). $(M, N, C \# U, S)$
λC (M, N, U, S). (M, *filter* (λL. *mset L* $\neq$ *mset C*) N, *filter* (λL. *mset L* $\neq$ *mset C*) U, S)
λ(k::*nat*) $(M, N, U, \text{-}, D)$. (M, N, U, k, D)
λD $(M, N, U, k, \text{-})$. (M, N, U, k, D)
λN. ([], N, [], *0*, *None*)
$\lambda(\text{-}, N, U, \text{-})$. ([], N, U, *0*, *None*)
⟨*proof*⟩

lemma *mmset-of-mlit'-mmset-of-mlit*: *mmset-of-mlit' l* = *mmset-of-mlit l*
⟨*proof*⟩

lemma *clauses-of-l-filter-removeAll*:
clauses-of-l [$L \leftarrow a$. *mset L* $\neq$ *mset C*] = *mset* (*removeAll* (*mset C*) (*map mset a*))

$\langle proof \rangle$

interpretation $state_W$
$mset::'v\ literal\ list \Rightarrow 'v\ clause$
$op\ \#\ remove1$

$clauses\text{-}of\text{-}l\ op\ @\ \lambda L\ C.\ L \in set\ C\ op\ \#\ \lambda C.\ remove1\text{-}cond\ (\lambda L.\ mset\ L = mset\ C)$

$mset\ \lambda xs\ ys.\ case\text{-}prod\ append\ (fold\ (\lambda x\ (ys,\ zs).\ (remove1\ x\ ys,\ x\ \#\ zs))\ xs\ (ys,\ []))$
$op\ \#\ remove1$

$id\ id$

$\lambda(M,\ \text{-}).\ map\ mmset\text{-}of\text{-}mlit'\ M\ \lambda(M,\ \text{-}).\ hd\ M$
$\lambda(\text{-},\ N,\ \text{-}).\ N$
$\lambda(\text{-},\ \text{-},\ U,\ \text{-}).\ U$
$\lambda(\text{-},\ \text{-},\ \text{-},\ k,\ \text{-}).\ k$
$\lambda(\text{-},\ \text{-},\ \text{-},\ \text{-},\ C).\ C$

$\lambda L\ (M,\ S).\ (L\ \#\ M,\ S)$
$\lambda(M,\ S).\ (tl\ M,\ S)$
$\lambda C\ (M,\ N,\ S).\ (M,\ C\ \#\ N,\ S)$
$\lambda C\ (M,\ N,\ U,\ S).\ (M,\ N,\ C\ \#\ U,\ S)$
$\lambda C\ (M,\ N,\ U,\ S).\ (M,\ filter\ (\lambda L.\ mset\ L \neq mset\ C)\ N,\ filter\ (\lambda L.\ mset\ L \neq mset\ C)\ U,\ S)$
$\lambda(k::nat)\ (M,\ N,\ U,\ \text{-},\ D).\ (M,\ N,\ U,\ k,\ D)$
$\lambda D\ (M,\ N,\ U,\ k,\ \text{-}).\ (M,\ N,\ U,\ k,\ D)$
$\lambda N.\ ([],\ N,\ [],\ 0,\ None)$
$\lambda(\text{-},\ N,\ U,\ \text{-}).\ ([],\ N,\ U,\ 0,\ None)$
$\langle proof \rangle$

global-interpretation $conflict\text{-}driven\text{-}clause\text{-}learning_W$
$mset::'v\ literal\ list \Rightarrow 'v\ clause$
$op\ \#\ remove1$

$clauses\text{-}of\text{-}l\ op\ @\ \lambda L\ C.\ L \in set\ C\ op\ \#\ \lambda C.\ remove1\text{-}cond\ (\lambda L.\ mset\ L = mset\ C)$

$mset\ \lambda xs\ ys.\ case\text{-}prod\ append\ (fold\ (\lambda x\ (ys,\ zs).\ (remove1\ x\ ys,\ x\ \#\ zs))\ xs\ (ys,\ []))$
$op\ \#\ remove1$

$id\ id$

$\lambda(M,\ \text{-}).\ map\ mmset\text{-}of\text{-}mlit'\ M\ \lambda(M,\ \text{-}).\ hd\ M$
$\lambda(\text{-},\ N,\ \text{-}).\ N$
$\lambda(\text{-},\ \text{-},\ U,\ \text{-}).\ U$
$\lambda(\text{-},\ \text{-},\ \text{-},\ k,\ \text{-}).\ k$
$\lambda(\text{-},\ \text{-},\ \text{-},\ \text{-},\ C).\ C$

$\lambda L\ (M,\ S).\ (L\ \#\ M,\ S)$
$\lambda(M,\ S).\ (tl\ M,\ S)$
$\lambda C\ (M,\ N,\ S).\ (M,\ C\ \#\ N,\ S)$
$\lambda C\ (M,\ N,\ U,\ S).\ (M,\ N,\ C\ \#\ U,\ S)$
$\lambda C\ (M,\ N,\ U,\ S).\ (M,\ filter\ (\lambda L.\ mset\ L \neq mset\ C)\ N,\ filter\ (\lambda L.\ mset\ L \neq mset\ C)\ U,\ S)$
$\lambda(k::nat)\ (M,\ N,\ U,\ \text{-},\ D).\ (M,\ N,\ U,\ k,\ D)$
$\lambda D\ (M,\ N,\ U,\ k,\ \text{-}).\ (M,\ N,\ U,\ k,\ D)$
$\lambda N.\ ([],\ N,\ [],\ 0,\ None)$

λ(-, N, U, -). ([], N, U, 0, $None$)
⟨*proof*⟩

declare *state-simp*[*simp del*] *raw-clauses-def*[*simp*] *state-eq-def*[*simp*]
notation *state-eq* (**infix** ∼ *50*)
term *reduce-trail-to*

lemma *reduce-trail-to-map*[*simp*]:
$reduce\text{-}trail\text{-}to\ (map\ f\ M1) = reduce\text{-}trail\text{-}to\ M1$
⟨*proof*⟩

20.3 CDCL Implementation

20.3.1 Definition of the rules

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

lemma *satisfiable-mset-remdups*[*simp*]:
$satisfiable\ ((mset \circ remdups)\ `\ N) \longleftrightarrow satisfiable\ (mset\ `\ N)$
⟨*proof*⟩

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

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

lemma *convert-Propagated*[*elim*!]:
$mmset\text{-}of\text{-}mlit'\ z = Propagated\ L\ C \Longrightarrow (\exists C'.\ z = Propagated\ L\ C' \wedge C = mset\ C')$
⟨*proof*⟩

lemma *get-rev-level-map-convert*:
$get\text{-}rev\text{-}level\ (map\ mmset\text{-}of\text{-}mlit'\ M)\ n\ x = get\text{-}rev\text{-}level\ M\ n\ x$
⟨*proof*⟩

lemma *get-level-map-convert*[*simp*]:
$get\text{-}level\ (map\ mmset\text{-}of\text{-}mlit'\ M) = get\text{-}level\ M$
⟨*proof*⟩

lemma *get-rev-level-map-mmsetof-mlit*[*simp*]:
$get\text{-}rev\text{-}level\ (map\ mmset\text{-}of\text{-}mlit\ M) = get\text{-}rev\text{-}level\ M$
⟨*proof*⟩

lemma *get-level-map-mmsetof-mlit*[*simp*]:
$get\text{-}level\ (map\ mmset\text{-}of\text{-}mlit\ M) = get\text{-}level\ M$
⟨*proof*⟩

lemma *get-maximum-level-map-convert*[*simp*]:
$get\text{-}maximum\text{-}level\ (map\ mmset\text{-}of\text{-}mlit'\ M)\ D = get\text{-}maximum\text{-}level\ M\ D$
⟨*proof*⟩

lemma *get-all-levels-of-marked-map-convert*[*simp*]:
$get\text{-}all\text{-}levels\text{-}of\text{-}marked\ (map\ mmset\text{-}of\text{-}mlit'\ M) = (get\text{-}all\text{-}levels\text{-}of\text{-}marked\ M)$
⟨*proof*⟩

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

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

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

Definition an abstract type

typedef *'v* $cdcl_W$*-state-inv* = {*S*::*'v* $cdcl_W$*-state-inv-st*. $cdcl_W$*-all-struct-inv* S}
morphisms *rough-state-of state-of*
⟨*proof*⟩

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

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

lemma *undefined-lit-map-convert*[*iff*]:
undefined-lit (*map mmset-of-mlit'* M) L ⟷ *undefined-lit* M L
⟨*proof*⟩

lemma *true-annot-map-convert*[*simp*]: *map mmset-of-mlit'* M ⊨*a* N ⟷ M ⊨*a* N
⟨*proof*⟩

lemma *true-annots-map-convert*[*simp*]: *map mmset-of-mlit'* 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* (M, N, U, k, *None*) (*Propagated* L C # M, N, U, k, *None*)
⟨*proof*⟩

20.3.2 The Transitions

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

$Some\ (L,\ C) \Rightarrow (Propagated\ L\ C\ \#\ M,\ N,\ U,\ k,\ None)$
$\quad |\ None \Rightarrow (M,\ N,\ U,\ k,\ None))$
$|\ S \Rightarrow S)$

lemma *do-propgate-step*:
$do\text{-}propagate\text{-}step\ S \neq S \Longrightarrow propagate\ S\ (do\text{-}propagate\text{-}step\ S)$
⟨*proof*⟩

lemma *do-propagate-step-option*[*simp*]:
$conflicting\ S \neq None \Longrightarrow do\text{-}propagate\text{-}step\ S = S$
⟨*proof*⟩
thm *prod-cases*

lemma *do-propagate-step-no-step*:
assumes $dist$: $\forall c{\in}set\ (raw\text{-}clauses\ S).\ distinct\ c$ **and**
$prop\text{-}step$: $do\text{-}propagate\text{-}step\ S = S$
shows $no\text{-}step\ propagate\ S$
⟨*proof*⟩

Conflict **fun** *find-conflict* **where**
$find\text{-}conflict\ M\ [] = None\ |$
$find\text{-}conflict\ M\ (N\ \#\ Ns) = (if\ (\forall c \in set\ N.\ -c \in lits\text{-}of\text{-}l\ M)\ then\ Some\ N\ else\ find\text{-}conflict\ M\ Ns)$

lemma *find-conflict-Some*:
$find\text{-}conflict\ M\ Ns = Some\ N \Longrightarrow N \in set\ Ns \land M \models as\ CNot\ (mset\ N)$
⟨*proof*⟩

lemma *find-conflict-None*:
$find\text{-}conflict\ M\ Ns = None \longleftrightarrow (\forall N \in set\ Ns.\ \neg M \models as\ CNot\ (mset\ N))$
⟨*proof*⟩

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

definition *do-conflict-step* **where**
$do\text{-}conflict\text{-}step\ S =$
$(case\ S\ of$
$(M,\ N,\ U,\ k,\ None) \Rightarrow$
$(case\ find\text{-}conflict\ M\ (N\ @\ U)\ of$
$Some\ a \Rightarrow (M,\ N,\ U,\ k,\ Some\ a)$
$|\ None \Rightarrow (M,\ N,\ U,\ k,\ None))$
$|\ S \Rightarrow S)$

lemma *do-conflict-step*:
$do\text{-}conflict\text{-}step\ S \neq S \Longrightarrow conflict\ S\ (do\text{-}conflict\text{-}step\ S)$
⟨*proof*⟩

lemma *do-conflict-step-no-step*:
$do\text{-}conflict\text{-}step\ S = S \Longrightarrow no\text{-}step\ conflict\ S$
⟨*proof*⟩

lemma *do-conflict-step-option*[*simp*]:
$conflicting\ S \neq None \Longrightarrow do\text{-}conflict\text{-}step\ S = S$
⟨*proof*⟩

lemma *do-conflict-step-conflicting*[*dest*]:
  $do\text{-}conflict\text{-}step\ S \neq S \Longrightarrow conflicting\ (do\text{-}conflict\text{-}step\ S) \neq None$
  ⟨*proof*⟩

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

lemma *cp-step-is-cdcl*$_W$*-cp*:
  assumes *H*: $do\text{-}cp\text{-}step\ S \neq S$
  shows $cdcl_W\text{-}cp\ S\ (do\text{-}cp\text{-}step\ S)$
⟨*proof*⟩

lemma *do-cp-step-eq-no-prop-no-confl*:
  $do\text{-}cp\text{-}step\ S = S \Longrightarrow do\text{-}conflict\text{-}step\ S = S \wedge do\text{-}propagate\text{-}step\ S = S$
  ⟨*proof*⟩

lemma *no-cdcl*$_W$*-cp-iff-no-propagate-no-conflict*:
  $no\text{-}step\ cdcl_W\text{-}cp\ S \longleftrightarrow no\text{-}step\ propagate\ S \wedge no\text{-}step\ conflict\ S$
  ⟨*proof*⟩

lemma *do-cp-step-eq-no-step*:
  assumes
    H: $do\text{-}cp\text{-}step\ S = S$ **and**
    $\forall c \in set\ (raw\text{-}init\text{-}clss\ S\ @\ raw\text{-}learned\text{-}clss\ S).\ distinct\ c$
  shows $no\text{-}step\ cdcl_W\text{-}cp\ S$
  ⟨*proof*⟩

lemma *cdcl*$_W$*-cp-cdcl*$_W$*-st*: $cdcl_W\text{-}cp\ S\ S' \Longrightarrow cdcl_W^{**}\ S\ S'$
  ⟨*proof*⟩

lemma *cdcl*$_W$*-all-struct-inv-rough-state*[*simp*]: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ (rough\text{-}state\text{-}of\ S)$
  ⟨*proof*⟩

lemma [*simp*]: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S \Longrightarrow rough\text{-}state\text{-}of\ (state\text{-}of\ S) = S$
  ⟨*proof*⟩

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

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

lemma *do-skip-step*:
  $do\text{-}skip\text{-}step\ S \neq S \Longrightarrow skip\ S\ (do\text{-}skip\text{-}step\ S)$
  ⟨*proof*⟩

lemma *do-skip-step-no*:
  $do\text{-}skip\text{-}step\ S = S \Longrightarrow no\text{-}step\ skip\ S$

$\langle proof \rangle$

lemma *do-skip-step-trail-is-None*[*iff*]:
$do\text{-}skip\text{-}step\ S = (a,\ b,\ c,\ d,\ None) \longleftrightarrow S = (a,\ b,\ c,\ d,\ None)$
$\langle proof \rangle$

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

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

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

lemma *do-resolve-step*:
$cdcl_W\text{-}all\text{-}struct\text{-}inv\ S \Longrightarrow do\text{-}resolve\text{-}step\ S \neq S$
$\Longrightarrow resolve\ S\ (do\text{-}resolve\text{-}step\ S)$
$\langle proof \rangle$

lemma *do-resolve-step-no*:
$do\text{-}resolve\text{-}step\ S = S \Longrightarrow no\text{-}step\ resolve\ S$
$\langle proof \rangle$

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

lemma *do-resolve-step-trail-is-None*[*iff*]:
$do\text{-}resolve\text{-}step\ S = (a,\ b,\ c,\ d,\ None) \longleftrightarrow S = (a,\ b,\ c,\ d,\ None)$
$\langle proof \rangle$

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

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

lemma *find-level-decomp-none*:
assumes $find\text{-}level\text{-}decomp\ M\ Ls\ E\ k = None$ **and** $mset\ (L\#D) = mset\ (Ls\ @\ E)$
shows $\neg(L \in set\ Ls \wedge get\text{-}maximum\text{-}level\ M\ (mset\ D) < k \wedge k = get\text{-}level\ M\ L)$
$\langle proof \rangle$

fun *bt-cut* **where**
bt-cut i (Propagated - - # Ls) = bt-cut i Ls |
bt-cut i (Marked K k # Ls) = (if k = Suc i then Some (Marked K k # Ls) else bt-cut i Ls) |
bt-cut i [] = None

lemma *bt-cut-some-decomp*:
  $bt\text{-}cut\ i\ M = Some\ M' \Longrightarrow \exists K\ M2\ M1.\ M = M2\ @\ M' \wedge M' = Marked\ K\ (i+1)\ \#\ M1$
  ⟨*proof*⟩

lemma *bt-cut-not-none*: $M = M2\ @\ Marked\ K\ (Suc\ i)\ \#\ M' \Longrightarrow bt\text{-}cut\ i\ M \neq None$
  ⟨*proof*⟩

lemma *get-all-marked-decomposition-ex*:
  $\exists N.\ (Marked\ K\ (Suc\ i)\ \#\ M',\ N) \in set\ (get\text{-}all\text{-}marked\text{-}decomposition\ (M2@Marked\ K\ (Suc\ i)\ \#\ M'))$
  ⟨*proof*⟩

lemma *bt-cut-in-get-all-marked-decomposition*:
  $bt\text{-}cut\ i\ M = Some\ M' \Longrightarrow \exists M2.\ (M',\ M2) \in set\ (get\text{-}all\text{-}marked\text{-}decomposition\ M)$
  ⟨*proof*⟩

fun *do-backtrack-step* **where**
do-backtrack-step (M, N, U, k, Some D) =
  (case find-level-decomp M D [] k of
    None ⇒ (M, N, U, k, Some D)
  | Some (L, j) ⇒
    (case bt-cut j M of
      Some (Marked - - # Ls) ⇒ (Propagated L D # Ls, N, D # U, j, None)
    | - ⇒ (M, N, U, k, Some D))
) |
do-backtrack-step S = S

lemma *get-all-marked-decomposition-map-convert*:
  (get-all-marked-decomposition (map mmset-of-mlit′ M)) =
    map (λ(a, b). (map mmset-of-mlit′ a, map mmset-of-mlit′ b)) (get-all-marked-decomposition M)
  ⟨*proof*⟩

lemma *do-backtrack-step*:
  assumes
    db: *do-backtrack-step S ≠ S* **and**
    inv: $cdcl_W$*-all-struct-inv S*
  shows *backtrack S (do-backtrack-step S)*
  ⟨*proof*⟩

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

lemma *map-mmset-of-mlit-eq-cons*:
  assumes *map mmset-of-mlit′ M = a @ c*
  obtains *a′ c′* **where**
    M = a′ @ c′ **and**
    a = map mmset-of-mlit′ a′ **and**
    c = map mmset-of-mlit′ c′

⟨proof⟩

lemma *do-backtrack-step-no*:
assumes
db: *do-backtrack-step S = S* **and**
inv: $cdcl_W$*-all-struct-inv S*
shows *no-step backtrack S*
⟨proof⟩

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

Decide **fun** *do-decide-step* **where**
do-decide-step (M, N, U, k, None) =
(case find-first-unused-var N (lits-of-l M) of
None ⇒ (M, N, U, k, None)
| Some L ⇒ (Marked L (Suc k) # M, N, U, k+1, None)) |
do-decide-step S = S

lemma *do-decide-step*:
fixes *S :: 'v* $cdcl_W$*-state-inv-st*
assumes *do-decide-step S ≠ S*
shows *decide S (do-decide-step S)*
⟨proof⟩

lemma *mmset-of-mlit'-eq-Marked*[*iff*]: *mmset-of-mlit' z = Marked x k ⟷ z = Marked x k*
⟨proof⟩

lemma *do-decide-step-no*:
do-decide-step S = S ⟹ no-step decide S
⟨proof⟩

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

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

20.3.3 Code generation

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

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

lemma [*code abstype*]:
Con (rough-state-of S) = S
⟨proof⟩

definition *do-cp-step'* **where**

$do\text{-}cp\text{-}step'\ S = state\text{-}of\ (do\text{-}cp\text{-}step\ (rough\text{-}state\text{-}of\ S))$

typedef $'v\ cdcl_W\text{-}state\text{-}inv\text{-}from\text{-}init\text{-}state = \{S::'v\ cdcl_W\text{-}state\text{-}inv\text{-}st.\ cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$
$\wedge\ cdcl_W\text{-}stgy^{**}\ (raw\text{-}S0\text{-}cdcl_W\ (raw\text{-}init\text{-}clss\ S))\ S\}$
morphisms *rough-state-from-init-state-of state-from-init-state-of*
⟨*proof*⟩

instantiation $cdcl_W$*-state-inv-from-init-state* :: (*type*) *equal*
begin
definition $equal\text{-}cdcl_W\text{-}state\text{-}inv\text{-}from\text{-}init\text{-}state :: 'v\ cdcl_W\text{-}state\text{-}inv\text{-}from\text{-}init\text{-}state \Rightarrow$
$'v\ cdcl_W\text{-}state\text{-}inv\text{-}from\text{-}init\text{-}state \Rightarrow bool$ **where**
$equal\text{-}cdcl_W\text{-}state\text{-}inv\text{-}from\text{-}init\text{-}state\ S\ S' \longleftrightarrow$
$(rough\text{-}state\text{-}from\text{-}init\text{-}state\text{-}of\ S = rough\text{-}state\text{-}from\text{-}init\text{-}state\text{-}of\ S')$
instance
⟨*proof*⟩
end

definition *ConI* **where**
$ConI\ S = state\text{-}from\text{-}init\text{-}state\text{-}of\ (if\ cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$
$\wedge\ cdcl_W\text{-}stgy^{**}\ (raw\text{-}S0\text{-}cdcl_W\ (raw\text{-}init\text{-}clss\ S))\ S\ then\ S\ else\ ([], [], [], 0, None))$

lemma [*code abstype*]:
$ConI\ (rough\text{-}state\text{-}from\text{-}init\text{-}state\text{-}of\ S) = S$
⟨*proof*⟩

definition $id\text{-}of\text{-}I\text{-}to:: 'v\ cdcl_W\text{-}state\text{-}inv\text{-}from\text{-}init\text{-}state \Rightarrow 'v\ cdcl_W\text{-}state\text{-}inv$ **where**
$id\text{-}of\text{-}I\text{-}to\ S = state\text{-}of\ (rough\text{-}state\text{-}from\text{-}init\text{-}state\text{-}of\ S)$

lemma [*code abstract*]:
$rough\text{-}state\text{-}of\ (id\text{-}of\text{-}I\text{-}to\ S) = rough\text{-}state\text{-}from\text{-}init\text{-}state\text{-}of\ S$
⟨*proof*⟩

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

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

lemma *in-clauses-rough-state-of-is-distinct*:
$c{\in}set\ (raw\text{-}init\text{-}clss\ (rough\text{-}state\text{-}of\ S)\ @\ raw\text{-}learned\text{-}clss\ (rough\text{-}state\text{-}of\ S)) \Longrightarrow distinct\ c$
⟨*proof*⟩

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

lemma [*code abstract*]:

rough-state-of (*do-cp-step'* S) = *do-cp-step* (*rough-state-of* S)
⟨*proof*⟩

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

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

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

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

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

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

definition *do-*$cdcl_W$*-stgy-step* **where**
do-$cdcl_W$*-stgy-step* S =
  (*let* T = *do-full1-cp-step* S *in*
    if $T \neq S$
    then T
    else
      (*let* U = (*do-other-step'* T) *in*
       (*do-full1-cp-step* U)))

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

lemma *toS-do-full1-cp-step-not-eq*: *do-full1-cp-step* $S \neq S \Longrightarrow$
    rough-state-of $S \neq$ *rough-state-of* (*do-full1-cp-step* S)
⟨*proof*⟩

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

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

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

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

lemma *do-full1-cp-step-induct*:
($\bigwedge S$. ($S \neq$ *do-cp-step'* $S \Longrightarrow P$ (*do-cp-step'* S)) $\Longrightarrow P\ S$) $\Longrightarrow P\ a0$
⟨*proof*⟩

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

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

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

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

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

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

lemma *do-other-step-not-conflicting-one-more-decide-bt*:
assumes
conflicting (*rough-state-of* S) $\neq$ *None* **and**
conflicting (*rough-state-of* (*do-other-step'* S)) = *None* **and**
do-other-step' $S \neq S$
shows *length* (*filter is-marked* (*raw-trail* (*rough-state-of* S)))

$> \mathit{length}\ (\mathit{filter}\ \mathit{is\text{-}marked}\ (\mathit{raw\text{-}trail}\ (\mathit{rough\text{-}state\text{-}of}\ (\mathit{do\text{-}other\text{-}step}'\ S))))$
⟨proof⟩

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

lemma *rough-state-of-state-of-do-skip-step-rough-state-of*[*simp*]:
$\mathit{rough\text{-}state\text{-}of}\ (\mathit{state\text{-}of}\ (\mathit{do\text{-}skip\text{-}step}\ (\mathit{rough\text{-}state\text{-}of}\ S))) = \mathit{do\text{-}skip\text{-}step}\ (\mathit{rough\text{-}state\text{-}of}\ S)$
⟨proof⟩

lemma *conflicting-do-resolve-step-iff*[*iff*]:
$\mathit{conflicting}\ (\mathit{do\text{-}resolve\text{-}step}\ S) = \mathit{None} \longleftrightarrow \mathit{conflicting}\ S = \mathit{None}$
⟨proof⟩

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

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

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

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

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

lemma *do-backtrack-step-eq-iff-trail-eq*:
$\mathit{do\text{-}backtrack\text{-}step}\ S = S \longleftrightarrow \mathit{raw\text{-}trail}\ (\mathit{do\text{-}backtrack\text{-}step}\ S) = \mathit{raw\text{-}trail}\ S$
⟨proof⟩

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

lemma *do-other-step-eq-iff-trail-eq*:
$\mathit{do\text{-}other\text{-}step}\ S = S \longleftrightarrow \mathit{raw\text{-}trail}\ (\mathit{do\text{-}other\text{-}step}\ S) = \mathit{raw\text{-}trail}\ S$

⟨proof⟩

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

lemma *do-cdcl$_W$-stgy-step-no*:
  assumes *S*: *do-cdcl$_W$-stgy-step S = S*
  shows *no-step cdcl$_W$-stgy (rough-state-of S)*
⟨*proof*⟩

lemma *toS-rough-state-of-state-of-rough-state-from-init-state-of*[*simp*]:
  rough-state-of (state-of (rough-state-from-init-state-of S))
  = rough-state-from-init-state-of S
  ⟨*proof*⟩

lemma *cdcl$_W$-cp-is-rtranclp-cdcl$_W$*: $cdcl_W\text{-}cp\ S\ T \Longrightarrow cdcl_W{}^{**}\ S\ T$
  ⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-cp-is-rtranclp-cdcl$_W$*: $cdcl_W\text{-}cp^{**}\ S\ T \Longrightarrow cdcl_W{}^{**}\ S\ T$
  ⟨*proof*⟩

lemma *cdcl$_W$-stgy-is-rtranclp-cdcl$_W$*:
  $cdcl_W\text{-}stgy\ S\ T \Longrightarrow cdcl_W{}^{**}\ S\ T$
  ⟨*proof*⟩

lemma *cdcl$_W$-stgy-init-clss*: $cdcl_W\text{-}stgy\ S\ T \Longrightarrow cdcl_W\text{-}M\text{-}level\text{-}inv\ S \Longrightarrow init\text{-}clss\ S = init\text{-}clss\ T$
  ⟨*proof*⟩

lemma *clauses-toS-rough-state-of-do-cdcl$_W$-stgy-step*[*simp*]:
  init-clss (rough-state-of (do-cdcl$_W$-stgy-step (state-of (rough-state-from-init-state-of S))))
  = init-clss (rough-state-from-init-state-of S) (**is** - = *init-clss ?S*)
⟨*proof*⟩

lemma *raw-init-clss-do-cp-step*[*simp*]:
  raw-init-clss (do-cp-step S) = raw-init-clss S
⟨*proof*⟩
lemma *raw-init-clss-do-cp-step'*[*simp*]:
  raw-init-clss (rough-state-of (do-cp-step' S)) = raw-init-clss (rough-state-of S)
  ⟨*proof*⟩

lemma *raw-init-clss-rough-state-of-do-full1-cp-step*[*simp*]:
  raw-init-clss (rough-state-of (do-full1-cp-step S))
  = raw-init-clss (rough-state-of S)
  ⟨*proof*⟩

lemma *raw-init-clss-do-skip-def*[*simp*]:
  raw-init-clss (do-skip-step S) = raw-init-clss S
  ⟨*proof*⟩

lemma *raw-init-clss-do-resolve-def*[*simp*]:
  raw-init-clss (do-resolve-step S) = raw-init-clss S
  ⟨*proof*⟩

lemma *raw-init-clss-do-backtrack-def*[*simp*]:
  raw-init-clss (do-backtrack-step S) = raw-init-clss S
  ⟨*proof*⟩

lemma *raw-init-clss-do-decide-def*[*simp*]:

raw-init-clss (*do-decide-step* S) = *raw-init-clss* S
⟨*proof*⟩

lemma *raw-init-clss-rough-state-of-do-other-step'*[*simp*]:
raw-init-clss (*rough-state-of* (*do-other-step'* S))
= *raw-init-clss* (*rough-state-of* S)
⟨*proof*⟩

lemma [*simp*]:
raw-init-clss (*rough-state-of* (*do-cdcl*$_W$*-stgy-step* (*state-of* (*rough-state-from-init-state-of* S))))
=
raw-init-clss (*rough-state-from-init-state-of* S)
⟨*proof*⟩

lemma *rough-state-from-init-state-of-do-cdcl*$_W$*-stgy-step'*[*code abstract*]:
rough-state-from-init-state-of (*do-cdcl*$_W$*-stgy-step'* S) =
rough-state-of (*do-cdcl*$_W$*-stgy-step* (*id-of-I-to* S))
⟨*proof*⟩

All rules together **function** *do-all-cdcl*$_W$*-stgy* **where**
do-all-cdcl$_W$*-stgy* S =
(*let* T = *do-cdcl*$_W$*-stgy-step'* S *in*
if $T = S$ *then* S *else* *do-all-cdcl*$_W$*-stgy* T)
⟨*proof*⟩
termination
⟨*proof*⟩

thm *do-all-cdcl*$_W$*-stgy.induct*
lemma *do-all-cdcl*$_W$*-stgy-induct*:
$(\bigwedge S.$ (*do-cdcl*$_W$*-stgy-step'* $S \neq S \Longrightarrow P$ (*do-cdcl*$_W$*-stgy-step'* S)) $\Longrightarrow P\ S) \Longrightarrow P\ a0$
⟨*proof*⟩

lemma [*simp*]: *raw-init-clss* (*rough-state-from-init-state-of* (*do-all-cdcl*$_W$*-stgy* S)) =
raw-init-clss (*rough-state-from-init-state-of* S)
⟨*proof*⟩

lemma *no-step-cdcl*$_W$*-stgy-cdcl*$_W$*-all*:
fixes S :: *'a cdcl*$_W$*-state-inv-from-init-state*
shows *no-step cdcl*$_W$*-stgy* (*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*** (*rough-state-from-init-state-of* S)
(*rough-state-from-init-state-of* (*do-all-cdcl*$_W$*-stgy* S))
⟨*proof*⟩

Final theorem:

lemma *consistent-interp-mmset-of-mlit*[*simp*]:
consistent-interp (*lit-of* ' *mmset-of-mlit'* ' *set M'*) $\longleftrightarrow$
consistent-interp (*lit-of* ' *set M'*)
⟨*proof*⟩

lemma *DPLL-tot-correct*:
assumes
r: *rough-state-from-init-state-of* (*do-all-cdcl*$_W$*-stgy* (*state-from-init-state-of*

$(([], map\ remdups\ N, [], 0, None)))) = S$ **and**
S: $(M', N', U', k, E) = S$
shows $(E \neq Some\ [] \wedge satisfiable\ (set\ (map\ mset\ N)))$
$\vee (E = Some\ [] \wedge unsatisfiable\ (set\ (map\ mset\ N)))$
⟨*proof*⟩

The Code The SML code is skipped in the documentation, but stays to ensure that some version of the exported code is working. The only difference between the generated code and the one used here is the export of the constructor ConI.

end
theory *CDCL-W-Merge*
imports *CDCL-W-Termination*
begin

21 Link between Weidenbach's and NOT's CDCL

21.1 Inclusion of the states

context $conflict\text{-}driven\text{-}clause\text{-}learning_W$
begin
declare $cdcl_W.intros[intro]$ $cdcl_W\text{-}bj.intros[intro]$ $cdcl_W\text{-}o.intros[intro]$

lemma $backtrack\text{-}no\text{-}cdcl_W\text{-}bj$:
assumes $cdcl$: $cdcl_W\text{-}bj\ T\ U$ **and** inv: $cdcl_W\text{-}M\text{-}level\text{-}inv\ V$
shows $\neg backtrack\ V\ T$
⟨*proof*⟩

inductive $skip\text{-}or\text{-}resolve :: 'st \Rightarrow 'st \Rightarrow bool$ **where**
$s\text{-}or\text{-}r\text{-}skip[intro]$: $skip\ S\ T \Longrightarrow skip\text{-}or\text{-}resolve\ S\ T\ |$
$s\text{-}or\text{-}r\text{-}resolve[intro]$: $resolve\ S\ T \Longrightarrow skip\text{-}or\text{-}resolve\ S\ T$

lemma $rtranclp\text{-}cdcl_W\text{-}bj\text{-}skip\text{-}or\text{-}resolve\text{-}backtrack$:
assumes $cdcl_W\text{-}bj^{**}\ S\ U$ **and** inv: $cdcl_W\text{-}M\text{-}level\text{-}inv\ S$
shows $skip\text{-}or\text{-}resolve^{**}\ S\ U \vee (\exists\ T.\ skip\text{-}or\text{-}resolve^{**}\ S\ T \wedge backtrack\ T\ U)$
⟨*proof*⟩

lemma $rtranclp\text{-}skip\text{-}or\text{-}resolve\text{-}rtranclp\text{-}cdcl_W$:
$skip\text{-}or\text{-}resolve^{**}\ S\ T \Longrightarrow cdcl_W^{**}\ S\ T$
⟨*proof*⟩

definition $backjump\text{-}l\text{-}cond :: 'v\ clause \Rightarrow 'v\ clause \Rightarrow 'v\ literal \Rightarrow 'st \Rightarrow 'st \Rightarrow bool$ **where**
$backjump\text{-}l\text{-}cond \equiv \lambda C\ C'\ L'\ S\ T.\ True$

definition $inv_{NOT} :: 'st \Rightarrow bool$ **where**
$inv_{NOT} \equiv \lambda S.\ no\text{-}dup\ (trail\ S)$

declare $inv_{NOT}\text{-}def[simp]$
end

context $conflict\text{-}driven\text{-}clause\text{-}learning_W$
begin

21.2 More lemmas conflict–propagate and backjumping

21.2.1 Termination

lemma *cdcl$_W$-cp-normalized-element-all-inv*:
  assumes *inv*: *cdcl$_W$-all-struct-inv S*
  obtains *T* **where** *full cdcl$_W$-cp S T*
  ⟨*proof*⟩
thm *backtrackE*

lemma *cdcl$_W$-bj-measure*:
  assumes *cdcl$_W$-bj S T* **and** *cdcl$_W$-M-level-inv S*
  shows *length* (*trail S*) + (*if conflicting S = None then 0 else 1*)
    > *length* (*trail T*) + (*if conflicting T = None then 0 else 1*)
  ⟨*proof*⟩

lemma *wf-cdcl$_W$-bj*:
  wf {(*b*,*a*). *cdcl$_W$-bj a b* ∧ *cdcl$_W$-M-level-inv a*}
  ⟨*proof*⟩

lemma *cdcl$_W$-bj-exists-normal-form*:
  assumes *lev*: *cdcl$_W$-M-level-inv S*
  shows ∃ *T*. *full cdcl$_W$-bj S T*
⟨*proof*⟩
lemma *rtranclp-skip-state-decomp*:
  assumes *skip** S T* **and** *no-dup* (*trail S*)
  shows
    ∃ *M*. *trail S = M @ trail T* ∧ (∀ *m*∈*set M*. ¬*is-marked m*)
    init-clss S = init-clss T
    learned-clss S = learned-clss T
    backtrack-lvl S = backtrack-lvl T
    conflicting S = conflicting T
  ⟨*proof*⟩

21.2.2 More backjumping

Backjumping after skipping or jump directly **lemma** *rtranclp-skip-backtrack-backtrack*:
  assumes
    skip** S T **and**
    backtrack T W **and**
    cdcl$_W$-all-struct-inv S
  shows *backtrack S W*
  ⟨*proof*⟩

lemma *fst-get-all-marked-decomposition-prepend-not-marked*:
  assumes ∀ *m*∈*set MS*. ¬ *is-marked m*
  shows *set* (*map fst* (*get-all-marked-decomposition M*))
    = *set* (*map fst* (*get-all-marked-decomposition* (*MS @ M*)))
    ⟨*proof*⟩

See also ⟦*skip** ?S ?T*; *backtrack ?T ?W*; *cdcl$_W$-all-struct-inv ?S*⟧ ⟹ *backtrack ?S ?W*

lemma *rtranclp-skip-backtrack-backtrack-end*:
  assumes
    skip: *skip** S T* **and**
    bt: *backtrack S W* **and**
    inv: *cdcl$_W$-all-struct-inv S*

shows *backtrack* T W
⟨*proof*⟩

lemma $cdcl_W$*-bj-decomp-resolve-skip-and-bj*:
assumes $cdcl_W$*-bj*** S T **and** *inv*: $cdcl_W$*-M-level-inv* S
shows (*skip-or-resolve*** S T
$\vee$ ($\exists$ U. *skip-or-resolve*** S U $\wedge$ *backtrack* U T))
⟨*proof*⟩

lemma *resolve-skip-deterministic*:
resolve S T $\Longrightarrow$ *skip* S U $\Longrightarrow$ *False*
⟨*proof*⟩

lemma *backtrack-unique*:
assumes
bt-T: *backtrack* S T **and**
bt-U: *backtrack* S U **and**
inv: $cdcl_W$*-all-struct-inv* S
shows $T \sim U$
⟨*proof*⟩

lemma *if-can-apply-backtrack-no-more-resolve*:
assumes
skip: *skip*** S U **and**
bt: *backtrack* S T **and**
inv: $cdcl_W$*-all-struct-inv* S
shows $\neg$*resolve* U V
⟨*proof*⟩

lemma *if-can-apply-resolve-no-more-backtrack*:
assumes
skip: *skip*** S U **and**
resolve: *resolve* S T **and**
inv: $cdcl_W$*-all-struct-inv* S
shows $\neg$*backtrack* U V
⟨*proof*⟩

lemma *if-can-apply-backtrack-skip-or-resolve-is-skip*:
assumes
bt: *backtrack* S T **and**
skip: *skip-or-resolve*** S U **and**
inv: $cdcl_W$*-all-struct-inv* S
shows *skip*** S U
⟨*proof*⟩

lemma $cdcl_W$*-bj-bj-decomp*:
assumes $cdcl_W$*-bj*** S W **and** $cdcl_W$*-all-struct-inv* S
shows
($\exists$ T U V. (λS T. *skip-or-resolve* S T $\wedge$ *no-step backtrack* S)** S T
$\wedge$ (λT U. *resolve* T U $\wedge$ *no-step backtrack* T) T U
$\wedge$ *skip*** U V $\wedge$ *backtrack* V W)
$\vee$ ($\exists$ T U. (λS T. *skip-or-resolve* S T $\wedge$ *no-step backtrack* S)** S T
$\wedge$ (λT U. *resolve* T U $\wedge$ *no-step backtrack* T) T U $\wedge$ *skip*** U W)
$\vee$ ($\exists$ T. *skip*** S T $\wedge$ *backtrack* T W)
$\vee$ *skip*** S W (**is** *?RB* S W $\vee$ *?R* S W $\vee$ *?SB* S W $\vee$ *?S* S W)

⟨*proof*⟩

The case distinction is needed, since $T \sim V$ does not imply that $R^{**}\ T\ V$.

lemma $cdcl_W$*-bj-strongly-confluent*:
assumes
$cdcl_W$*-bj*** S V **and**
$cdcl_W$*-bj*** S T **and**
n-s: *no-step* $cdcl_W$*-bj* V **and**
inv: $cdcl_W$*-all-struct-inv* S
shows $T \sim V \lor cdcl_W\text{-}bj^{**}\ T\ V$
⟨*proof*⟩

lemma $cdcl_W$*-bj-unique-normal-form*:
assumes
ST: $cdcl_W$*-bj*** S T **and** SU: $cdcl_W$*-bj*** S U **and**
n-s-U: *no-step* $cdcl_W$*-bj* U **and**
n-s-T: *no-step* $cdcl_W$*-bj* T **and**
inv: $cdcl_W$*-all-struct-inv* S
shows $T \sim U$
⟨*proof*⟩

lemma *full-*$cdcl_W$*-bj-unique-normal-form*:
assumes *full* $cdcl_W$*-bj* S T **and** *full* $cdcl_W$*-bj* S U **and**
inv: $cdcl_W$*-all-struct-inv* S
shows $T \sim U$
⟨*proof*⟩

21.3 CDCL FW

inductive $cdcl_W$*-merge-restart* :: $'st \Rightarrow 'st \Rightarrow bool$ **where**
fw-r-propagate: *propagate* S S' $\Longrightarrow$ $cdcl_W$*-merge-restart* S S' |
fw-r-conflict: *conflict* S T $\Longrightarrow$ *full* $cdcl_W$*-bj* T U $\Longrightarrow$ $cdcl_W$*-merge-restart* S U |
fw-r-decide: *decide* S S' $\Longrightarrow$ $cdcl_W$*-merge-restart* S S'|
fw-r-rf: $cdcl_W$*-rf* S S' $\Longrightarrow$ $cdcl_W$*-merge-restart* S S'

lemma *rtranclp-*$cdcl_W$*-bj-rtranclp-*$cdcl_W$:
$cdcl_W$*-bj*** S T $\Longrightarrow$ $cdcl_W{}^{**}$ S T
⟨*proof*⟩

lemma $cdcl_W$*-merge-restart-*$cdcl_W$:
assumes $cdcl_W$*-merge-restart* S T
shows $cdcl_W{}^{**}$ S T
⟨*proof*⟩

lemma $cdcl_W$*-merge-restart-conflicting-true-or-no-step*:
assumes $cdcl_W$*-merge-restart* S T
shows *conflicting* T = *None* $\lor$ *no-step* $cdcl_W$ T
⟨*proof*⟩

inductive $cdcl_W$*-merge* :: $'st \Rightarrow 'st \Rightarrow bool$ **where**
fw-propagate: *propagate* S S' $\Longrightarrow$ $cdcl_W$*-merge* S S' |
fw-conflict: *conflict* S T $\Longrightarrow$ *full* $cdcl_W$*-bj* T U $\Longrightarrow$ $cdcl_W$*-merge* S U |
fw-decide: *decide* S S' $\Longrightarrow$ $cdcl_W$*-merge* S S'|
fw-forget: *forget* S S' $\Longrightarrow$ $cdcl_W$*-merge* S S'

lemma *cdcl*$_W$*-merge-cdcl*$_W$*-merge-restart*:
cdcl$_W$*-merge* S T $\Longrightarrow$ *cdcl*$_W$*-merge-restart* S T
⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-merge-tranclp-cdcl*$_W$*-merge-restart*:
cdcl$_W$*-merge*** S T $\Longrightarrow$ *cdcl*$_W$*-merge-restart*** S T
⟨*proof*⟩

lemma *cdcl*$_W$*-merge-rtranclp-cdcl*$_W$:
cdcl$_W$*-merge* S T $\Longrightarrow$ *cdcl*$_W$** S T
⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-merge-rtranclp-cdcl*$_W$:
cdcl$_W$*-merge*** S T $\Longrightarrow$ *cdcl*$_W$** S T
⟨*proof*⟩

lemmas *rulesE* =
skipE resolveE backtrackE propagateE conflictE decideE restartE forgetE

lemma *cdcl*$_W$*-all-struct-inv-tranclp-cdcl*$_W$*-merge-tranclp-cdcl*$_W$*-merge-cdcl*$_W$*-all-struct-inv*:
assumes
inv: *cdcl*$_W$*-all-struct-inv* b
cdcl$_W$*-merge*$^{++}$ b a
shows (λS T. *cdcl*$_W$*-all-struct-inv* S $\wedge$ *cdcl*$_W$*-merge* S T)$^{++}$ b a
⟨*proof*⟩

lemma *backtrack-is-full1-cdcl*$_W$*-bj*:
assumes *bt*: *backtrack* S T **and** *inv*: *cdcl*$_W$*-M-level-inv* S
shows *full1 cdcl*$_W$*-bj* S T
⟨*proof*⟩

lemma *rtrancl-cdcl*$_W$*-conflicting-true-cdcl*$_W$*-merge-restart*:
assumes *cdcl*$_W$** S V **and** *inv*: *cdcl*$_W$*-M-level-inv* S **and** *conflicting* S = *None*
shows (*cdcl*$_W$*-merge-restart*** S V $\wedge$ *conflicting* V = *None*)
$\vee$ ($\exists$ T U. *cdcl*$_W$*-merge-restart*** S T $\wedge$ *conflicting* V $\neq$ *None* $\wedge$ *conflict* T U $\wedge$ *cdcl*$_W$*-bj*** U V)
⟨*proof*⟩

lemma *no-step-cdcl*$_W$*-no-step-cdcl*$_W$*-merge-restart*: *no-step cdcl*$_W$ S $\Longrightarrow$ *no-step cdcl*$_W$*-merge-restart* S
⟨*proof*⟩

lemma *no-step-cdcl*$_W$*-merge-restart-no-step-cdcl*$_W$:
assumes
conflicting S = *None* **and**
cdcl$_W$*-M-level-inv* S **and**
no-step cdcl$_W$*-merge-restart* S
shows *no-step cdcl*$_W$ S
⟨*proof*⟩

lemma *cdcl*$_W$*-merge-restart-no-step-cdcl*$_W$*-bj*:
assumes
cdcl$_W$*-merge-restart* S T
shows *no-step cdcl*$_W$*-bj* T
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-merge-restart-no-step-cdcl$_W$-bj*:
assumes
cdcl$_W$-merge-restart** S T **and**
conflicting S = *None*
shows *no-step cdcl$_W$-bj* T
⟨*proof*⟩

If *conflicting* $S \neq$ *None*, we cannot say anything.

Remark that this theorem does not say anything about well-foundedness: even if you know that one relation is well-founded, it only states that the normal forms are shared.

lemma *conflicting-true-full-cdcl$_W$-iff-full-cdcl$_W$-merge*:
assumes *confl*: *conflicting* S = *None* **and** *lev*: *cdcl$_W$-M-level-inv* S
shows *full cdcl$_W$* S V $\longleftrightarrow$ *full cdcl$_W$-merge-restart* S V
⟨*proof*⟩

lemma *init-state-true-full-cdcl$_W$-iff-full-cdcl$_W$-merge*:
shows *full cdcl$_W$* (*init-state* N) V $\longleftrightarrow$ *full cdcl$_W$-merge-restart* (*init-state* N) V
⟨*proof*⟩

21.4 FW with strategy

21.4.1 The intermediate step

inductive *cdcl$_W$-s$'$* :: $'st \Rightarrow {'st} \Rightarrow$ *bool* **where**
conflict$'$: *full1 cdcl$_W$-cp* S S' $\Longrightarrow$ *cdcl$_W$-s$'$* S S' |
decide$'$: *decide* S S' $\Longrightarrow$ *no-step cdcl$_W$-cp* S $\Longrightarrow$ *full cdcl$_W$-cp* S' S'' $\Longrightarrow$ *cdcl$_W$-s$'$* S S'' |
bj$'$: *full1 cdcl$_W$-bj* S S' $\Longrightarrow$ *no-step cdcl$_W$-cp* S $\Longrightarrow$ *full cdcl$_W$-cp* S' S'' $\Longrightarrow$ *cdcl$_W$-s$'$* S S''

inductive-cases *cdcl$_W$-s$'$E*: *cdcl$_W$-s$'$* S T

lemma *rtranclp-cdcl$_W$-bj-full1-cdclp-cdcl$_W$-stgy*:
cdcl$_W$-bj** S S' $\Longrightarrow$ *full cdcl$_W$-cp* S' S'' $\Longrightarrow$ *cdcl$_W$-stgy*** S S''
⟨*proof*⟩

lemma *cdcl$_W$-s$'$-is-rtranclp-cdcl$_W$-stgy*:
cdcl$_W$-s$'$ S T $\Longrightarrow$ *cdcl$_W$-stgy*** S T
⟨*proof*⟩

lemma *cdcl$_W$-cp-cdcl$_W$-bj-bissimulation*:
assumes
full cdcl$_W$-cp T U **and**
cdcl$_W$-bj** T T' **and**
cdcl$_W$-all-struct-inv T **and**
no-step cdcl$_W$-bj T'
shows *full cdcl$_W$-cp* T' U
$\vee$ ($\exists$ U' U''. *full cdcl$_W$-cp* T' U'' $\wedge$ *full1 cdcl$_W$-bj* U U' $\wedge$ *full cdcl$_W$-cp* U' U''
$\wedge$ *cdcl$_W$-s$'$*** U U'')
⟨*proof*⟩

lemma *cdcl$_W$-cp-cdcl$_W$-bj-bissimulation$'$*:
assumes
full cdcl$_W$-cp T U **and**
cdcl$_W$-bj** T T' **and**
cdcl$_W$-all-struct-inv T **and**
no-step cdcl$_W$-bj T'

shows *full* $cdcl_W$*-cp* T' U
$\lor$ ($\exists$ U'. *full1* $cdcl_W$*-bj* U U' $\land$ ($\forall$ U''. *full* $cdcl_W$*-cp* U' U'' $\longrightarrow$ *full* $cdcl_W$*-cp* T' U''
$\land$ $cdcl_W$*-s'*** U U''))
⟨*proof*⟩

lemma $cdcl_W$*-stgy-*$cdcl_W$*-s'-connected*:
assumes $cdcl_W$*-stgy* S U **and** $cdcl_W$*-all-struct-inv* S
shows $cdcl_W$*-s'* S U
$\lor$ ($\exists$ U'. *full1* $cdcl_W$*-bj* U U' $\land$ ($\forall$ U''. *full* $cdcl_W$*-cp* U' U'' $\longrightarrow$ $cdcl_W$*-s'* S U''))
⟨*proof*⟩

lemma $cdcl_W$*-stgy-*$cdcl_W$*-s'-connected'*:
assumes $cdcl_W$*-stgy* S U **and** $cdcl_W$*-all-struct-inv* S
shows $cdcl_W$*-s'* S U
$\lor$ ($\exists$ U' U''. $cdcl_W$*-s'* S U'' $\land$ *full1* $cdcl_W$*-bj* U U' $\land$ *full* $cdcl_W$*-cp* U' U'')
⟨*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 $\lor$ ($\exists$ T. $cdcl_W$*-s'*** S T $\land$ $cdcl_W$*-bj*$^{++}$ T U $\land$ *conflicting* U $\neq$ *None*)
⟨*proof*⟩

lemma *n-step-*$cdcl_W$*-stgy-iff-no-step-*$cdcl_W$*-cl-*$cdcl_W$*-o*:
assumes *inv*: $cdcl_W$*-all-struct-inv* S
shows *no-step* $cdcl_W$*-s'* S $\longleftrightarrow$ *no-step* $cdcl_W$*-cp* S $\land$ *no-step* $cdcl_W$*-o* S (**is** *?S'* S $\longleftrightarrow$ *?C* S $\land$ *?O* S)
⟨*proof*⟩

lemma $cdcl_W$*-s'-tranclp-*$cdcl_W$:
$cdcl_W$*-s'* S S' $\Longrightarrow$ $cdcl_W{}^{++}$ S S'
⟨*proof*⟩

lemma *tranclp-*$cdcl_W$*-s'-tranclp-*$cdcl_W$:
$cdcl_W$*-s'*$^{++}$ S S' $\Longrightarrow$ $cdcl_W{}^{++}$ S S'
⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-s'-rtranclp-*$cdcl_W$:
$cdcl_W$*-s'*** S S' $\Longrightarrow$ $cdcl_W{}^{**}$ S S'
⟨*proof*⟩

lemma *full-*$cdcl_W$*-stgy-iff-full-*$cdcl_W$*-s'*:
assumes *inv*: $cdcl_W$*-all-struct-inv* S
shows *full* $cdcl_W$*-stgy* S T $\longleftrightarrow$ *full* $cdcl_W$*-s'* S T (**is** *?S* $\longleftrightarrow$ *?S'*)
⟨*proof*⟩

lemma *conflict-step-*$cdcl_W$*-stgy-step*:
assumes
conflict S T
$cdcl_W$*-all-struct-inv* S
shows $\exists$ T. $cdcl_W$*-stgy* S T
⟨*proof*⟩

lemma *decide-step-cdcl$_W$-stgy-step*:
  assumes
    decide S T
    cdcl$_W$-all-struct-inv S
  shows $\exists T.$ *cdcl$_W$-stgy S T*
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-cp-conflicting-Some*:
  cdcl$_W$-cp** *S T* $\Longrightarrow$ *conflicting S = Some D* $\Longrightarrow$ *S = T*
  ⟨*proof*⟩

inductive *cdcl$_W$-merge-cp* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
conflict': *conflict S T* $\Longrightarrow$ *full cdcl$_W$-bj T U* $\Longrightarrow$ *cdcl$_W$-merge-cp S U* |
propagate': *propagate*$^{++}$ *S S'* $\Longrightarrow$ *cdcl$_W$-merge-cp S S'*

lemma *cdcl$_W$-merge-restart-cases*[*consumes 1*, *case-names conflict propagate*]:
  assumes
    cdcl$_W$-merge-cp S U **and**
    $\bigwedge T.$ *conflict S T* $\Longrightarrow$ *full cdcl$_W$-bj T U* $\Longrightarrow$ *P* **and**
    propagate^{++} *S U* $\Longrightarrow$ *P*
  shows *P*
  ⟨*proof*⟩

lemma *cdcl$_W$-merge-cp-tranclp-cdcl$_W$-merge*:
  cdcl$_W$-merge-cp S T $\Longrightarrow$ *cdcl$_W$-merge*$^{++}$ *S T*
  ⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-merge-cp-rtranclp-cdcl$_W$*:
  cdcl$_W$-merge-cp** *S T* $\Longrightarrow$ *cdcl$_W$*** *S T*
⟨*proof*⟩

lemma *full1-cdcl$_W$-bj-no-step-cdcl$_W$-bj*:
  full1 cdcl$_W$-bj S T $\Longrightarrow$ *no-step cdcl$_W$-cp S*
  ⟨*proof*⟩

inductive *cdcl$_W$-s'-without-decide* **where**
conflict'-without-decide[*intro*]: *full1 cdcl$_W$-cp S S'* $\Longrightarrow$ *cdcl$_W$-s'-without-decide S S'* |
bj'-without-decide[*intro*]: *full1 cdcl$_W$-bj S S'* $\Longrightarrow$ *no-step cdcl$_W$-cp S* $\Longrightarrow$ *full cdcl$_W$-cp S' S''*
    $\Longrightarrow$ *cdcl$_W$-s'-without-decide S S''*

lemma *rtranclp-cdcl$_W$-s'-without-decide-rtranclp-cdcl$_W$*:
  cdcl$_W$-s'-without-decide** *S T* $\Longrightarrow$ *cdcl$_W$*** *S T*
  ⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-s'-without-decide-rtranclp-cdcl$_W$-s'*:
  cdcl$_W$-s'-without-decide** *S T* $\Longrightarrow$ *cdcl$_W$-s'*** *S T*
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-merge-cp-is-rtranclp-cdcl$_W$-s'-without-decide*:
  assumes
    cdcl$_W$-merge-cp** *S V*
    conflicting S = None
  shows
    (*cdcl$_W$-s'-without-decide*** *S V*)

$\lor$ ($\exists$ T. *cdcl*$_W$*-s'-without-decide*** S T $\land$ *propagate*$^{++}$ T V)
$\lor$ ($\exists$ T U. *cdcl*$_W$*-s'-without-decide*** S T $\land$ *full1 cdcl*$_W$*-bj* T U $\land$ *propagate*** U V)
⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-s'-without-decide-is-rtranclp-cdcl*$_W$*-merge-cp*:
assumes
cdcl$_W$*-s'-without-decide*** S V **and**
confl: *conflicting* S = *None*
shows
(*cdcl*$_W$*-merge-cp*** S V $\land$ *conflicting* V = *None*)
$\lor$ (*cdcl*$_W$*-merge-cp*** S V $\land$ *conflicting* V $\neq$ *None* $\land$ *no-step cdcl*$_W$*-cp* V $\land$ *no-step cdcl*$_W$*-bj* V)
$\lor$ ($\exists$ T. *cdcl*$_W$*-merge-cp*** S T $\land$ *conflict* T V)
⟨*proof*⟩

lemma *no-step-cdcl*$_W$*-s'-no-ste-cdcl*$_W$*-merge-cp*:
assumes
cdcl$_W$*-all-struct-inv* S
conflicting S = *None*
no-step cdcl$_W$*-s'* S
shows *no-step cdcl*$_W$*-merge-cp* S
⟨*proof*⟩

The *no-step decide* S is needed, since *cdcl*$_W$*-merge-cp* is *cdcl*$_W$*-s'* without *decide*.

lemma *conflicting-true-no-step-cdcl*$_W$*-merge-cp-no-step-s'-without-decide*:
assumes
confl: *conflicting* S = *None* **and**
inv: *cdcl*$_W$*-M-level-inv* S **and**
n-s: *no-step cdcl*$_W$*-merge-cp* S
shows *no-step cdcl*$_W$*-s'-without-decide* S
⟨*proof*⟩

lemma *conflicting-true-no-step-s'-without-decide-no-step-cdcl*$_W$*-merge-cp*:
assumes
inv: *cdcl*$_W$*-all-struct-inv* S **and**
n-s: *no-step cdcl*$_W$*-s'-without-decide* S
shows *no-step cdcl*$_W$*-merge-cp* S
⟨*proof*⟩

lemma *no-step-cdcl*$_W$*-merge-cp-no-step-cdcl*$_W$*-cp*:
no-step cdcl$_W$*-merge-cp* S $\Longrightarrow$ *cdcl*$_W$*-M-level-inv* S $\Longrightarrow$ *no-step cdcl*$_W$*-cp* S
⟨*proof*⟩

lemma *conflicting-not-true-rtranclp-cdcl*$_W$*-merge-cp-no-step-cdcl*$_W$*-bj*:
assumes
conflicting S = *None* **and**
cdcl$_W$*-merge-cp*** S T
shows *no-step cdcl*$_W$*-bj* T
⟨*proof*⟩

lemma *conflicting-true-full-cdcl*$_W$*-merge-cp-iff-full-cdcl*$_W$*-s'-without-decode*:
assumes
confl: *conflicting* S = *None* **and**
inv: *cdcl*$_W$*-all-struct-inv* S
shows
full cdcl$_W$*-merge-cp* S V $\longleftrightarrow$ *full cdcl*$_W$*-s'-without-decide* S V (**is** *?fw* $\longleftrightarrow$ *?s'*)

$\langle proof \rangle$

lemma *conflicting-true-full1-cdcl*$_W$*-merge-cp-iff-full1-cdcl*$_W$*-s'-without-decode*:
assumes
confl: *conflicting* $S = None$ **and**
inv: *cdcl*$_W$*-all-struct-inv* S
shows
full1 cdcl$_W$*-merge-cp* S V $\longleftrightarrow$ *full1 cdcl*$_W$*-s'-without-decide* S V
$\langle proof \rangle$

lemma *conflicting-true-full1-cdcl*$_W$*-merge-cp-imp-full1-cdcl*$_W$*-s'-without-decode*:
assumes
fw: *full1 cdcl*$_W$*-merge-cp* S V **and**
inv: *cdcl*$_W$*-all-struct-inv* S
shows
full1 cdcl$_W$*-s'-without-decide* S V
$\langle proof \rangle$

inductive *cdcl*$_W$*-merge-stgy* **where**
fw-s-cp[*intro*]: *full1 cdcl*$_W$*-merge-cp* S T $\Longrightarrow$ *cdcl*$_W$*-merge-stgy* S T |
fw-s-decide[*intro*]: *decide* S T $\Longrightarrow$ *no-step cdcl*$_W$*-merge-cp* S $\Longrightarrow$ *full cdcl*$_W$*-merge-cp* T U
$\Longrightarrow$ *cdcl*$_W$*-merge-stgy* S U

lemma *cdcl*$_W$*-merge-stgy-tranclp-cdcl*$_W$*-merge*:
assumes *fw*: *cdcl*$_W$*-merge-stgy* S T
shows *cdcl*$_W$*-merge*$^{++}$ S T
$\langle proof \rangle$

lemma *rtranclp-cdcl*$_W$*-merge-stgy-rtranclp-cdcl*$_W$*-merge*:
assumes *fw*: *cdcl*$_W$*-merge-stgy*** S T
shows *cdcl*$_W$*-merge*** S T
$\langle proof \rangle$

lemma *cdcl*$_W$*-merge-stgy-rtranclp-cdcl*$_W$:
cdcl$_W$*-merge-stgy* S T $\Longrightarrow$ *cdcl*$_W$** S T
$\langle proof \rangle$

lemma *rtranclp-cdcl*$_W$*-merge-stgy-rtranclp-cdcl*$_W$:
cdcl$_W$*-merge-stgy*** S T $\Longrightarrow$ *cdcl*$_W$** S T
$\langle proof \rangle$

lemma *cdcl*$_W$*-merge-stgy-cases*[*consumes 1*, *case-names fw-s-cp fw-s-decide*]:
assumes
cdcl$_W$*-merge-stgy* S U
full1 cdcl$_W$*-merge-cp* S U $\Longrightarrow$ P
$\bigwedge T$. *decide* S T $\Longrightarrow$ *no-step cdcl*$_W$*-merge-cp* S $\Longrightarrow$ *full cdcl*$_W$*-merge-cp* T U $\Longrightarrow$ P
shows P
$\langle proof \rangle$

inductive *cdcl*$_W$*-s'-w* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
conflict': *full1 cdcl*$_W$*-s'-without-decide* S S' $\Longrightarrow$ *cdcl*$_W$*-s'-w* S S' |
decide': *decide* S S' $\Longrightarrow$ *no-step cdcl*$_W$*-s'-without-decide* S $\Longrightarrow$ *full cdcl*$_W$*-s'-without-decide* S' S''
$\Longrightarrow$ *cdcl*$_W$*-s'-w* S S''

lemma *cdcl*$_W$*-s'-w-rtranclp-cdcl*$_W$:

$cdcl_W$-s'-w S T $\Longrightarrow$ $cdcl_W$** S T
⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-s'-w-rtranclp-cdcl*$_W$:
$cdcl_W$-s'-w** S T $\Longrightarrow$ $cdcl_W$** S T
⟨*proof*⟩

lemma *no-step-cdcl*$_W$*-cp-no-step-cdcl*$_W$*-s'-without-decide*:
assumes *no-step cdcl*$_W$*-cp* S **and** *conflicting* S = *None* **and** *inv*: *cdcl*$_W$*-M-level-inv* S
shows *no-step cdcl*$_W$*-s'-without-decide* S
⟨*proof*⟩

lemma *no-step-cdcl*$_W$*-cp-no-step-cdcl*$_W$*-merge-restart*:
assumes *no-step cdcl*$_W$*-cp* S **and** *conflicting* S = *None*
shows *no-step cdcl*$_W$*-merge-cp* S
⟨*proof*⟩
lemma *after-cdcl*$_W$*-s'-without-decide-no-step-cdcl*$_W$*-cp*:
assumes *cdcl*$_W$*-s'-without-decide* S T
shows *no-step cdcl*$_W$*-cp* T
⟨*proof*⟩

lemma *no-step-cdcl*$_W$*-s'-without-decide-no-step-cdcl*$_W$*-cp*:
cdcl$_W$*-all-struct-inv* S $\Longrightarrow$ *no-step cdcl*$_W$*-s'-without-decide* S $\Longrightarrow$ *no-step cdcl*$_W$*-cp* S
⟨*proof*⟩

lemma *after-cdcl*$_W$*-s'-w-no-step-cdcl*$_W$*-cp*:
assumes *cdcl*$_W$*-s'-w* S T **and** *cdcl*$_W$*-all-struct-inv* S
shows *no-step cdcl*$_W$*-cp* T
⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-s'-w-no-step-cdcl*$_W$*-cp-or-eq*:
assumes *cdcl*$_W$*-s'-w*** S T **and** *cdcl*$_W$*-all-struct-inv* S
shows $S = T \lor$ *no-step cdcl*$_W$*-cp* T
⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-merge-stgy'-no-step-cdcl*$_W$*-cp-or-eq*:
assumes *cdcl*$_W$*-merge-stgy*** S T **and** *inv*: *cdcl*$_W$*-all-struct-inv* S
shows $S = T \lor$ *no-step cdcl*$_W$*-cp* T
⟨*proof*⟩

lemma *no-step-cdcl*$_W$*-s'-without-decide-no-step-cdcl*$_W$*-bj*:
assumes *no-step cdcl*$_W$*-s'-without-decide* S **and** *inv*: *cdcl*$_W$*-all-struct-inv* S
shows *no-step cdcl*$_W$*-bj* S
⟨*proof*⟩

lemma *cdcl*$_W$*-s'-w-no-step-cdcl*$_W$*-bj*:
assumes *cdcl*$_W$*-s'-w* S T **and** *cdcl*$_W$*-all-struct-inv* S
shows *no-step cdcl*$_W$*-bj* T
⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-s'-w-no-step-cdcl*$_W$*-bj-or-eq*:
assumes *cdcl*$_W$*-s'-w*** S T **and** *cdcl*$_W$*-all-struct-inv* S
shows $S = T \lor$ *no-step cdcl*$_W$*-bj* T
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-s'-no-step-cdcl$_W$-s'-without-decide-decomp-into-cdcl$_W$-merge*:
assumes
cdcl$_W$-s'** R V **and**
conflicting R = None **and**
inv: *cdcl$_W$-all-struct-inv R*
shows (*cdcl$_W$-merge-stgy** R V* ∧ *conflicting V = None*)
∨ (*cdcl$_W$-merge-stgy** R V* ∧ *conflicting V ≠ None* ∧ *no-step cdcl$_W$-bj V*)
∨ (∃ *S T U*. *cdcl$_W$-merge-stgy** R S* ∧ *no-step cdcl$_W$-merge-cp S* ∧ *decide S T*
∧ *cdcl$_W$-merge-cp** T U* ∧ *conflict U V*)
∨ (∃ *S T*. *cdcl$_W$-merge-stgy** R S* ∧ *no-step cdcl$_W$-merge-cp S* ∧ *decide S T*
∧ *cdcl$_W$-merge-cp** T V*
∧ *conflicting V = None*)
∨ (*cdcl$_W$-merge-cp** R V* ∧ *conflicting V = None*)
∨ (∃ *U*. *cdcl$_W$-merge-cp** R U* ∧ *conflict U V*)
⟨*proof*⟩

lemma *decide-rtranclp-cdcl$_W$-s'-rtranclp-cdcl$_W$-s'*:
assumes
dec: *decide S T* **and**
cdcl$_W$-s'** T U **and**
n-s-S: *no-step cdcl$_W$-cp S* **and**
no-step cdcl$_W$-cp U
shows *cdcl$_W$-s'** S U*
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-merge-stgy-rtranclp-cdcl$_W$-s'*:
assumes
cdcl$_W$-merge-stgy** R V **and**
inv: *cdcl$_W$-all-struct-inv R*
shows *cdcl$_W$-s'** R V*
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-merge-stgy-distinct-mset-clauses*:
assumes *invR*: *cdcl$_W$-all-struct-inv R* **and**
st: *cdcl$_W$-merge-stgy** R S* **and**
dist: *distinct-mset* (*clauses R*) **and**
R: *trail R* = []
shows *distinct-mset* (*clauses S*)
⟨*proof*⟩

lemma *no-step-cdcl$_W$-s'-no-step-cdcl$_W$-merge-stgy*:
assumes
inv: *cdcl$_W$-all-struct-inv R* **and** *s'*: *no-step cdcl$_W$-s' R*
shows *no-step cdcl$_W$-merge-stgy R*
⟨*proof*⟩
end

We will discharge the assumption later.

locale *conflict-driven-clause-learning$_W$-termination* =
conflict-driven-clause-learning$_W$ +
assumes *wf-cdcl$_W$-merge-inv*: *wf* {(*T*, *S*). *cdcl$_W$-all-struct-inv S* ∧ *cdcl$_W$-merge S T*}
begin

lemma *wf-tranclp-cdcl$_W$-merge*: *wf* {(*T*, *S*). *cdcl$_W$-all-struct-inv S* ∧ *cdcl$_W$-merge^{++} S T*}
⟨*proof*⟩

lemma *wf-cdcl$_W$-merge-cp*:
wf$\{(T, S)$. *cdcl$_W$-all-struct-inv* $S \wedge$ *cdcl$_W$-merge-cp* S $T\}$
⟨*proof*⟩

lemma *wf-cdcl$_W$-merge-stgy*:
wf$\{(T, S)$. *cdcl$_W$-all-struct-inv* $S \wedge$ *cdcl$_W$-merge-stgy* S $T\}$
⟨*proof*⟩

lemma *cdcl$_W$-merge-cp-obtain-normal-form*:
assumes *inv*: *cdcl$_W$-all-struct-inv R*
obtains *S* **where** *full cdcl$_W$-merge-cp R S*
⟨*proof*⟩

lemma *no-step-cdcl$_W$-merge-stgy-no-step-cdcl$_W$-s'*:
assumes
inv: *cdcl$_W$-all-struct-inv R* **and**
confl: *conflicting* $R = $ *None* **and**
n-s: *no-step cdcl$_W$-merge-stgy R*
shows *no-step cdcl$_W$-s' R*
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-merge-cp-no-step-cdcl$_W$-bj*:
assumes *conflicting* $R = $ *None* **and** *cdcl$_W$-merge-cp*** *R S*
shows *no-step cdcl$_W$-bj S*
⟨*proof*⟩

lemma *rtranclp-cdcl$_W$-merge-stgy-no-step-cdcl$_W$-bj*:
assumes *confl*: *conflicting* $R = $ *None* **and** *cdcl$_W$-merge-stgy*** *R S*
shows *no-step cdcl$_W$-bj S*
⟨*proof*⟩

end

end
theory *CDCL-W-Restart*
imports *CDCL-W-Merge*
begin

21.5 Adding Restarts

locale *cdcl$_W$-restart* =
conflict-driven-clause-learning$_W$
— functions for clauses:
mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss

— functions for the conflicting clause:
mset-ccls union-ccls insert-ccls remove-clit

— conversion
ccls-of-cls cls-of-ccls

— functions for the state:
— access functions:
trail hd-raw-trail raw-init-clss raw-learned-clss backtrack-lvl raw-conflicting

— changing state:
cons-trail tl-trail add-init-cls add-learned-cls remove-cls update-backtrack-lvl update-conflicting

— get state:
init-state
restart-state
for
mset-cls:: *'cls* ⇒ *'v clause* **and**
insert-cls :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**
remove-lit :: *'v literal* ⇒ *'cls* ⇒ *'cls* **and**

mset-clss:: *'clss* ⇒ *'v clauses* **and**
union-clss :: *'clss* ⇒ *'clss* ⇒ *'clss* **and**
in-clss :: *'cls* ⇒ *'clss* ⇒ *bool* **and**
insert-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**
remove-from-clss :: *'cls* ⇒ *'clss* ⇒ *'clss* **and**

mset-ccls:: *'ccls* ⇒ *'v clause* **and**
union-ccls :: *'ccls* ⇒ *'ccls* ⇒ *'ccls* **and**
insert-ccls :: *'v literal* ⇒ *'ccls* ⇒ *'ccls* **and**
remove-clit :: *'v literal* ⇒ *'ccls* ⇒ *'ccls* **and**

ccls-of-cls :: *'cls* ⇒ *'ccls* **and**
cls-of-ccls :: *'ccls* ⇒ *'cls* **and**

trail :: *'st* ⇒ (*'v*, *nat*, *'v clause*) *marked-lits* **and**
hd-raw-trail :: *'st* ⇒ (*'v*, *nat*, *'cls*) *marked-lit* **and**
raw-init-clss :: *'st* ⇒ *'clss* **and**
raw-learned-clss :: *'st* ⇒ *'clss* **and**
backtrack-lvl :: *'st* ⇒ *nat* **and**
raw-conflicting :: *'st* ⇒ *'ccls option* **and**

cons-trail :: (*'v*, *nat*, *'cls*) *marked-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒ *'st* **and**
add-init-cls :: *'cls* ⇒ *'st* ⇒ *'st* **and**
add-learned-cls :: *'cls* ⇒ *'st* ⇒ *'st* **and**
remove-cls :: *'cls* ⇒ *'st* ⇒ *'st* **and**
update-backtrack-lvl :: *nat* ⇒ *'st* ⇒ *'st* **and**
update-conflicting :: *'ccls option* ⇒ *'st* ⇒ *'st* **and**

init-state :: *'clss* ⇒ *'st* **and**
restart-state :: *'st* ⇒ *'st* +
fixes *f* :: *nat* ⇒ *nat*
assumes *f*: *unbounded f*
begin

The condition of the differences of cardinality has to be strict. Otherwise, you could be in a strange state, where nothing remains to do, but a restart is done. See the proof of well-foundedness.

inductive $cdcl_W$*-merge-with-restart* **where**
restart-step:
($cdcl_W$*-merge-stgy*^^(*card* (*set-mset* (*learned-clss T*)) − *card* (*set-mset* (*learned-clss S*)))) *S T*
⟹ *card* (*set-mset* (*learned-clss T*)) − *card* (*set-mset* (*learned-clss S*)) > *f n*
⟹ *restart T U* ⟹ $cdcl_W$*-merge-with-restart* (*S*, *n*) (*U*, *Suc n*) |

restart-full: *full1 cdcl*$_W$*-merge-stgy S T* $\Longrightarrow$ *cdcl*$_W$*-merge-with-restart* (*S*, *n*) (*T*, *Suc n*)

lemma *cdcl*$_W$*-merge-with-restart S T* $\Longrightarrow$ *cdcl*$_W$*-merge-restart*** (*fst S*) (*fst T*)
⟨*proof*⟩

lemma *cdcl*$_W$*-merge-with-restart-rtranclp-cdcl*$_W$:
cdcl$_W$*-merge-with-restart S T* $\Longrightarrow$ *cdcl*$_W$** (*fst S*) (*fst T*)
⟨*proof*⟩

lemma *cdcl*$_W$*-merge-with-restart-increasing-number*:
cdcl$_W$*-merge-with-restart S T* $\Longrightarrow$ *snd T* = *1* + *snd S*
⟨*proof*⟩

lemma *full1 cdcl*$_W$*-merge-stgy S T* $\Longrightarrow$ *cdcl*$_W$*-merge-with-restart* (*S*, *n*) (*T*, *Suc n*)
⟨*proof*⟩

lemma *cdcl*$_W$*-all-struct-inv-learned-clss-bound*:
assumes *inv*: *cdcl*$_W$*-all-struct-inv S*
shows *set-mset* (*learned-clss S*) $\subseteq$ *simple-clss* (*atms-of-mm* (*init-clss S*))
⟨*proof*⟩

lemma *cdcl*$_W$*-merge-with-restart-init-clss*:
cdcl$_W$*-merge-with-restart S T* $\Longrightarrow$ *cdcl*$_W$*-M-level-inv* (*fst S*) $\Longrightarrow$
init-clss (*fst S*) = *init-clss* (*fst T*)
⟨*proof*⟩

lemma
wf {(*T*, *S*). *cdcl*$_W$*-all-struct-inv* (*fst S*) $\wedge$ *cdcl*$_W$*-merge-with-restart S T*}
⟨*proof*⟩

lemma *cdcl*$_W$*-merge-with-restart-distinct-mset-clauses*:
assumes *invR*: *cdcl*$_W$*-all-struct-inv* (*fst R*) **and**
st: *cdcl*$_W$*-merge-with-restart R S* **and**
dist: *distinct-mset* (*clauses* (*fst R*)) **and**
R: *trail* (*fst R*) = []
shows *distinct-mset* (*clauses* (*fst S*))
⟨*proof*⟩

inductive *cdcl*$_W$*-with-restart* **where**
restart-step:
(*cdcl*$_W$*-stgy*^^(*card* (*set-mset* (*learned-clss T*)) − *card* (*set-mset* (*learned-clss S*)))) *S T* $\Longrightarrow$
card (*set-mset* (*learned-clss T*)) − *card* (*set-mset* (*learned-clss S*)) > *f n* $\Longrightarrow$
restart T U $\Longrightarrow$
cdcl$_W$*-with-restart* (*S*, *n*) (*U*, *Suc n*) |
restart-full: *full1 cdcl*$_W$*-stgy S T* $\Longrightarrow$ *cdcl*$_W$*-with-restart* (*S*, *n*) (*T*, *Suc n*)

lemma *cdcl*$_W$*-with-restart-rtranclp-cdcl*$_W$:
cdcl$_W$*-with-restart S T* $\Longrightarrow$ *cdcl*$_W$** (*fst S*) (*fst T*)
⟨*proof*⟩

lemma *cdcl*$_W$*-with-restart-increasing-number*:
cdcl$_W$*-with-restart S T* $\Longrightarrow$ *snd T* = *1* + *snd S*
⟨*proof*⟩

lemma *full1 cdcl*$_W$*-stgy S T* $\Longrightarrow$ *cdcl*$_W$*-with-restart* (*S*, *n*) (*T*, *Suc n*)

$\langle proof \rangle$

lemma *cdcl$_W$-with-restart-init-clss*:
$cdcl_W\text{-}with\text{-}restart\ S\ T \Longrightarrow\ cdcl_W\text{-}M\text{-}level\text{-}inv\ (fst\ S) \Longrightarrow init\text{-}clss\ (fst\ S) = init\text{-}clss\ (fst\ T)$
$\langle proof \rangle$

lemma
$wf\ \{(T, S).\ cdcl_W\text{-}all\text{-}struct\text{-}inv\ (fst\ S) \wedge cdcl_W\text{-}with\text{-}restart\ S\ T\}$
$\langle proof \rangle$

lemma *cdcl$_W$-with-restart-distinct-mset-clauses*:
assumes *invR*: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ (fst\ R)$ **and**
st: $cdcl_W\text{-}with\text{-}restart\ R\ S$ **and**
dist: $distinct\text{-}mset\ (clauses\ (fst\ R))$ **and**
R: $trail\ (fst\ R) = []$
shows $distinct\text{-}mset\ (clauses\ (fst\ S))$
$\langle proof \rangle$
end

locale *luby-sequence* =
fixes $ur :: nat$
assumes $ur > 0$
begin

lemma *exists-luby-decomp*:
fixes $i :: nat$
shows $\exists k::nat.\ (2\ \hat{}\ (k - 1) \leq i \wedge i < 2\ \hat{}\ k - 1) \vee i = 2\ \hat{}\ k - 1$
$\langle proof \rangle$

Luby sequences are defined by:

- $2^k - 1$, if $i = (2::'a)^k - (1::'a)$
- $luby\text{-}sequence\text{-}core\ (i - 2^{k-1} + 1)$, if $(2::'a)^{k-1} \leq i$ and $i \leq (2::'a)^k - (1::'a)$

Then the sequence is then scaled by a constant unit run (called *ur* here), strictly positive.

function $luby\text{-}sequence\text{-}core :: nat \Rightarrow nat$ **where**
$luby\text{-}sequence\text{-}core\ i =$
$(if\ \exists k.\ i = 2\hat{}k - 1$
$then\ 2\hat{}((SOME\ k.\ i = 2\hat{}k - 1) - 1)$
$else\ luby\text{-}sequence\text{-}core\ (i - 2\hat{}((SOME\ k.\ 2\hat{}(k-1) \leq i \wedge i < 2\hat{}k - 1) - 1) + 1))$
$\langle proof \rangle$
termination
$\langle proof \rangle$

function $natlog2 :: nat \Rightarrow nat$ **where**
$natlog2\ n = (if\ n = 0\ then\ 0\ else\ 1 + natlog2\ (n\ div\ 2))$
$\langle proof \rangle$
termination $\langle proof \rangle$

declare *natlog2.simps*[*simp del*]

declare *luby-sequence-core.simps*[*simp del*]

lemma *two-pover-n-eq-two-power-n'-eq*:

assumes H: $(2::nat)$ ^ $(k::nat) - 1 = 2$ ^ $k' - 1$
shows $k' = k$
⟨proof⟩

lemma *luby-sequence-core-two-power-minus-one*:
luby-sequence-core $(2\hat{}k - 1) = 2\hat{}(k-1)$ (**is** $?L = ?K$)
⟨proof⟩

lemma *different-luby-decomposition-false*:
assumes
H: 2 ^ $(k - Suc\ 0) \leq i$ **and**
k': $i < 2$ ^ $k' - Suc\ 0$ **and**
k-k': $k > k'$
shows *False*
⟨proof⟩

lemma *luby-sequence-core-not-two-power-minus-one*:
assumes
k-i: 2 ^ $(k - 1) \leq i$ **and**
i-k: $i < 2\hat{}k - 1$
shows *luby-sequence-core* i = *luby-sequence-core* $(i - 2$ ^ $(k - 1) + 1)$
⟨proof⟩

lemma *unbounded-luby-sequence-core*: *unbounded luby-sequence-core*
⟨proof⟩

abbreviation *luby-sequence* :: *nat* ⇒ *nat* **where**
luby-sequence $n \equiv$ *ur* $*$ *luby-sequence-core* n

lemma *bounded-luby-sequence*: *unbounded luby-sequence*
⟨proof⟩

lemma *luby-sequence-core-0*: *luby-sequence-core* $0 = 1$
⟨proof⟩

lemma *luby-sequence-core* $n \geq 1$
⟨proof⟩
end

locale *luby-sequence-restart* =
luby-sequence ur +
conflict-driven-clause-learning$_W$ — functions for clauses:
mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss

— functions for the conflicting clause:
mset-ccls union-ccls insert-ccls remove-clit

— conversion
ccls-of-cls cls-of-ccls

— functions for the state:
— access functions:
trail hd-raw-trail raw-init-clss raw-learned-clss backtrack-lvl raw-conflicting
— changing state:

cons-trail tl-trail add-init-cls add-learned-cls remove-cls update-backtrack-lvl
update-conflicting

— get state:
init-state
restart-state
for
ur :: *nat* **and**
mset-cls:: $'cls \Rightarrow 'v$ *clause* **and**
insert-cls :: $'v$ *literal* $\Rightarrow 'cls \Rightarrow 'cls$ **and**
remove-lit :: $'v$ *literal* $\Rightarrow 'cls \Rightarrow 'cls$ **and**

mset-clss:: $'clss \Rightarrow 'v$ *clauses* **and**
union-clss :: $'clss \Rightarrow 'clss \Rightarrow 'clss$ **and**
in-clss :: $'cls \Rightarrow 'clss \Rightarrow$ *bool* **and**
insert-clss :: $'cls \Rightarrow 'clss \Rightarrow 'clss$ **and**
remove-from-clss :: $'cls \Rightarrow 'clss \Rightarrow 'clss$ **and**

mset-ccls:: $'ccls \Rightarrow 'v$ *clause* **and**
union-ccls :: $'ccls \Rightarrow 'ccls \Rightarrow 'ccls$ **and**
insert-ccls :: $'v$ *literal* $\Rightarrow 'ccls \Rightarrow 'ccls$ **and**
remove-clit :: $'v$ *literal* $\Rightarrow 'ccls \Rightarrow 'ccls$ **and**

ccls-of-cls :: $'cls \Rightarrow 'ccls$ **and**
cls-of-ccls :: $'ccls \Rightarrow 'cls$ **and**

trail :: $'st \Rightarrow ('v, nat, 'v$ *clause*$)$ *marked-lits* **and**
hd-raw-trail :: $'st \Rightarrow ('v, nat, 'cls)$ *marked-lit* **and**
raw-init-clss :: $'st \Rightarrow 'clss$ **and**
raw-learned-clss :: $'st \Rightarrow 'clss$ **and**
backtrack-lvl :: $'st \Rightarrow nat$ **and**
raw-conflicting :: $'st \Rightarrow 'ccls$ *option* **and**

cons-trail :: $('v, nat, 'cls)$ *marked-lit* $\Rightarrow 'st \Rightarrow 'st$ **and**
tl-trail :: $'st \Rightarrow 'st$ **and**
add-init-cls :: $'cls \Rightarrow 'st \Rightarrow 'st$ **and**
add-learned-cls :: $'cls \Rightarrow 'st \Rightarrow 'st$ **and**
remove-cls :: $'cls \Rightarrow 'st \Rightarrow 'st$ **and**
update-backtrack-lvl :: $nat \Rightarrow 'st \Rightarrow 'st$ **and**
update-conflicting :: $'ccls$ *option* $\Rightarrow 'st \Rightarrow 'st$ **and**

init-state :: $'clss \Rightarrow 'st$ **and**
restart-state :: $'st \Rightarrow 'st$
begin

sublocale $cdcl_W$*-restart* - *luby-sequence*
⟨*proof*⟩

end
end
theory *CDCL-WNOT*
imports *CDCL-NOT CDCL-W-Termination CDCL-W-Merge*
begin

22 Link between Weidenbach's and NOT's CDCL

22.1 Inclusion of the states

declare *upt.simps(2)[simp del]*

fun *convert-marked-lit-from-W* **where**
convert-marked-lit-from-W (Propagated L -) = Propagated L () |
convert-marked-lit-from-W (Marked L -) = Marked L ()

abbreviation *convert-trail-from-W* ::
 ('v, 'lvl, 'a) marked-lit list
 ⇒ *('v, unit, unit) marked-lit list* **where**
convert-trail-from-W ≡ map convert-marked-lit-from-W

lemma *lits-of-l-convert-trail-from-W[simp]*:
 lits-of-l (convert-trail-from-W M) = lits-of-l M
 ⟨*proof*⟩

lemma *lit-of-convert-trail-from-W[simp]*:
 lit-of (convert-marked-lit-from-W L) = lit-of L
 ⟨*proof*⟩

lemma *no-dup-convert-from-W[simp]*:
 no-dup (convert-trail-from-W M) ⟷ no-dup M
 ⟨*proof*⟩

lemma *convert-trail-from-W-true-annots[simp]*:
 convert-trail-from-W M ⊨as C ⟷ M ⊨as C
 ⟨*proof*⟩

lemma *defined-lit-convert-trail-from-W[simp]*:
 defined-lit (convert-trail-from-W S) L ⟷ defined-lit S L
 ⟨*proof*⟩

The values *0* and {#} are dummy values.

consts *dummy-cls* :: *'cls*
fun *convert-marked-lit-from-NOT*
 :: *('a, 'e, 'b) marked-lit ⇒ ('a, nat, 'cls) marked-lit* **where**
convert-marked-lit-from-NOT (Propagated L -) = Propagated L dummy-cls |
convert-marked-lit-from-NOT (Marked L -) = Marked L 0

abbreviation *convert-trail-from-NOT* **where**
convert-trail-from-NOT ≡ map convert-marked-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-marked-lit-from-W (*convert-marked-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-marked-lit-from-NOT*[*iff*]:
lit-of (*convert-marked-lit-from-NOT L*) = *lit-of L*
⟨*proof*⟩

sublocale $state_W$ ⊆ *dpll-state-ops*
mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss
λ*S*. *convert-trail-from-W* (*trail S*)
raw-clauses
λ*L S*. *cons-trail* (*convert-marked-lit-from-NOT L*) *S*
λ*S*. *tl-trail S*
λ*C S*. *add-learned-cls C S*
λ*C S*. *remove-cls C S*
⟨*proof*⟩

context $state_W$
begin
lemma *convert-marked-lit-from-W-convert-marked-lit-from-NOT*[*simp*]:
convert-marked-lit-from-W (*mmset-of-mlit* (*convert-marked-lit-from-NOT L*)) = *L*
⟨*proof*⟩
end

sublocale $state_W$ ⊆ *dpll-state*
mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss
λ*S*. *convert-trail-from-W* (*trail S*)
raw-clauses
λ*L S*. *cons-trail* (*convert-marked-lit-from-NOT L*) *S*
λ*S*. *tl-trail S*
λ*C S*. *add-learned-cls C S*
λ*C S*. *remove-cls C S*
⟨*proof*⟩

context $state_W$
begin
declare $state\text{-}simp_{NOT}$[*simp del*]

end

sublocale $conflict\text{-}driven\text{-}clause\text{-}learning_W$ ⊆ $cdcl_{NOT}$*-merge-bj-learn-ops*
mset-cls insert-cls remove-lit

mset-clss union-clss in-clss insert-clss remove-from-clss
λS. *convert-trail-from-W* (*trail* S)
raw-clauses
$\lambda L\ S$. *cons-trail* (*convert-marked-lit-from-NOT* L) S
λS. *tl-trail* S
$\lambda C\ S$. *add-learned-cls* $C\ S$
$\lambda C\ S$. *remove-cls* $C\ S$
λ- -. *True*
λ- S. *raw-conflicting* S = *None*
$\lambda C\ C'\ L'\ S\ T$. *backjump-l-cond* $C\ C'\ L'\ S\ T$
$\wedge$ *distinct-mset* $(C' + \{\#L'\#\})$ $\wedge$ $\neg$*tautology* $(C' + \{\#L'\#\})$
⟨*proof*⟩

thm $cdcl_{NOT}$*-merge-bj-learn-proxy.axioms*
sublocale *conflict-driven-clause-learning*$_W$ $\subseteq$ $cdcl_{NOT}$*-merge-bj-learn-proxy*
mset-cls insert-cls remove-lit
mset-clss union-clss in-clss insert-clss remove-from-clss
λS. *convert-trail-from-W* (*trail* S)
raw-clauses
$\lambda L\ S$. *cons-trail* (*convert-marked-lit-from-NOT* L) S
λS. *tl-trail* S
$\lambda C\ S$. *add-learned-cls* $C\ S$
$\lambda C\ S$. *remove-cls* $C\ S$

λ- -. *True*
λ- S. *raw-conflicting* S = *None*
backjump-l-cond
inv_{NOT}
⟨*proof*⟩

sublocale *conflict-driven-clause-learning*$_W$ $\subseteq$ $cdcl_{NOT}$*-merge-bj-learn-proxy2* - - - - - - - -
λS. *convert-trail-from-W* (*trail* S)
raw-clauses
$\lambda L\ S$. *cons-trail* (*convert-marked-lit-from-NOT* L) S
λS. *tl-trail* S
$\lambda C\ S$. *add-learned-cls* $C\ S$
$\lambda C\ S$. *remove-cls* $C\ S$
λ- -. *True*
λ- S. *raw-conflicting* S = *None* *backjump-l-cond* inv_{NOT}
⟨*proof*⟩

sublocale *conflict-driven-clause-learning*$_W$ $\subseteq$ $cdcl_{NOT}$*-merge-bj-learn* - - - - - - - -
λS. *convert-trail-from-W* (*trail* S)
raw-clauses
$\lambda L\ S$. *cons-trail* (*convert-marked-lit-from-NOT* L) S
λS. *tl-trail* S
$\lambda C\ S$. *add-learned-cls* $C\ S$
$\lambda C\ S$. *remove-cls* $C\ S$
backjump-l-cond
λ- -. *True*
λ- S. *raw-conflicting* S = *None* inv_{NOT}
⟨*proof*⟩

context *conflict-driven-clause-learning*$_W$
begin

Notations are lost while proving locale inclusion:

notation *state-eq*$_{NOT}$ (**infix** $\sim_{NOT}$ *50*)

22.2 Additional Lemmas between NOT and W states

lemma *trail*$_W$*-eq-reduce-trail-to*$_{NOT}$*-eq*:
$trail\ S = trail\ T \Longrightarrow trail\ (reduce\text{-}trail\text{-}to_{NOT}\ F\ S) = trail\ (reduce\text{-}trail\text{-}to_{NOT}\ F\ T)$
⟨*proof*⟩

lemma *trail-reduce-trail-to*$_{NOT}$*-add-learned-cls*:
$no\text{-}dup\ (trail\ S) \Longrightarrow$
$trail\ (reduce\text{-}trail\text{-}to_{NOT}\ M\ (add\text{-}learned\text{-}cls\ D\ S)) = trail\ (reduce\text{-}trail\text{-}to_{NOT}\ M\ S)$
⟨*proof*⟩

lemma *reduce-trail-to*$_{NOT}$*-reduce-trail-convert*:
$reduce\text{-}trail\text{-}to_{NOT}\ C\ S = reduce\text{-}trail\text{-}to\ (convert\text{-}trail\text{-}from\text{-}NOT\ C)\ S$
⟨*proof*⟩

lemma *reduce-trail-to-map*[*simp*]:
$reduce\text{-}trail\text{-}to\ (map\ f\ M)\ S = reduce\text{-}trail\text{-}to\ M\ S$
⟨*proof*⟩

lemma *reduce-trail-to*$_{NOT}$*-map*[*simp*]:
$reduce\text{-}trail\text{-}to_{NOT}\ (map\ f\ M)\ S = reduce\text{-}trail\text{-}to_{NOT}\ M\ S$
⟨*proof*⟩

lemma *skip-or-resolve-state-change*:
assumes $skip\text{-}or\text{-}resolve^{**}\ S\ T$
shows
$\exists M.\ trail\ S = M\ @\ trail\ T \wedge (\forall m \in set\ M.\ \neg is\text{-}marked\ m)$
$clauses\ S = clauses\ T$
$backtrack\text{-}lvl\ S = backtrack\text{-}lvl\ T$
⟨*proof*⟩

22.3 More lemmas conflict–propagate and backjumping

22.4 CDCL FW

lemma *cdcl*$_W$*-merge-is-cdcl*$_{NOT}$*-merged-bj-learn*:
assumes
inv: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$ **and**
$cdcl_W$:$cdcl_W\text{-}merge\ S\ T$
shows $cdcl_{NOT}\text{-}merged\text{-}bj\text{-}learn\ S\ T$
$\vee\ (no\text{-}step\ cdcl_W\text{-}merge\ T \wedge conflicting\ T \neq None)$
⟨*proof*⟩

abbreviation $cdcl_{NOT}$*-restart* **where**
$cdcl_{NOT}\text{-}restart \equiv restart\text{-}ops.cdcl_{NOT}\text{-}raw\text{-}restart\ cdcl_{NOT}\ restart$

lemma *cdcl*$_W$*-merge-restart-is-cdcl*$_{NOT}$*-merged-bj-learn-restart-no-step*:
assumes
inv: $cdcl_W\text{-}all\text{-}struct\text{-}inv\ S$ **and**
$cdcl_W$:$cdcl_W\text{-}merge\text{-}restart\ S\ T$
shows $cdcl_{NOT}\text{-}restart^{**}\ S\ T \vee (no\text{-}step\ cdcl_W\text{-}merge\ T \wedge conflicting\ T \neq None)$
⟨*proof*⟩

abbreviation μ_{FW} :: *'st* $\Rightarrow$ *nat* **where**
μ_{FW} $S \equiv$ (*if no-step* $cdcl_W$*-merge* S *then 0 else* $1+\mu_{CDCL}$*'-merged* (*set-mset* (*init-clss* S)) S)

lemma $cdcl_W$*-merge-*μ_{FW}*-decreasing*:
  assumes
    inv: $cdcl_W$*-all-struct-inv* S **and**
    fw: $cdcl_W$*-merge* S T
  shows μ_{FW} $T < \mu_{FW}$ S
⟨*proof*⟩

lemma *wf-*$cdcl_W$*-merge*: *wf* {(T, S). $cdcl_W$*-all-struct-inv* S $\wedge$ $cdcl_W$*-merge* S T}
  ⟨*proof*⟩

sublocale *conflict-driven-clause-learning*$_W$*-termination*
  ⟨*proof*⟩

lemma *full-*$cdcl_W$*-s'-full-*$cdcl_W$*-merge-restart*:
  assumes
    conflicting R = *None* **and**
    inv: $cdcl_W$*-all-struct-inv* R
  shows *full* $cdcl_W$*-s'* R V $\longleftrightarrow$ *full* $cdcl_W$*-merge-stgy* R V (**is** *?s'* $\longleftrightarrow$ *?fw*)
⟨*proof*⟩

lemma *full-*$cdcl_W$*-stgy-full-*$cdcl_W$*-merge*:
  assumes
    conflicting R = *None* **and**
    inv: $cdcl_W$*-all-struct-inv* R
  shows *full* $cdcl_W$*-stgy* R V $\longleftrightarrow$ *full* $cdcl_W$*-merge-stgy* R V
  ⟨*proof*⟩

lemma *full-*$cdcl_W$*-merge-stgy-final-state-conclusive'*:
  fixes S' :: *'st*
  assumes *full*: *full* $cdcl_W$*-merge-stgy* (*init-state* N) S'
  and *no-d*: *distinct-mset-mset* (*mset-clss* N)
  shows (*conflicting* S' = *Some* {#} $\wedge$ *unsatisfiable* (*set-mset* (*mset-clss* N)))
    $\vee$ (*conflicting* S' = *None* $\wedge$ *trail* S' $\models$*asm mset-clss* N $\wedge$ *satisfiable* (*set-mset* (*mset-clss* N)))
⟨*proof*⟩
end

end
theory *CDCL-W-Incremental*
imports *CDCL-W-Termination*
begin

23 Incremental SAT solving

context *conflict-driven-clause-learning*$_W$
begin

This invariant holds all the invariant related to the strategy. See the structural invariant in $cdcl_W$*-all-struct-inv*

definition $cdcl_W$*-stgy-invariant* **where**
$cdcl_W$*-stgy-invariant* S $\longleftrightarrow$
  conflict-is-false-with-level S
  $\wedge$ *no-clause-is-false* S

$\wedge$ *no-smaller-confl S*
$\wedge$ *no-clause-is-false S*

lemma *$cdcl_W$-stgy-$cdcl_W$-stgy-invariant*:
assumes
$cdcl_W$: *$cdcl_W$-stgy S T* **and**
inv-s: *$cdcl_W$-stgy-invariant S* **and**
inv: *$cdcl_W$-all-struct-inv S*
shows
$cdcl_W$-stgy-invariant T
⟨*proof*⟩

lemma *rtranclp-$cdcl_W$-stgy-$cdcl_W$-stgy-invariant*:
assumes
$cdcl_W$: *$cdcl_W$-stgy** S T* **and**
inv-s: *$cdcl_W$-stgy-invariant S* **and**
inv: *$cdcl_W$-all-struct-inv S*
shows
$cdcl_W$-stgy-invariant T
⟨*proof*⟩

abbreviation *decr-bt-lvl* **where**
decr-bt-lvl S ≡ update-backtrack-lvl (backtrack-lvl S − 1) S

When we add a new clause, we reduce the trail until we get to tho first literal included in C. Then we can mark the conflict.

fun *cut-trail-wrt-clause* **where**
cut-trail-wrt-clause C [] S = S |
cut-trail-wrt-clause C (Marked L - # M) S =
(if −L ∈# C then S
else cut-trail-wrt-clause C M (decr-bt-lvl (tl-trail S))) |
cut-trail-wrt-clause C (Propagated L - # M) S =
(if −L ∈# C then S
else cut-trail-wrt-clause C M (tl-trail S))

definition *add-new-clause-and-update* :: *'ccls ⇒ 'st ⇒ 'st* **where**
add-new-clause-and-update C S =
(if trail S ⊨as CNot (mset-ccls C)
then update-conflicting (Some C) (add-init-cls (cls-of-ccls C)
(cut-trail-wrt-clause (mset-ccls C) (trail S) S))
else add-init-cls (cls-of-ccls C) S)

thm *cut-trail-wrt-clause.induct*
lemma *init-clss-cut-trail-wrt-clause*[*simp*]:
init-clss (cut-trail-wrt-clause C M S) = init-clss S
⟨*proof*⟩

lemma *learned-clss-cut-trail-wrt-clause*[*simp*]:
learned-clss (cut-trail-wrt-clause C M S) = learned-clss S
⟨*proof*⟩

lemma *conflicting-clss-cut-trail-wrt-clause*[*simp*]:
conflicting (cut-trail-wrt-clause C M S) = conflicting S
⟨*proof*⟩

lemma *trail-cut-trail-wrt-clause*:
$\exists M.$ *trail* $S = M$ @ *trail* (*cut-trail-wrt-clause* C (*trail* S) S)
⟨*proof*⟩

lemma *n-dup-no-dup-trail-cut-trail-wrt-clause*[*simp*]:
assumes *n-d*: *no-dup* (*trail* T)
shows *no-dup* (*trail* (*cut-trail-wrt-clause* C (*trail* T) T))
⟨*proof*⟩

lemma *cut-trail-wrt-clause-backtrack-lvl-length-marked*:
assumes
backtrack-lvl T = *length* (*get-all-levels-of-marked* (*trail* T))
shows
backtrack-lvl (*cut-trail-wrt-clause* C (*trail* T) T) =
length (*get-all-levels-of-marked* (*trail* (*cut-trail-wrt-clause* C (*trail* T) T)))
⟨*proof*⟩

lemma *cut-trail-wrt-clause-get-all-levels-of-marked*:
assumes *get-all-levels-of-marked* (*trail* T) = *rev* [*Suc 0*..<
Suc (*length* (*get-all-levels-of-marked* (*trail* T)))]
shows
get-all-levels-of-marked (*trail* ((*cut-trail-wrt-clause* C (*trail* T) T))) = *rev* [*Suc 0*..<
Suc (*length* (*get-all-levels-of-marked* (*trail* ((*cut-trail-wrt-clause* C (*trail* T) T)))))]
⟨*proof*⟩

lemma *cut-trail-wrt-clause-CNot-trail*:
assumes *trail* T ⊨*as CNot* C
shows
(*trail* ((*cut-trail-wrt-clause* C (*trail* T) T))) ⊨*as CNot* C
⟨*proof*⟩

lemma *cut-trail-wrt-clause-hd-trail-in-or-empty-trail*:
(($\forall L$ ∈#C. $-L \notin$ *lits-of-l* (*trail* T)) ∧ *trail* (*cut-trail-wrt-clause* C (*trail* T) T) = [])
∨ (−*lit-of* (*hd* (*trail* (*cut-trail-wrt-clause* C (*trail* T) T))) ∈# C
∧ *length* (*trail* (*cut-trail-wrt-clause* C (*trail* T) T)) ≥ *1*)
⟨*proof*⟩

We can fully run $cdcl_W$*-s* or add a clause. Remark that we use $cdcl_W$*-s* to avoid an explicit *skip*, *resolve*, and *backtrack* normalisation to get rid of the conflict C if possible.

inductive *incremental-*$cdcl_W$ :: *'st* ⇒ *'st* ⇒ *bool* **for** S **where**
add-confl:
trail S ⊨*asm init-clss* S ⟹ *distinct-mset* (*mset-ccls* C) ⟹ *conflicting* S = *None* ⟹
trail S ⊨*as CNot* (*mset-ccls* C) ⟹
full $cdcl_W$*-stgy*
(*update-conflicting* (*Some* C)
(*add-init-cls* (*cls-of-ccls* C) (*cut-trail-wrt-clause* (*mset-ccls* C) (*trail* S) S))) T ⟹
incremental-$cdcl_W$ S T |
add-no-confl:
trail S ⊨*asm init-clss* S ⟹ *distinct-mset* (*mset-ccls* C) ⟹ *conflicting* S = *None* ⟹
¬*trail* S ⊨*as CNot* (*mset-ccls* C) ⟹
full $cdcl_W$*-stgy* (*add-init-cls* (*cls-of-ccls* C) S) T ⟹
incremental-$cdcl_W$ S T

lemma $cdcl_W$*-all-struct-inv-add-new-clause-and-update-*$cdcl_W$*-all-struct-inv*:
assumes

inv-T: $cdcl_W$*-all-struct-inv* T **and**
tr-T-N[*simp*]: *trail* T $\models$*asm* N **and**
tr-C[*simp*]: *trail* T $\models$*as* *CNot* (*mset-ccls* C) **and**
[*simp*]: *distinct-mset* (*mset-ccls* C)
shows $cdcl_W$*-all-struct-inv* (*add-new-clause-and-update* C T) (**is** $cdcl_W$*-all-struct-inv* *?T'*)
⟨*proof*⟩

lemma $cdcl_W$*-all-struct-inv-add-new-clause-and-update-*$cdcl_W$*-stgy-inv*:
assumes
inv-s: $cdcl_W$*-stgy-invariant* T **and**
inv: $cdcl_W$*-all-struct-inv* T **and**
tr-T-N[*simp*]: *trail* T $\models$*asm* N **and**
tr-C[*simp*]: *trail* T $\models$*as* *CNot* (*mset-ccls* C) **and**
[*simp*]: *distinct-mset* (*mset-ccls* C)
shows $cdcl_W$*-stgy-invariant* (*add-new-clause-and-update* C T)
(**is** $cdcl_W$*-stgy-invariant* *?T'*)
⟨*proof*⟩

lemma *full-*$cdcl_W$*-stgy-inv-normal-form*:
assumes
full: *full* $cdcl_W$*-stgy* S T **and**
inv-s: $cdcl_W$*-stgy-invariant* S **and**
inv: $cdcl_W$*-all-struct-inv* S
shows *conflicting* T = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss* S))
∨ *conflicting* T = *None* ∧ *trail* T $\models$*asm* *init-clss* S ∧ *satisfiable* (*set-mset* (*init-clss* S))
⟨*proof*⟩

lemma *incremental-*$cdcl_W$*-inv*:
assumes
inc: *incremental-*$cdcl_W$ S T **and**
inv: $cdcl_W$*-all-struct-inv* S **and**
s-inv: $cdcl_W$*-stgy-invariant* S
shows
$cdcl_W$*-all-struct-inv* T **and**
$cdcl_W$*-stgy-invariant* T
⟨*proof*⟩

lemma *rtranclp-incremental-*$cdcl_W$*-inv*:
assumes
inc: *incremental-*$cdcl_W^{**}$ S T **and**
inv: $cdcl_W$*-all-struct-inv* S **and**
s-inv: $cdcl_W$*-stgy-invariant* S
shows
$cdcl_W$*-all-struct-inv* T **and**
$cdcl_W$*-stgy-invariant* T
⟨*proof*⟩

lemma *incremental-conclusive-state*:
assumes
inc: *incremental-*$cdcl_W$ S T **and**
inv: $cdcl_W$*-all-struct-inv* S **and**
s-inv: $cdcl_W$*-stgy-invariant* S
shows *conflicting* T = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss* T))
∨ *conflicting* T = *None* ∧ *trail* T $\models$*asm* *init-clss* T ∧ *satisfiable* (*set-mset* (*init-clss* T))
⟨*proof*⟩

lemma *tranclp-incremental-correct*:
  assumes
    inc: *incremental-cdcl*$_W$$^{++}$ *S T* **and**
    inv: *cdcl*$_W$*-all-struct-inv S* **and**
    s-inv: *cdcl*$_W$*-stgy-invariant S*
  shows *conflicting T* = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss T*))
    ∨ *conflicting T* = *None* ∧ *trail T* ⊨*asm init-clss T* ∧ *satisfiable* (*set-mset* (*init-clss T*))
  ⟨*proof*⟩

end

end

24 2-Watched-Literal

theory *CDCL-Two-Watched-Literals*
imports *CDCL-WNOT*
begin

We will directly on the two-watched literals datastructure with lists: it could be also seen as a state over some abstract clause representation we would later refine as lists. However, as we need a way to select element from a clause, working on lists is better.

24.1 Datastructure and Access Functions

Only the 2-watched literals have to be verified here: the backtrack level and the trail that appear in the state are not related to the 2-watched algoritm.

datatype *'v twl-clause* =
  TWL-Clause (*watched*: *'v literal list*) (*unwatched*: *'v literal list*)

datatype *'v twl-state* =
  TWL-State (*raw-trail*: (*'v*, *nat*, *'v twl-clause*) *marked-lit list*)
    (*raw-init-clss*: *'v twl-clause list*)
    (*raw-learned-clss*: *'v twl-clause list*) (*backtrack-lvl*: *nat*)
    (*raw-conflicting*: *'v literal list option*)

fun *mmset-of-mlit'* :: (*'v*, *nat*, *'v twl-clause*) *marked-lit* ⇒ (*'v*, *nat*, *'v clause*) *marked-lit*
  where
mmset-of-mlit' (*Propagated L C*) = *Propagated L* (*mset* (*watched C* @ *unwatched C*)) |
mmset-of-mlit' (*Marked L i*) = *Marked L i*

lemma *lit-of-mmset-of-mlit'*[*simp*]: *lit-of* (*mmset-of-mlit' x*) = *lit-of x*
  ⟨*proof*⟩

lemma *lits-of-mmset-of-mlit'*[*simp*]: *lits-of* (*mmset-of-mlit'* ‘ *S*) = *lits-of S*
  ⟨*proof*⟩

abbreviation *trail* **where**
trail S ≡ *map mmset-of-mlit'* (*raw-trail S*)

abbreviation *clauses-of-l* **where**
  clauses-of-l ≡ λ*L*. *mset* (*map mset L*)

definition *raw-clause* :: *'v twl-clause* ⇒ *'v literal list* **where**
raw-clause C ≡ *watched C* @ *unwatched C*

abbreviation *raw-clss* :: *'v twl-state* ⇒ *'v clauses* **where**
raw-clss S ≡ *clauses-of-l* (*map raw-clause* (*raw-init-clss S* @ *raw-learned-clss S*))

interpretation *raw-cls*
λ*C*. *mset* (*raw-clause C*)
λ*L C*. *TWL-Clause* (*watched C*) (*L* # *unwatched C*)
λ*L C*. *TWL-Clause* [] (*remove1 L* (*raw-clause C*))
⟨*proof*⟩

lemma *XXX*:
mset (*map* (λ*x*. *mset* (*unwatched x*) + *mset* (*watched x*))
(*remove1-cond* (λ*D*. *mset* (*raw-clause D*) = *mset* (*raw-clause a*)) *Cs*)) =
remove1-mset (*mset* (*raw-clause a*)) (*mset* (*map* (λ*x*. *mset* (*raw-clause x*)) *Cs*))
⟨*proof*⟩

interpretation *raw-clss*
λ*C*. *mset* (*raw-clause C*)

λ*L C*. *TWL-Clause* (*watched C*) (*L* # *unwatched C*)
λ*L C*. *TWL-Clause* [] (*remove1 L* (*raw-clause C*))
λ*C*. *clauses-of-l* (*map raw-clause C*) *op* @
λ*L C*. *L* ∈ *set C* *op* # λ*C*. *remove1-cond* (λ*D*. *mset* (*raw-clause D*) = *mset* (*raw-clause C*))
⟨*proof*⟩

lemma *ex-mset-unwatched-watched*:
∃ *a*. *mset* (*unwatched a*) + *mset* (*watched a*) = *E*
⟨*proof*⟩

thm *CDCL-Two-Watched-Literals.raw-cls-axioms*
interpretation *twl*: $state_W$*-ops*
λ*C*. *mset* (*raw-clause C*)

λ*L C*. *TWL-Clause* (*watched C*) (*L* # *unwatched C*)
λ*L C*. *TWL-Clause* [] (*remove1 L* (*raw-clause C*))
λ*C*. *clauses-of-l* (*map raw-clause C*) *op* @
λ*L C*. *L* ∈ *set C* *op* # λ*C*. *remove1-cond* (λ*D*. *mset* (*raw-clause D*) = *mset* (*raw-clause C*))

mset λ*xs ys*. *case-prod append* (*fold* (λ*x* (*ys*, *zs*). (*remove1 x ys*, *x* # *zs*)) *xs* (*ys*, []))
op # *remove1*

raw-clause λ*C*. *TWL-Clause* [] *C*
trail λ*S*. *hd* (*raw-trail S*)
raw-init-clss raw-learned-clss backtrack-lvl raw-conflicting

⟨*proof*⟩
declare *CDCL-Two-Watched-Literals.twl.mset-ccls-ccls-of-cls*[*simp del*]

lemma *mmset-of-mlit'-mmset-of-mlit*[*simp*]:
twl.mmset-of-mlit L = *mmset-of-mlit' L*
⟨*proof*⟩

definition

candidates-propagate :: $'v$ *twl-state* $\Rightarrow$ ($'v$ *literal* $\times$ $'v$ *twl-clause*) *set*
where
candidates-propagate S =
{(L, C) | L C.
$C \in$ *set* (*twl.raw-clauses* S) $\wedge$
set (*watched* C) $-$ (*uminus* ‘ *lits-of-l* (*trail* S)) = {L} $\wedge$
undefined-lit (*raw-trail* S) L}

definition *candidates-conflict* :: $'v$ *twl-state* $\Rightarrow$ $'v$ *twl-clause set* **where**
candidates-conflict S =
{C. $C \in$ *set* (*twl.raw-clauses* S) $\wedge$
set (*watched* C) $\subseteq$ *uminus* ‘ *lits-of-l* (*raw-trail* S)}

primrec (*nonexhaustive*) *index* :: $'a$ *list* $\Rightarrow$ $'a$ $\Rightarrow$ *nat* **where**
index (a # l) c = (*if* $a = c$ *then* 0 *else* 1+*index* l c)

lemma *index-nth*:
$a \in$ *set* $l \Longrightarrow l$! (*index* l a) = a
⟨*proof*⟩

24.2 Invariants

We need the following property about updates: if there is a literal L with $-L$ in the trail, and L is not watched, then it stays unwatched; i.e., while updating with *rewatch* it does not get swap with a watched literal L' such that $-L'$ is in the trail.

primrec *watched-decided-most-recently* :: ($'v$, $'lvl$, $'mark$) *marked-lit list* $\Rightarrow$
$'v$ *twl-clause* $\Rightarrow$ *bool*
where
watched-decided-most-recently M (*TWL-Clause* W UW) $\longleftrightarrow$
($\forall L' \in$ *set* W. $\forall L \in$ *set* UW.
$-L' \in$ *lits-of-l* $M \longrightarrow -L \in$ *lits-of-l* $M \longrightarrow L \notin\#$ *mset* $W \longrightarrow$
index (*map lit-of* M) ($-L'$) $\leq$ *index* (*map lit-of* M) ($-L$))

Here are the invariant strictly related to the 2-WL data structure.

primrec *wf-twl-cls* :: ($'v$, $'lvl$, $'mark$) *marked-lit list* $\Rightarrow$ $'v$ *twl-clause* $\Rightarrow$ *bool* **where**
wf-twl-cls M (*TWL-Clause* W UW) $\longleftrightarrow$
distinct W $\wedge$ *length* $W \leq 2$ $\wedge$ (*length* $W < 2 \longrightarrow$ *set* $UW \subseteq$ *set* W) $\wedge$
($\forall L \in$ *set* W. $-L \in$ *lits-of-l* $M \longrightarrow$ ($\forall L' \in$ *set* UW. $L' \notin$ *set* $W \longrightarrow -L' \in$ *lits-of-l* M)) $\wedge$
watched-decided-most-recently M (*TWL-Clause* W UW)

lemma *size-mset-2*: *size* $x1 = 2 \longleftrightarrow (\exists\, a\ b.\ x1 = \{\#a, b\#\})$
⟨*proof*⟩

lemma *distinct-mset-size-2*: *distinct-mset* $\{\#a, b\#\} \longleftrightarrow a \neq b$
⟨*proof*⟩

lemma *wf-twl-cls-annotation-independant*:
assumes M: *map lit-of* M = *map lit-of* M'
shows *wf-twl-cls* M (*TWL-Clause* W UW) $\longleftrightarrow$ *wf-twl-cls* M' (*TWL-Clause* W UW)
⟨*proof*⟩

lemma *wf-twl-cls-wf-twl-cls-tl*:
assumes *wf*: *wf-twl-cls* M C **and** *n-d*: *no-dup* M
shows *wf-twl-cls* (*tl* M) C
⟨*proof*⟩

lemma *wf-twl-cls-append*:
  assumes
    n-d: *no-dup* $(M' @ M)$ **and**
    wf: *wf-twl-cls* $(M' @ M)$ C
  shows *wf-twl-cls* M C
  ⟨*proof*⟩

definition *wf-twl-state* :: *'v twl-state* ⇒ *bool* **where**
  wf-twl-state S ⟷
    (∀ C ∈ *set* (*twl.raw-clauses* S). *wf-twl-cls* (*raw-trail* S) C) ∧ *no-dup* (*raw-trail* S)

lemma *wf-candidates-propagate-sound*:
  assumes *wf*: *wf-twl-state* S **and**
    cand: (L, C) ∈ *candidates-propagate* S
  shows *raw-trail* S ⊨*as* *CNot* (*mset* (*removeAll* L (*raw-clause* C))) ∧ *undefined-lit* (*raw-trail* S) L
    (**is** *?Not* ∧ *?undef*)
⟨*proof*⟩

lemma *wf-candidates-propagate-complete*:
  assumes *wf*: *wf-twl-state* S **and**
    c-mem: C ∈ *set* (*twl.raw-clauses* S) **and**
    l-mem: L ∈ *set* (*raw-clause* C) **and**
    unsat: *trail* S ⊨*as* *CNot* (*mset-set* (*set* (*raw-clause* C) − $\{L\}$)) **and**
    undef: *undefined-lit* (*raw-trail* S) L
  shows (L, C) ∈ *candidates-propagate* S
⟨*proof*⟩

lemma *wf-candidates-conflict-sound*:
  assumes *wf*: *wf-twl-state* S **and**
    cand: C ∈ *candidates-conflict* S
  shows *trail* S ⊨*as* *CNot* (*mset* (*raw-clause* C)) ∧ C ∈ *set* (*twl.raw-clauses* S)
⟨*proof*⟩

lemma *wf-candidates-conflict-complete*:
  assumes *wf*: *wf-twl-state* S **and**
    c-mem: C ∈ *set* (*twl.raw-clauses* S) **and**
    unsat: *trail* S ⊨*as* *CNot* (*mset* (*raw-clause* C))
  shows C ∈ *candidates-conflict* S
⟨*proof*⟩

typedef *'v wf-twl* = {*S*::*'v twl-state*. *wf-twl-state* S}
morphisms *rough-state-of-twl* *twl-of-rough-state*
⟨*proof*⟩

lemma [*code abstype*]:
  twl-of-rough-state (*rough-state-of-twl* S) = S
  ⟨*proof*⟩

lemma *wf-twl-state-rough-state-of-twl*[*simp*]: *wf-twl-state* (*rough-state-of-twl* S)
  ⟨*proof*⟩

abbreviation *candidates-conflict-twl* :: *'v wf-twl* ⇒ *'v twl-clause set* **where**
candidates-conflict-twl S ≡ *candidates-conflict* (*rough-state-of-twl* S)

abbreviation *candidates-propagate-twl* :: *'v wf-twl* ⇒ (*'v literal* × *'v twl-clause*) *set* **where**
candidates-propagate-twl S ≡ *candidates-propagate* (*rough-state-of-twl S*)

abbreviation *raw-trail-twl* :: *'a wf-twl* ⇒ (*'a*, *nat*, *'a twl-clause*) *marked-lit list* **where**
raw-trail-twl S ≡ *raw-trail* (*rough-state-of-twl S*)

abbreviation *trail-twl* :: *'a wf-twl* ⇒ (*'a*, *nat*, *'a literal multiset*) *marked-lit list* **where**
trail-twl S ≡ *trail* (*rough-state-of-twl S*)

abbreviation *raw-clauses-twl* :: *'a wf-twl* ⇒ *'a twl-clause list* **where**
raw-clauses-twl S ≡ *twl.raw-clauses* (*rough-state-of-twl S*)

abbreviation *raw-init-clss-twl* :: *'a wf-twl* ⇒ *'a twl-clause list* **where**
raw-init-clss-twl S ≡ *raw-init-clss* (*rough-state-of-twl S*)

abbreviation *raw-learned-clss-twl* :: *'a wf-twl* ⇒ *'a twl-clause list* **where**
raw-learned-clss-twl S ≡ *raw-learned-clss* (*rough-state-of-twl S*)

abbreviation *backtrack-lvl-twl* **where**
backtrack-lvl-twl S ≡ *backtrack-lvl* (*rough-state-of-twl S*)

abbreviation *raw-conflicting-twl* **where**
raw-conflicting-twl S ≡ *raw-conflicting* (*rough-state-of-twl S*)

lemma *wf-candidates-twl-conflict-complete*:
  assumes
    c-mem: *C* ∈ *set* (*raw-clauses-twl S*) **and**
    unsat: *trail-twl S* ⊨*as CNot* (*mset* (*raw-clause C*))
  shows *C* ∈ *candidates-conflict-twl S*
  ⟨*proof*⟩

abbreviation *update-backtrack-lvl* **where**
  update-backtrack-lvl k S ≡
   TWL-State (*raw-trail S*) (*raw-init-clss S*) (*raw-learned-clss S*) *k* (*raw-conflicting S*)

abbreviation *update-conflicting* **where**
  update-conflicting C S ≡
    TWL-State (*raw-trail S*) (*raw-init-clss S*) (*raw-learned-clss S*) (*backtrack-lvl S*) *C*

24.3 Abstract 2-WL

definition *tl-trail* **where**
  tl-trail S =
   TWL-State (*tl* (*raw-trail S*)) (*raw-init-clss S*) (*raw-learned-clss S*) (*backtrack-lvl S*)
   (*raw-conflicting S*)

locale *abstract-twl* =
  fixes
    watch :: *'v twl-state* ⇒ *'v literal list* ⇒ *'v twl-clause* **and**
    rewatch :: *'v literal* ⇒ *'v twl-state* ⇒
      'v twl-clause ⇒ *'v twl-clause* **and**
    restart-learned :: *'v twl-state* ⇒ *'v twl-clause list*
  assumes
    clause-watch: *no-dup* (*raw-trail S*) ⟹ *mset* (*raw-clause* (*watch S C*)) = *mset C* **and**
    wf-watch: *no-dup* (*raw-trail S*) ⟹ *wf-twl-cls* (*raw-trail S*) (*watch S C*) **and**
    clause-rewatch: *mset* (*raw-clause* (*rewatch L' S C'*)) = *mset* (*raw-clause C'*) **and**

wf-rewatch:
$no\text{-}dup\ (raw\text{-}trail\ S) \Longrightarrow undefined\text{-}lit\ (raw\text{-}trail\ S)\ (lit\text{-}of\ L) \Longrightarrow$
$wf\text{-}twl\text{-}cls\ (raw\text{-}trail\ S)\ C' \Longrightarrow$
$wf\text{-}twl\text{-}cls\ (L\ \#\ raw\text{-}trail\ S)\ (rewatch\ (lit\text{-}of\ L)\ S\ C')$
and
restart-learned: $mset\ (restart\text{-}learned\ S) \subseteq\#\ mset\ (raw\text{-}learned\text{-}clss\ S)$ — We need *mset* and not *set* to take care of duplicates.
begin

definition
$cons\text{-}trail :: ('v,\ nat,\ 'v\ twl\text{-}clause)\ marked\text{-}lit \Rightarrow 'v\ twl\text{-}state \Rightarrow 'v\ twl\text{-}state$
where
$cons\text{-}trail\ L\ S =$
$TWL\text{-}State\ (L\ \#\ raw\text{-}trail\ S)\ (map\ (rewatch\ (lit\text{-}of\ L)\ S)\ (raw\text{-}init\text{-}clss\ S))$
$(map\ (rewatch\ (lit\text{-}of\ L)\ S)\ (raw\text{-}learned\text{-}clss\ S))\ (backtrack\text{-}lvl\ S)\ (raw\text{-}conflicting\ S)$

definition
$add\text{-}init\text{-}cls :: 'v\ literal\ list \Rightarrow 'v\ twl\text{-}state \Rightarrow 'v\ twl\text{-}state$
where
$add\text{-}init\text{-}cls\ C\ S =$
$TWL\text{-}State\ (raw\text{-}trail\ S)\ (watch\ S\ C\ \#\ raw\text{-}init\text{-}clss\ S)\ (raw\text{-}learned\text{-}clss\ S)\ (backtrack\text{-}lvl\ S)$
$(raw\text{-}conflicting\ S)$

definition
$add\text{-}learned\text{-}cls :: 'v\ literal\ list \Rightarrow 'v\ twl\text{-}state \Rightarrow 'v\ twl\text{-}state$
where
$add\text{-}learned\text{-}cls\ C\ S =$
$TWL\text{-}State\ (raw\text{-}trail\ S)\ (raw\text{-}init\text{-}clss\ S)\ (watch\ S\ C\ \#\ raw\text{-}learned\text{-}clss\ S)\ (backtrack\text{-}lvl\ S)$
$(raw\text{-}conflicting\ S)$

definition
$remove\text{-}cls :: 'v\ literal\ list \Rightarrow 'v\ twl\text{-}state \Rightarrow 'v\ twl\text{-}state$
where
$remove\text{-}cls\ C\ S =$
$TWL\text{-}State\ (raw\text{-}trail\ S)$
$(removeAll\text{-}cond\ (\lambda D.\ mset\ (raw\text{-}clause\ D) = mset\ C)\ (raw\text{-}init\text{-}clss\ S))$
$(removeAll\text{-}cond\ (\lambda D.\ mset\ (raw\text{-}clause\ D) = mset\ C)\ (raw\text{-}learned\text{-}clss\ S))$
$(backtrack\text{-}lvl\ S)$
$(raw\text{-}conflicting\ S)$

definition $init\text{-}state :: 'v\ literal\ list\ list \Rightarrow 'v\ twl\text{-}state$ **where**
$init\text{-}state\ N = fold\ add\text{-}init\text{-}cls\ N\ (TWL\text{-}State\ []\ []\ []\ 0\ None)$

lemma *unchanged-fold-add-init-cls*:
$raw\text{-}trail\ (fold\ add\text{-}init\text{-}cls\ Cs\ (TWL\text{-}State\ M\ N\ U\ k\ C)) = M$
$raw\text{-}learned\text{-}clss\ (fold\ add\text{-}init\text{-}cls\ Cs\ (TWL\text{-}State\ M\ N\ U\ k\ C)) = U$
$backtrack\text{-}lvl\ (fold\ add\text{-}init\text{-}cls\ Cs\ (TWL\text{-}State\ M\ N\ U\ k\ C)) = k$
$raw\text{-}conflicting\ (fold\ add\text{-}init\text{-}cls\ Cs\ (TWL\text{-}State\ M\ N\ U\ k\ C)) = C$
$\langle proof \rangle$

lemma *unchanged-init-state*[*simp*]:
$raw\text{-}trail\ (init\text{-}state\ N) = []$
$raw\text{-}learned\text{-}clss\ (init\text{-}state\ N) = []$
$backtrack\text{-}lvl\ (init\text{-}state\ N) = 0$
$raw\text{-}conflicting\ (init\text{-}state\ N) = None$

⟨proof⟩

lemma *clauses-init-fold-add-init*:
no-dup M $\Longrightarrow$
twl.init-clss (*fold add-init-cls Cs* (*TWL-State M N U k C*)) =
clauses-of-l Cs + *clauses-of-l* (*map raw-clause N*)
⟨proof⟩

lemma *init-clss-init-state*[*simp*]: *twl.init-clss* (*init-state N*) = *clauses-of-l N*
⟨proof⟩

definition *restart'* **where**
restart' S = *TWL-State* [] (*raw-init-clss S*) (*restart-learned S*) *0 None*

end

24.4 Instanciation of the previous locale

definition *watch-nat* :: *'v twl-state* ⇒ *'v literal list* ⇒ *'v twl-clause* **where**
watch-nat S C =
(*let*
C' = *remdups C*;
neg-not-assigned = *filter* (λ*L*. −*L* ∉ *lits-of-l* (*raw-trail S*)) *C'*;
neg-assigned-sorted-by-trail = *filter* (λ*L*. *L* ∈ *set C*) (*map* (λ*L*. −*lit-of L*) (*raw-trail S*));
W = *take 2* (*neg-not-assigned* @ *neg-assigned-sorted-by-trail*);
UW = *foldr remove1 W C*
in TWL-Clause W UW)

lemma *list-cases2*:
fixes *l* :: *'a list*
assumes
l = [] $\Longrightarrow$ *P* **and**
$\bigwedge$*x*. *l* = [*x*] $\Longrightarrow$ *P* **and**
$\bigwedge$*x y xs*. *l* = *x* # *y* # *xs* $\Longrightarrow$ *P*
shows *P*
⟨proof⟩

lemma *filter-in-list-prop-verifiedD*:
assumes [*L*←*P* . *Q L*] = *l*
shows ∀ *x* ∈ *set l*. *x* ∈ *set P* ∧ *Q x*
⟨proof⟩

lemma *no-dup-filter-diff*:
assumes *n-d*: *no-dup M* **and** *H*: [*L*←*map* (λ*L*. − *lit-of L*) *M*. *L* ∈ *set C*] = *l*
shows *distinct l*
⟨proof⟩

lemma *watch-nat-lists-disjointD*:
assumes
l: [*L*←*remdups C*. − *L* ∉ *lits-of-l* (*raw-trail S*)] = *l* **and**
l': [*L*←*map* (λ*L*. − *lit-of L*) (*raw-trail S*) . *L* ∈ *set C*] = *l'*
shows ∀ *x* ∈ *set l*. ∀ *y* ∈ *set l'*. *x* ≠ *y*
⟨proof⟩

lemma *watch-nat-list-cases-witness*[*consumes 2*, *case-names nil-nil nil-single nil-other single-nil single-other other*]:

fixes
  C :: $'v$ *literal list* **and**
  S :: $'v$ *twl-state*
defines
  $xs \equiv [L{\leftarrow}remdups\ C.\ -\ L \notin lits\text{-}of\text{-}l\ (raw\text{-}trail\ S)]$ **and**
  $ys \equiv [L{\leftarrow}map\ (\lambda L.\ -\ lit\text{-}of\ L)\ (raw\text{-}trail\ S)\ .\ L \in set\ C]$
assumes
  n-d: $no\text{-}dup\ (raw\text{-}trail\ S)$ **and**
  nil-nil: $xs = [] \Longrightarrow ys = [] \Longrightarrow P$ **and**
  nil-single:
    $\bigwedge a.\ xs = [] \Longrightarrow ys = [a] \Longrightarrow a \in set\ C \Longrightarrow P$ **and**
  nil-other: $\bigwedge a\ b\ ys'.\ xs = [] \Longrightarrow ys = a\ \#\ b\ \#\ ys' \Longrightarrow a \neq b \Longrightarrow P$ **and**
  single-nil: $\bigwedge a.\ xs = [a] \Longrightarrow ys = [] \Longrightarrow P$ **and**
  single-other: $\bigwedge a\ b\ ys'.\ xs = [a] \Longrightarrow ys = b\ \#\ ys' \Longrightarrow a \neq b \Longrightarrow P$ **and**
  other: $\bigwedge a\ b\ xs'.\ xs = a\ \#\ b\ \#\ xs' \Longrightarrow a \neq b \Longrightarrow P$
shows P
⟨*proof*⟩

lemma *watch-nat-list-cases* [*consumes 1*, *case-names nil-nil nil-single nil-other single-nil single-other other*]:
fixes
  C :: $'v$ *literal list* **and**
  S :: $'v$ *twl-state*
defines
  $xs \equiv [L{\leftarrow}remdups\ C\ .\ -\ L \notin lits\text{-}of\text{-}l\ (raw\text{-}trail\ S)]$ **and**
  $ys \equiv [L{\leftarrow}map\ (\lambda L.\ -\ lit\text{-}of\ L)\ (raw\text{-}trail\ S)\ .\ L \in set\ C]$
assumes
  n-d: $no\text{-}dup\ (raw\text{-}trail\ S)$ **and**
  nil-nil: $xs = [] \Longrightarrow ys = [] \Longrightarrow P$ **and**
  nil-single:
    $\bigwedge a.\ xs = [] \Longrightarrow ys = [a] \Longrightarrow a \in set\ C \Longrightarrow P$ **and**
  nil-other: $\bigwedge a\ b\ ys'.\ xs = [] \Longrightarrow ys = a\ \#\ b\ \#\ ys' \Longrightarrow a \neq b \Longrightarrow P$ **and**
  single-nil: $\bigwedge a.\ xs = [a] \Longrightarrow ys = [] \Longrightarrow P$ **and**
  single-other: $\bigwedge a\ b\ ys'.\ xs = [a] \Longrightarrow ys = b\ \#\ ys' \Longrightarrow a \neq b \Longrightarrow P$ **and**
  other: $\bigwedge a\ b\ xs'.\ xs = a\ \#\ b\ \#\ xs' \Longrightarrow a \neq b \Longrightarrow P$
shows P
⟨*proof*⟩

lemma *watch-nat-lists-set-union-witness*:
fixes
  C :: $'v$ *literal list* **and**
  S :: $'v$ *twl-state*
defines
  $xs \equiv [L{\leftarrow}remdups\ C.\ -\ L \notin lits\text{-}of\text{-}l\ (raw\text{-}trail\ S)]$ **and**
  $ys \equiv [L{\leftarrow}map\ (\lambda L.\ -\ lit\text{-}of\ L)\ (raw\text{-}trail\ S)\ .\ L \in set\ C]$
assumes *n-d*: $no\text{-}dup\ (raw\text{-}trail\ S)$
shows $set\ C = set\ xs \cup set\ ys$
⟨*proof*⟩

lemma *mset-intersection-inclusion*: $A + (B - A) = B \longleftrightarrow A \subseteq\#\ B$
⟨*proof*⟩

lemma *clause-watch-nat*:
assumes $no\text{-}dup\ (raw\text{-}trail\ S)$
shows $mset\ (raw\text{-}clause\ (watch\text{-}nat\ S\ C)) = mset\ C$

$\langle proof \rangle$

lemma *index-uminus-index-map-uminus*:
$-a \in set\ L \Longrightarrow index\ L\ (-a) = index\ (map\ uminus\ L)\ (a{::}'a\ literal)$
$\langle proof \rangle$

lemma *index-filter*:
$a \in set\ L \Longrightarrow b \in set\ L \Longrightarrow P\ a \Longrightarrow P\ b \Longrightarrow$
$index\ L\ a \leq index\ L\ b \longleftrightarrow index\ (filter\ P\ L)\ a \leq index\ (filter\ P\ L)\ b$
$\langle proof \rangle$

lemma *foldr-remove1-W-Nil*[*simp*]: $foldr\ remove1\ W\ [] = []$
$\langle proof \rangle$

lemma *image-lit-of-mmset-of-mlit'*[*simp*]:
$lit\text{-}of\ `\ mmset\text{-}of\text{-}mlit'\ `\ A = lit\text{-}of\ `\ A$
$\langle proof \rangle$

lemma *distinct-filter-eq*:
assumes *distinct xs*
shows $[L \leftarrow xs.\ L = a] = (if\ a \in set\ xs\ then\ [a]\ else\ [])$
$\langle proof \rangle$

lemma *no-dup-distinct-map-uminus-lit-of*:
$no\text{-}dup\ xs \Longrightarrow distinct\ (map\ (\lambda L.\ -\ lit\text{-}of\ L)\ xs)$
$\langle proof \rangle$

lemma *wf-watch-witness*:
fixes C :: *'v literal list* **and**
S :: *'v twl-state*
defines
ass: $neg\text{-}not\text{-}assigned \equiv filter\ (\lambda L.\ -L \notin lits\text{-}of\text{-}l\ (raw\text{-}trail\ S))\ (remdups\ C)$ **and**
tr: $neg\text{-}assigned\text{-}sorted\text{-}by\text{-}trail \equiv filter\ (\lambda L.\ L \in set\ C)\ (map\ (\lambda L.\ -lit\text{-}of\ L)\ (raw\text{-}trail\ S))$
defines
W: $W \equiv take\ 2\ (neg\text{-}not\text{-}assigned\ @\ neg\text{-}assigned\text{-}sorted\text{-}by\text{-}trail)$
assumes
n-d[*simp*]: $no\text{-}dup\ (raw\text{-}trail\ S)$
shows $wf\text{-}twl\text{-}cls\ (raw\text{-}trail\ S)\ (TWL\text{-}Clause\ W\ (foldr\ remove1\ W\ C))$
$\langle proof \rangle$

lemma *wf-watch-nat*: $no\text{-}dup\ (raw\text{-}trail\ S) \Longrightarrow wf\text{-}twl\text{-}cls\ (raw\text{-}trail\ S)\ (watch\text{-}nat\ S\ C)$
$\langle proof \rangle$

definition
rewatch-nat ::
$'v\ literal \Rightarrow 'v\ twl\text{-}state \Rightarrow 'v\ twl\text{-}clause \Rightarrow 'v\ twl\text{-}clause$
where
$rewatch\text{-}nat\ L\ S\ C =$
$(if\ -\ L \in set\ (watched\ C)\ then$
$case\ filter\ (\lambda L'.\ L' \notin set\ (watched\ C) \land -\ L' \notin insert\ L\ (lits\text{-}of\text{-}l\ (trail\ S)))$
$(unwatched\ C)\ of$
$[] \Rightarrow C$
$|\ L'\ \#\ \text{-} \Rightarrow$
$TWL\text{-}Clause\ (L'\ \#\ remove1\ (-L)\ (watched\ C))\ (-L\ \#\ remove1\ L'\ (unwatched\ C))$
else

C)

lemma *clause-rewatch-nat*:
  fixes UW :: $'v$ *literal list* **and**
    S :: $'v$ *twl-state* **and**
    L :: $'v$ *literal* **and** C :: $'v$ *twl-clause*
  shows *mset* (*raw-clause* (*rewatch-nat* L S C)) = *mset* (*raw-clause* C)
  ⟨*proof*⟩

lemma *filter-sorted-list-of-multiset-Nil*:
  $[x \leftarrow$ *sorted-list-of-multiset* $M.\ p\ x] = [] \longleftrightarrow (\forall x \in\# M.\ \neg\ p\ x)$
  ⟨*proof*⟩

lemma *filter-sorted-list-of-multiset-ConsD*:
  $[x \leftarrow$ *sorted-list-of-multiset* $M.\ p\ x] = x\ \#\ xs \Longrightarrow p\ x$
  ⟨*proof*⟩

lemma *mset-minus-single-eq-mempty*:
  $a - \{\#b\#\} = \{\#\} \longleftrightarrow a = \{\#b\#\} \lor a = \{\#\}$
  ⟨*proof*⟩

lemma *size-mset-le-2-cases*:
  assumes *size* $W \leq 2$
  shows $W = \{\#\} \lor (\exists a.\ W = \{\#a\#\}) \lor (\exists a\ b.\ W = \{\#a,b\#\})$
  ⟨*proof*⟩

lemma *filter-sorted-list-of-multiset-eqD*:
  assumes $[x \leftarrow$ *sorted-list-of-multiset* $A.\ p\ x] = x\ \#\ xs$ (**is** *?comp* = -)
  shows $x \in\# A$
⟨*proof*⟩

lemma *clause-rewatch-witness'*:
  assumes
    wf: *wf-twl-cls* (*raw-trail* S) C **and**
    undef: *undefined-lit* (*raw-trail* S) (*lit-of* L)
  shows *wf-twl-cls* (L # *raw-trail* S) (*rewatch-nat* (*lit-of* L) S C)
⟨*proof*⟩

interpretation *twl*: *abstract-twl watch-nat rewatch-nat raw-learned-clss*
  ⟨*proof*⟩

interpretation *twl2*: *abstract-twl watch-nat rewatch-nat* λ-. []
  ⟨*proof*⟩

end

25 Invariants for 2 Watched-Literals

theory *CDCL-Two-Watched-Literals-Invariant*
imports *CDCL-Two-Watched-Literals DPLL-CDCL-W-Implementation*
begin

25.1 Interpretation for *conflict-driven-clause-learning*$_W$*.cdcl*$_W$

We define here the 2-WL with the invariant and show the role of the candidates.

context *abstract-twl*
begin

25.1.1 Direct Interpretation

lemma *mset-map-removeAll-cond*:
  mset (*map* (λx. *mset* (*raw-clause x*))
    (*removeAll-cond* (λD. *mset* (*raw-clause D*) = *mset* (*raw-clause C*)) *N*))
  = *mset* (*removeAll* (*mset* (*raw-clause C*)) (*map* (λx. *mset* (*raw-clause x*)) *N*))
  ⟨*proof*⟩

lemma *mset-raw-init-clss-init-state*:
  mset (*map* (λx. *mset* (*raw-clause x*)) (*raw-init-clss* (*init-state* (*map raw-clause N*))))
  = *mset* (*map* (λx. *mset* (*raw-clause x*)) *N*)
  ⟨*proof*⟩

interpretation *rough-cdcl*: *state*$_W$
  λC. *mset* (*raw-clause C*)

  λL C. *TWL-Clause* (*watched C*) (*L* # *unwatched C*)
  λL C. *TWL-Clause* [] (*remove1 L* (*raw-clause C*))
  λC. *clauses-of-l* (*map raw-clause C*) *op* @
  λL C. $L \in$ *set C op* # λC. *remove1-cond* (λD. *mset* (*raw-clause D*) = *mset* (*raw-clause C*))

  mset λ*xs ys*. *case-prod append* (*fold* (λx (*ys*, *zs*). (*remove1 x ys*, *x* # *zs*)) *xs* (*ys*, []))
  op # *remove1*

  raw-clause λC. *TWL-Clause* [] *C*
  trail λS. *hd* (*raw-trail S*)
  raw-init-clss raw-learned-clss backtrack-lvl raw-conflicting
  cons-trail tl-trail λC. *add-init-cls* (*raw-clause C*) λC. *add-learned-cls* (*raw-clause C*)
  λC. *remove-cls* (*raw-clause C*)
  update-backtrack-lvl
  update-conflicting λN. *init-state* (*map raw-clause N*) *restart'*
  ⟨*proof*⟩

interpretation *rough-cdcl*: *conflict-driven-clause-learning*$_W$
  λC. *mset* (*raw-clause C*)

  λL C. *TWL-Clause* (*watched C*) (*L* # *unwatched C*)
  λL C. *TWL-Clause* [] (*remove1 L* (*raw-clause C*))
  λC. *clauses-of-l* (*map raw-clause C*) *op* @
  λL C. $L \in$ *set C op* # λC. *remove1-cond* (λD. *mset* (*raw-clause D*) = *mset* (*raw-clause C*))

  mset λ*xs ys*. *case-prod append* (*fold* (λx (*ys*, *zs*). (*remove1 x ys*, *x* # *zs*)) *xs* (*ys*, []))
  op # *remove1*

  λC. *raw-clause C* λC. *TWL-Clause* [] *C*
  trail λS. *hd* (*raw-trail S*)
  raw-init-clss raw-learned-clss backtrack-lvl raw-conflicting
  cons-trail tl-trail λC. *add-init-cls* (*raw-clause C*) λC. *add-learned-cls* (*raw-clause C*)
  λC. *remove-cls* (*raw-clause C*)

update-backtrack-lvl
update-conflicting λN. *init-state* (*map raw-clause* N) *restart'*
⟨*proof*⟩

declare *local.rough-cdcl.mset-ccls-ccls-of-cls*[*simp del*]

25.1.2 Opaque Type with Invariant

declare *rough-cdcl.state-simp*[*simp del*]

definition *cons-trail-twl* :: (*'v*, *nat*, *'v twl-clause*) *marked-lit* ⇒ *'v wf-twl* ⇒ *'v wf-twl*
where
cons-trail-twl L S ≡ *twl-of-rough-state* (*cons-trail* L (*rough-state-of-twl* S))

lemma *wf-twl-state-cons-trail*:
assumes
undef: *undefined-lit* (*raw-trail* S) (*lit-of* L) **and**
wf: *wf-twl-state* S
shows *wf-twl-state* (*cons-trail* L S)
⟨*proof*⟩

lemma *rough-state-of-twl-cons-trail*:
undefined-lit (*raw-trail-twl* S) (*lit-of* L) ⟹
rough-state-of-twl (*cons-trail-twl* L S) = *cons-trail* L (*rough-state-of-twl* S)
⟨*proof*⟩

abbreviation *add-init-cls-twl* **where**
add-init-cls-twl C S ≡ *twl-of-rough-state* (*add-init-cls* C (*rough-state-of-twl* S))

lemma *wf-twl-add-init-cls*: *wf-twl-state* S ⟹ *wf-twl-state* (*add-init-cls* L S)
⟨*proof*⟩

lemma *rough-state-of-twl-add-init-cls*:
rough-state-of-twl (*add-init-cls-twl* L S) = *add-init-cls* L (*rough-state-of-twl* S)
⟨*proof*⟩

abbreviation *add-learned-cls-twl* **where**
add-learned-cls-twl C S ≡ *twl-of-rough-state* (*add-learned-cls* C (*rough-state-of-twl* S))

lemma *wf-twl-add-learned-cls*: *wf-twl-state* S ⟹ *wf-twl-state* (*add-learned-cls* L S)
⟨*proof*⟩

lemma *rough-state-of-twl-add-learned-cls*:
rough-state-of-twl (*add-learned-cls-twl* L S) = *add-learned-cls* L (*rough-state-of-twl* S)
⟨*proof*⟩

abbreviation *remove-cls-twl* **where**
remove-cls-twl C S ≡ *twl-of-rough-state* (*remove-cls* C (*rough-state-of-twl* S))

lemma *set-removeAll-condD*: $x \in$ *set* (*removeAll-cond f xs*) ⟹ $x \in$ *set xs*
⟨*proof*⟩

lemma *wf-twl-remove-cls*: *wf-twl-state* S ⟹ *wf-twl-state* (*remove-cls* L S)
⟨*proof*⟩

lemma *rough-state-of-twl-remove-cls*:

rough-state-of-twl (*remove-cls-twl* L S) = *remove-cls* L (*rough-state-of-twl* S)
⟨*proof*⟩

abbreviation *init-state-twl* **where**
init-state-twl N ≡ *twl-of-rough-state* (*init-state* N)

lemma *wf-twl-state-wf-twl-state-fold-add-init-cls*:
assumes *wf-twl-state* S
shows *wf-twl-state* (*fold add-init-cls* N S)
⟨*proof*⟩

lemma *wf-twl-state-epsilon-state*[*simp*]:
wf-twl-state (*TWL-State* [] [] [] *0 None*)
⟨*proof*⟩

lemma *wf-twl-init-state*: *wf-twl-state* (*init-state* N)
⟨*proof*⟩

lemma *rough-state-of-twl-init-state*:
rough-state-of-twl (*init-state-twl* N) = *init-state* N
⟨*proof*⟩

abbreviation *tl-trail-twl* **where**
tl-trail-twl S ≡ *twl-of-rough-state* (*tl-trail* (*rough-state-of-twl* S))

lemma *wf-twl-state-tl-trail*: *wf-twl-state* S ⟹ *wf-twl-state* (*tl-trail* S)
⟨*proof*⟩

lemma *rough-state-of-twl-tl-trail*:
rough-state-of-twl (*tl-trail-twl* S) = *tl-trail* (*rough-state-of-twl* S)
⟨*proof*⟩

abbreviation *update-backtrack-lvl-twl* **where**
update-backtrack-lvl-twl k S ≡ *twl-of-rough-state* (*update-backtrack-lvl* k (*rough-state-of-twl* S))

lemma *wf-twl-state-update-backtrack-lvl*:
wf-twl-state S ⟹ *wf-twl-state* (*update-backtrack-lvl* k S)
⟨*proof*⟩

lemma *rough-state-of-twl-update-backtrack-lvl*:
rough-state-of-twl (*update-backtrack-lvl-twl* k S) = *update-backtrack-lvl* k
(*rough-state-of-twl* S)
⟨*proof*⟩

abbreviation *update-conflicting-twl* **where**
update-conflicting-twl k S ≡ *twl-of-rough-state* (*update-conflicting* k (*rough-state-of-twl* S))

lemma *wf-twl-state-update-conflicting*:
wf-twl-state S ⟹ *wf-twl-state* (*update-conflicting* k S)
⟨*proof*⟩

lemma *rough-state-of-twl-update-conflicting*:
rough-state-of-twl (*update-conflicting-twl* k S) = *update-conflicting* k
(*rough-state-of-twl* S)
⟨*proof*⟩

abbreviation *raw-clauses-twl* **where**
raw-clauses-twl S ≡ *twl.raw-clauses* (*rough-state-of-twl S*)

abbreviation *restart-twl* **where**
restart-twl S ≡ *twl-of-rough-state* (*restart′* (*rough-state-of-twl S*))
lemma *mset-union-mset-setD*:
mset A ⊆# *mset B* ⟹ *set A* ⊆ *set B*
⟨*proof*⟩

lemma *wf-wf-restart′*: *wf-twl-state S* ⟹ *wf-twl-state* (*restart′ S*)
⟨*proof*⟩

lemma *rough-state-of-twl-restart-twl*:
rough-state-of-twl (*restart-twl S*) = *restart′* (*rough-state-of-twl S*)
⟨*proof*⟩

lemma *undefined-lit-trail-twl-raw-trail*[*iff*]:
undefined-lit (*trail-twl S*) *L* ⟷ *undefined-lit* (*raw-trail-twl S*) *L*
⟨*proof*⟩

sublocale *conflict-driven-clause-learning$_W$*
λ*C*. *mset* (*raw-clause C*)

λ*L C*. *TWL-Clause* (*watched C*) (*L* # *unwatched C*)
λ*L C*. *TWL-Clause* [] (*remove1 L* (*raw-clause C*))
λ*C*. *clauses-of-l* (*map raw-clause C*) *op* @
λ*L C*. *L* ∈ *set C op* # λ*C*. *remove1-cond* (λ*D*. *mset* (*raw-clause D*) = *mset* (*raw-clause C*))

mset λ*xs ys*. *case-prod append* (*fold* (λ*x* (*ys*, *zs*). (*remove1 x ys*, *x* # *zs*)) *xs* (*ys*, []))
op # *remove1*

λ*C*. *raw-clause C* λ*C*. *TWL-Clause* [] *C*
trail-twl λ*S*. *hd* (*raw-trail-twl S*)
raw-init-clss-twl
raw-learned-clss-twl
backtrack-lvl-twl
raw-conflicting-twl
cons-trail-twl
tl-trail-twl
λ*C*. *add-init-cls-twl* (*raw-clause C*)
λ*C*. *add-learned-cls-twl* (*raw-clause C*)
λ*C*. *remove-cls-twl* (*raw-clause C*)
update-backtrack-lvl-twl
update-conflicting-twl
λ*N*. *init-state-twl* (*map raw-clause N*)
restart-twl
⟨*proof*⟩

declare *local.rough-cdcl.mset-ccls-ccls-of-cls*[*simp del*]
abbreviation *state-eq-twl* (**infix** ∼*TWL 51*) **where**
state-eq-twl S S′ ≡ *rough-cdcl.state-eq* (*rough-state-of-twl S*) (*rough-state-of-twl S′*)
notation *state-eq* (**infix** ∼ *51*)
declare *state-simp*[*simp del*]

To avoid ambiguities:

no-notation *state-eq-twl* (**infix** $\sim$ *51*)

inductive *propagate-twl* :: *'v wf-twl* $\Rightarrow$ *'v wf-twl* $\Rightarrow$ *bool* **where**
propagate-twl-rule: $(L, C) \in$ *candidates-propagate-twl S* $\Longrightarrow$
 $S' \sim$ *cons-trail-twl* (*Propagated L C*) *S* $\Longrightarrow$
 raw-conflicting-twl S = *None* $\Longrightarrow$
 propagate-twl S S'

inductive-cases *propagate-twlE*: *propagate-twl S T*
thm *propagateE*

lemma *distinct-filter-eq-if*:
 distinct C $\Longrightarrow$ *length* (*filter* (*op* = *L*) *C*) = (*if L* $\in$ *set C then 1 else 0*)
 ⟨*proof*⟩

lemma *distinct-mset-remove1-All*:
 distinct-mset C $\Longrightarrow$ *remove1-mset L C* = *removeAll-mset L C*
 ⟨*proof*⟩

lemma *propagate-twl-iff-propagate*:
 assumes *inv*: $cdcl_W$*-all-struct-inv S*
 shows *propagate S T* $\longleftrightarrow$ *propagate-twl S T* (**is** *?P* $\longleftrightarrow$ *?T*)
⟨*proof*⟩

no-notation *twl.state-eq-twl* (**infix** $\sim$*TWL 51*)

inductive *conflict-twl* **where**
conflict-twl-rule:
$C \in$ *candidates-conflict-twl S* $\Longrightarrow$
 $S' \sim$ *update-conflicting-twl* (*Some* (*raw-clause C*)) *S* $\Longrightarrow$
 raw-conflicting-twl S = *None* $\Longrightarrow$
 conflict-twl S S'

inductive-cases *conflict-twlE*: *conflict-twl S T*

lemma *conflict-twl-iff-conflict*:
 shows *conflict S T* $\longleftrightarrow$ *conflict-twl S T* (**is** *?C* $\longleftrightarrow$ *?T*)
⟨*proof*⟩

inductive $cdcl_W$*-twl* :: *'v wf-twl* $\Rightarrow$ *'v wf-twl* $\Rightarrow$ *bool* **for** *S* :: *'v wf-twl* **where**
propagate: *propagate-twl S S'* $\Longrightarrow$ $cdcl_W$*-twl S S'* |
conflict: *conflict-twl S S'* $\Longrightarrow$ $cdcl_W$*-twl S S'* |
other: $cdcl_W$*-o S S'* $\Longrightarrow$ $cdcl_W$*-twl S S'*|
rf: $cdcl_W$*-rf S S'* $\Longrightarrow$ $cdcl_W$*-twl S S'*

lemma $cdcl_W$*-twl-iff-*$cdcl_W$:
 assumes $cdcl_W$*-all-struct-inv S*
 shows $cdcl_W$*-twl S T* $\longleftrightarrow$ $cdcl_W$ *S T*
 ⟨*proof*⟩

lemma *rtranclp-*$cdcl_W$*-twl-all-struct-inv-inv*:
 assumes $cdcl_W$*-twl*** *S T* **and** $cdcl_W$*-all-struct-inv S*
 shows $cdcl_W$*-all-struct-inv T*
 ⟨*proof*⟩

lemma *rtranclp-cdcl*$_W$*-twl-iff-rtranclp-cdcl*$_W$:
  assumes *cdcl*$_W$*-all-struct-inv S*
  shows $cdcl_W\text{-}twl^{**}\ S\ T \longleftrightarrow cdcl_W{}^{**}\ S\ T$ (**is** $?T \longleftrightarrow ?W$)
⟨*proof*⟩

end

end

26 Implementation for 2 Watched-Literals

theory *CDCL-Two-Watched-Literals-Implementation*
imports *CDCL-Two-Watched-Literals-Invariant*
begin

The general idea is the following:

1. Build a "propagate" queue and a conflict clause.
2. While updating the data-structure: if you find a conflicting clause, update the conflict clause. Otherwise prepend the propagated clause.
3. While updating, when looking for conflicts and propagation, work with respect to the trail of the state and the propagate queue (and not only the trail of the state).
4. As long as the propagate queue is not empty, dequeue the first element, push it on the trail (with the *conflict-driven-clause-learning*$_W$*.propagate* rule), propagate, and update the data-structure.
5. if a conflict has been found such that it is entailed by the trail only (i.e. without the propagate queue), then apply the *conflict-driven-clause-learning*$_W$*.conflict* rule.

It is important to remember that a conflicting clause with respect to the trail and the queue might not be the earliest conflicting clause, meaning that the proof of non-redundancy should not work anymore.

However, once a conflict has been found, we can stop adding literals to the queue: we just have to finish updating the data-structure (both to keep the invariant and find a potentially better conflict). A conflict is better when it involves less literals, i.e. less propagations are needed before finding the conflict.

datatype *'v candidate* =
  Prop-Or-Conf
    (*prop-queue*: (*'v literal* × *'v twl-clause*) *list*)
    (*conflict*: *'v twl-clause option*)

Morally instead of (*'v literal* × *'v twl-clause*) *list*, we should use (*'v*, *nat*, *'v twl-clause*) *marked-lits* with only *Propagated*. However, we do not want to define the function for *Marked* too. The following function makes the conversion from the pair to the trail:

abbreviation *get-trail-of-cand* **where**
get-trail-of-cand C ≡ *map* (*case-prod Propagated*) (*prop-queue C*)

datatype *'v twl-state-cands* =
  TWL-State-Cand (*twl-state*: *'v twl-state*)

($cand$: $'v$ $candidate$)

fun *find-earliest-conflict* :: ($'v$, *nat*, $'v$ *twl-clause*) *marked-lits* $\Rightarrow$
$'v$ *twl-clause option* $\Rightarrow$ $'v$ *twl-clause option* $\Rightarrow$ $'v$ *twl-clause option* **where**
find-earliest-conflict - *None* $C = C$ |
find-earliest-conflict - C *None* $= C$ |
find-earliest-conflict [] C - $= C$ |
find-earliest-conflict (L # M) (*Some* C) (*Some* D) =
(*case* ($M \models a$ *mset* (*raw-clause* C), $\neg M \models a$ *mset* (*raw-clause* D)) *of*
(*True*, *True*) $\Rightarrow$ *find-earliest-conflict* M (*Some* C) (*Some* D)
| (*False*, *True*) $\Rightarrow$ *Some* D
| (*True*, *False*) $\Rightarrow$ *Some* C
| - $\Rightarrow$ *Some* C)

lemma *find-earliest-conflict-cases*:
find-earliest-conflict M (*Some* C) (*Some* D) = *Some* C $\vee$
find-earliest-conflict M (*Some* C) (*Some* D) = *Some* D
$\langle proof \rangle$

While updating the clauses, there are several cases:

- L is not watched and there is nothing to do;
- there is a literal to be watched: there are swapped;
- there is no literal to be watched, the other literal is not assigned: the clause is a propagate or a conflict candidate;
- there is no literal to be watched, the other literal is $-\ L$: the clause is a tautology and nothing special is done;
- there is no literal to be watched, but the other literal is true: there is nothing to do;
- there is no literal to be watched, but the other literal is false: the clause is a conflict candidate.

  The function returns a couple composed of a list of clauses and a candidate.

fun
rewatch-nat-cand-single-clause ::
$'v$ *literal* $\Rightarrow$ ($'v$, *nat*, $'v$ *twl-clause*) *marked-lits* $\Rightarrow$ $'v$ *twl-clause* $\Rightarrow$
$'v$ *twl-clause list* $\times$ $'v$ *candidate* $\Rightarrow$
$'v$ *twl-clause list* $\times$ $'v$ *candidate*
where
rewatch-nat-cand-single-clause L M C (*Cs*, *Ks*) =
(*if* $-\ L \in$ *set* (*watched* C) *then*
case filter ($\lambda L'$. $L' \notin$ *set* (*watched* C) $\wedge$ $-\ L' \notin$ *insert* L (*lits-of-l* M)) (*unwatched* C) *of*
[] $\Rightarrow$
(*case remove1* ($-L$) (*watched* C) *of* (* *contains at most a single element* *)
[] $\Rightarrow$ (C # *Cs*, *Prop-Or-Conf* (*prop-queue Ks*)
(*find-earliest-conflict* (*get-trail-of-cand Ks* @ M) (*Some* C) (*conflict Ks*)))
| L' # - $\Rightarrow$
if undefined-lit (*get-trail-of-cand Ks* @ M) L' $\wedge$ *atm-of* $L \neq$ *atm-of* L'
then (C # *Cs*, *Prop-Or-Conf* ((L', C) # *prop-queue Ks*) (*conflict Ks*))
else
(*if* $-L' \in$ *lits-of-l* (*get-trail-of-cand Ks* @ M)

then $(C \# Cs,$ *Prop-Or-Conf* (*prop-queue Ks*)
(*find-earliest-conflict* (*get-trail-of-cand Ks* @ M) (*Some* C) (*conflict Ks*)))
else $(C \# Cs,$ *Ks*)))
| $L' \#$ - $\Rightarrow$
(*TWL-Clause* ($L' \#$ *remove1* $(-L)$ (*watched* C)) $(-L \#$ *remove1* L' (*unwatched* C)) $\# Cs$, *Ks*)
else
$(C \# Cs,$ *Ks*))

declare *rewatch-nat-cand-single-clause.simps*[*simp del*]

lemma *CNot-mset-replicate*[*simp*]:
CNot (*mset* (*replicate* n $(-\ L)$)) = (*if* $n = 0$ *then* {} *else* {{#L#}})
⟨*proof*⟩

lemma *wf-rewatch-nat-cand-single-clause-cases*[*consumes 1*, *case-names wf lit-notin propagate conflict no-conflict update-cls*]:
assumes
wf: *wf-twl-cls* M C **and**
lit-notin: $-L \notin$ *set* (*watched* C) $\Longrightarrow$
rewatch-nat-cand-single-clause L M C (*Cs*, *Ks*) = $(C \# Cs,$ *Ks*) $\Longrightarrow$
P
and
single-lit-watched: $-L \in$ *set* (*watched* C) $\Longrightarrow$
filter ($\lambda L'.\ L' \notin$ *set* (*watched* C) $\wedge -\ L' \notin$ *insert* L (*lits-of-l* M)) (*unwatched* C) = [] $\Longrightarrow$
watched $C = [-L] \Longrightarrow$
set (*unwatched* C) $\subseteq \{-L\} \Longrightarrow$
rewatch-nat-cand-single-clause L M C (*Cs*, *Ks*) = $(C \# Cs,$ *Prop-Or-Conf* (*prop-queue Ks*)
(*find-earliest-conflict* (*get-trail-of-cand Ks* @ M) (*Some* C) (*conflict Ks*))) $\Longrightarrow$
P
and
propagate: $\bigwedge L'.\ -L \in$ *set* (*watched* C) $\Longrightarrow$
filter ($\lambda L'.\ L' \notin$ *set* (*watched* C) $\wedge -\ L' \notin$ *insert* L (*lits-of-l* M)) (*unwatched* C) = [] $\Longrightarrow$
set (*watched* C) = $\{-L, L'\} \Longrightarrow$
undefined-lit (*get-trail-of-cand Ks* @ M) $L' \Longrightarrow$
atm-of $L \neq$ *atm-of* $L' \Longrightarrow$
rewatch-nat-cand-single-clause L M C (*Cs*, *Ks*) =
$(C \# Cs,$ *Prop-Or-Conf* $((L', C) \#$ *prop-queue Ks*) (*conflict Ks*)) $\Longrightarrow$
P
and
conflict: $\bigwedge L'.\ -L \in$ *set* (*watched* C) $\Longrightarrow$
filter ($\lambda L'.\ L' \notin$ *set* (*watched* C) $\wedge -\ L' \notin$ *insert* L (*lits-of-l* M)) (*unwatched* C) = [] $\Longrightarrow$
set (*watched* C) = $\{-L, L'\} \Longrightarrow$
$-L' \in$ *insert* L (*lits-of-l* (*get-trail-of-cand Ks* @ M)) $\Longrightarrow$
rewatch-nat-cand-single-clause L M C (*Cs*, *Ks*) = $(C \# Cs,$ *Prop-Or-Conf* (*prop-queue Ks*)
(*find-earliest-conflict* (*get-trail-of-cand Ks* @ M) (*Some* C) (*conflict Ks*))) $\Longrightarrow$
P
and
no-conflict: $\bigwedge L'.\ -L \in$ *set* (*watched* C) $\Longrightarrow$
filter ($\lambda L'.\ L' \notin$ *set* (*watched* C) $\wedge -\ L' \notin$ *insert* L (*lits-of-l* M)) (*unwatched* C) = [] $\Longrightarrow$
set (*watched* C) = $\{-L, L'\} \Longrightarrow$
$L' \in$ *insert* L (*lits-of-l* (*get-trail-of-cand Ks* @ M)) $\Longrightarrow$
rewatch-nat-cand-single-clause L M C (*Cs*, *Ks*) = $(C \# Cs,$ *Ks*) $\Longrightarrow$
P
and
update-cls: $\bigwedge L'\ fUW.\ -L \in$ *set* (*watched* C) $\Longrightarrow$

filter $(\lambda L'.\ L' \notin$ *set* (*watched* C) $\wedge - L' \notin$ *insert* L (*lits-of-l* M)) (*unwatched* C) $= L'$ # *fUW*
$\Longrightarrow$
rewatch-nat-cand-single-clause $L\ M\ C$ (Cs, Ks) =
(*TWL-Clause* (L' # *remove1* ($-L$) (*watched* C)) ($-L$ # *remove1* L' (*unwatched* C)) # Cs, Ks)
$\Longrightarrow$
P
shows P
⟨*proof*⟩

lemmas *rewatch-nat-cand-single-clause-cases* =
wf-rewatch-nat-cand-single-clause-cases[*OF wf-twl-cls-append*[*of get-trail-of-cand* -], *consumes 2*,
case-names wf lit-notin propagate conflict no-conflict update-cls]

lemma *lit-of-case-Propagated*[*simp*]: *lit-of* (*case* x *of* (x, xa) $\Rightarrow$ *Propagated* x xa) = *fst* x
⟨*proof*⟩

lemma *no-dup-rewatch-nat-cand-single-clause*:
fixes L :: *'v literal*
assumes
L: $L \in$ *lits-of-l* M **and**
wf: *wf-twl-cls* (*get-trail-of-cand* Ks @ M) C **and**
n-d: *no-dup* (*get-trail-of-cand* Ks @ M)
shows *no-dup* (M @ *get-trail-of-cand* (*snd* (*rewatch-nat-cand-single-clause* $L\ M\ C$ (Cs, Ks))))
⟨*proof*⟩

lemma *wf-twl-cls-prop-in-trailD*:
assumes *wf-twl-cls* M (*TWL-Clause* W UW)
shows $\forall L \in$ *set* W. $-L \in$ *lits-of-l* $M \longrightarrow (\forall L' \in$ *set* UW. $L' \notin$ *set* $W \longrightarrow -L' \in$ *lits-of-l* M)
⟨*proof*⟩

lemma *rewatch-nat-cand-single-clause-conflict*:
assumes
L: $L \in$ *lits-of-l* M **and**
wf: *wf-twl-cls* (*get-trail-of-cand* Ks @ M) C **and**
conf: *conflict* Ks = *Some* D **and**
conf': *conflict* (*snd* (*rewatch-nat-cand-single-clause* $L\ M\ C$ (Cs, Ks))) = *Some* D' **and**
n-d: *no-dup* (*get-trail-of-cand* Ks @ M) **and**
confI: *get-trail-of-cand* Ks @ M $\models$*as CNot* (*mset* (*raw-clause* D))
shows *get-trail-of-cand* Ks @ M $\models$*as CNot* (*mset* (*raw-clause* D'))
⟨*proof*⟩

lemma *rewatch-nat-cand-single-clause-conflict-found*:
assumes
L: $L \in$ *lits-of-l* M **and**
wf: *wf-twl-cls* (*get-trail-of-cand* Ks @ M) C **and**
n-d: *no-dup* (*get-trail-of-cand* Ks @ M) **and**
conf: *conflict* Ks = *None* **and**
conf': *conflict* (*snd* (*rewatch-nat-cand-single-clause* $L\ M\ C$ (Cs, Ks))) = *Some* D'
shows *get-trail-of-cand* Ks @ M $\models$*as CNot* (*mset* (*raw-clause* D'))
⟨*proof*⟩

lemma *rewatch-nat-cand-single-clause-clauses*:
assumes
wf: *wf-twl-cls* (*get-trail-of-cand* Ks @ M) C **and**

n-d: *no-dup* (*get-trail-of-cand Ks* @ *M*)
shows *clauses-of-l* (*map raw-clause* (*fst* (*rewatch-nat-cand-single-clause L M C* (*Cs*, *Ks*)))) =
clauses-of-l (*map raw-clause* (*C* # *Cs*))
⟨*proof*⟩

This lemma is *wrong*: we are speaking of half-update data-structure, meaning that *wf-twl-cls* (*get-trail-of-cand K* @ *M*) *C* is the wrong assumption to use.

lemma
fixes *Ks* :: *'v candidate* **and** *M* :: (*'v*, *nat*, *'v twl-clause*) *marked-lit list*
and *L* :: *'v literal* **and** *Cs* :: *'v twl-clause list* **and** *C* :: *'v twl-clause*
defines *S* ≡ *rewatch-nat-cand-single-clause L M C* (*Cs*, *Ks*)
assumes *wf*: *wf-twl-cls* (*get-trail-of-cand Ks* @ *M*) *C* **and**
n-d: *no-dup* (*get-trail-of-cand Ks* @ *M*)
shows *wf-twl-cls* (*get-trail-of-cand* (*snd S*) @ *M*) *C*
⟨*proof*⟩

lemma *wf-rewatch-nat-cand-single-clause*:
fixes *Ks* :: *'v candidate* **and** *M* :: (*'v*, *nat*, *'v twl-clause*) *marked-lit list* **and**
L :: (*'v*, *nat*, *'v twl-clause*) *marked-lit* **and** *Cs* :: *'v twl-clause list* **and**
C :: *'v twl-clause*
defines *S* ≡ *rewatch-nat-cand-single-clause* (*lit-of L*) *M C* (*Cs*, *Ks*)
assumes
wf: *wf-twl-cls M C* **and**
n-d: *no-dup* (*get-trail-of-cand Ks* @ *M*) **and**
undef: *undefined-lit* (*get-trail-of-cand Ks* @ *M*) (*lit-of L*)
shows *wf-twl-cls* (*L* # *M*) (*hd* (*fst S*))
⟨*proof*⟩

lemma *rewatch-nat-cand-single-clause-no-dup*:
fixes *Ks* :: *'v candidate* **and** *M* :: (*'v*, *nat*, *'v twl-clause*) *marked-lit list*
and *L* :: *'v literal* **and** *Cs* :: *'v twl-clause list* **and** *C* :: *'v twl-clause*
defines *S* ≡ *rewatch-nat-cand-single-clause L M C* (*Cs*, *Ks*)
assumes *wf*: *wf-twl-cls M C* **and**
n-d: *no-dup* (*get-trail-of-cand Ks* @ *M*) **and**
undef: *undefined-lit* (*get-trail-of-cand Ks* @ *M*) *L*
shows *no-dup* (*get-trail-of-cand* (*snd S*) @ *M*)
⟨*proof*⟩

fun
rewatch-nat-cand-clss ::
'v literal ⇒ (*'v*, *nat*, *'v twl-clause*) *marked-lits* ⇒
'v twl-clause list × *'v candidate* ⇒
'v twl-clause list × *'v candidate*
where
rewatch-nat-cand-clss L M (*Cs*, *Ks*) =
foldr (*rewatch-nat-cand-single-clause L M*) *Cs* ([], *Ks*)

lemma *wf-foldr-rewatch-nat-cand-single-clause*:
fixes *Ks* :: *'v candidate* **and** *M* :: (*'v*, *nat*, *'v twl-clause*) *marked-lits* **and**
L :: (*'v*, *nat*, *'v twl-clause*) *marked-lit* **and** *Cs* :: *'v twl-clause list* **and**
C :: *'v twl-clause*
defines *S* ≡ *rewatch-nat-cand-clss* (*lit-of L*) *M* (*Cs*, *Ks*)
assumes
wf: ∀ *C* ∈ *set Cs*. *wf-twl-cls M C* **and**
n-d: *no-dup* (*get-trail-of-cand Ks* @ *M*) **and**

undef: *undefined-lit* (*get-trail-of-cand Ks* @ *M*) (*lit-of L*)
shows
(∀ *C* ∈ *set* (*fst S*). *wf-twl-cls* (*L* # *M*) *C*) ∧
undefined-lit (*get-trail-of-cand* (*snd S*) @ *M*) (*lit-of L*) ∧
no-dup (*get-trail-of-cand* (*snd S*) @ *M*) (**is** *?wf S* ∧ *?undef S* ∧ *?n-d S*)
⟨*proof*⟩

declare *rewatch-nat-cand-clss.simps*[*simp del*]

fun *rewatch-nat-cand* :: *'a literal* ⇒ *'a twl-state-cands* ⇒ *'a twl-state-cands* **where**
rewatch-nat-cand L (*TWL-State-Cand S Ks*) =
(*let*
(*N*, *K*) = *rewatch-nat-cand-clss L* (*raw-trail S*) (*raw-init-clss S*, *Ks*);
(*U*, *K'*) = *rewatch-nat-cand-clss L* (*raw-trail S*) (*raw-learned-clss S*, *K*) *in*
TWL-State-Cand
(*TWL-State* (*raw-trail S*) *N U* (*backtrack-lvl S*) (*raw-conflicting S*))
K')

fun *raw-cons-trail* **where**
raw-cons-trail L (*TWL-State M N U k C*) = *TWL-State* (*L* # *M*) *N U k C*

lemma *length-raw-trail-raw-cons-trails*[*simp*]:
length (*raw-trail* (*raw-cons-trail* (*Propagated L C'*) *S*)) = *Suc* (*length* (*raw-trail S*))
⟨*proof*⟩

fun *raw-cons-trail-pq* **where**
raw-cons-trail-pq L (*TWL-State-Cand S Q*) = *TWL-State-Cand* (*raw-cons-trail L S*) *Q*

fun *update-conflicting-pq* **where**
update-conflicting-pq L (*TWL-State-Cand S Q*) = *TWL-State-Cand* (*update-conflicting L S*) *Q*

lemma
fixes *Ks* :: *'v candidate* **and** *M* :: (*'v*, *nat*, *'v twl-clause*) *marked-lits* **and**
L :: (*'v*, *nat*, *'v twl-clause*) *marked-lit* **and** *Cs* :: *'v twl-clause list* **and**
C :: *'v twl-clause* **and** *S* :: *'v twl-state*
defines *T* ≡ *rewatch-nat-cand* (*lit-of L*) (*TWL-State-Cand S Ks*)
assumes
wf: *wf-twl-state S* **and**
n-d: *no-dup* (*get-trail-of-cand Ks* @ *raw-trail S*) **and**
undef: *undefined-lit* (*get-trail-of-cand Ks* @ *raw-trail S*) (*lit-of L*)
shows *wf-twl-state* (*raw-cons-trail L* (*twl-state T*))
⟨*proof*⟩

function *do-propagate-or-conflict-step* :: *'a twl-state-cands* ⇒ *'a twl-state-cands* **where**
do-propagate-or-conflict-step (*TWL-State-Cand S* (*Prop-Or-Conf* [] (*Some D*))) =
(*if trail S* ⊨*as CNot* (*mset* (*raw-clause D*))
then do-propagate-or-conflict-step
(*update-conflicting-pq* (*Some* (*raw-clause D*)) (*TWL-State-Cand S* (*Prop-Or-Conf* [] *None*)))
else TWL-State-Cand S (*Prop-Or-Conf* [] (*Some D*))) |
do-propagate-or-conflict-step (*TWL-State-Cand S* (*Prop-Or-Conf* [] *None*)) =
TWL-State-Cand S (*Prop-Or-Conf* [] *None*) |
do-propagate-or-conflict-step (*TWL-State-Cand S* (*Prop-Or-Conf* (*l* @ [(*L*, *C'*)]) (*Some D*))) =
(*if trail S* ⊨*as CNot* (*mset* (*raw-clause D*))
then do-propagate-or-conflict-step
(*update-conflicting-pq* (*Some* (*raw-clause D*)) (*TWL-State-Cand S* (*Prop-Or-Conf l None*)))

else do-propagate-or-conflict-step
  (*raw-cons-trail-pq* (*Propagated L C'*) (*TWL-State-Cand S* (*Prop-Or-Conf l* (*Some D*))))) |
do-propagate-or-conflict-step (*TWL-State-Cand S* (*Prop-Or-Conf* (*l* @ [(*L*, *C'*)]) *None*)) =
do-propagate-or-conflict-step
  (*raw-cons-trail-pq* (*Propagated L C'*) (*TWL-State-Cand S* (*Prop-Or-Conf l None*)))
⟨*proof*⟩

termination ⟨*proof*⟩
end
theory *Prop-Superposition*
imports *Partial-Clausal-Logic ../lib/Herbrand-Interpretation*
begin

27 Superposition

no-notation *Herbrand-Interpretation.true-cls* (**infix** ⊨ *50*)
notation *Herbrand-Interpretation.true-cls* (**infix** ⊨*h 50*)

no-notation *Herbrand-Interpretation.true-clss* (**infix** ⊨*s 50*)
notation *Herbrand-Interpretation.true-clss* (**infix** ⊨*hs 50*)

lemma *herbrand-interp-iff-partial-interp-cls*:
  $S \models h\ C \longleftrightarrow \{Pos\ P|P.\ P{\in}S\} \cup \{Neg\ P|P.\ P{\notin}S\} \models C$
  ⟨*proof*⟩

lemma *herbrand-consistent-interp*:
  consistent-interp $(\{Pos\ P|P.\ P{\in}S\} \cup \{Neg\ P|P.\ P{\notin}S\})$
  ⟨*proof*⟩

lemma *herbrand-total-over-set*:
  total-over-set $(\{Pos\ P|P.\ P{\in}S\} \cup \{Neg\ P|P.\ P{\notin}S\})\ T$
  ⟨*proof*⟩

lemma *herbrand-total-over-m*:
  total-over-m $(\{Pos\ P|P.\ P{\in}S\} \cup \{Neg\ P|P.\ P{\notin}S\})\ T$
  ⟨*proof*⟩

lemma *herbrand-interp-iff-partial-interp-clss*:
  $S \models hs\ C \longleftrightarrow \{Pos\ P|P.\ P{\in}S\} \cup \{Neg\ P|P.\ P{\notin}S\} \models s\ C$
  ⟨*proof*⟩

definition *clss-lt* :: *'a::wellorder clauses* ⇒ *'a clause* ⇒ *'a clauses* **where**
clss-lt N C = $\{D \in N.\ D\ \#{\subset}\#\ C\}$

notation (*latex* **output**)
 clss-lt (-<^bsup>-<^esup>)

locale *selection* =
  fixes *S* :: *'a clause* ⇒ *'a clause*
  assumes
    S-selects-subseteq: $\bigwedge C.\ S\ C \leq\#\ C$ **and**
    S-selects-neg-lits: $\bigwedge C\ L.\ L \in\#\ S\ C \Longrightarrow$ *is-neg L*

locale *ground-resolution-with-selection* =

selection S **for** $S :: ('a :: wellorder)\ clause \Rightarrow 'a\ clause$
begin

context
fixes $N :: 'a\ clause\ set$
begin

We do not create an equivalent of δ, but we directly defined N_C by inlining the definition.

function
$production :: 'a\ clause \Rightarrow 'a\ interp$
where
$production\ C =$
$\{A.\ C \in N \wedge C \neq \{\#\} \wedge Max\ (set\text{-}mset\ C) = Pos\ A \wedge count\ C\ (Pos\ A) \leq 1$
$\wedge \neg\ (\bigcup D \in \{D.\ D\ \#{\subset}\#\ C\}.\ production\ D) \models h\ C \wedge S\ C = \{\#\}\}$
$\langle proof \rangle$
termination $\langle proof \rangle$

declare *production.simps*[*simp del*]

definition $interp :: 'a\ clause \Rightarrow 'a\ interp$ **where**
$interp\ C = (\bigcup D \in \{D.\ D\ \#{\subset}\#\ C\}.\ production\ D)$

lemma *production-unfold*:
$production\ C = \{A.\ C \in N \wedge C \neq \{\#\} \wedge Max\ (set\text{-}mset\ C) = Pos\ A \wedge\ count\ C\ (Pos\ A) \leq 1 \wedge \neg\ interp\ C \models h\ C \wedge S\ C = \{\#\}\}$
$\langle proof \rangle$

abbreviation $productive\ A \equiv (production\ A \neq \{\})$

abbreviation $produces :: 'a\ clause \Rightarrow 'a \Rightarrow bool$ **where**
$produces\ C\ A \equiv production\ C = \{A\}$

lemma *producesD*:
$produces\ C\ A \Longrightarrow C \in N \wedge C \neq \{\#\} \wedge Pos\ A = Max\ (set\text{-}mset\ C) \wedge count\ C\ (Pos\ A) \leq 1 \wedge$
$\neg\ interp\ C \models h\ C \wedge S\ C = \{\#\}$
$\langle proof \rangle$

lemma $produces\ C\ A \Longrightarrow Pos\ A \in\#\ C$
$\langle proof \rangle$

lemma *interp′-def-in-set*:
$interp\ C = (\bigcup D \in \{D \in N.\ D\ \#{\subset}\#\ C\}.\ production\ D)$
$\langle proof \rangle$

lemma *production-iff-produces*:
$produces\ D\ A \longleftrightarrow A \in production\ D$
$\langle proof \rangle$

definition $Interp :: 'a\ clause \Rightarrow 'a\ interp$ **where**
$Interp\ C = interp\ C \cup production\ C$

lemma
assumes $produces\ C\ P$
shows $Interp\ C \models h\ C$
$\langle proof \rangle$

definition *INTERP* :: *'a interp* **where**
INTERP = ($\bigcup$ *D* ∈*N*. *production D*)

lemma *interp-subseteq-Interp*[*simp*]: *interp C* ⊆ *Interp C*
⟨*proof*⟩

lemma *Interp-as-UNION*: *Interp C* = ($\bigcup$ *D* ∈ {*D*. *D* #⊆# *C*}. *production D*)
⟨*proof*⟩

lemma *productive-not-empty*: *productive C* ⟹ *C* ≠ {#}
⟨*proof*⟩

lemma *productive-imp-produces-Max-literal*: *productive C* ⟹ *produces C* (*atm-of* (*Max* (*set-mset C*)))
⟨*proof*⟩

lemma *productive-imp-produces-Max-atom*: *productive C* ⟹ *produces C* (*Max* (*atms-of C*))
⟨*proof*⟩

lemma *produces-imp-Max-literal*: *produces C A* ⟹ *A* = *atm-of* (*Max* (*set-mset C*))
⟨*proof*⟩

lemma *produces-imp-Max-atom*: *produces C A* ⟹ *A* = *Max* (*atms-of C*)
⟨*proof*⟩

lemma *produces-imp-Pos-in-lits*: *produces C A* ⟹ *Pos A* ∈# *C*
⟨*proof*⟩

lemma *productive-in-N*: *productive C* ⟹ *C* ∈ *N*
⟨*proof*⟩

lemma *produces-imp-atms-leq*: *produces C A* ⟹ *B* ∈ *atms-of C* ⟹ *B* ≤ *A*
⟨*proof*⟩

lemma *produces-imp-neg-notin-lits*: *produces C A* ⟹ ¬ *Neg A* ∈# *C*
⟨*proof*⟩

lemma *less-eq-imp-interp-subseteq-interp*: *C* #⊆# *D* ⟹ *interp C* ⊆ *interp D*
⟨*proof*⟩

lemma *less-eq-imp-interp-subseteq-Interp*: *C* #⊆# *D* ⟹ *interp C* ⊆ *Interp D*
⟨*proof*⟩

lemma *less-imp-production-subseteq-interp*: *C* #⊂# *D* ⟹ *production C* ⊆ *interp D*
⟨*proof*⟩

lemma *less-eq-imp-production-subseteq-Interp*: *C* #⊆# *D* ⟹ *production C* ⊆ *Interp D*
⟨*proof*⟩

lemma *less-imp-Interp-subseteq-interp*: *C* #⊂# *D* ⟹ *Interp C* ⊆ *interp D*
⟨*proof*⟩

lemma *less-eq-imp-Interp-subseteq-Interp*: *C* #⊆# *D* ⟹ *Interp C* ⊆ *Interp D*
⟨*proof*⟩

lemma *false-Interp-to-true-interp-imp-less-multiset*: $A \notin \mathit{Interp}\ C \Longrightarrow A \in \mathit{interp}\ D \Longrightarrow C\ \#{\subset}\#\ D$
⟨*proof*⟩

lemma *false-interp-to-true-interp-imp-less-multiset*: $A \notin \mathit{interp}\ C \Longrightarrow A \in \mathit{interp}\ D \Longrightarrow C\ \#{\subset}\#\ D$
⟨*proof*⟩

lemma *false-Interp-to-true-Interp-imp-less-multiset*: $A \notin \mathit{Interp}\ C \Longrightarrow A \in \mathit{Interp}\ D \Longrightarrow C\ \#{\subset}\#\ D$
⟨*proof*⟩

lemma *false-interp-to-true-Interp-imp-le-multiset*: $A \notin \mathit{interp}\ C \Longrightarrow A \in \mathit{Interp}\ D \Longrightarrow C\ \#{\subseteq}\#\ D$
⟨*proof*⟩

lemma *interp-subseteq-INTERP*: $\mathit{interp}\ C \subseteq \mathit{INTERP}$
⟨*proof*⟩

lemma *production-subseteq-INTERP*: $\mathit{production}\ C \subseteq \mathit{INTERP}$
⟨*proof*⟩

lemma *Interp-subseteq-INTERP*: $\mathit{Interp}\ C \subseteq \mathit{INTERP}$
⟨*proof*⟩

This lemma corresponds to theorem 2.7.6 page 66 of CW.

lemma *produces-imp-in-interp*:
assumes *a-in-c*: $\mathit{Neg}\ A \in\#\ C$ **and** *d*: $\mathit{produces}\ D\ A$
shows $A \in \mathit{interp}\ C$
⟨*proof*⟩

lemma *neg-notin-Interp-not-produce*: $\mathit{Neg}\ A \in\#\ C \Longrightarrow A \notin \mathit{Interp}\ D \Longrightarrow C\ \#{\subseteq}\#\ D \Longrightarrow \neg\ \mathit{produces}\ D''\ A$
⟨*proof*⟩

lemma *in-production-imp-produces*: $A \in \mathit{production}\ C \Longrightarrow \mathit{produces}\ C\ A$
⟨*proof*⟩

lemma *not-produces-imp-notin-production*: $\neg\ \mathit{produces}\ C\ A \Longrightarrow A \notin \mathit{production}\ C$
⟨*proof*⟩

lemma *not-produces-imp-notin-interp*: $(\bigwedge D.\ \neg\ \mathit{produces}\ D\ A) \Longrightarrow A \notin \mathit{interp}\ C$
⟨*proof*⟩

The results below corresponds to Lemma 3.4.
Nitpicking: If $D = D'$ and D is productive, $I^D \subseteq I_{D'}$ does not hold.

lemma *true-Interp-imp-general*:
assumes
c-le-d: $C\ \#{\subseteq}\#\ D$ **and**
d-lt-d': $D\ \#{\subset}\#\ D'$ **and**
c-at-d: $\mathit{Interp}\ D \models h\ C$ **and**
subs: $\mathit{interp}\ D' \subseteq (\bigcup C \in CC.\ \mathit{production}\ C)$
shows $(\bigcup C \in CC.\ \mathit{production}\ C) \models h\ C$
⟨*proof*⟩

lemma *true-Interp-imp-interp*: $C\ \#{\subseteq}\#\ D \Longrightarrow D\ \#{\subset}\#\ D' \Longrightarrow \mathit{Interp}\ D \models h\ C \Longrightarrow \mathit{interp}\ D' \models h\ C$
⟨*proof*⟩

lemma *true-Interp-imp-Interp*: $C \ \#\subseteq\# \ D \Longrightarrow D \ \#\subset\# \ D' \Longrightarrow Interp \ D \models h \ C \Longrightarrow Interp \ D' \models h \ C$
⟨*proof*⟩

lemma *true-Interp-imp-INTERP*: $C \ \#\subseteq\# \ D \Longrightarrow Interp \ D \models h \ C \Longrightarrow INTERP \models h \ C$
⟨*proof*⟩

lemma *true-interp-imp-general*:
assumes
c-le-d: $C \ \#\subseteq\# \ D$ **and**
d-lt-d': $D \ \#\subset\# \ D'$ **and**
c-at-d: $interp \ D \models h \ C$ **and**
subs: $interp \ D' \subseteq (\bigcup C \in CC. \ production \ C)$
shows $(\bigcup C \in CC. \ production \ C) \models h \ C$
⟨*proof*⟩

This lemma corresponds to theorem 2.7.6 page 66 of CW. Here the strict maximality is important

lemma *true-interp-imp-interp*: $C \ \#\subseteq\# \ D \Longrightarrow D \ \#\subset\# \ D' \Longrightarrow interp \ D \models h \ C \Longrightarrow interp \ D' \models h \ C$
⟨*proof*⟩

lemma *true-interp-imp-Interp*: $C \ \#\subseteq\# \ D \Longrightarrow D \ \#\subset\# \ D' \Longrightarrow interp \ D \models h \ C \Longrightarrow Interp \ D' \models h \ C$
⟨*proof*⟩

lemma *true-interp-imp-INTERP*: $C \ \#\subseteq\# \ D \Longrightarrow interp \ D \models h \ C \Longrightarrow INTERP \models h \ C$
⟨*proof*⟩

lemma *productive-imp-false-interp*: $productive \ C \Longrightarrow \neg \ interp \ C \models h \ C$
⟨*proof*⟩

This lemma corresponds to theorem 2.7.6 page 66 of CW. Here the strict maximality is important

lemma *cls-gt-double-pos-no-production*:
assumes D: $\{\#Pos \ P, \ Pos \ P\#\} \ \#\subset\# \ C$
shows $\neg produces \ C \ P$
⟨*proof*⟩

This lemma corresponds to theorem 2.7.6 page 66 of CW.

lemma
assumes D: $C+\{\#Neg \ P\#\} \ \#\subset\# \ D$
shows $production \ D \neq \{P\}$
⟨*proof*⟩

lemma *in-interp-is-produced*:
assumes $P \in INTERP$
shows $\exists D. \ D +\{\#Pos \ P\#\} \in N \wedge produces \ (D +\{\#Pos \ P\#\}) \ P$
⟨*proof*⟩

end
end

abbreviation $MMax \ M \equiv Max \ (set\text{-}mset \ M)$

27.1 We can now define the rules of the calculus

inductive *superposition-rules* :: $'a \ clause \Rightarrow \ 'a \ clause \Rightarrow \ 'a \ clause \Rightarrow bool$ **where**
factoring: $superposition\text{-}rules \ (C + \{\#Pos \ P\#\} + \{\#Pos \ P\#\}) \ B \ (C + \{\#Pos \ P\#\}) \ |$

superposition-l: *superposition-rules* $(C_1 + \{\#Pos\ P\#\})\ (C_2 + \{\#Neg\ P\#\})\ (C_1 + C_2)$

inductive *superposition* :: *'a clauses* ⇒ *'a clauses* ⇒ *bool* **where**
superposition: $A \in N \Longrightarrow B \in N \Longrightarrow$ *superposition-rules* $A\ B\ C$
  $\Longrightarrow$ *superposition* $N\ (N \cup \{C\})$

definition *abstract-red* :: *'a::wellorder clause* ⇒ *'a clauses* ⇒ *bool* **where**
abstract-red $C\ N = ($*clss-lt* $N\ C \models p\ C)$

lemma *less-multiset*[*iff*]: $M < N \longleftrightarrow M\ \#{\subset}\#\ N$
  ⟨*proof*⟩

lemma *less-eq-multiset*[*iff*]: $M \leq N \longleftrightarrow M\ \#{\subseteq}\#\ N$
  ⟨*proof*⟩

lemma *herbrand-true-clss-true-clss-cls-herbrand-true-clss*:
  assumes
    AB: $A \models hs\ B$ **and**
    BC: $B \models p\ C$
  shows $A \models h\ C$
⟨*proof*⟩

lemma *abstract-red-subset-mset-abstract-red*:
  assumes
    abstr: *abstract-red* $C\ N$ **and**
    c-lt-d: $C \subseteq\#\ D$
  shows *abstract-red* $D\ N$
⟨*proof*⟩

lemma *true-clss-cls-extended*:
  assumes
    $A \models p\ B$ **and**
    tot: *total-over-m* $I\ (A)$ **and**
    cons: *consistent-interp* I **and**
    I-A: $I \models s\ A$
  shows $I \models B$
⟨*proof*⟩
lemma
  assumes
    CP: ¬ *clss-lt* $N\ (\{\#C\#\} + \{\#E\#\}) \models p\ \{\#C\#\} + \{\#Neg\ P\#\}$ **and**
     clss-lt $N\ (\{\#C\#\} + \{\#E\#\}) \models p\ \{\#E\#\} + \{\#Pos\ P\#\} \vee$ *clss-lt* $N\ (\{\#C\#\} + \{\#E\#\}) \models p$
$\{\#C\#\} + \{\#Neg\ P\#\}$
  shows *clss-lt* $N\ (\{\#C\#\} + \{\#E\#\}) \models p\ \{\#E\#\} + \{\#Pos\ P\#\}$

⟨*proof*⟩

locale *ground-ordered-resolution-with-redundancy* =
  ground-resolution-with-selection +
  fixes *redundant* :: *'a::wellorder clause* ⇒ *'a clauses* ⇒ *bool*
  assumes
    redundant-iff-abstract: *redundant* $A\ N \longleftrightarrow$ *abstract-red* $A\ N$
begin
definition *saturated* :: *'a clauses* ⇒ *bool* **where**
saturated $N \longleftrightarrow (\forall A\ B\ C.\ A \in N \longrightarrow B \in N \longrightarrow \neg$*redundant* $A\ N \longrightarrow \neg$*redundant* $B\ N$

$\longrightarrow$ *superposition-rules* A B C $\longrightarrow$ *redundant* C N $\vee$ $C \in N$)

lemma
assumes
saturated: *saturated* N **and**
finite: *finite* N **and**
empty: $\{\#\} \notin N$
shows *INTERP* $N \models hs$ N
⟨*proof*⟩

end

lemma *tautology-is-redundant*:
assumes *tautology* C
shows *abstract-red* C N
⟨*proof*⟩

lemma *subsumed-is-redundant*:
assumes *AB*: $A \subset\# B$
and *AN*: $A \in N$
shows *abstract-red* B N
⟨*proof*⟩

inductive *redundant* :: $'a$ *clause* $\Rightarrow$ $'a$ *clauses* $\Rightarrow$ *bool* **where**
subsumption: $A \in N \Longrightarrow A \subset\# B \Longrightarrow$ *redundant* B N

lemma *redundant-is-redundancy-criterion*:
fixes A :: $'a$ :: *wellorder clause* **and** N :: $'a$ :: *wellorder clauses*
assumes *redundant* A N
shows *abstract-red* A N
⟨*proof*⟩

lemma *redundant-mono*:
redundant A $N \Longrightarrow A \subseteq\# B \Longrightarrow$ *redundant* B N
⟨*proof*⟩

locale *truc* =
selection S **for** S :: *nat clause* $\Rightarrow$ *nat clause*
begin

end

end