Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

April 24, 2020

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*: ‹$a \in$ *uminus* ‘ $A \longleftrightarrow -a \in A$› **for** a :: ‹*'v literal*›
  ⟨*proof*⟩

lemma *uminus-lit-swap*: $- a = b \longleftrightarrow (a$::*'a literal*$) = - b$
  ⟨*proof*⟩

lemma *atm-of-notin-atms-of-iff*: ‹*atm-of* $L \notin$ *atms-of* $C' \longleftrightarrow L \notin\# C' \wedge -L \notin\# C'$› **for** L C'
  ⟨*proof*⟩

lemma *atm-of-notin-atms-of-iff-Pos-Neg*:
  ‹$L \notin$ *atms-of* $C' \longleftrightarrow$ *Pos* $L \notin\# C' \wedge$ *Neg* $L \notin\# C'$› **for** L C'
  ⟨*proof*⟩

lemma *atms-of-uminus*[*simp*]: ‹*atms-of* (*uminus* ‘# C) = *atms-of* C›
  ⟨*proof*⟩

lemma *distinct-mset-atm-ofD*:
  ‹*distinct-mset* (*atm-of* ‘# *mset xc*) $\Longrightarrow$ *distinct xc*›
  ⟨*proof*⟩

lemma *atms-of-cong-set-mset*:
  ‹*set-mset* D = *set-mset* $D' \Longrightarrow$ *atms-of* D = *atms-of* D'›
  ⟨*proof*⟩

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

1.1.2 Clauses

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

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

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

1.1.3 Partial Interpretations

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

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

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

Consistency

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

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

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

lemma *Ex-consistent-interp*: ‹*Ex consistent-interp*›
⟨*proof*⟩

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

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

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

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

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

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

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

lemma (**in** −) *distinct-consistent-distinct-atm*:
‹$\textit{distinct } M \Longrightarrow \textit{consistent-interp } (\textit{set } M) \Longrightarrow \textit{distinct-mset } (\textit{atm-of } \text{‘}\#\ \textit{mset } M)$›
⟨*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* ⇒ *'a set* **where**
$\textit{atms-of-ms } \psi s = \bigcup (\textit{atms-of } \text{‘}\ \psi s)$

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

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

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

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

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

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

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

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

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

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

Totality

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *total-union-2*:
assumes $\textit{total-over-m}\ I\ \psi$
and $\textit{total-over-m}\ I'\ \psi'$

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

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

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

Interpretations

definition *true-cls* :: *'a partial-interp* ⇒ *'a clause* ⇒ *bool* (**infix** ⊨ *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*]: $I \models$ *add-mset* a $D \longleftrightarrow a \in I \lor I \models D$
⟨*proof*⟩

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

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

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

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

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

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

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

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

definition *true-clss* :: *'a partial-interp* ⇒ *'a clause-set* ⇒ *bool* (**infix** ⊨*s 50*) **where**
I ⊨*s CC* ⟷ (∀ *C* ∈ *CC*. *I* ⊨ *C*)

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

lemma *true-clss-singleton*[*iff*]: *I* ⊨*s* {*C*} ⟷ *I* ⊨ *C*
⟨*proof*⟩

lemma *true-clss-empty-entails-empty*[*iff*]: {} ⊨*s N* ⟷ *N* = {}
⟨*proof*⟩

lemma *true-cls-insert-l* [*simp*]:
M ⊨ *A* ⟹ *insert L M* ⊨ *A*
⟨*proof*⟩

lemma *true-clss-union*[*iff*]: *I* ⊨*s CC* ∪ *DD* ⟷ *I* ⊨*s CC* ∧ *I* ⊨*s DD*
⟨*proof*⟩

lemma *true-clss-insert*[*iff*]: *I* ⊨*s insert C DD* ⟷ *I* ⊨ *C* ∧ *I* ⊨*s DD*
⟨*proof*⟩

lemma *true-clss-mono*: *DD* ⊆ *CC* ⟹ *I* ⊨*s CC* ⟹ *I* ⊨*s DD*
⟨*proof*⟩

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

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

lemma *true-clss-commute-l*:
(*I* ∪ *I'* ⊨*s ψ*) ⟷ (*I'* ∪ *I* ⊨*s ψ*)
⟨*proof*⟩

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

lemma *model-remove-minus*[*simp*]: *I* ⊨*s N* ⟹ *I* ⊨*s N* − *A*
⟨*proof*⟩

lemma *notin-vars-union-true-cls-true-cls*:
assumes ∀ *x*∈*I'*. *atm-of x* ∉ *atms-of-ms A*
and *atms-of L* ⊆ *atms-of-ms A*
and *I* ∪ *I'* ⊨ *L*
shows *I* ⊨ *L*
⟨*proof*⟩

lemma *notin-vars-union-true-clss-true-clss*:
assumes ∀ *x*∈*I'*. *atm-of x* ∉ *atms-of-ms A*
and *atms-of-ms L* ⊆ *atms-of-ms A*
and *I* ∪ *I'* ⊨*s L*

shows $I \models s\ L$
$\langle proof \rangle$

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

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

Satisfiability

definition *satisfiable* :: *'a clause set* ⇒ *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*]:
$satisfiable\ \{\{\#L\#\}\}$
$\langle proof \rangle$

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

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

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

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** $?sat \longleftrightarrow ?B$)
$\langle proof \rangle$

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

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

lemma *unsatisfiable-mono*:
‹$N \subseteq N' \Longrightarrow unsatisfiable\ N \Longrightarrow unsatisfiable\ N'$›
$\langle proof \rangle$

Entailment for Multisets of Clauses

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

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

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

⟨*proof*⟩

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

lemma *true-cls-mset-add-mset*[*iff*]: $I \models m\ \mathit{add\text{-}mset}\ C\ CC \longleftrightarrow I \models C \land I \models m\ CC$
⟨*proof*⟩

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

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

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

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

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

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

A simple application of the previous theorem:

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

lemma *multiset-not-empty*:
assumes $M \neq \{\#\}$
and $x \in\#\ M$
shows $\exists A.\ x = \mathit{Pos}\ A \lor x = \mathit{Neg}\ A$
⟨*proof*⟩

lemma *atms-of-ms-empty*:
fixes $\psi :: {}'v\ \mathit{clause\text{-}set}$
assumes $\mathit{atms\text{-}of\text{-}ms}\ \psi = \{\}$
shows $\psi = \{\} \lor \psi = \{\{\#\}\}$
⟨*proof*⟩

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

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

lemma *true-cls-remove-hd-if-notin-vars*:
  assumes *insert a M*′⊨ *D*
  and *atm-of a* ∉ *atms-of D*
  shows *M*′ ⊨ *D*
  ⟨*proof*⟩

lemma *total-over-set-atm-of*:
  fixes *I* :: ′*v partial-interp* **and** *K* :: ′*v set*
  shows *total-over-set I K* ⟷ (∀ *l* ∈ *K*. *l* ∈ (*atm-of* ‘ *I*))
  ⟨*proof*⟩

lemma *true-cls-mset-true-clss-iff*:
  ‹*finite C* ⟹ *I* ⊨*m mset-set C* ⟷ *I* ⊨*s C*›
  ‹*I* ⊨*m D* ⟷ *I* ⊨*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*) ≡ ∀ *I*. *total-over-set I* (*atms-of ψ*) ⟶ *I* ⊨ *ψ*

lemma *tautology-Pos-Neg*[*intro*]:
  assumes *Pos p* ∈# *A* **and** *Neg p* ∈# *A*
  shows *tautology A*
  ⟨*proof*⟩

lemma *tautology-minus*[*simp*]:
  assumes *L* ∈# *A* **and** −*L* ∈# *A*
  shows *tautology A*
  ⟨*proof*⟩

lemma *tautology-exists-Pos-Neg*:
  assumes *tautology ψ*
  shows ∃ *p*. *Pos p* ∈# *ψ* ∧ *Neg p* ∈# *ψ*
⟨*proof*⟩

lemma *tautology-decomp*:
  tautology ψ ⟷ (∃ *p*. *Pos p* ∈# *ψ* ∧ *Neg p* ∈# *ψ*)
  ⟨*proof*⟩

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

lemma *not-tautology-minus*:

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

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

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

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

lemma *tautology-union*:
$\langle \mathit{tautology}\ (A + B) \longleftrightarrow \mathit{tautology}\ A \lor \mathit{tautology}\ B \lor (\exists\, a.\ a \in\#\ A \land -a \in\#\ B)\rangle$
$\langle \mathit{proof}\rangle$

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

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

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

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

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

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

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

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

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

lemma *tautology-distinct-atm-iff*:
  ‹$\textit{distinct-mset}\ C \Longrightarrow \textit{tautology}\ C \longleftrightarrow \neg\textit{distinct-mset}\ (\textit{atm-of}\ \text{‘\#}\ C)$›
  ⟨*proof*⟩

lemma *not-tautology-minusD*:
  ‹$\textit{tautology}\ (A - B) \Longrightarrow \textit{tautology}\ A$›
  ⟨*proof*⟩

Entailment for clauses and propositions

We also need entailment of clauses by other clauses.

definition *true-cls-cls* :: $'a\ \textit{clause} \Rightarrow 'a\ \textit{clause} \Rightarrow \textit{bool}$ (**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* :: $'a\ \textit{clause} \Rightarrow 'a\ \textit{clause-set} \Rightarrow \textit{bool}$ (**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* :: $'a\ \textit{clause-set} \Rightarrow 'a\ \textit{clause} \Rightarrow \textit{bool}$ (**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* :: $'a\ \textit{clause-set} \Rightarrow 'a\ \textit{clause-set} \Rightarrow \textit{bool}$ (**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*]:
  $A \models f\ A$
  ⟨*proof*⟩

lemma *true-clss-cls-empty-empty*[*iff*]:
  ‹$\{\} \models p\ \{\#\} \longleftrightarrow \textit{False}$›
  ⟨*proof*⟩

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

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

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

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

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

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

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

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*]:
$A \models ps\ C \cup D \longleftrightarrow (A \models ps\ C \land A \models ps\ D)$
⟨*proof*⟩

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

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

Better suited as intro rule:

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

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

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

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

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

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

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

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

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

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

shows $N \models p\ D \cup\#\ C$
$\langle proof \rangle$

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

lemma *true-cls-mset-empty-iff* [*simp*]: $\langle \{\} \models m\ C \longleftrightarrow C = \{\#\} \rangle$
$\langle proof \rangle$

lemma *true-clss-mono-left*:
$\langle I \models s\ A \Longrightarrow I \subseteq J \Longrightarrow J \models s\ A \rangle$
$\langle proof \rangle$

lemma *true-cls-remove-alien*:
$\langle I \models N \longleftrightarrow \{L.\ L \in I \wedge atm\text{-}of\ L \in atms\text{-}of\ N\} \models N \rangle$
$\langle proof \rangle$

lemma *true-clss-remove-alien*:
$\langle I \models s\ N \longleftrightarrow \{L.\ L \in I \wedge atm\text{-}of\ L \in atms\text{-}of\text{-}ms\ N\} \models s\ N \rangle$
$\langle proof \rangle$

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

lemma *true-clss-alt-def2*:
assumes $\langle \neg tautology\ \chi \rangle$
shows $\langle N \models p\ \chi \longleftrightarrow (\forall I.\ atms\text{-}of\text{-}s\ I = atms\text{-}of\text{-}ms\ N \longrightarrow consistent\text{-}interp\ I \longrightarrow I \models s\ N \longrightarrow I \models \chi) \rangle$ (**is** $\langle ?A \longleftrightarrow ?B \rangle$)
$\langle proof \rangle$

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

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 $\langle N \models p\ C \rangle$
$\langle I \models s\ N \rangle$ **and**
cons: $\langle consistent\text{-}interp\ I \rangle$ **and**
tauto: $\langle \neg tautology\ C \rangle$
shows $\langle I \models C \rangle$
$\langle proof \rangle$

1.1.4 Subsumptions

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

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

lemma *subsumption-chained*:
$\quad$**assumes**
$\quad\quad$∀I. *total-over-m* I {D} ⟶ I ⊨ D ⟶ I ⊨ φ **and**
$\quad\quad$$C$ ⊆# D
$\quad$**shows** (∀I. *total-over-m* I {C} ⟶ I ⊨ C ⟶ I ⊨ φ) ∨ *tautology* φ
$\quad$⟨*proof*⟩

1.1.5 Removing Duplicates

lemma *tautology-remdups-mset*[*iff*]:
$\quad$*tautology* (*remdups-mset* C) ⟷ *tautology* C
$\quad$⟨*proof*⟩

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

lemma *true-cls-remdups-mset*[*iff*]: I ⊨ *remdups-mset* C ⟷ I ⊨ C
$\quad$⟨*proof*⟩

lemma *true-clss-cls-remdups-mset*[*iff*]: A ⊨*p* *remdups-mset* C ⟷ A ⊨*p* C
$\quad$⟨*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-clss atms = {C. *atms-of* C ⊆ *atms* ∧ ¬*tautology* C ∧ *distinct-mset* C}

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

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

lemma *simple-clss-finite*:

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

lemma *simple-clssE*:
assumes
x ∈ *simple-clss atms*
shows *atms-of x* ⊆ *atms* ∧ ¬*tautology x* ∧ *distinct-mset x*
⟨*proof*⟩

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

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

lemma *simple-clss-mono*:
assumes *incl*: *atms* ⊆ *atms'*
shows *simple-clss atms* ⊆ *simple-clss atms'*
⟨*proof*⟩

lemma *distinct-mset-not-tautology-implies-in-simple-clss*:
assumes *distinct-mset* χ **and** ¬*tautology* χ
shows χ ∈ *simple-clss* (*atms-of* χ)
⟨*proof*⟩

lemma *simplified-in-simple-clss*:
assumes *distinct-mset-set* ψ **and** ∀ χ ∈ ψ. ¬*tautology* χ
shows ψ ⊆ *simple-clss* (*atms-of-ms* ψ)
⟨*proof*⟩

lemma *simple-clss-element-mono*:
‹*x* ∈ *simple-clss A* ⟹ *y* ⊆# *x* ⟹ *y* ∈ *simple-clss A*›
⟨*proof*⟩

1.1.7 Experiment: Expressing the Entailments as Locales

locale *entail* =
fixes *entail* :: *'a set* ⇒ *'b* ⇒ *bool* (**infix** ⊨*e 50*)
assumes *entail-insert*[*simp*]: *I* ≠ {} ⟹ *insert L I* ⊨*e x* ⟷ {*L*} ⊨*e x* ∨ *I* ⊨*e x*
assumes *entail-union*[*simp*]: *I* ⊨*e A* ⟹ *I* ∪ *I'* ⊨*e A*
begin

definition *entails* :: *'a set* ⇒ *'b set* ⇒ *bool* (**infix** ⊨*es 50*) **where**
I ⊨*es A* ⟷ (∀ *a* ∈ *A*. *I* ⊨*e a*)

lemma *entails-empty*[*simp*]:
I ⊨*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* ⊨*sext N* − {*C*}
⟨*proof*⟩

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

lemma *not-consistent-true-clss-ext*:
assumes ¬*consistent-interp I*
shows *I* ⊨*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**
‹⋀*L xs*. *P xs* ⟹ *P* (*Decided L* # *xs*)› **and**
‹⋀*L m xs*. *P xs* ⟹ *P* (*Propagated L m* # *xs*)›
shows ‹*P xs*›
⟨*proof*⟩

lemma *is-decided-ex-Decided*:
‹*is-decided L* ⟹ (⋀*K*. *L* = *Decided K* ⟹ *P*) ⟹ *P*›
⟨*proof*⟩

lemma *is-propedE*: ‹*is-proped L* ⟹ (⋀*K C*. *L* = *Propagated K C* ⟹ *P*) ⟹ *P*›
⟨*proof*⟩

lemma *is-decided-no-proped-iff*: ‹*is-decided L* ⟷ ¬*is-proped L*›
⟨*proof*⟩

lemma *not-is-decidedE*:
‹¬*is-decided E* ⟹ (⋀*K C*. *E* = *Propagated K C* ⟹ *thesis*) ⟹ *thesis*›
⟨*proof*⟩

type-synonym (*'v*, *'m*) *ann-lits* = ‹(*'v*, *'m*) *ann-lit list*›

fun *lit-of* :: ‹(*'a*, *'a*, *'mark*) *annotated-lit* ⇒ *'a*› **where**
‹*lit-of* (*Decided L*) = *L*› |
‹*lit-of* (*Propagated L* -) = *L*›

definition *lits-of* :: ‹(*'a*, *'b*) *ann-lit set* ⇒ *'a literal set*› **where**
‹*lits-of Ls* = *lit-of* ‘ *Ls*›

abbreviation *lits-of-l* :: ‹(*'a*, *'b*) *ann-lits* ⇒ *'a literal set*› **where**
‹*lits-of-l Ls* ≡ *lits-of* (*set Ls*)›

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

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

lemma *lits-of-l-Un*[*simp*]:
‹*lits-of* (*l* ∪ *l'*) = *lits-of l* ∪ *lits-of l'*›
⟨*proof*⟩

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

abbreviation *unmark* **where**
‹*unmark* ≡ (λ*a*. {#*lit-of a*#})›

abbreviation *unmark-s* **where**
‹*unmark-s M* ≡ *unmark* ‘ *M*›

abbreviation *unmark-l* **where**
‹*unmark-l M* ≡ *unmark-s* (*set M*)›

lemma *atms-of-ms-lambda-lit-of-is-atm-of-lit-of*[*simp*]:
‹*atms-of-ms* (*unmark-l M'*) = *atm-of* ‘ *lits-of-l M'*›
⟨*proof*⟩

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

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

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

Entailment

definition *true-annot* :: ‹(*'a*, *'m*) *ann-lits* ⇒ *'a clause* ⇒ *bool*› (**infix** ⊨*a 49*) **where**
‹*I* ⊨*a C* ⟷ (*lits-of-l I*) ⊨ *C*›

definition *true-annots* :: ‹(*'a*, *'m*) *ann-lits* ⇒ *'a clause-set* ⇒ *bool*› (**infix** ⊨*as 49*) **where**
‹*I* ⊨*as CC* ⟷ (∀ *C* ∈ *CC*. *I* ⊨*a C*)›

lemma *true-annot-empty-model*[*simp*]:
‹¬[] ⊨*a* ψ›
⟨*proof*⟩

lemma *true-annot-empty*[*simp*]:
‹¬*I* ⊨*a* {#}›
⟨*proof*⟩

lemma *empty-true-annots-def*[*iff*]:
‹[] ⊨*as* ψ ⟷ ψ = {}›
⟨*proof*⟩

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

lemma *true-annots-single-true-annot*[*iff*]:
‹*I* ⊨*as* {*C*} ⟷ *I* ⊨*a C*›
⟨*proof*⟩

lemma *true-annot-insert-l*[*simp*]:
‹*M* ⊨*a A* ⟹ *L* # *M* ⊨*a A*›
⟨*proof*⟩

lemma *true-annots-insert-l* [*simp*]:
‹*M* ⊨*as A* ⟹ *L* # *M* ⊨*as A*›
⟨*proof*⟩

lemma *true-annots-union*[*iff*]:
‹*M* ⊨*as A* ∪ *B* ⟷ (*M* ⊨*as A* ∧ *M* ⊨*as B*)›
⟨*proof*⟩

lemma *true-annots-insert*[*iff*]:
‹*M* ⊨*as insert a A* ⟷ (*M* ⊨*a a* ∧ *M* ⊨*as A*)›
⟨*proof*⟩

lemma *true-annot-append-l*:
‹*M* ⊨*a A* ⟹ *M'* @ *M* ⊨*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*:
‹$I \models as\ CC \longleftrightarrow \mathit{lits\text{-}of\text{-}l}\ I \models s\ CC$›
⟨*proof*⟩

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

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

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

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

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

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

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

lemma *true-annots-true-clss-clss*:
‹$A \models as\ \Psi \Longrightarrow \mathit{unmark\text{-}l}\ A \models ps\ \Psi$›
⟨*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*]:
‹$\mathit{set}\ I \subseteq \mathit{set}\ I' \Longrightarrow I \models a\ N \Longrightarrow I' \models a\ N$›
⟨*proof*⟩

lemma *true-annots-mono*:
‹$\mathit{set}\ I \subseteq \mathit{set}\ I' \Longrightarrow I \models as\ N \Longrightarrow I' \models as\ N$›
⟨*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 ∈ set (fst (backtrack-split l)) ⟹ ¬is-decided a›
  ⟨proof⟩
```

```
lemma backtrack-split-snd-hd-decided:
  ‹snd (backtrack-split l) ≠ [] ⟹ is-decided (hd (snd (backtrack-split l)))›
  ⟨proof⟩
```

```
lemma backtrack-split-list-eq[simp]:
  ‹fst (backtrack-split l) @ (snd (backtrack-split l)) = l›
  ⟨proof⟩
```

```
lemma backtrack-snd-empty-not-decided:
  ‹backtrack-split M = (M'', []) ⟹ ∀ l∈set M. ¬ is-decided l›
  ⟨proof⟩
```

```
lemma backtrack-split-some-is-decided-then-snd-has-hd:
  ‹∃ l∈set M. is-decided l ⟹ ∃ M' L' M''. backtrack-split M = (M'', L' # M')›
  ⟨proof⟩
```

Another characterisation of the result of *backtrack-split*. This view allows some simpler proofs, since *takeWhile* and *dropWhile* are highly automated:

```
lemma backtrack-split-takeWhile-dropWhile:
  ‹backtrack-split M = (takeWhile (Not o is-decided) M, dropWhile (Not o is-decided) M)›
  ⟨proof⟩
```

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*]:
‹[] = *get-all-ann-decomposition M* ⟷ *False*›
⟨*proof*⟩

lemma *get-all-ann-decomposition-decomp*:
‹*hd* (*get-all-ann-decomposition S*) = (*a*, *c*) ⟹ *S* = *c* @ *a*›
⟨*proof*⟩

lemma *get-all-ann-decomposition-backtrack-split*:
‹*backtrack-split S* = (*M*, *M'*) ⟷ *hd* (*get-all-ann-decomposition S*) = (*M'*, *M*)›
⟨*proof*⟩

lemma *get-all-ann-decomposition-Nil-backtrack-split-snd-Nil*:
‹*get-all-ann-decomposition S* = [([], *A*)] ⟹ *snd* (*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 ‹*get-all-ann-decomposition M* = (*a*, *b*) # []›
shows ‹*a* = [] ∨ (*length a* = *1* ∧ *is-decided* (*hd a*) ∧ *hd a* ∈ *set M*)›
⟨*proof*⟩

lemma *get-all-ann-decomposition-fst-empty-or-hd-in-M*:
assumes ‹*get-all-ann-decomposition M* = (*a*, *b*) # *l*›
shows ‹*a* = [] ∨ (*is-decided* (*hd a*) ∧ *hd a* ∈ *set M*)›
⟨*proof*⟩

lemma *get-all-ann-decomposition-snd-not-decided*:
assumes ‹(*a*, *b*) ∈ *set* (*get-all-ann-decomposition M*)›
and ‹*L* ∈ *set b*›
shows ‹¬*is-decided L*›
⟨*proof*⟩

lemma *tl-get-all-ann-decomposition-skip-some*:
assumes ‹*x* ∈ *set* (*tl* (*get-all-ann-decomposition M1*))›
shows ‹*x* ∈ *set* (*tl* (*get-all-ann-decomposition* (*M0* @ *M1*)))›
⟨*proof*⟩

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

lemma *in-get-all-ann-decomposition-in-get-all-ann-decomposition-prepend*:
‹(*a*, *b*) ∈ *set* (*get-all-ann-decomposition M'*) ⟹
∃ *b'*. (*a*, *b'* @ *b*) ∈ *set* (*get-all-ann-decomposition* (*M* @ *M'*))›
⟨*proof*⟩

lemma *in-get-all-ann-decomposition-decided-or-empty*:
assumes ‹(*a*, *b*) ∈ *set* (*get-all-ann-decomposition M*)›
shows ‹*a* = [] ∨ (*is-decided* (*hd a*))›
⟨*proof*⟩

lemma *get-all-ann-decomposition-remove-undecided-length*:
  assumes ‹∀ l ∈ 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 ‹∀ l ∈ 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 ‹∀ l ∈ 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 ‹∀ l ∈ 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-all-ann-decomposition M*))
  = *set* (*map fst* (*get-all-ann-decomposition* (*MS* @ *M*)))›
⟨*proof*⟩

lemma *no-decision-get-all-ann-decomposition*:
‹∀ *l*∈*set M*. ¬ *is-decided l* ⟹ *get-all-ann-decomposition M* = [([], *M*)]›
⟨*proof*⟩

Entailment of the Propagated by the Decided Literal

lemma *get-all-ann-decomposition-snd-union*:
‹*set M* = ⋃(*set* ‘ *snd* ‘ *set* (*get-all-ann-decomposition M*)) ∪ {*L* |*L*. *is-decided L* ∧ *L* ∈ *set M*}›
(**is** ‹*?M M* = *?U M* ∪ *?Ls M*›)
⟨*proof*⟩

definition *all-decomposition-implies* :: ‹*'a clause set*
⇒ ((*'a*, *'m*) *ann-lits* × (*'a*, *'m*) *ann-lits*) *list* ⇒ *bool*› **where**
‹*all-decomposition-implies N S* ⟷ (∀ (*Ls*, *seen*) ∈ *set S*. *unmark-l Ls* ∪ *N* ⊨*ps unmark-l seen*)›

lemma *all-decomposition-implies-empty*[*iff*]:
‹*all-decomposition-implies N* []› ⟨*proof*⟩

lemma *all-decomposition-implies-single*[*iff*]:
‹*all-decomposition-implies N* [(*Ls*, *seen*)] ⟷ *unmark-l Ls* ∪ *N* ⊨*ps unmark-l seen*›
⟨*proof*⟩

lemma *all-decomposition-implies-append*[*iff*]:
‹*all-decomposition-implies N* (*S* @ *S'*)
  ⟷ (*all-decomposition-implies N S* ∧ *all-decomposition-implies N S'*)›
⟨*proof*⟩

lemma *all-decomposition-implies-cons-pair*[*iff*]:
‹*all-decomposition-implies N* ((*Ls*, *seen*) # *S'*)
  ⟷ (*all-decomposition-implies N* [(*Ls*, *seen*)] ∧ *all-decomposition-implies N S'*)›
⟨*proof*⟩

lemma *all-decomposition-implies-cons-single*[*iff*]:
‹*all-decomposition-implies N* (*l* # *S'*) ⟷
  (*unmark-l* (*fst l*) ∪ *N* ⊨*ps unmark-l* (*snd l*) ∧
   all-decomposition-implies N S')›
⟨*proof*⟩

lemma *all-decomposition-implies-trail-is-implied*:
assumes ‹*all-decomposition-implies N* (*get-all-ann-decomposition M*)›
shows ‹*N* ∪ {*unmark L* |*L*. *is-decided L* ∧ *L* ∈ *set M*}
  ⊨*ps unmark* ‘ ⋃(*set* ‘ *snd* ‘ *set* (*get-all-ann-decomposition M*))›
⟨*proof*⟩

lemma *all-decomposition-implies-propagated-lits-are-implied*:
assumes ‹*all-decomposition-implies N* (*get-all-ann-decomposition M*)›
shows ‹*N* ∪ {*unmark L* |*L*. *is-decided L* ∧ *L* ∈ *set M*} ⊨*ps unmark-l M*›
(**is** ‹*?I* ⊨*ps ?A*›)
⟨*proof*⟩

lemma *all-decomposition-implies-insert-single*:
‹*all-decomposition-implies N M* ⟹ *all-decomposition-implies* (*insert C N*) *M*›
⟨*proof*⟩

lemma *all-decomposition-implies-union*:
‹*all-decomposition-implies N M* ⟹ *all-decomposition-implies* (*N* ∪ *N′*) *M*›
⟨*proof*⟩

lemma *all-decomposition-implies-mono*:
‹*N* ⊆ *N′* ⟹ *all-decomposition-implies N M* ⟹ *all-decomposition-implies N′ M*›
⟨*proof*⟩

lemma *all-decomposition-implies-mono-right*:
‹*all-decomposition-implies I* (*get-all-ann-decomposition* (*M′* @ *M*)) ⟹
all-decomposition-implies I (*get-all-ann-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 ψ* = { {#−*L*#} | *L*. *L* ∈# *ψ* }›

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

lemma *in-CNot-uminus*[*iff*]:
shows ‹{#*L*#} ∈ *CNot ψ* ⟷ −*L* ∈# *ψ*›
⟨*proof*⟩

lemma
shows
CNot-add-mset[*simp*]: ‹*CNot* (*add-mset L ψ*) = *insert* {#−*L*#} (*CNot ψ*)› **and**
CNot-empty[*simp*]: ‹*CNot* {#} = {}› **and**
CNot-plus[*simp*]: ‹*CNot* (*A* + *B*) = *CNot A* ∪ *CNot B*›
⟨*proof*⟩

lemma *CNot-eq-empty*[*iff*]:
‹*CNot D* = {} ⟷ *D* = {#}›
⟨*proof*⟩

lemma *in-CNot-implies-uminus*:
assumes ‹*L* ∈# *D*› **and** ‹*M* ⊨*as CNot D*›
shows ‹*M* ⊨*a* {#−*L*#}› **and** ‹−*L* ∈ *lits-of-l M*›
⟨*proof*⟩

lemma *CNot-remdups-mset*[*simp*]:
‹*CNot* (*remdups-mset A*) = *CNot A*›
⟨*proof*⟩

lemma *Ball-CNot-Ball-mset*[*simp*]:
‹(∀ *x*∈*CNot D*. *P x*) ⟷ (∀ *L*∈# *D*. *P* {#−*L*#})›
⟨*proof*⟩

lemma *consistent-CNot-not*:
assumes ⟨*consistent-interp I*⟩
shows ⟨*I* ⊨*s CNot* φ ⟹ ¬*I* ⊨ φ⟩
⟨*proof*⟩

lemma *total-not-true-cls-true-clss-CNot*:
assumes ⟨*total-over-m I* {φ}⟩ **and** ⟨¬*I* ⊨ φ⟩
shows ⟨*I* ⊨*s CNot* φ⟩
⟨*proof*⟩

lemma *total-not-CNot*:
assumes ⟨*total-over-m I* {φ}⟩ **and** ⟨¬*I* ⊨*s CNot* φ⟩
shows ⟨*I* ⊨ φ⟩
⟨*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* ∈ *D* ⟹ *D* ⊨*ps CNot C* ⟹ *D* ⊨*p* {#}⟩
⟨*proof*⟩

lemma *true-annots-CNot-all-atms-defined*:
assumes ⟨*M* ⊨*as CNot T*⟩ **and** *a1*: ⟨*L* ∈# *T*⟩
shows ⟨*atm-of L* ∈ *atm-of* ‘ *lits-of-l M*⟩
⟨*proof*⟩

lemma *true-annots-CNot-all-uminus-atms-defined*:
assumes ⟨*M* ⊨*as CNot T*⟩ **and** *a1*: ⟨−*L* ∈# *T*⟩
shows ⟨*atm-of L* ∈ *atm-of* ‘ *lits-of-l M*⟩
⟨*proof*⟩

lemma *true-clss-clss-false-left-right*:
assumes ⟨{{#*L*#}} ∪ *B* ⊨*p* {#}⟩
shows ⟨*B* ⊨*ps CNot* {#*L*#}⟩
⟨*proof*⟩

lemma *true-annots-true-cls-def-iff-negation-in-model*:
⟨*M* ⊨*as CNot C* ⟷ (∀ *L* ∈# *C*. −*L* ∈ *lits-of-l M*)⟩
⟨*proof*⟩

lemma *true-clss-def-iff-negation-in-model*:
⟨*M* ⊨*s CNot C* ⟷ (∀ *l* ∈# *C*. −*l* ∈ *M*)⟩
⟨*proof*⟩

lemma *true-annots-CNot-definedD*:
⟨*M* ⊨*as CNot C* ⟹ ∀ *L* ∈# *C*. *defined-lit M L*⟩
⟨*proof*⟩

lemma *true-annot-CNot-diff*:
⟨*I* ⊨*as CNot C* ⟹ *I* ⊨*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* ⊨*p I* ⟷ *N* ∪ (λ*L*. {#−*L*#}) ‘ *set-mset I* ⊨*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* ∈ *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* ⟷ *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-dup M* ⟹ − *lit-of xa* = *lit-of x* ⟹ *x* ∈ *set M* ⟹ *xa* ∉ *set M*›
⟨*proof*⟩

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

lemma *true-cls-mset-restrict*:
‹{*L* ∈ *I*. *atm-of L* ∈ *atms-of-mm N*} ⊨*m N* ⟷ *I* ⊨*m N*›
⟨*proof*⟩

lemma *true-clss-restrict*:
‹{*L* ∈ *I*. *atm-of L* ∈ *atms-of-mm N*} ⊨*sm N* ⟷ *I* ⊨*sm N*›
⟨*proof*⟩

lemma *total-over-m-atms-incl*:
assumes ‹*total-over-m M* (*set-mset N*)›
shows
‹*x* ∈ *atms-of-mm N* ⟹ *x* ∈ *atms-of-s M*›
⟨*proof*⟩

lemma *true-clss-restrict-iff*:
assumes ‹¬*tautology* χ›
shows ‹*N* ⊨*p* χ ⟷ *N* ⊨*p* {#*L* ∈# χ. *atm-of L* ∈ *atms-of-ms N*#}› (**is** ‹*?A* ⟷ *?B*›)
⟨*proof*⟩

1.2.8 Negation of annotated clauses

definition *negate-ann-lits* :: ‹('*v literal*, '*v literal*, '*mark*) *annotated-lits* ⇒ '*v literal multiset*› **where**
‹*negate-ann-lits M* = (λ*L*. − *lit-of L*) ‘# *mset M*›

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

lemma *entails-CNot-negate-ann-lits*:
‹*M* ⊨*as CNot D* ⟷ *set-mset D* ⊆ *set-mset* (*negate-ann-lits M*)›
⟨*proof*⟩

Pointwise negation of a clause:

definition *pNeg* :: ‹'*v clause* ⇒ '*v clause*› **where**
‹*pNeg C* = {#−*D*. *D* ∈# *C*#}›

lemma *pNeg-simps*:
‹*pNeg* (*add-mset A C*) = *add-mset* (−*A*) (*pNeg C*)›
‹*pNeg* (*C* + *D*) = *pNeg C* + *pNeg D*›
⟨*proof*⟩

lemma *atms-of-pNeg*[*simp*]: ‹*atms-of* (*pNeg C*) = *atms-of C*›
⟨*proof*⟩

lemma *negate-ann-lits-pNeg-lit-of*: ‹*negate-ann-lits* = *pNeg o image-mset lit-of o mset*›
⟨*proof*⟩

lemma *negate-ann-lits-empty-iff*: ‹*negate-ann-lits M* ≠ {#} ⟷ *M* ≠ []›
⟨*proof*⟩

lemma *atms-of-negate-ann-lits*[*simp*]: ‹*atms-of* (*negate-ann-lits M*) = *atm-of* ‘ (*lits-of-l M*)›
⟨*proof*⟩

lemma *tautology-pNeg*[*simp*]:
‹*tautology* (*pNeg C*) ⟷ *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-mset n L*) = *replicate-mset n* (−*L*)›
⟨*proof*⟩

lemma *distinct-mset-pNeg-iff*[*iff*]: ‹*distinct-mset* (*pNeg x*) ⟷ *distinct-mset x*›
⟨*proof*⟩

lemma *pNeg-simple-clss-iff*[*simp*]:
‹*pNeg M* ∈ *simple-clss N* ⟷ *M* ∈ *simple-clss N*›
⟨*proof*⟩

lemma *atms-of-ms-pNeg*[*simp*]: ‹*atms-of-ms* (*pNeg* ‘ *N*) = *atms-of-ms N*›
⟨*proof*⟩

definition *DECO-clause* :: ‹(*'v*, *'a*) *ann-lits* ⇒ *'v clause*› **where**
‹*DECO-clause M* = (*uminus o lit-of*) ‘# (*filter-mset is-decided* (*mset M*))›

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

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

lemma *in-negate-trial-iff*: ‹*L* ∈# *negate-ann-lits M* ⟷ − *L* ∈ *lits-of-l M*›
⟨*proof*⟩

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

lemma *uminus-simple-clss-iff* [*simp*]:
⟨*uminus* ‘# *M* ∈ *simple-clss N* ⟷ *M* ∈ *simple-clss N*⟩
⟨*proof*⟩

lemma *pNeg-mono*: ⟨*C* ⊆# *C'* ⟹ *pNeg C* ⊆# *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* ⇒ *'a partial-interp*⟩ **where**
⟨*partial-model-of I* = *Pos* ‘ *I* ∪ *Neg* ‘ {*x*. *x* ∉ *I*}⟩

definition *total-model-of* :: ⟨*'a partial-interp* ⇒ *'a interp*⟩ **where**
⟨*total-model-of I* = {*x*. *Pos x* ∈ *I*}⟩

lemma *total-over-set-partial-model-of*:
⟨*total-over-set* (*partial-model-of I*) *UNIV*⟩
⟨*proof*⟩

lemma *consistent-interp-partial-model-of*:
⟨*consistent-interp* (*partial-model-of I*)⟩
⟨*proof*⟩

lemma *consistent-interp-alt-def*:
⟨*consistent-interp I* ⟷ (∀ *L*. ¬(*Pos L* ∈ *I* ∧ *Neg L* ∈ *I*))⟩
⟨*proof*⟩

context
fixes *I* :: ⟨*'a partial-interp*⟩
assumes *cons*: ⟨*consistent-interp I*⟩
begin

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

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

lemma *partial-implies-total-true-clss-total-model-of*:
assumes ⟨*Partial-Herbrand-Interpretation.true-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* ⟷ *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} ∪ {CT} ∪ {CVar x | 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 φ ψ :: 'v propo
  assumes nullary: ⋀φ x. φ = FF ∨ φ = FT ∨ φ = FVar x ⟹ P φ
  and unary: ⋀ψ. P ψ ⟹ P (FNot ψ)
  and binary: ⋀φ ψ1 ψ2. P ψ1 ⟹ P ψ2 ⟹ φ = FAnd ψ1 ψ2 ∨ φ = FOr ψ1 ψ2 ∨ φ = FImp ψ1
ψ2
    ∨ φ = FEq ψ1 ψ2 ⟹ P φ
  shows P ψ
  ⟨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* ⇒ *′v propo list* ⇒ *′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*: ⋀*x*. *c* = *CT* ∨ *c* = *CF* ∨ *c* = *CVar x* ⟹ *P*
  and *binary*: *c* ∈ *binary-connectives* ⟹ *P*
  and *unary*: *c* = *CNot* ⟹ *P*
  shows *P*
  ⟨*proof*⟩

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

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

inductive *wf-conn* :: *′v connective* ⇒ *′v propo list* ⇒ *bool* **for** *c* :: *′v connective* **where**
wf-conn-nullary[*simp*]: (*c* = *CT* ∨ *c* = *CF* ∨ *c* = *CVar v*) ⟹ *wf-conn c* [] |
wf-conn-unary[*simp*]: *c* = *CNot* ⟹ *wf-conn c* [ψ] |
wf-conn-binary[*simp*]: *c* ∈ *binary-connectives* ⟹ *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**
    ⋀*v*. *c* = *CT* ⟹ *P* [] **and**
    ⋀*v*. *c* = *CF* ⟹ *P* [] **and**
    ⋀*v*. *c* = *CVar v* ⟹ *P* [] **and**
    ⋀ψ. *c* = *CNot* ⟹ *P* [ψ] **and**
    ⋀ψ ψ′. *c* = *COr* ⟹ *P* [ψ, ψ′] **and**
    ⋀ψ ψ′. *c* = *CAnd* ⟹ *P* [ψ, ψ′] **and**
    ⋀ψ ψ′. *c* = *CImp* ⟹ *P* [ψ, ψ′] **and**
    ⋀ψ ψ′. *c* = *CEq* ⟹ *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*]:

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

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

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

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

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

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

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

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

Decomposing the Not into an element is moreover very useful.

lemma *wf-conn-Not-decomp*:
fixes $l :: \textit{'v propo list}$ **and** $a :: \textit{'v}$
assumes *corr*: *wf-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*:
$\textit{length l} = \textit{length l'} \Longrightarrow \textit{wf-conn c l} \longleftrightarrow \textit{wf-conn c l'}$
⟨*proof*⟩

lemma *wf-conn-no-arity-change-helper*:
$\mathit{length}\ (\xi\ @\ \varphi\ \#\ \xi') = \mathit{length}\ (\xi\ @\ \varphi'\ \#\ \xi')$
$\langle \mathit{proof} \rangle$

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*: $\mathit{conn}\ c\ l = \mathit{FNot}\ \psi$
shows $c = \mathit{CNot}$ **and** $l = [\psi]$
$\langle \mathit{proof} \rangle$

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

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: $\psi \in \mathit{set}\ l \Longrightarrow \mathit{wf\text{-}conn}\ c\ l \Longrightarrow \varphi \preceq \psi \Longrightarrow \varphi \preceq \mathit{conn}\ c\ l$

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*: $\mathit{FNot}\ \varphi \preceq \psi \Longrightarrow \varphi \preceq \psi$
$\langle \mathit{proof} \rangle$

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

lemma *subformula-trans*:
$\psi \preceq \psi' \Longrightarrow \varphi \preceq \psi \Longrightarrow \varphi \preceq \psi'$
$\langle \mathit{proof} \rangle$

lemma *subformula-leaf*:
fixes $\varphi\ \psi$ :: *'v propo*
assumes *incl*: $\varphi \preceq \psi$
and *simple*: $\psi = \mathit{FT} \vee \psi = \mathit{FF} \vee \psi = \mathit{FVar}\ x$
shows $\varphi = \psi$
$\langle \mathit{proof} \rangle$

lemma *subfurmula-not-incl-eq*:

assumes $\varphi \preceq \mathit{conn}\ c\ l$
and $\mathit{wf\text{-}conn}\ c\ l$
and $\forall \psi.\ \psi \in \mathit{set}\ l \longrightarrow \neg\ \varphi \preceq \psi$
shows $\varphi = \mathit{conn}\ c\ l$
$\langle \mathit{proof} \rangle$

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

lemma *subformula-decomp-explicit*[*simp*]:
$\varphi \preceq \mathit{FAnd}\ \psi\ \psi' \longleftrightarrow (\varphi = \mathit{FAnd}\ \psi\ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi')$ (**is** *?P FAnd*)
$\varphi \preceq \mathit{FOr}\ \psi\ \psi' \longleftrightarrow (\varphi = \mathit{FOr}\ \psi\ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi')$
$\varphi \preceq \mathit{FEq}\ \psi\ \psi' \longleftrightarrow (\varphi = \mathit{FEq}\ \psi\ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi')$
$\varphi \preceq \mathit{FImp}\ \psi\ \psi' \longleftrightarrow (\varphi = \mathit{FImp}\ \psi\ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi')$
$\langle \mathit{proof} \rangle$

lemma *wf-conn-helper-facts*[*iff*]:
$\mathit{wf\text{-}conn}\ \mathit{CNot}\ [\varphi]$
$\mathit{wf\text{-}conn}\ \mathit{CT}\ []$
$\mathit{wf\text{-}conn}\ \mathit{CF}\ []$
$\mathit{wf\text{-}conn}\ (\mathit{CVar}\ x)\ []$
$\mathit{wf\text{-}conn}\ \mathit{CAnd}\ [\varphi, \psi]$
$\mathit{wf\text{-}conn}\ \mathit{COr}\ [\varphi, \psi]$
$\mathit{wf\text{-}conn}\ \mathit{CImp}\ [\varphi, \psi]$
$\mathit{wf\text{-}conn}\ \mathit{CEq}\ [\varphi, \psi]$
$\langle \mathit{proof} \rangle$

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

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

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

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

primrec *vars-of-prop*:: *'v propo* ⇒ *'v set* **where**
$\mathit{vars\text{-}of\text{-}prop}\ \mathit{FT} = \{\}\ |$
$\mathit{vars\text{-}of\text{-}prop}\ \mathit{FF} = \{\}\ |$
$\mathit{vars\text{-}of\text{-}prop}\ (\mathit{FVar}\ x) = \{x\}\ |$
$\mathit{vars\text{-}of\text{-}prop}\ (\mathit{FNot}\ \varphi) = \mathit{vars\text{-}of\text{-}prop}\ \varphi\ |$
$\mathit{vars\text{-}of\text{-}prop}\ (\mathit{FAnd}\ \varphi\ \psi) = \mathit{vars\text{-}of\text{-}prop}\ \varphi \cup \mathit{vars\text{-}of\text{-}prop}\ \psi\ |$
$\mathit{vars\text{-}of\text{-}prop}\ (\mathit{FOr}\ \varphi\ \psi) = \mathit{vars\text{-}of\text{-}prop}\ \varphi \cup \mathit{vars\text{-}of\text{-}prop}\ \psi\ |$
$\mathit{vars\text{-}of\text{-}prop}\ (\mathit{FImp}\ \varphi\ \psi) = \mathit{vars\text{-}of\text{-}prop}\ \varphi \cup \mathit{vars\text{-}of\text{-}prop}\ \psi\ |$
$\mathit{vars\text{-}of\text{-}prop}\ (\mathit{FEq}\ \varphi\ \psi) = \mathit{vars\text{-}of\text{-}prop}\ \varphi \cup \mathit{vars\text{-}of\text{-}prop}\ \psi$

lemma *vars-of-prop-incl-conn*:
fixes $\xi\ \xi'$ :: *'v propo list* **and** ψ :: *'v propo* **and** c :: *'v connective*
assumes *corr*: $\mathit{wf\text{-}conn}\ c\ l$ **and** *incl*: $\psi \in \mathit{set}\ l$

```
  shows vars-of-prop ψ ⊆ vars-of-prop (conn c l)
⟨proof⟩
```

The set of variables is compatible with the subformula order.

```
lemma subformula-vars-of-prop:
  φ ≼ ψ ⟹ vars-of-prop φ ⊆ 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) ≤ prop-size φ

  ⟨proof⟩
```

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

```
inductive path-to :: sign list ⇒ 'v propo ⇒ 'v propo ⇒ bool where
path-to-refl[intro]: path-to [] φ φ |
path-to-l: c∈binary-connectives ∨ c = CNot ⟹ wf-conn c (φ#l) ⟹ path-to p φ φ'⟹
  path-to (L#p) (conn c (φ#l)) φ' |
path-to-r: c∈binary-connectives ⟹ wf-conn c (ψ#φ#[]) ⟹ path-to p φ φ' ⟹
  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' ⟹ φ' ⪯ φ
  ⟨proof⟩
```

```
lemma subformula-path-exists:
  fixes φ φ':: 'v propo
  shows φ' ⪯ φ ⟹ ∃ p. path-to p φ φ'
⟨proof⟩
```

```
fun replace-at :: sign list ⇒ 'v propo ⇒ 'v propo ⇒ '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 ⇒ bool) ⇒ 'v propo ⇒ bool (infix ⊨ 50) where
𝒜 ⊨ FT = True |
𝒜 ⊨ FF = False |
𝒜 ⊨ FVar v = (𝒜 v) |
𝒜 ⊨ FNot φ = (¬(𝒜⊨ φ)) |
𝒜 ⊨ FAnd φ_1 φ_2 = (𝒜⊨φ_1 ∧ 𝒜⊨φ_2) |
𝒜 ⊨ FOr φ_1 φ_2 = (𝒜⊨φ_1 ∨ 𝒜⊨φ_2) |
𝒜 ⊨ FImp φ_1 φ_2 = (𝒜⊨φ_1 ⟶ 𝒜⊨φ_2) |
𝒜 ⊨ FEq φ_1 φ_2 = (𝒜⊨φ_1 ⟷ 𝒜 ⊨φ_2)
```

```
definition evalf (infix ⊨f 50) where
evalf φ ψ = (∀ A. A ⊨ φ ⟶ A ⊨ ψ)
```

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

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

A shorter proof:

lemma $\varphi \models\!f\ \psi \longleftrightarrow (\forall A.\ A\models \mathit{FImp}\ \varphi\ \psi)$
  ⟨*proof*⟩

definition *same-over-set*:: $('v \Rightarrow \mathit{bool}) \Rightarrow ('v \Rightarrow \mathit{bool}) \Rightarrow\ 'v\ \mathit{set} \Rightarrow \mathit{bool}$ **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