Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

December 6, 2019

Contents

Chapter 1

Definition of Entailment

This chapter defines various form of entailment.

end

1.1 Partial Herbrand Interpretation

theory *Partial-Herbrand-Interpretation*
  imports
    Weidenbach-Book-Base.WB-List-More
    Ordered-Resolution-Prover.Clausal-Logic
begin

1.1.1 More Literals

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 *in-image-uminus-uminus*: $\langle a \in uminus \text{ ` } A \longleftrightarrow -a \in A\rangle$ **for** $a :: \langle 'v\ literal\rangle$
  $\langle proof\rangle$

lemma *uminus-lit-swap*: $- a = b \longleftrightarrow (a{::}'a\ literal) = - b$
  $\langle proof\rangle$

lemma *atm-of-notin-atms-of-iff*: $\langle atm\text{-}of\ L \notin atms\text{-}of\ C' \longleftrightarrow L \notin\# C' \wedge -L \notin\# C'\rangle$ **for** $L\ C'$
  $\langle proof\rangle$

lemma *atm-of-notin-atms-of-iff-Pos-Neg*:
  $\langle L \notin atms\text{-}of\ C' \longleftrightarrow Pos\ L \notin\# C' \wedge Neg\ L \notin\# C'\rangle$ **for** $L\ C'$
  $\langle proof\rangle$

lemma *atms-of-uminus*[*simp*]: $\langle atms\text{-}of\ (uminus \text{ `}\#\ C) = atms\text{-}of\ C\rangle$
  $\langle proof\rangle$

lemma *distinct-mset-atm-ofD*:
  $\langle distinct\text{-}mset\ (atm\text{-}of \text{ `}\#\ mset\ xc) \Longrightarrow distinct\ xc\rangle$
  $\langle proof\rangle$

lemma *atms-of-cong-set-mset*:
  $\langle set\text{-}mset\ D = set\text{-}mset\ D' \Longrightarrow atms\text{-}of\ D = atms\text{-}of\ D'\rangle$
  $\langle proof\rangle$

lemma *lit-in-set-iff-atm*:
  ‹$NO\text{-}MATCH\ (Pos\ x)\ l \Longrightarrow NO\text{-}MATCH\ (Neg\ x)\ l \Longrightarrow$
    $l \in M \longleftrightarrow (\exists l'.\ (l = Pos\ l' \wedge Pos\ l' \in M) \vee (l = Neg\ l' \wedge Neg\ l' \in M))$›
  ⟨*proof*⟩

We define here entailment by a set of literals. This is an Herbrand interpretation, but not the same as used for the resolution prover. Both has different properties. One key difference is that such a set can be inconsistent (i.e. containing both L and $-\ L$).

Satisfiability is defined by the existence of a total and consistent model.

lemma *lit-eq-Neg-Pos-iff*:
  ‹$x \neq Neg\ (atm\text{-}of\ x) \longleftrightarrow is\text{-}pos\ x$›
  ‹$x \neq Pos\ (atm\text{-}of\ x) \longleftrightarrow is\text{-}neg\ x$›
  ‹$-x \neq Neg\ (atm\text{-}of\ x) \longleftrightarrow is\text{-}neg\ x$›
  ‹$-x \neq Pos\ (atm\text{-}of\ x) \longleftrightarrow is\text{-}pos\ x$›
  ‹$Neg\ (atm\text{-}of\ x) \neq x \longleftrightarrow is\text{-}pos\ x$›
  ‹$Pos\ (atm\text{-}of\ x) \neq x \longleftrightarrow is\text{-}neg\ x$›
  ‹$Neg\ (atm\text{-}of\ x) \neq -x \longleftrightarrow is\text{-}neg\ x$›
  ‹$Pos\ (atm\text{-}of\ x) \neq -x \longleftrightarrow is\text{-}pos\ x$›
  ⟨*proof*⟩

1.1.2 Clauses

Clauses are set of literals or (finite) multisets of literals.

type-synonym $'v\ clause\text{-}set = {'v}\ clause\ set$
type-synonym $'v\ clauses = {'v}\ clause\ multiset$

lemma *is-neg-neg-not-is-neg*: $is\text{-}neg\ (-\ L) \longleftrightarrow \neg\ is\text{-}neg\ L$
  ⟨*proof*⟩

1.1.3 Partial Interpretations

type-synonym $'a\ partial\text{-}interp = {'a}\ literal\ set$

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

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

Consistency

definition $consistent\text{-}interp :: {'a}\ literal\ set \Rightarrow bool$ **where**
$consistent\text{-}interp\ I \longleftrightarrow (\forall L.\ \neg(L \in I \wedge -\ L \in I))$

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

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

lemma *Ex-consistent-interp*: ‹$Ex\ consistent\text{-}interp$›
  ⟨*proof*⟩

lemma *consistent-interp-subset*:
  assumes
    $A \subseteq B$ **and**

$consistent\text{-}interp\ B$
shows $consistent\text{-}interp\ A$
⟨*proof*⟩

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

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

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

lemma *consistent-interp-unionD*: ‹$consistent\text{-}interp\ (I \cup I') \Longrightarrow consistent\text{-}interp\ I'$›
⟨*proof*⟩

lemma *consistent-interp-insert-iff*:
‹$consistent\text{-}interp\ (insert\ L\ C) \longleftrightarrow consistent\text{-}interp\ C \wedge -L \notin C$›
⟨*proof*⟩

lemma (**in** −) *distinct-consistent-distinct-atm*:
‹$distinct\ M \Longrightarrow consistent\text{-}interp\ (set\ M) \Longrightarrow distinct\text{-}mset\ (atm\text{-}of\ \text{‘}\#\ mset\ M)$›
⟨*proof*⟩

Atoms

We define here various lifting of *atm-of* (applied to a single literal) to set and multisets of literals.

definition *atms-of-ms* :: $'a\ clause\ set \Rightarrow\ 'a\ set$ **where**
$atms\text{-}of\text{-}ms\ \psi s = \bigcup(atms\text{-}of\ \text{‘}\ \psi s)$

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

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

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

lemma *atms-of-ms-emtpy-set*[*simp*]:
$atms\text{-}of\text{-}ms\ \{\} = \{\}$
⟨*proof*⟩

lemma *atms-of-ms-memtpy*[*simp*]:
$atms\text{-}of\text{-}ms\ \{\{\#\}\} = \{\}$
⟨*proof*⟩

lemma *atms-of-ms-mono*:

$A \subseteq B \Longrightarrow$ *atms-of-ms* $A \subseteq$ *atms-of-ms* B
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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 *add-mset* 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*]:
$atms\text{-}of\text{-}s\ \{L\} = \{atm\text{-}of\ L\}$
⟨*proof*⟩

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

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

lemma *atm-of-in-atm-of-set-in-uminus*:
$atm\text{-}of\ L' \in atm\text{-}of\ `\ B \Longrightarrow L' \in B \lor -\ L' \in B$
⟨*proof*⟩

lemma *finite-atms-of-s*[*simp*]:
‹$finite\ M \Longrightarrow finite\ (atms\text{-}of\text{-}s\ M)$›
⟨*proof*⟩

lemma
atms-of-s-empty [*simp*]:
‹$atms\text{-}of\text{-}s\ \{\} = \{\}$› **and**
atms-of-s-empty-iff[*simp*]:
‹$atms\text{-}of\text{-}s\ x = \{\} \longleftrightarrow x = \{\}$›
⟨*proof*⟩

Totality

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

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

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

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

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

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

lemma *total-over-set-union*[*iff*]:
$total\text{-}over\text{-}set\ I\ (Ls \cup Ls') \longleftrightarrow (total\text{-}over\text{-}set\ I\ Ls \land total\text{-}over\text{-}set\ I\ Ls')$
⟨*proof*⟩

lemma *total-over-m-subset*:
$A \subseteq B \Longrightarrow$ *total-over-m I B* $\Longrightarrow$ *total-over-m I A*
⟨*proof*⟩

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

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

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

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

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

lemma *total-over-set-atms-of-m*[*simp*]:
total-over-set Ia (*atms-of-s Ia*)
⟨*proof*⟩

lemma *total-over-set-literal-defined*:
assumes *add-mset A D* $\in \psi s$
and *total-over-set I* (*atms-of-ms* ψs)
shows $A \in I \vee -A \in I$
⟨*proof*⟩

lemma *tot-over-m-remove*:
assumes *total-over-m* $(I \cup \{L\})$ $\{\psi\}$
and *L*: $L \notin\# \psi$ $-L \notin\# \psi$
shows *total-over-m I* $\{\psi\}$
⟨*proof*⟩

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 $\textit{total-over-m}\ (I \cup I')\ (\psi \cup \psi')$
⟨*proof*⟩

lemma *total-over-m-alt-def*: ‹$\textit{total-over-m}\ I\ S \longleftrightarrow \textit{atms-of-ms}\ S \subseteq \textit{atms-of-s}\ I$›
⟨*proof*⟩

lemma *total-over-set-alt-def*: ‹$\textit{total-over-set}\ M\ A \longleftrightarrow A \subseteq \textit{atms-of-s}\ M$›
⟨*proof*⟩

Interpretations

definition *true-cls* :: $'a\ \textit{partial-interp} \Rightarrow\ 'a\ \textit{clause} \Rightarrow \textit{bool}$ (**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-add-mset*[*iff*]: $I \models \textit{add-mset}\ a\ D \longleftrightarrow a \in I \vee I \models D$
⟨*proof*⟩

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

lemma *true-cls-mono-set-mset*: $\textit{set-mset}\ C \subseteq \textit{set-mset}\ D \Longrightarrow I \models C \Longrightarrow I \models D$
⟨*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*]: $I \models \textit{replicate-mset}\ n\ L \longleftrightarrow n \neq 0 \wedge I \models l\ L$
⟨*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 partial-interp* ⇒ *'a clause-set* ⇒ *bool* (**infix** ⊨*s 50*) **where**
I ⊨*s* $CC \longleftrightarrow (\forall\, C \in CC.\ I \models C)$

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

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

lemma *true-clss-empty-entails-empty*[*iff*]: $\{\}$ ⊨*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 ⊨*s* $CC \cup DD \longleftrightarrow I$ ⊨*s* $CC \wedge I$ ⊨*s* DD
⟨*proof*⟩

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

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

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

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

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

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

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

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

shows $I \models s\ L$
⟨*proof*⟩

lemma *true-cls-def-set-mset-eq*:
‹$set\text{-}mset\ A = set\text{-}mset\ B \Longrightarrow I \models A \longleftrightarrow I \models B$›
⟨*proof*⟩

lemma *true-cls-add-mset-strict*: ‹$I \models add\text{-}mset\ L\ C \longleftrightarrow L \in I \lor I \models (removeAll\text{-}mset\ L\ C)$›
⟨*proof*⟩

Satisfiability

definition *satisfiable* :: $'a\ clause\ set \Rightarrow bool$ **where**
$satisfiable\ CC \longleftrightarrow (\exists I.\ (I \models s\ CC \land consistent\text{-}interp\ I \land total\text{-}over\text{-}m\ I\ CC))$

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

lemma *satisfiable-empty*[*simp*]: ‹$satisfiable\ \{\}$›
⟨*proof*⟩

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

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

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

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

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

lemma *unsatisfiable-mono*:
‹$N \subseteq N' \Longrightarrow unsatisfiable\ N \Longrightarrow unsatisfiable\ N'$›
⟨*proof*⟩

Entailment for Multisets of Clauses

definition *true-cls-mset* :: $'a\ partial\text{-}interp \Rightarrow 'a\ clause\ multiset \Rightarrow bool$ (**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*]: $I \models m\ \{\#C\#\} \longleftrightarrow I \models C$

$\langle proof \rangle$

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

lemma *true-cls-mset-add-mset*[*iff*]: $I \models m\ add\text{-}mset\ C\ CC \longleftrightarrow I \models C \wedge I \models m\ CC$
$\langle proof \rangle$

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

lemma *true-cls-mset-mono*: $set\text{-}mset\ DD \subseteq set\text{-}mset\ CC \Longrightarrow I \models m\ CC \Longrightarrow I \models m\ DD$
$\langle proof \rangle$

lemma *true-clss-set-mset*[*iff*]: $I \models s\ set\text{-}mset\ CC \longleftrightarrow I \models m\ CC$
$\langle proof \rangle$

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

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

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 $\psi :: 'v\ clause\text{-}set$
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-interp* $(A \cup I)$
$\langle proof \rangle$

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

lemma *true-cls-remove-hd-if-notin-vars*:
assumes *insert a* $M' \models D$
and *atm-of a* $\notin$ *atms-of D*
shows $M' \models D$
⟨*proof*⟩

lemma *total-over-set-atm-of*:
fixes I :: *'v partial-interp* **and** K :: *'v set*
shows *total-over-set I K* $\longleftrightarrow (\forall l \in K.\ l \in$ (*atm-of* ‘ I))
⟨*proof*⟩

lemma *true-cls-mset-true-clss-iff*:
‹*finite C* $\Longrightarrow I \models m$ *mset-set C* $\longleftrightarrow I \models s\ C$›
‹$I \models m\ D \longleftrightarrow I \models s$ *set-mset D*›
⟨*proof*⟩

Tautologies

We define tautologies as clause entailed by every total model and show later that is equivalent to containing a literal and its negation.

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

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-union-add-iff*[*simp*]:
‹*tautology* ($A \cup\# B$) $\longleftrightarrow$ *tautology* ($A + B$)›
⟨*proof*⟩
lemma *tautology-add-mset-union-add-iff*[*simp*]:
‹*tautology* (*add-mset L* ($A \cup\# B$)) $\longleftrightarrow$ *tautology* (*add-mset L* ($A + B$))›
⟨*proof*⟩

lemma *not-tautology-minus*:

$\langle \neg tautology\ A \Longrightarrow \neg tautology\ (A - B) \rangle$
$\langle proof \rangle$

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

lemma *tautology-add-mset*:
$tautology\ (add\text{-}mset\ a\ L) \longleftrightarrow tautology\ L \vee -a \in\#\ L$
$\langle proof \rangle$

lemma *tautology-single*[*simp*]: $\langle \neg tautology\ \{\#L\#\} \rangle$
$\langle proof \rangle$

lemma *tautology-union*:
$\langle tautology\ (A + B) \longleftrightarrow tautology\ A \vee tautology\ B \vee (\exists\, a.\ a \in\#\ A \wedge -a \in\#\ B) \rangle$
$\langle proof \rangle$

lemma
tautology-poss[*simp*]: $\langle \neg tautology\ (poss\ A) \rangle$ **and**
tautology-negs[*simp*]: $\langle \neg tautology\ (negs\ A) \rangle$
$\langle proof \rangle$

lemma *tautology-uminus*[*simp*]:
$\langle tautology\ (uminus\ `\#\ w) \longleftrightarrow tautology\ w \rangle$
$\langle proof \rangle$

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

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

lemma *tautology-imp-tautology*:
fixes $\chi\ \chi' :: 'v\ clause$
assumes $\forall\, I.\ total\text{-}over\text{-}m\ I\ \{\chi\} \longrightarrow I \models \chi \longrightarrow I \models \chi'$ **and** $tautology\ \chi$
shows $tautology\ \chi'$ $\langle proof \rangle$

lemma *not-tautology-mono*: $\langle D' \subseteq\#\ D \Longrightarrow \neg tautology\ D \Longrightarrow \neg tautology\ D' \rangle$
$\langle proof \rangle$

lemma *tautology-decomp'*:
$\langle tautology\ C \longleftrightarrow (\exists\, L.\ L \in\#\ C \wedge -\ L \in\#\ C) \rangle$
$\langle proof \rangle$

lemma *consistent-interp-tautology*:
$\langle consistent\text{-}interp\ (set\ M') \longleftrightarrow \neg tautology\ (mset\ M') \rangle$
$\langle proof \rangle$

lemma *consistent-interp-tuatology-mset-set*:
$\langle finite\ x \Longrightarrow consistent\text{-}interp\ x \longleftrightarrow \neg tautology\ (mset\text{-}set\ x) \rangle$
$\langle proof \rangle$

lemma *tautology-distinct-atm-iff*:
$\langle$*distinct-mset* $C \Longrightarrow$ *tautology* $C \longleftrightarrow \neg$*distinct-mset* (*atm-of* ‘# C)$\rangle$
$\langle$*proof*$\rangle$

lemma *not-tautology-minusD*:
$\langle$*tautology* $(A - B) \Longrightarrow$ *tautology* $A\rangle$
$\langle$*proof*$\rangle$

Entailment for clauses and propositions

We also need entailment of clauses by other clauses.

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

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

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

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

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

lemma *true-clss-cls-empty-empty*[*iff*]:
$\langle\{\} \models p\ \{\#\} \longleftrightarrow \textit{False}\rangle$
$\langle$*proof*$\rangle$

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

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

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

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

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

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

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

This version of $[\![?A \subseteq ?B;\ ?A \models p\ ?CC]\!] \Longrightarrow ?B \models p\ ?CC$ is useful as intro rule.

lemma (**in** $-$)*true-clss-cls-subsetI*: ‹$I \models p\ A \Longrightarrow I \subseteq I' \Longrightarrow I' \models p\ A$›
⟨*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-cls-mono-add-mset*[*simp*]:
$A \models p\ CC \Longrightarrow A \models p\ add\text{-}mset\ L\ CC$
⟨*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*]:
$CC \in A \Longrightarrow A \models p\ CC$
⟨*proof*⟩

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

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

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

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

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

Better suited as intro rule:

lemma *true-clss-clss-subsetI*:
$A \models ps\ CC \Longrightarrow A \subseteq B \Longrightarrow B \models ps\ CC$
⟨*proof*⟩

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

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

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

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

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

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

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

lemma *true-clss-cls-or-true-clss-cls-or-not-true-clss-cls-or*:
assumes D: $N \models p$ *add-mset* $(-L)\ D$
and C: $N \models p$ *add-mset* $L\ C$
shows $N \models p\ D + C$
⟨*proof*⟩

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

lemma *true-clss-cls-sup-iff-add*: $N \models p\ C \cup\#\ D \longleftrightarrow N \models p\ C + D$
⟨*proof*⟩

lemma *true-clss-cls-union-mset-true-clss-cls-or-not-true-clss-cls-or*:
assumes
D: $N \models p$ *add-mset* $(-L)\ D$ **and**
C: $N \models p$ *add-mset* $L\ C$

shows $N \models p\ D \cup\#\ C$
⟨*proof*⟩

lemma *true-clss-cls-tautology-iff*:
‹$\{\} \models p\ a \longleftrightarrow$ *tautology* a› (**is** ‹$?A \longleftrightarrow ?B$›)
⟨*proof*⟩

lemma *true-cls-mset-empty-iff* [*simp*]: ‹$\{\} \models m\ C \longleftrightarrow C = \{\#\}$›
⟨*proof*⟩

lemma *true-clss-mono-left*:
‹$I \models s\ A \Longrightarrow I \subseteq J \Longrightarrow J \models s\ A$›
⟨*proof*⟩

lemma *true-cls-remove-alien*:
‹$I \models N \longleftrightarrow \{L.\ L \in I \wedge$ *atm-of* $L \in$ *atms-of* $N\} \models N$›
⟨*proof*⟩

lemma *true-clss-remove-alien*:
‹$I \models s\ N \longleftrightarrow \{L.\ L \in I \wedge$ *atm-of* $L \in$ *atms-of-ms* $N\} \models s\ N$›
⟨*proof*⟩

lemma *true-clss-alt-def*:
‹$N \models p\ \chi \longleftrightarrow$
($\forall I.$ *atms-of-s* $I =$ *atms-of-ms* $(N \cup \{\chi\}) \longrightarrow$ *consistent-interp* $I \longrightarrow I \models s\ N \longrightarrow I \models \chi)$›
⟨*proof*⟩

lemma *true-clss-alt-def2*:
assumes ‹$\neg$*tautology* χ›
shows ‹$N \models p\ \chi \longleftrightarrow (\forall I.$ *atms-of-s* $I =$ *atms-of-ms* $N \longrightarrow$ *consistent-interp* $I \longrightarrow I \models s\ N \longrightarrow I \models \chi)$› (**is** ‹$?A \longleftrightarrow ?B$›)
⟨*proof*⟩

lemma *true-clss-restrict-iff*:
assumes ‹$\neg$*tautology* χ›
shows ‹$N \models p\ \chi \longleftrightarrow N \models p\ \{\#L \in\#\ \chi.$ *atm-of* $L \in$ *atms-of-ms* $N\#\}$› (**is** ‹$?A \longleftrightarrow ?B$›)
⟨*proof*⟩

This is a slightly restrictive theorem, that encompasses most useful cases. The assumption $\neg$ *tautology* C can be removed if the model I is total over the clause.

lemma *true-clss-cls-true-clss-true-cls*:
assumes ‹$N \models p\ C$›
‹$I \models s\ N$› **and**
cons: ‹*consistent-interp* I› **and**
tauto: ‹$\neg$*tautology* C›
shows ‹$I \models C$›
⟨*proof*⟩

1.1.4 Subsumptions

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

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

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

1.1.5 Removing Duplicates

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

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

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

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

1.1.6 Set of all Simple Clauses

A simple clause with respect to a set of atoms is such that

1. its atoms are included in the considered set of atoms;
2. it is not a tautology;
3. it does not contains duplicate literals.

   It corresponds to the clauses that cannot be simplified away in a calculus without considering the other clauses.

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

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

lemma *simple-clss-insert*:
assumes $l \notin atms$
shows $simple\text{-}clss\ (insert\ l\ atms) =$
$((+)\ \{\#Pos\ l\#\})$ ‘ $(simple\text{-}clss\ atms)$
$\cup ((+)\ \{\#Neg\ l\#\})$ ‘ $(simple\text{-}clss\ atms)$
$\cup\ simple\text{-}clss\ atms$(**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*⟩

lemma *simple-clss-element-mono*:
‹$x \in$ *simple-clss A* $\Longrightarrow y \subseteq\# x \Longrightarrow y \in$ *simple-clss A*›
⟨*proof*⟩

1.1.7 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* $\vee$ $I \models$*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*]: $I \models es\ CC \cup DD \longleftrightarrow I \models es\ CC \wedge I \models es\ DD$
⟨*proof*⟩

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

1.1.8 Entailment to be extended

In some cases we want a more general version of entailment to have for example $\{\} \models \{\#L, -L\#\}$. This is useful when the model we are building might not be total (the literal L might have been definitely removed from the set of clauses), but we still want to have a property of entailment considering that theses removed literals are not important.

We can given a model I consider all the natural extensions: C is entailed by an extended I, if for all total extension of I, this model entails C.

definition *true-clss-ext* :: *'a literal set* ⇒ *'a clause set* ⇒ *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*⟩

lemma *inj-on-Pos*: ‹*inj-on Pos A*› **and**
inj-on-Neg: ‹*inj-on Neg A*›
⟨*proof*⟩

lemma *inj-on-uminus-lit*: ‹*inj-on uminus A*› **for** A :: ‹*'a literal set*›
⟨*proof*⟩

end

1.2 Partial Annotated Herbrand Interpretation

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

theory *Partial-Annotated-Herbrand-Interpretation*
imports
Partial-Herbrand-Interpretation
begin

1.2.1 Decided Literals

Definition

datatype (*'v*, *'w*, *'mark*) *annotated-lit* =
is-decided: *Decided* (*lit-dec*: *'v*) |
is-proped: *Propagated* (*lit-prop*: *'w*) (*mark-of*: *'mark*)

type-synonym (*'v*, *'w*, *'mark*) *annotated-lits* = ‹(*'v*, *'w*, *'mark*) *annotated-lit list*›
type-synonym (*'v*, *'mark*) *ann-lit* = ‹(*'v literal*, *'v literal*, *'mark*) *annotated-lit*›

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

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

lemma *is-propedE*: ‹$is\text{-}proped\ L \Longrightarrow (\bigwedge K\ C.\ L = Propagated\ K\ C \Longrightarrow P) \Longrightarrow P$›
⟨*proof*⟩

lemma *is-decided-no-proped-iff*: ‹$is\text{-}decided\ L \longleftrightarrow \neg is\text{-}proped\ L$›
⟨*proof*⟩

lemma *not-is-decidedE*:
‹$\neg is\text{-}decided\ E \Longrightarrow (\bigwedge K\ C.\ E = Propagated\ K\ C \Longrightarrow thesis) \Longrightarrow thesis$›
⟨*proof*⟩

type-synonym $('v, 'm)\ ann\text{-}lits$ = ‹$('v, 'm)\ ann\text{-}lit\ list$›

fun *lit-of* :: ‹$('a, 'a, 'mark)\ annotated\text{-}lit \Rightarrow 'a$› **where**
‹$lit\text{-}of\ (Decided\ L) = L$› |
‹$lit\text{-}of\ (Propagated\ L\ \text{-}) = L$›

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

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

lemma *lits-of-l-empty*[*simp*]:
‹$lits\text{-}of\ \{\} = \{\}$›
⟨*proof*⟩

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

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

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

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

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

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

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

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

lemma *in-unmark-l-in-lits-of-l-iff*: ‹$\{\#L\#\} \in$ *unmark-l* $M \longleftrightarrow L \in$ *lits-of-l* M›
⟨*proof*⟩

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

Entailment

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

definition *true-annots* :: ‹$('a, 'm)$ *ann-lits* $\Rightarrow$ $'a$ *clause-set* $\Rightarrow$ *bool*› (**infix** $\models as$ *49*) **where**
‹$I \models as\ CC \longleftrightarrow (\forall C \in CC.\ I \models a\ C)$›

lemma *true-annot-empty-model*[*simp*]:
‹$\neg[] \models a\ \psi$›
⟨*proof*⟩

lemma *true-annot-empty*[*simp*]:
‹$\neg I \models a\ \{\#\}$›
⟨*proof*⟩

lemma *empty-true-annots-def*[*iff*]:
‹$[] \models as\ \psi \longleftrightarrow \psi = \{\}$›
⟨*proof*⟩

lemma *true-annots-empty*[*simp*]:
‹$I \models as\ \{\}$›
⟨*proof*⟩

lemma *true-annots-single-true-annot*[*iff*]:
‹$I \models as\ \{C\} \longleftrightarrow I \models a\ C$›
⟨*proof*⟩

lemma *true-annot-insert-l*[*simp*]:
‹$M \models a\ A \Longrightarrow L\ \#\ M \models a\ A$›
⟨*proof*⟩

lemma *true-annots-insert-l* [*simp*]:
‹$M \models as\ A \Longrightarrow L\ \#\ M \models as\ A$›
⟨*proof*⟩

lemma *true-annots-union*[*iff*]:
‹$M \models as\ A \cup B \longleftrightarrow (M \models as\ A \wedge M \models as\ B)$›
⟨*proof*⟩

lemma *true-annots-insert*[*iff*]:
‹$M \models as$ *insert a A* $\longleftrightarrow (M \models a\ a \wedge M \models as\ A)$›
⟨*proof*⟩

lemma *true-annot-append-l*:
‹$M \models a\ A \Longrightarrow M'\ @\ M \models a\ A$›
⟨*proof*⟩

lemma *true-annots-append-l*:

‹$M \models as\ A \Longrightarrow M' @ M \models as\ A$›
⟨*proof*⟩

Link between $\models as$ and $\models s$:

lemma *true-annots-true-cls*:
‹$I \models as\ CC \longleftrightarrow lits\text{-}of\text{-}l\ I \models s\ CC$›
⟨*proof*⟩

lemma *in-lit-of-true-annot*:
‹$a \in lits\text{-}of\text{-}l\ M \longleftrightarrow M \models a\ \{\#a\#\}$›
⟨*proof*⟩

lemma *true-annot-lit-of-notin-skip*:
‹$L\ \#\ M \models a\ A \Longrightarrow lit\text{-}of\ L \notin\#\ A \Longrightarrow M \models a\ A$›
⟨*proof*⟩

lemma *true-clss-singleton-lit-of-implies-incl*:
‹$I \models s\ unmark\text{-}l\ MLs \Longrightarrow lits\text{-}of\text{-}l\ MLs \subseteq I$›
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

Defined and Undefined Literals

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

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

definition *defined-lit* :: ‹('*a literal*, '*a literal*, '*m*) *annotated-lits* ⇒ '*a literal* ⇒ *bool*›
where
‹*defined-lit I L* ⟷ (*Decided L* ∈ *set I*) ∨ (∃ *P*. *Propagated L P* ∈ *set I*)
∨ (*Decided* (−*L*) ∈ *set I*) ∨ (∃ *P*. *Propagated* (−*L*) *P* ∈ *set I*)›

abbreviation *undefined-lit* :: ‹('*a literal*, '*a literal*, '*m*) *annotated-lits* ⇒ '*a literal* ⇒ *bool*›
where ‹*undefined-lit I L* ≡ ¬*defined-lit I L*›

lemma *defined-lit-rev*[*simp*]:
‹*defined-lit* (*rev M*) *L* ⟷ *defined-lit M L*›
⟨*proof*⟩

lemma *atm-imp-decided-or-proped*:
assumes ‹*x* ∈ *set I*›
shows
‹(*Decided* (− *lit-of x*) ∈ *set I*)
∨ (*Decided* (*lit-of x*) ∈ *set I*)
∨ (∃ *l*. *Propagated* (− *lit-of x*) *l* ∈ *set I*)
∨ (∃ *l*. *Propagated* (*lit-of x*) *l* ∈ *set I*)›
⟨*proof*⟩

lemma *literal-is-lit-of-decided*:
assumes ‹*L* = *lit-of x*›
shows ‹(*x* = *Decided L*) ∨ (∃ *l'*. *x* = *Propagated L l'*)›
⟨*proof*⟩

lemma *true-annot-iff-decided-or-true-lit*:
‹*defined-lit I L* ⟷ (*lits-of-l I* ⊨*l L* ∨ *lits-of-l I* ⊨*l* −*L*)›
⟨*proof*⟩

lemma *consistent-inter-true-annots-satisfiable*:
‹*consistent-interp* (*lits-of-l I*) ⟹ *I* ⊨*as N* ⟹ *satisfiable N*›
⟨*proof*⟩

lemma *defined-lit-map*:
‹*defined-lit Ls L* ⟷ *atm-of L* ∈ (λ*l*. *atm-of* (*lit-of l*)) ‘ *set Ls*›
⟨*proof*⟩

lemma *defined-lit-uminus*[*iff*]:
‹*defined-lit I* (−*L*) ⟷ *defined-lit I L*›
⟨*proof*⟩

lemma *Decided-Propagated-in-iff-in-lits-of-l*:
‹*defined-lit I L* ⟷ (*L* ∈ *lits-of-l I* ∨ −*L* ∈ *lits-of-l I*)›
⟨*proof*⟩

lemma *consistent-add-undefined-lit-consistent*[*simp*]:
assumes
‹*consistent-interp* (*lits-of-l Ls*)› **and**
‹*undefined-lit Ls L*›

shows ‹*consistent-interp* (*insert* L (*lits-of-l* Ls))›
⟨*proof*⟩

lemma *decided-empty*[*simp*]:
‹¬*defined-lit* [] L›
⟨*proof*⟩

lemma *undefined-lit-single*[*iff*]:
‹*defined-lit* [L] K ⟷ *atm-of* (*lit-of* L) = *atm-of* K›
⟨*proof*⟩

lemma *undefined-lit-cons*[*iff*]:
‹*undefined-lit* (L # M) K ⟷ *atm-of* (*lit-of* L) ≠ *atm-of* K ∧ *undefined-lit* M K›
⟨*proof*⟩

lemma *undefined-lit-append*[*iff*]:
‹*undefined-lit* (M @ M') K ⟷ *undefined-lit* M K ∧ *undefined-lit* M' K›
⟨*proof*⟩

lemma *defined-lit-cons*:
‹*defined-lit* (L # M) K ⟷ *atm-of* (*lit-of* L) = *atm-of* K ∨ *defined-lit* M K›
⟨*proof*⟩

lemma *defined-lit-append*:
‹*defined-lit* (M @ M') K ⟷ *defined-lit* M K ∨ *defined-lit* M' K›
⟨*proof*⟩

lemma *in-lits-of-l-defined-litD*: ‹*L-max* ∈ *lits-of-l* M ⟹ *defined-lit* M *L-max*›
⟨*proof*⟩

lemma *undefined-notin*: ‹*undefined-lit* M (*lit-of* x) ⟹ x ∉ *set* M› **for** x M
⟨*proof*⟩

lemma *uminus-lits-of-l-definedD*:
‹−x ∈ *lits-of-l* M ⟹ *defined-lit* M x›
⟨*proof*⟩

lemma *defined-lit-Neg-Pos-iff*:
‹*defined-lit* M (*Neg* A) ⟷ *defined-lit* M (*Pos* A)›
⟨*proof*⟩

lemma *defined-lit-Pos-atm-iff*[*simp*]:
‹*defined-lit* *M1* (*Pos* (*atm-of* x)) ⟷ *defined-lit* *M1* x›
⟨*proof*⟩

lemma *defined-lit-mono*:
‹*defined-lit* *M2* L ⟹ *set* *M2* ⊆ *set* *M3* ⟹ *defined-lit* *M3* L›
⟨*proof*⟩

lemma *defined-lit-nth*:
‹n < *length* *M2* ⟹ *defined-lit* *M2* (*lit-of* (*M2* ! n))›
⟨*proof*⟩

1.2.2 Backtracking

fun *backtrack-split* :: ‹($'a$, $'v$, $'m$) *annotated-lits*

⇒ ('a, 'v, 'm) *annotated-lits* × ('a, 'v, 'm) *annotated-lits*› **where**
‹*backtrack-split* [] = ([], [])› |
‹*backtrack-split* (*Propagated L P* # *mlits*) = *apfst* ((#) (*Propagated L P*)) (*backtrack-split mlits*)› |
‹*backtrack-split* (*Decided L* # *mlits*) = ([], *Decided L* # *mlits*)›

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

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

lemma *backtrack-split-list-eq*[*simp*]:
‹$fst\ (backtrack\text{-}split\ l)\ @\ (snd\ (backtrack\text{-}split\ l)) = l$›
⟨*proof*⟩

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

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

1.2.3 Decomposition with respect to the First Decided Literals

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

Definition

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

fun *get-all-ann-decomposition* :: ‹('a, 'b, 'm) *annotated-lits*
⇒ (('a, 'b, 'm) *annotated-lits* × ('a, 'b, 'm) *annotated-lits*) *list*› **where**
‹*get-all-ann-decomposition* (*Decided L* # *Ls*) =
(*Decided L* # *Ls*, []) # *get-all-ann-decomposition Ls*› |
‹*get-all-ann-decomposition* (*Propagated L P*# *Ls*) =
(*apsnd* ((#) (*Propagated L P*)) (*hd* (*get-all-ann-decomposition Ls*)))
\# *tl* (*get-all-ann-decomposition Ls*)› |
‹*get-all-ann-decomposition* [] = [([], [])]›

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

Now we can prove several simple properties about the function.

lemma *get-all-ann-decomposition-never-empty*[*iff*]:
‹*get-all-ann-decomposition M* = [] ⟷ *False*›

⟨*proof*⟩

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

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

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

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

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

lemma *get-all-ann-decomposition-length-1-fst-empty-or-length-1*:
assumes ‹$\textit{get-all-ann-decomposition } M = (a,\ b)\ \#\ []$›
shows ‹$a = [] \lor (\textit{length } a = 1 \land \textit{is-decided } (\textit{hd } a) \land \textit{hd } a \in \textit{set } M)$›
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

lemma *get-all-ann-decomposition-exists-prepend*[*dest*]:
assumes ‹(a, b) ∈ *set* (*get-all-ann-decomposition* M)›
shows ‹∃ c. M = c @ b @ a›
⟨*proof*⟩

lemma *get-all-ann-decomposition-incl*:
assumes ‹(a, b) ∈ *set* (*get-all-ann-decomposition* M)›
shows ‹*set* b ⊆ *set* M› **and** ‹*set* a ⊆ *set* M›
⟨*proof*⟩

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

lemma *union-in-get-all-ann-decomposition-is-subset*:
assumes ‹(a, b) ∈ *set* (*get-all-ann-decomposition* M)›
shows ‹*set* a ∪ *set* b ⊆ *set* M›
⟨*proof*⟩

lemma *Decided-cons-in-get-all-ann-decomposition-append-Decided-cons*:
‹∃ c''. (*Decided* K # c, c'') ∈ *set* (*get-all-ann-decomposition* (c' @ *Decided* K # c))›
⟨*proof*⟩

lemma *fst-get-all-ann-decomposition-prepend-not-decided*:
assumes ‹∀ m∈*set* MS. ¬ *is-decided* m›

shows ‹$set\ (map\ fst\ (get\text{-}all\text{-}ann\text{-}decomposition\ M))$
$= set\ (map\ fst\ (get\text{-}all\text{-}ann\text{-}decomposition\ (MS\ @\ M)))$›
⟨*proof*⟩

lemma *no-decision-get-all-ann-decomposition*:
‹$\forall\ l{\in}set\ M.\ \neg\ is\text{-}decided\ l \Longrightarrow\ get\text{-}all\text{-}ann\text{-}decomposition\ M = [([], M)]$›
⟨*proof*⟩

Entailment of the Propagated by the Decided Literal

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

definition *all-decomposition-implies* :: ‹$'a\ clause\ set$
$\Rightarrow (('a, 'm)\ ann\text{-}lits \times ('a, 'm)\ ann\text{-}lits)\ list \Rightarrow bool$› **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*]:
‹$all\text{-}decomposition\text{-}implies\ N\ ((Ls, seen)\ \#\ S')$
$\longleftrightarrow (all\text{-}decomposition\text{-}implies\ N\ [(Ls, seen)] \wedge all\text{-}decomposition\text{-}implies\ N\ S')$›
⟨*proof*⟩

lemma *all-decomposition-implies-cons-single*[*iff*]:
‹$all\text{-}decomposition\text{-}implies\ N\ (l\ \#\ S') \longleftrightarrow$
$(unmark\text{-}l\ (fst\ l) \cup N \models ps\ unmark\text{-}l\ (snd\ l) \wedge$
$all\text{-}decomposition\text{-}implies\ N\ S')$›
⟨*proof*⟩

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

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

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

lemma *all-decomposition-implies-union*:
‹$all\text{-}decomposition\text{-}implies\ N\ M \Longrightarrow all\text{-}decomposition\text{-}implies\ (N \cup N')\ M$›
⟨*proof*⟩

lemma *all-decomposition-implies-mono*:
‹$N \subseteq N' \Longrightarrow all\text{-}decomposition\text{-}implies\ N\ M \Longrightarrow all\text{-}decomposition\text{-}implies\ N'\ M$›
⟨*proof*⟩

lemma *all-decomposition-implies-mono-right*:
‹$all\text{-}decomposition\text{-}implies\ I\ (get\text{-}all\text{-}ann\text{-}decomposition\ (M' @ M)) \Longrightarrow$
$all\text{-}decomposition\text{-}implies\ I\ (get\text{-}all\text{-}ann\text{-}decomposition\ M)$›
⟨*proof*⟩

1.2.4 Negation of a Clause

We define the negation of a *'a clause*: it converts a single clause to a set of clauses, where each clause is a single literal (whose negation is in the original clause).

definition *CNot* :: ‹*'v clause* ⇒ *'v clause-set*› **where**
‹$CNot\ \psi = \{\ \{\#-L\#\}\ |\ L.\ L \in\#\ \psi\ \}$›

lemma *finite-CNot*[*simp*]: ‹$finite\ (CNot\ C)$›
⟨*proof*⟩

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

lemma
shows
CNot-add-mset[*simp*]: ‹$CNot\ (add\text{-}mset\ L\ \psi) = insert\ \{\#-L\#\}\ (CNot\ \psi)$› **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** ‹$-L \in lits\text{-}of\text{-}l\ M$›
⟨*proof*⟩

lemma *CNot-remdups-mset*[*simp*]:
‹$CNot\ (remdups\text{-}mset\ A) = CNot\ A$›
⟨*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* ∈ *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* ∈ *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-clss-def-iff-negation-in-model*:
  ‹$M \models s\ CNot\ C \longleftrightarrow (\forall l \in\#\ C.\ -l \in M)$›
  ⟨*proof*⟩

lemma *true-annots-CNot-definedD*:
  ‹$M \models as\ CNot\ C \Longrightarrow \forall L \in\#\ C.$ *defined-lit M L*›
  ⟨*proof*⟩

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

lemma *CNot-mset-replicate*[*simp*]:

‹*CNot* (*mset* (*replicate n L*)) = (*if n* = *0 then* {} *else* {{#−*L*#}})›
⟨*proof*⟩

lemma *consistent-CNot-not-tautology*:
‹*consistent-interp M* ⟹ *M* ⊨*s CNot D* ⟹ ¬*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*⟩

lemma *true-clss-cls-plus-CNot*:
assumes
CC-L: ‹*A* ⊨*p add-mset L CC*› **and**
CNot-CC: ‹*A* ⊨*ps CNot CC*›
shows ‹*A* ⊨*p* {#*L*#}›
⟨*proof*⟩

lemma *true-annots-CNot-lit-of-notin-skip*:
assumes *LM*: ‹*L* # *M* ⊨*as CNot A*› **and** *LA*: ‹*lit-of L* ∉# *A*› ‹−*lit-of L* ∉# *A*›
shows ‹*M* ⊨*as CNot A*›
⟨*proof*⟩

lemma *true-clss-clss-union-false-true-clss-clss-cnot*:
‹*A* ∪ {*B*} ⊨*ps* {{#}} ⟷ *A* ⊨*ps CNot B*›
⟨*proof*⟩

lemma *true-annot-remove-hd-if-notin-vars*:
assumes ‹*a* # *M'*⊨*a D*› **and** ‹*atm-of* (*lit-of a*) ∉ *atms-of D*›
shows ‹*M'* ⊨*a D*›
⟨*proof*⟩

lemma *true-annot-remove-if-notin-vars*:
assumes ‹*M* @ *M'*⊨*a D*› **and** ‹∀ *x*∈*atms-of D*. *x* ∉ *atm-of* ‘ *lits-of-l M*›
shows ‹*M'* ⊨*a D*›
⟨*proof*⟩

lemma *true-annots-remove-if-notin-vars*:
assumes ‹*M* @ *M'*⊨*as D*› **and** ‹∀ *x*∈*atms-of-ms D*. *x* ∉ *atm-of* ‘ *lits-of-l M*›
shows ‹*M'* ⊨*as D*› ⟨*proof*⟩

lemma *all-variables-defined-not-imply-cnot*:
assumes
‹∀ *s* ∈ *atms-of-ms* {*B*}. *s* ∈ *atm-of* ‘ *lits-of-l A*› **and**
‹¬ *A* ⊨*a B*›
shows ‹*A* ⊨*as CNot B*›
⟨*proof*⟩

lemma *CNot-union-mset*[*simp*]:
‹*CNot* (*A* ∪# *B*) = *CNot A* ∪ *CNot B*›
⟨*proof*⟩

lemma *true-clss-clss-true-clss-cls-true-clss-clss*:

assumes
⟨A ⊨*ps unmark-l M*⟩ **and** ⟨*M* ⊨*as D*⟩
shows ⟨*A* ⊨*ps D*⟩
⟨*proof*⟩

lemma *true-clss-clss-CNot-true-clss-cls-unsatisfiable*:
assumes ⟨*A* ⊨*ps CNot D*⟩ **and** ⟨*A* ⊨*p D*⟩
shows ⟨*unsatisfiable A*⟩
⟨*proof*⟩

lemma *true-clss-cls-neg*:
⟨$N \models p\ I \longleftrightarrow N \cup (\lambda L.\ \{\#-L\#\})$ ‘ *set-mset* $I \models p\ \{\#\}$⟩
⟨*proof*⟩

lemma *all-decomposition-implies-conflict-DECO-clause*:
assumes ⟨*all-decomposition-implies N* (*get-all-ann-decomposition M*)⟩ **and**
⟨*M* ⊨*as CNot C*⟩ **and**
⟨$C \in N$⟩
shows ⟨*N* ⊨*p* (*uminus o lit-of*) ‘# (*filter-mset is-decided* (*mset M*))⟩
(**is** ⟨*?I* ⊨*p ?A*⟩)
⟨*proof*⟩

1.2.5 Other

definition ⟨*no-dup L* ≡ *distinct* (*map* (λ*l*. *atm-of* (*lit-of l*)) *L*)⟩

lemma *no-dup-nil*[*simp*]:
⟨*no-dup* []⟩
⟨*proof*⟩

lemma *no-dup-cons*[*simp*]:
⟨*no-dup* (*L* # *M*) ⟷ *undefined-lit M* (*lit-of L*) ∧ *no-dup M*⟩
⟨*proof*⟩

lemma *no-dup-append-cons*[*iff*]:
⟨*no-dup* (*M* @ *L* # *M′*) ⟷ *undefined-lit* (*M* @ *M′*) (*lit-of L*) ∧ *no-dup* (*M* @ *M′*)⟩
⟨*proof*⟩

lemma *no-dup-append-append-cons*[*iff*]:
⟨*no-dup* (*M0* @ *M* @ *L* # *M′*) ⟷ *undefined-lit* (*M0* @ *M* @ *M′*) (*lit-of L*) ∧ *no-dup* (*M0* @ *M* @ *M′*)⟩
⟨*proof*⟩

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

lemma *no-dup-appendD*:
⟨*no-dup* (*a* @ *b*) ⟹ *no-dup b*⟩
⟨*proof*⟩

lemma *no-dup-appendD1*:
⟨*no-dup* (*a* @ *b*) ⟹ *no-dup a*⟩
⟨*proof*⟩

lemma *no-dup-length-eq-card-atm-of-lits-of-l*:

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

lemma *distinct-consistent-interp*:
‹*no-dup M* ⟹ *consistent-interp* (*lits-of-l M*)›
⟨*proof*⟩

lemma *same-mset-no-dup-iff*:
‹*mset M* = *mset M'* ⟹ *no-dup M* ⟷ *no-dup M'*›
⟨*proof*⟩

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

lemma *true-annots-lit-of-notin-skip*:
assumes ‹*L* # *M* ⊨*as CNot A*›
and ‹−*lit-of L* ∉# *A*›
and ‹*no-dup* (*L* # *M*)›
shows ‹*M* ⊨*as CNot A*›
⟨*proof*⟩

lemma *no-dup-imp-distinct*: ‹*no-dup M* ⟹ *distinct M*›
⟨*proof*⟩

lemma *no-dup-tlD*: ‹*no-dup a* ⟹ *no-dup* (*tl a*)›
⟨*proof*⟩

lemma *defined-lit-no-dupD*:
‹*defined-lit M1 L* ⟹ *no-dup* (*M2* @ *M1*) ⟹ *undefined-lit M2 L*›
‹*defined-lit M1 L* ⟹ *no-dup* (*M2'* @ *M2* @ *M1*) ⟹ *undefined-lit M2' L*›
‹*defined-lit M1 L* ⟹ *no-dup* (*M2'* @ *M2* @ *M1*) ⟹ *undefined-lit M2 L*›
⟨*proof*⟩

lemma *no-dup-consistentD*:
‹*no-dup M* ⟹ *L* ∈ *lits-of-l M* ⟹ −*L* ∉ *lits-of-l M*›
⟨*proof*⟩

lemma *no-dup-not-tautology*: ‹*no-dup M* ⟹ ¬*tautology* (*image-mset lit-of* (*mset M*))›
⟨*proof*⟩

lemma *no-dup-distinct*: ‹*no-dup M* ⟹ *distinct-mset* (*image-mset lit-of* (*mset M*))›
⟨*proof*⟩

lemma *no-dup-not-tautology-uminus*: ‹*no-dup M* ⟹ ¬*tautology* {#−*lit-of L*. *L* ∈# *mset M*#}›
⟨*proof*⟩

lemma *no-dup-distinct-uminus*: ‹*no-dup M* ⟹ *distinct-mset* {#−*lit-of L*. *L* ∈# *mset M*#}›
⟨*proof*⟩

lemma *no-dup-map-lit-of*: ‹*no-dup M* ⟹ *distinct* (*map lit-of M*)›
⟨*proof*⟩

lemma *no-dup-alt-def*:
‹*no-dup M* $\longleftrightarrow$ *distinct-mset* {#*atm-of* (*lit-of x*). *x* ∈# *mset M*#}›
⟨*proof*⟩

lemma *no-dup-append-in-atm-notin*:
assumes ‹*no-dup* (*M* @ *M'*)› **and** ‹*L* ∈ *lits-of-l M'*›
shows ‹*undefined-lit M L*›
⟨*proof*⟩

lemma *no-dup-uminus-append-in-atm-notin*:
assumes ‹*no-dup* (*M* @ *M'*)› **and** ‹−*L* ∈ *lits-of-l M'*›
shows ‹*undefined-lit M L*›
⟨*proof*⟩

1.2.6 Extending Entailments to multisets

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

abbreviation *true-annots-mset* (**infix** ⊨*asm 50*) **where**
‹*I* ⊨*asm C* ≡ *I* ⊨*as* (*set-mset C*)›

abbreviation *true-clss-clss-m* :: ‹*'v clause multiset* ⇒ *'v clause multiset* ⇒ *bool*› (**infix** ⊨*psm 50*) **where**
‹*I* ⊨*psm C* ≡ *set-mset I* ⊨*ps* (*set-mset C*)›

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

lemma *true-clss-clssm-subsetE*: ‹*N* ⊨*psm B* ⟹ *A* ⊆# *B* ⟹ *N* ⊨*psm A*›
⟨*proof*⟩

abbreviation *true-clss-cls-m*:: ‹*'a clause multiset* ⇒ *'a clause* ⇒ *bool*› (**infix** ⊨*pm 50*) **where**
‹*I* ⊨*pm C* ≡ *set-mset I* ⊨*p C*›

abbreviation *distinct-mset-mset* :: ‹*'a multiset multiset* ⇒ *bool*› **where**
‹*distinct-mset-mset* Σ ≡ *distinct-mset-set* (*set-mset* Σ)›

abbreviation *all-decomposition-implies-m* **where**
‹*all-decomposition-implies-m A B* ≡ *all-decomposition-implies* (*set-mset A*) *B*›

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

Other definition using ⋃#

lemma *atms-of-mm-alt-def*: ‹*atms-of-mm U* = *set-mset* (⋃# (*image-mset* (*image-mset atm-of*) *U*))›
⟨*proof*⟩

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

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

lemma *true-clss-cls-cong-set-mset*:
‹*N* ⊨*pm D* ⟹ *set-mset D* = *set-mset D'* ⟹ *N* ⊨*pm D'*›
⟨*proof*⟩

1.2.7 More Lemmas

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

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

lemma *true-cls-mset-restrict*:
‹$\{L \in I.\ atm\text{-}of\ L \in atms\text{-}of\text{-}mm\ N\} \models m\ N \longleftrightarrow I \models m\ N$›
⟨*proof*⟩

lemma *true-clss-restrict*:
‹$\{L \in I.\ atm\text{-}of\ L \in atms\text{-}of\text{-}mm\ N\} \models sm\ N \longleftrightarrow I \models sm\ N$›
⟨*proof*⟩

lemma *total-over-m-atms-incl*:
assumes ‹$total\text{-}over\text{-}m\ M\ (set\text{-}mset\ N)$›
shows
‹$x \in atms\text{-}of\text{-}mm\ N \Longrightarrow x \in atms\text{-}of\text{-}s\ M$›
⟨*proof*⟩

lemma *true-clss-restrict-iff*:
assumes ‹$\neg tautology\ \chi$›
shows ‹$N \models p\ \chi \longleftrightarrow N \models p\ \{\#L \in\#\ \chi.\ atm\text{-}of\ L \in atms\text{-}of\text{-}ms\ N\#\}$› (**is** ‹$?A \longleftrightarrow\ ?B$›)
⟨*proof*⟩

1.2.8 Negation of annotated clauses

definition *negate-ann-lits* :: ‹$('v,\ 'v\ clause)\ ann\text{-}lits \Rightarrow\ 'v\ literal\ multiset$› **where**
‹$negate\text{-}ann\text{-}lits\ M = (\lambda L.\ -\ lit\text{-}of\ L)\ `\#\ mset\ M$›

lemma *negate-ann-lits-empty*[*simp*]: ‹$negate\text{-}ann\text{-}lits\ [] = \{\#\}$›
⟨*proof*⟩

lemma *entails-CNot-negate-ann-lits*:
‹$M \models as\ CNot\ D \longleftrightarrow set\text{-}mset\ D \subseteq set\text{-}mset\ (negate\text{-}ann\text{-}lits\ M)$›
⟨*proof*⟩

Pointwise negation of a clause:

definition *pNeg* :: ‹$'v\ clause \Rightarrow\ 'v\ clause$› **where**
‹$pNeg\ C = \{\#-D.\ D \in\#\ C\#\}$›

lemma *pNeg-simps*:
‹$pNeg\ (add\text{-}mset\ A\ C) = add\text{-}mset\ (-A)\ (pNeg\ C)$›
‹$pNeg\ (C + D) = pNeg\ C + pNeg\ D$›
⟨*proof*⟩

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

lemma *negate-ann-lits-pNeg-lit-of*: ‹$negate\text{-}ann\text{-}lits = pNeg\ o\ image\text{-}mset\ lit\text{-}of\ o\ mset$›
⟨*proof*⟩

lemma *negate-ann-lits-empty-iff*: ‹$negate\text{-}ann\text{-}lits\ M \neq \{\#\} \longleftrightarrow M \neq []$›
⟨*proof*⟩

lemma *atms-of-negate-ann-lits*[*simp*]: ‹$atms\text{-}of\ (negate\text{-}ann\text{-}lits\ M) = atm\text{-}of\ `\ (lits\text{-}of\text{-}l\ M)$›
⟨*proof*⟩

lemma *tautology-pNeg*[*simp*]:
‹$tautology\ (pNeg\ C) \longleftrightarrow tautology\ C$›
⟨*proof*⟩

lemma *pNeg-convolution*[*simp*]:
‹$pNeg\ (pNeg\ C) = C$›
⟨*proof*⟩

lemma *pNeg-minus*[*simp*]: ‹$pNeg\ (A - B) = pNeg\ A - pNeg\ B$›
⟨*proof*⟩

lemma *pNeg-empty*[*simp*]: ‹$pNeg\ \{\#\} = \{\#\}$›
⟨*proof*⟩

lemma *pNeg-replicate-mset*[*simp*]: ‹$pNeg\ (replicate\text{-}mset\ n\ L) = replicate\text{-}mset\ n\ (-L)$›
⟨*proof*⟩

lemma *distinct-mset-pNeg-iff*[*iff*]: ‹$distinct\text{-}mset\ (pNeg\ x) \longleftrightarrow distinct\text{-}mset\ x$›
⟨*proof*⟩

lemma *pNeg-simple-clss-iff*[*simp*]:
‹$pNeg\ M \in simple\text{-}clss\ N \longleftrightarrow M \in simple\text{-}clss\ N$›
⟨*proof*⟩

lemma *atms-of-ms-pNeg*[*simp*]: ‹$atms\text{-}of\text{-}ms\ (pNeg\ `\ N) = atms\text{-}of\text{-}ms\ N$›
⟨*proof*⟩

definition *DECO-clause* :: ‹$('v, 'a)\ ann\text{-}lits \Rightarrow 'v\ clause$› **where**
‹$DECO\text{-}clause\ M = (uminus\ o\ lit\text{-}of)\ `\#\ (filter\text{-}mset\ is\text{-}decided\ (mset\ M))$›

lemma
DECO-clause-cons-Decide[*simp*]:
‹$DECO\text{-}clause\ (Decided\ L\ \#\ M) = add\text{-}mset\ (-L)\ (DECO\text{-}clause\ M)$› **and**
DECO-clause-cons-Proped[*simp*]:
‹$DECO\text{-}clause\ (Propagated\ L\ C\ \#\ M) = DECO\text{-}clause\ M$›
⟨*proof*⟩

lemma *no-dup-distinct-mset*[*intro!*]:
assumes *n-d*: ‹$no\text{-}dup\ M$›
shows ‹$distinct\text{-}mset\ (negate\text{-}ann\text{-}lits\ M)$›
⟨*proof*⟩

lemma *in-negate-trial-iff*: ‹$L \in\#\ negate\text{-}ann\text{-}lits\ M \longleftrightarrow -\ L \in lits\text{-}of\text{-}l\ M$›
⟨*proof*⟩

lemma *negate-ann-lits-cons*[*simp*]:
‹$negate\text{-}ann\text{-}lits\ (L\ \#\ M) = add\text{-}mset\ (-\ lit\text{-}of\ L)\ (negate\text{-}ann\text{-}lits\ M)$›
⟨*proof*⟩

lemma *uminus-simple-clss-iff* [*simp*]:
‹$uminus$ ‘# $M \in simple\text{-}clss\ N \longleftrightarrow M \in simple\text{-}clss\ N$›
⟨*proof*⟩

lemma *pNeg-mono*: ‹$C \subseteq\# C' \Longrightarrow pNeg\ C \subseteq\# pNeg\ C'$›
⟨*proof*⟩

end
theory *Partial-And-Total-Herbrand-Interpretation*
imports *Partial-Herbrand-Interpretation*
Ordered-Resolution-Prover.Herbrand-Interpretation
begin

1.3 Bridging of total and partial Herbrand interpretation

This theory has mostly be written as a sanity check between the two entailment notion.

definition *partial-model-of* :: ‹$'a\ interp \Rightarrow 'a\ partial\text{-}interp$› **where**
‹$partial\text{-}model\text{-}of\ I = Pos$ ‘ $I \cup Neg$ ‘ $\{x.\ x \notin I\}$›

definition *total-model-of* :: ‹$'a\ partial\text{-}interp \Rightarrow 'a\ interp$› **where**
‹$total\text{-}model\text{-}of\ I = \{x.\ Pos\ x \in I\}$›

lemma *total-over-set-partial-model-of*:
‹$total\text{-}over\text{-}set\ (partial\text{-}model\text{-}of\ I)\ UNIV$›
⟨*proof*⟩

lemma *consistent-interp-partial-model-of*:
‹$consistent\text{-}interp\ (partial\text{-}model\text{-}of\ I)$›
⟨*proof*⟩

lemma *consistent-interp-alt-def*:
‹$consistent\text{-}interp\ I \longleftrightarrow (\forall L.\ \neg(Pos\ L \in I \land Neg\ L \in I))$›
⟨*proof*⟩

context
fixes I :: ‹$'a\ partial\text{-}interp$›
assumes *cons*: ‹$consistent\text{-}interp\ I$›
begin

lemma *partial-implies-total-true-cls-total-model-of*:
assumes ‹$Partial\text{-}Herbrand\text{-}Interpretation.true\text{-}cls\ I\ C$›
shows ‹$Herbrand\text{-}Interpretation.true\text{-}cls\ (total\text{-}model\text{-}of\ I)\ C$›
⟨*proof*⟩

lemma *total-implies-partial-true-cls-total-model-of*:
assumes ‹$Herbrand\text{-}Interpretation.true\text{-}cls\ (total\text{-}model\text{-}of\ I)\ C$› **and**
‹$total\text{-}over\text{-}set\ I\ (atms\text{-}of\ C)$›
shows ‹$Partial\text{-}Herbrand\text{-}Interpretation.true\text{-}cls\ I\ C$›
⟨*proof*⟩

lemma *partial-implies-total-true-clss-total-model-of*:
assumes ‹$Partial\text{-}Herbrand\text{-}Interpretation.true\text{-}clss\ I\ C$›

shows ‹*Herbrand-Interpretation.true-clss* (*total-model-of* I) C›
⟨*proof*⟩

lemma *total-implies-partial-true-clss-total-model-of*:
assumes ‹*Herbrand-Interpretation.true-clss* (*total-model-of* I) C› **and**
‹*total-over-m* I C›
shows ‹*Partial-Herbrand-Interpretation.true-clss* I C›
⟨*proof*⟩

end

lemma *total-implies-partial-true-cls-partial-model-of*:
assumes ‹*Herbrand-Interpretation.true-cls* I C›
shows ‹*Partial-Herbrand-Interpretation.true-cls* (*partial-model-of* I) C›
⟨*proof*⟩

lemma *total-implies-partial-true-clss-partial-model-of*:
assumes ‹*Herbrand-Interpretation.true-clss* I C›
shows ‹*Partial-Herbrand-Interpretation.true-clss* (*partial-model-of* I) C›
⟨*proof*⟩

lemma *partial-total-satisfiable-iff*:
‹*Partial-Herbrand-Interpretation.satisfiable* $N \longleftrightarrow$ *Herbrand-Interpretation.satisfiable* N›
⟨*proof*⟩

end
theory *Prop-Logic*
imports *Main*
begin

Chapter 2

Normalisation

We define here the normalisation from formula towards conjunctive and disjunctive normal form, including normalisation towards multiset of multisets to represent CNF.

2.1 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.

2.1.1 Definition and Abstraction

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

datatype *'v propo* =
  FT | *FF* | *FVar 'v* | *FNot 'v propo* | *FAnd 'v propo 'v propo* | *FOr 'v propo 'v propo*
  | *FImp 'v propo 'v propo* | *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* | *CF* | *CVar 'v* | *CNot* | *CAnd* | *COr* | *CImp* | *CEq*

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

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*: $\bigwedge\varphi\ x.\ \varphi = FF \vee \varphi = FT \vee \varphi = FVar\ x \Longrightarrow P\ \varphi$
  and *unary*: $\bigwedge\psi.\ P\ \psi \Longrightarrow P\ (FNot\ \psi)$
  and *binary*: $\bigwedge\varphi\ \psi 1\ \psi 2.\ P\ \psi 1 \Longrightarrow P\ \psi 2 \Longrightarrow \varphi = FAnd\ \psi 1\ \psi 2 \vee \varphi = FOr\ \psi 1\ \psi 2 \vee \varphi = FImp\ \psi 1$
$\psi 2$
    $\vee\ \varphi = FEq\ \psi 1\ \psi 2 \Longrightarrow P\ \varphi$
  shows $P\ \psi$
  ⟨*proof*⟩

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* |
conn CF [] = *FF* |
conn (*CVar v*) [] = *FVar v* |
conn CNot [φ] = *FNot* φ |
conn CAnd (φ # [ψ]) = *FAnd* φ ψ |
conn COr (φ # [ψ]) = *FOr* φ ψ |
conn CImp (φ # [ψ]) = *FImp* φ ψ |
conn CEq (φ # [ψ]) = *FEq* φ ψ |
conn - - = *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*: $\bigwedge x.\ c = CT \lor c = CF \lor c = CVar\ x \Longrightarrow P$
  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* $\Rightarrow$ $'v$ *propo list* $\Rightarrow$ *bool* **for** c :: $'v$ *connective* **where**
wf-conn-nullary[*simp*]: $(c = CT \lor c = CF \lor c = CVar\ v) \Longrightarrow$ *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**
    $\bigwedge \psi.\ c = CNot \Longrightarrow P$ [ψ] **and**
    $\bigwedge \psi\ \psi'.\ c = COr \Longrightarrow P$ [ψ, ψ'] **and**
    $\bigwedge \psi\ \psi'.\ c = CAnd \Longrightarrow P$ [ψ, ψ'] **and**
    $\bigwedge \psi\ \psi'.\ c = CImp \Longrightarrow P$ [ψ, ψ'] **and**
    $\bigwedge \psi\ \psi'.\ c = CEq \Longrightarrow P$ [ψ, ψ']
  shows $P\ x$
  ⟨*proof*⟩

2.1.2 Properties of the Abstraction

First we can define simplification rules.

lemma *wf-conn-conn*[*simp*]:

$wf\text{-}conn\ CT\ l \Longrightarrow conn\ CT\ l = FT$
$wf\text{-}conn\ CF\ l \Longrightarrow conn\ CF\ l = FF$
$wf\text{-}conn\ (CVar\ x)\ l \Longrightarrow conn\ (CVar\ x)\ l = FVar\ x$
⟨*proof*⟩

lemma *wf-conn-list-decomp*[*simp*]:
$wf\text{-}conn\ CT\ l \longleftrightarrow l = []$
$wf\text{-}conn\ CF\ l \longleftrightarrow l = []$
$wf\text{-}conn\ (CVar\ x)\ l \longleftrightarrow l = []$
$wf\text{-}conn\ CNot\ (\xi\ @\ \varphi\ \#\ \xi') \longleftrightarrow \xi = [] \wedge \xi' = []$
⟨*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-conn c l*
and *conn*: *conn c l* = *FNot* ψ
shows c = *CNot* **and** $l = [\psi]$
⟨*proof*⟩

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

2.1.3 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* ⇒ *'v propo* ⇒ *bool* (**infix** $\preceq$ *45*) **for** φ **where**
subformula-refl[*simp*]: $\varphi \preceq \varphi$ |
subformula-into-subformula: $\psi \in$ *set l* $\Longrightarrow$ *wf-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-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 φ ψ :: *'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-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-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*)
$\varphi \preceq FOr\ \psi\ \psi' \longleftrightarrow (\varphi = FOr\ \psi\ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi')$
$\varphi \preceq FEq\ \psi\ \psi' \longleftrightarrow (\varphi = FEq\ \psi\ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi')$
$\varphi \preceq FImp\ \psi\ \psi' \longleftrightarrow (\varphi = FImp\ \psi\ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi')$
⟨*proof*⟩

lemma *wf-conn-helper-facts*[*iff*]:
wf-conn CNot $[\varphi]$
wf-conn CT []
wf-conn CF []
wf-conn (*CVar* x) []
wf-conn CAnd $[\varphi, \psi]$
wf-conn COr $[\varphi, \psi]$
wf-conn CImp $[\varphi, \psi]$
wf-conn CEq $[\varphi, \psi]$
⟨*proof*⟩

lemma *exists-c-conn*: $\exists\ c\ l.\ \varphi = conn\ c\ l \wedge wf\text{-}conn\ c\ l$
⟨*proof*⟩

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

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

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 $(FNot\ \varphi)$ = *vars-of-prop* φ |
vars-of-prop $(FAnd\ \varphi\ \psi)$ = *vars-of-prop* $\varphi \cup$ *vars-of-prop* ψ |
vars-of-prop $(FOr\ \varphi\ \psi)$ = *vars-of-prop* $\varphi \cup$ *vars-of-prop* ψ |
vars-of-prop $(FImp\ \varphi\ \psi)$ = *vars-of-prop* $\varphi \cup$ *vars-of-prop* ψ |
vars-of-prop $(FEq\ \varphi\ \psi)$ = *vars-of-prop* $\varphi \cup$ *vars-of-prop* ψ

lemma *vars-of-prop-incl-conn*:
fixes $\xi\ \xi'$ :: *'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* $\psi \subseteq$ *vars-of-prop* (*conn c l*)
⟨*proof*⟩

The set of variables is compatible with the subformula order.

lemma *subformula-vars-of-prop*:
$\varphi \preceq \psi \Longrightarrow$ *vars-of-prop* $\varphi \subseteq$ *vars-of-prop* ψ
⟨*proof*⟩

2.1.4 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* ⇒ *sign list set* **where**
pos FF = {[]} |
pos FT = {[]} |
pos (*FVar x*) = {[]} |
pos (*FAnd* φ ψ) = {[]} ∪ { *L* # *p* | *p*. *p*∈ *pos* φ} ∪ { *R* # *p* | *p*. *p*∈ *pos* ψ} |
pos (*FOr* φ ψ) = {[]} ∪ { *L* # *p* | *p*. *p*∈ *pos* φ} ∪ { *R* # *p* | *p*. *p*∈ *pos* ψ} |
pos (*FEq* φ ψ) = {[]} ∪ { *L* # *p* | *p*. *p*∈ *pos* φ} ∪ { *R* # *p* | *p*. *p*∈ *pos* ψ} |
pos (*FImp* φ ψ) = {[]} ∪ { *L* # *p* | *p*. *p*∈ *pos* φ} ∪ { *R* # *p* | *p*. *p*∈ *pos* ψ} |
pos (*FNot* φ) = {[]} ∪ { *L* # *p* | *p*. *p*∈ *pos* φ}

lemma *finite-pos*: *finite* (*pos* φ)
⟨*proof*⟩

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

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

lemma *finite-insert-nil-cons*:
finite s ⟹ *card* (*insert* [] {*L* # *p* |*p*. *p* ∈ *s*}) = *1* + *card* {*L* # *p* |*p*. *p* ∈ *s*}
⟨*proof*⟩

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

lemma *card-seperate*:
assumes *finite s1* **and** *finite s2*
shows *card* ({*L* # *p* |*p*. *p* ∈ *s1*} ∪ {*R* # *p* |*p*. *p* ∈ *s2*}) = *card* ({*L* # *p* |*p*. *p* ∈ *s1*})
+ *card*({*R* # *p* |*p*. *p* ∈ *s2*}) (**is** *card* (*?L*∪*?R*) = *card ?L* + *card ?R*)
⟨*proof*⟩

definition *prop-size* **where** *prop-size* φ = *card* (*pos* φ)

lemma *prop-size-vars-of-prop*:
  fixes φ :: *'v propo*
  shows *card* (*vars-of-prop* φ) $\leq$ *prop-size* φ

  $\langle proof \rangle$

value *pos* (*FImp* (*FAnd* (*FVar P*) (*FVar Q*)) (*FOr* (*FVar P*) (*FVar Q*)))

inductive *path-to* :: *sign list* $\Rightarrow$ *'v propo* $\Rightarrow$ *'v propo* $\Rightarrow$ *bool* **where**
path-to-refl[*intro*]: *path-to* [] φ φ |
path-to-l: $c \in$ *binary-connectives* $\vee$ c = *CNot* $\Longrightarrow$ *wf-conn* c (φ#l) $\Longrightarrow$ *path-to* p φ $\varphi'$$\Longrightarrow$
  path-to (L#p) (*conn* c (φ#l)) φ' |
path-to-r: $c \in$ *binary-connectives* $\Longrightarrow$ *wf-conn* c (ψ#φ#[]) $\Longrightarrow$ *path-to* p φ φ' $\Longrightarrow$
  path-to (R#p) (*conn* c (ψ#φ#[])) φ'

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-to p φ φ' $\Longrightarrow$ $\varphi' \preceq \varphi$
  $\langle proof \rangle$

lemma *subformula-path-exists*:
  fixes φ φ':: *'v propo*
  shows $\varphi' \preceq \varphi \Longrightarrow \exists p.$ *path-to* p φ φ'
$\langle proof \rangle$

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

2.2 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* φ = ($\neg$($\mathcal{A}\models \varphi$)) |
$\mathcal{A} \models$ *FAnd* φ_1 φ_2 = ($\mathcal{A}\models\varphi_1 \wedge \mathcal{A}\models\varphi_2$) |
$\mathcal{A} \models$ *FOr* φ_1 φ_2 = ($\mathcal{A}\models\varphi_1 \vee \mathcal{A}\models\varphi_2$) |
$\mathcal{A} \models$ *FImp* φ_1 φ_2 = ($\mathcal{A}\models\varphi_1 \longrightarrow \mathcal{A}\models\varphi_2$) |
$\mathcal{A} \models$ *FEq* φ_1 φ_2 = ($\mathcal{A}\models\varphi_1 \longleftrightarrow \mathcal{A} \models\varphi_2$)

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

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