

Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

April 25, 2020

Contents

1	Definition of Entailment	5
1.1	Partial Herbrand Interpretation	5
1.1.1	More Literals	5
1.1.2	Clauses	6
1.1.3	Partial Interpretations	6
1.1.4	Subsumptions	20
1.1.5	Removing Duplicates	21
1.1.6	Set of all Simple Clauses	21
1.1.7	Experiment: Expressing the Entailments as Locales	22
1.1.8	Entailment to be extended	23
1.2	Partial Annotated Herbrand Interpretation	24
1.2.1	Decided Literals	24
1.2.2	Backtracking	29
1.2.3	Decomposition with respect to the First Decided Literals	30
1.2.4	Negation of a Clause	34
1.2.5	Other	37
1.2.6	Extending Entailments to multisets	39
1.2.7	More Lemmas	40
1.2.8	Negation of annotated clauses	40
1.3	Bridging of total and partial Herbrand interpretation	42
2	Normalisation	45
2.1	Logics	45
2.1.1	Definition and Abstraction	45
2.1.2	Properties of the Abstraction	46
2.1.3	Subformulas and Properties	48
2.1.4	Positions	50
2.2	Semantics over the Syntax	51

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

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

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

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

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

lemma *distinct-mset-atm-ofD*:
 $\langle \text{distinct-mset } (\text{atm-of } '\# \text{ mset } xc) \implies \text{distinct } xc \rangle$
<proof>

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

lemma *lit-in-set-iff-atm*:

$\langle \text{NO-MATCH } (Pos\ x)\ l \implies \text{NO-MATCH } (Neg\ x)\ l \implies$
 $l \in M \longleftrightarrow (\exists l'. (l = Pos\ l' \wedge Pos\ l' \in M) \vee (l = Neg\ l' \wedge Neg\ l' \in M)) \rangle$
 $\langle \text{proof} \rangle$

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 $\neg L$).

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

lemma *lit-eq-Neg-Pos-iff*:

$\langle x \neq Neg\ (atm-of\ x) \longleftrightarrow is-pos\ x \rangle$
 $\langle x \neq Pos\ (atm-of\ x) \longleftrightarrow is-neg\ x \rangle$
 $\langle \neg x \neq Neg\ (atm-of\ x) \longleftrightarrow is-neg\ x \rangle$
 $\langle \neg x \neq Pos\ (atm-of\ x) \longleftrightarrow is-pos\ x \rangle$
 $\langle Neg\ (atm-of\ x) \neq x \longleftrightarrow is-pos\ x \rangle$
 $\langle Pos\ (atm-of\ x) \neq x \longleftrightarrow is-neg\ x \rangle$
 $\langle Neg\ (atm-of\ x) \neq \neg x \longleftrightarrow is-neg\ x \rangle$
 $\langle Pos\ (atm-of\ x) \neq \neg x \longleftrightarrow is-pos\ x \rangle$
 $\langle \text{proof} \rangle$

1.1.2 Clauses

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

type-synonym *'v clause-set* = *'v clause set*

type-synonym *'v clauses* = *'v clause multiset*

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

1.1.3 Partial Interpretations

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

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

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

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

lemma *Ex-consistent-interp*: $\langle \text{Ex consistent-interp} \rangle$
 $\langle \text{proof} \rangle$

lemma *consistent-interp-subset*:

assumes

$A \subseteq B$ **and**

consistent-interp B
shows *consistent-interp A*
 ⟨proof⟩

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

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

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

lemma *consistent-interp-unionD*: $\langle \text{consistent-interp } (I \cup I') \implies \text{consistent-interp } I' \rangle$
 ⟨proof⟩

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

lemma (**in** $-$) *distinct-consistent-distinct-atm*:
 $\langle \text{distinct } M \implies \text{consistent-interp } (\text{set } M) \implies \text{distinct-mset } (\text{atm-of } \# \text{ mset } M) \rangle$
 ⟨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**
 $\text{atms-of-ms } \psi s = \bigcup (\text{atms-of } ' \psi s)$

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

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

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

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

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

lemma *atms-of-ms-mono*:

$A \subseteq B \implies \text{atms-of-ms } A \subseteq \text{atms-of-ms } B$
 $\langle \text{proof} \rangle$

lemma *atms-of-ms-finite[simp]*:
 $\text{finite } \psi s \implies \text{finite } (\text{atms-of-ms } \psi s)$
 $\langle \text{proof} \rangle$

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

lemma *atms-of-ms-insert[simp]*:
 $\text{atms-of-ms } (\text{insert } \psi s \chi s) = \text{atms-of-ms } \psi s \cup \text{atms-of-ms } \chi s$
 $\langle \text{proof} \rangle$

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

lemma *atms-of-atms-of-ms-mono[simp]*:
 $A \in \psi \implies \text{atms-of-ms } A \subseteq \text{atms-of-ms } \psi$
 $\langle \text{proof} \rangle$

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

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

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

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

lemma *in-implies-atm-of-on-atms-of-ms*:
assumes $L \in \# C$ **and** $C \in N$
shows $\text{atm-of } L \in \text{atms-of-ms } N$
 $\langle \text{proof} \rangle$

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

lemma *in-m-in-literals*:
assumes $\text{add-mset } A D \in \psi s$
shows $\text{atm-of } A \in \text{atms-of-ms } \psi s$
 $\langle \text{proof} \rangle$

lemma *atms-of-s-union[simp]*:
 $\text{atms-of-s } (Ia \cup Ib) = \text{atms-of-s } Ia \cup \text{atms-of-s } Ib$
 $\langle \text{proof} \rangle$

lemma *atms-of-s-single*[simp]:

atms-of-s {*L*} = {*atm-of* *L*}

⟨*proof*⟩

lemma *atms-of-s-insert*[simp]:

atms-of-s (*insert* *L* *Ib*) = {*atm-of* *L*} ∪ *atms-of-s* *Ib*

⟨*proof*⟩

lemma *in-atms-of-s-decomp*[iff]:

P ∈ *atms-of-s* *I* ⟷ (*Pos* *P* ∈ *I* ∨ *Neg* *P* ∈ *I*) (**is** ?*P* ⟷ ?*Q*)

⟨*proof*⟩

lemma *atm-of-in-atm-of-set-in-uminus*:

atm-of *L'* ∈ *atm-of* ' *B* ⟹ *L'* ∈ *B* ∨ − *L'* ∈ *B*

⟨*proof*⟩

lemma *finite-atms-of-s*[simp]:

⟨*finite* *M* ⟹ *finite* (*atms-of-s* *M*)

⟨*proof*⟩

lemma

atms-of-s-empty [simp]:

⟨*atms-of-s* {} = {}⟩ **and**

atms-of-s-empty-iff[simp]:

⟨*atms-of-s* *x* = {} ⟷ *x* = {}⟩

⟨*proof*⟩

Totality

definition *total-over-set* :: 'a *partial-interp* ⇒ 'a *set* ⇒ bool **where**

total-over-set *I* *S* = (∀ *l* ∈ *S*. *Pos* *l* ∈ *I* ∨ *Neg* *l* ∈ *I*)

definition *total-over-m* :: 'a *literal set* ⇒ 'a *clause set* ⇒ bool **where**

total-over-m *I* *ψs* = *total-over-set* *I* (*atms-of-ms* *ψs*)

lemma *total-over-set-empty*[simp]:

total-over-set *I* {}

⟨*proof*⟩

lemma *total-over-m-empty*[simp]:

total-over-m *I* {}

⟨*proof*⟩

lemma *total-over-set-single*[iff]:

total-over-set *I* {*L*} ⟷ (*Pos* *L* ∈ *I* ∨ *Neg* *L* ∈ *I*)

⟨*proof*⟩

lemma *total-over-set-insert*[iff]:

total-over-set *I* (*insert* *L* *Ls*) ⟷ ((*Pos* *L* ∈ *I* ∨ *Neg* *L* ∈ *I*) ∧ *total-over-set* *I* *Ls*)

⟨*proof*⟩

lemma *total-over-set-union*[iff]:

total-over-set *I* (*Ls* ∪ *Ls'*) ⟷ (*total-over-set* *I* *Ls* ∧ *total-over-set* *I* *Ls'*)

⟨*proof*⟩

lemma *total-over-m-subset*:

$A \subseteq B \implies \text{total-over-m } I B \implies \text{total-over-m } I A$
 $\langle \text{proof} \rangle$

lemma *total-over-m-sum[iff]*:

shows $\text{total-over-m } I \{C + D\} \longleftrightarrow (\text{total-over-m } I \{C\} \wedge \text{total-over-m } I \{D\})$
 $\langle \text{proof} \rangle$

lemma *total-over-m-union[iff]*:

$\text{total-over-m } I (A \cup B) \longleftrightarrow (\text{total-over-m } I A \wedge \text{total-over-m } I B)$
 $\langle \text{proof} \rangle$

lemma *total-over-m-insert[iff]*:

$\text{total-over-m } I (\text{insert } a A) \longleftrightarrow (\text{total-over-set } I (\text{atms-of } a) \wedge \text{total-over-m } I A)$
 $\langle \text{proof} \rangle$

lemma *total-over-m-extension*:

fixes $I :: 'v \text{ literal set}$ **and** $A :: 'v \text{ clause-set}$

assumes $\text{total}: \text{total-over-m } I A$

shows $\exists I'. \text{total-over-m } (I \cup I') (A \cup B)$

$\wedge (\forall x \in I'. \text{atm-of } x \in \text{atms-of-ms } B \wedge \text{atm-of } x \notin \text{atms-of-ms } A)$

$\langle \text{proof} \rangle$

lemma *total-over-m-consistent-extension*:

fixes $I :: 'v \text{ literal set}$ **and** $A :: 'v \text{ clause-set}$

assumes

$\text{total}: \text{total-over-m } I A$ **and**

$\text{cons}: \text{consistent-interp } I$

shows $\exists I'. \text{total-over-m } (I \cup I') (A \cup B)$

$\wedge (\forall x \in I'. \text{atm-of } x \in \text{atms-of-ms } B \wedge \text{atm-of } x \notin \text{atms-of-ms } A) \wedge \text{consistent-interp } (I \cup I')$

$\langle \text{proof} \rangle$

lemma *total-over-set-atms-of-m[simp]*:

$\text{total-over-set } I a (\text{atms-of-s } I a)$

$\langle \text{proof} \rangle$

lemma *total-over-set-literal-defined*:

assumes $\text{add-mset } A D \in \psi s$

and $\text{total-over-set } I (\text{atms-of-ms } \psi s)$

shows $A \in I \vee -A \in I$

$\langle \text{proof} \rangle$

lemma *tot-over-m-remove*:

assumes $\text{total-over-m } (I \cup \{L\}) \{\psi\}$

and $L: L \notin \# \psi \text{ } -L \notin \# \psi$

shows $\text{total-over-m } I \{\psi\}$

$\langle \text{proof} \rangle$

lemma *total-union*:

assumes $\text{total-over-m } I \psi$

shows $\text{total-over-m } (I \cup I') \psi$

$\langle \text{proof} \rangle$

lemma *total-union-2*:

assumes $\text{total-over-m } I \psi$

and $\text{total-over-m } I' \psi'$

shows *total-over-m* ($I \cup I'$) ($\psi \cup \psi'$)
 $\langle \text{proof} \rangle$

lemma *total-over-m-alt-def*: $\langle \text{total-over-m } I \ S \longleftrightarrow \text{atms-of-ms } S \subseteq \text{atms-of-s } I \rangle$
 $\langle \text{proof} \rangle$

lemma *total-over-set-alt-def*: $\langle \text{total-over-set } M \ A \longleftrightarrow A \subseteq \text{atms-of-s } M \rangle$
 $\langle \text{proof} \rangle$

Interpretations

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

lemma *true-cls-empty*[*iff*]: $\neg I \models \{\#\}$
 $\langle \text{proof} \rangle$

lemma *true-cls-singleton*[*iff*]: $I \models \{\#L\# \} \longleftrightarrow I \models_l L$
 $\langle \text{proof} \rangle$

lemma *true-cls-add-mset*[*iff*]: $I \models \text{add-mset } a \ D \longleftrightarrow a \in I \vee I \models D$
 $\langle \text{proof} \rangle$

lemma *true-cls-union*[*iff*]: $I \models C + D \longleftrightarrow I \models C \vee I \models D$
 $\langle \text{proof} \rangle$

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

lemma *true-cls-mono-leD*[*dest*]: $A \subseteq \# \ B \Longrightarrow I \models A \Longrightarrow I \models B$
 $\langle \text{proof} \rangle$

lemma
assumes $I \models \psi$
shows
true-cls-union-increase[*simp*]: $I \cup I' \models \psi$ **and**
true-cls-union-increase'[*simp*]: $I' \cup I \models \psi$
 $\langle \text{proof} \rangle$

lemma *true-cls-mono-set-mset-l*:
assumes $A \models \psi$
and $A \subseteq B$
shows $B \models \psi$
 $\langle \text{proof} \rangle$

lemma *true-cls-replicate-mset*[*iff*]: $I \models \text{replicate-mset } n \ L \longleftrightarrow n \neq 0 \wedge I \models_l L$
 $\langle \text{proof} \rangle$

lemma *true-cls-empty-entails*[*iff*]: $\neg \{\} \models N$
 $\langle \text{proof} \rangle$

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

definition *true-clss* :: 'a partial-interp \Rightarrow 'a clause-set \Rightarrow bool (**infix** \models_s 50) **where**
 $I \models_s CC \longleftrightarrow (\forall C \in CC. I \models C)$

lemma *true-clss-empty[simp]*: $I \models_s \{\}$
 $\langle proof \rangle$

lemma *true-clss-singleton[iff]*: $I \models_s \{C\} \longleftrightarrow I \models C$
 $\langle proof \rangle$

lemma *true-clss-empty-entails-empty[iff]*: $\{\} \models_s N \longleftrightarrow N = \{\}$
 $\langle proof \rangle$

lemma *true-clss-insert-l [simp]*:
 $M \models A \implies insert\ L\ M \models A$
 $\langle proof \rangle$

lemma *true-clss-union[iff]*: $I \models_s CC \cup DD \longleftrightarrow I \models_s CC \wedge I \models_s DD$
 $\langle proof \rangle$

lemma *true-clss-insert[iff]*: $I \models_s insert\ C\ DD \longleftrightarrow I \models C \wedge I \models_s DD$
 $\langle proof \rangle$

lemma *true-clss-mono*: $DD \subseteq CC \implies I \models_s CC \implies I \models_s DD$
 $\langle proof \rangle$

lemma *true-clss-union-increase[simp]*:
assumes $I \models_s \psi$
shows $I \cup I' \models_s \psi$
 $\langle proof \rangle$

lemma *true-clss-union-increase'[simp]*:
assumes $I' \models_s \psi$
shows $I \cup I' \models_s \psi$
 $\langle proof \rangle$

lemma *true-clss-commute-l*:
 $(I \cup I' \models_s \psi) \longleftrightarrow (I' \cup I \models_s \psi)$
 $\langle proof \rangle$

lemma *model-remove[simp]*: $I \models_s N \implies I \models_s Set.remove\ a\ N$
 $\langle proof \rangle$

lemma *model-remove-minus[simp]*: $I \models_s N \implies I \models_s N - A$
 $\langle proof \rangle$

lemma *notin-vars-union-true-clss-true-clss*:
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 $I \models L$
 $\langle proof \rangle$

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 $I \cup I' \models_s L$

shows $I \models_s L$
 $\langle \text{proof} \rangle$

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

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

Satisfiability

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

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

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

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

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

lemma *satisfiable-def-min*:
satisfiable CC
 $\longleftrightarrow (\exists I. I \models_s CC \wedge \text{consistent-interp } I \wedge \text{total-over-m } I \ CC \wedge \text{atm-of } I = \text{atms-of-ms } CC)$
(is ?sat $\longleftrightarrow ?B$)
 $\langle \text{proof} \rangle$

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

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

lemma *unsatisfiable-mono*:
 $\langle N \subseteq N' \implies \text{unsatisfiable } N \implies \text{unsatisfiable } N' \rangle$
 $\langle \text{proof} \rangle$

Entailment for Multisets of Clauses

definition *true-cls-mset* :: 'a partial-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]*: $I \models_m \{\#\}$
 $\langle \text{proof} \rangle$

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*: $set\text{-}mset\ DD \subseteq set\text{-}mset\ CC \implies I \models_m CC \implies I \models_m DD$

$\langle proof \rangle$

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*]:

$I \models_m CC \implies I \cup J \models_m CC$

$\langle proof \rangle$

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 $consI$: $consistent\text{-}interp\ I$

and $disj$: $atms\text{-}of\text{-}s\ A \cap atms\text{-}of\text{-}s\ I = \{\}$

and $consA$: $consistent\text{-}interp\ A$

shows $consistent\text{-}interp\ (A \cup I)$

$\langle proof \rangle$

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

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

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

lemma *true-cls-mset-true-clss-iff*:
 $\langle \text{finite } C \implies I \models_m \text{mset-set } C \longleftrightarrow I \models_s C \rangle$
 $\langle I \models_m D \longleftrightarrow I \models_s \text{set-mset } D \rangle$
 $\langle \text{proof} \rangle$

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

lemma *tautology-Pos-Neg[intro]*:
assumes *Pos* $p \in \# A$ **and** *Neg* $p \in \# A$
shows *tautology* A
 $\langle \text{proof} \rangle$

lemma *tautology-minus[simp]*:
assumes $L \in \# A$ **and** $-L \in \# A$
shows *tautology* A
 $\langle \text{proof} \rangle$

lemma *tautology-exists-Pos-Neg*:
assumes *tautology* ψ
shows $\exists p. \text{Pos } p \in \# \psi \wedge \text{Neg } p \in \# \psi$
 $\langle \text{proof} \rangle$

lemma *tautology-decomp*:
 $\text{tautology } \psi \longleftrightarrow (\exists p. \text{Pos } p \in \# \psi \wedge \text{Neg } p \in \# \psi)$
 $\langle \text{proof} \rangle$

lemma *tautology-union-add-iff[simp]*:
 $\langle \text{tautology } (A \cup \# B) \longleftrightarrow \text{tautology } (A + B) \rangle$
 $\langle \text{proof} \rangle$

lemma *tautology-add-mset-union-add-iff[simp]*:
 $\langle \text{tautology } (\text{add-mset } L (A \cup \# B)) \longleftrightarrow \text{tautology } (\text{add-mset } L (A + B)) \rangle$
 $\langle \text{proof} \rangle$

lemma *not-tautology-minus*:

$\langle \neg \text{tautology } A \implies \neg \text{tautology } (A - B) \rangle$
 $\langle \text{proof} \rangle$

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *tautology-distinct-atm-iff*:

$\langle \text{distinct-mset } C \implies \text{tautology } C \longleftrightarrow \neg \text{distinct-mset } (\text{atm-of } \# \ C) \rangle$
 $\langle \text{proof} \rangle$

lemma *not-tautology-minusD*:

$\langle \text{tautology } (A - B) \implies \text{tautology } A \rangle$
 $\langle \text{proof} \rangle$

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**

$\psi \models_f \chi \longleftrightarrow (\forall I. \text{total-over-m } I (\{\psi\} \cup \{\chi\}) \longrightarrow \text{consistent-interp } I \longrightarrow I \models \psi \longrightarrow I \models \chi)$

definition *true-cls-clss* :: 'a clause \Rightarrow 'a clause-set \Rightarrow bool (**infix** \models_{fs} 49) **where**

$\psi \models_{fs} \chi \longleftrightarrow (\forall I. \text{total-over-m } I (\{\psi\} \cup \chi) \longrightarrow \text{consistent-interp } I \longrightarrow I \models \psi \longrightarrow I \models \chi)$

definition *true-clss-cls* :: 'a clause-set \Rightarrow 'a clause \Rightarrow bool (**infix** \models_p 49) **where**

$N \models_p \chi \longleftrightarrow (\forall I. \text{total-over-m } I (N \cup \{\chi\}) \longrightarrow \text{consistent-interp } I \longrightarrow I \models_s N \longrightarrow I \models \chi)$

definition *true-clss-clss* :: 'a clause-set \Rightarrow 'a clause-set \Rightarrow bool (**infix** \models_{ps} 49) **where**

$N \models_{ps} N' \longleftrightarrow (\forall I. \text{total-over-m } I (N \cup N') \longrightarrow \text{consistent-interp } I \longrightarrow I \models_s N \longrightarrow I \models_s N')$

lemma *true-cls-cls-refl[simp]*:

$A \models_f A$
 $\langle \text{proof} \rangle$

lemma *true-clss-cls-empty-empty[iff]*:

$\langle \{\} \models_p \{\# \} \longleftrightarrow \text{False} \rangle$
 $\langle \text{proof} \rangle$

lemma *true-cls-cls-insert-l[simp]*:

$a \models_f C \implies \text{insert } a \ A \models_p C$
 $\langle \text{proof} \rangle$

lemma *true-cls-clss-empty[iff]*:

$N \models_{fs} \{\}$
 $\langle \text{proof} \rangle$

lemma *true-prop-true-clause[iff]*:

$\{\varphi\} \models_p \psi \longleftrightarrow \varphi \models_f \psi$
 $\langle \text{proof} \rangle$

lemma *true-clss-clss-true-clss-cls[iff]*:

$N \models_{ps} \{\psi\} \longleftrightarrow N \models_p \psi$
 $\langle \text{proof} \rangle$

lemma *true-clss-clss-true-cls-clss[iff]*:

$\{\chi\} \models_{ps} \psi \longleftrightarrow \chi \models_{fs} \psi$
 $\langle \text{proof} \rangle$

lemma *true-clss-clss-empty[simp]*:

$N \models_{ps} \{\}$
 $\langle \text{proof} \rangle$

lemma *true-clss-cls-subset*:

$$A \subseteq B \implies A \models_p CC \implies B \models_p CC$$

$\langle \text{proof} \rangle$

This version of $\llbracket ?A \subseteq ?B; ?A \models_p ?CC \rrbracket \implies ?B \models_p ?CC$ is useful as intro rule.

lemma *(in -)true-clss-cls-subsetI*: $\langle I \models_p A \implies I \subseteq I' \implies I' \models_p A \rangle$

$\langle \text{proof} \rangle$

lemma *true-clss-cs-mono-l[simp]*:

$$A \models_p CC \implies A \cup B \models_p CC$$

$\langle \text{proof} \rangle$

lemma *true-clss-cs-mono-l2[simp]*:

$$B \models_p CC \implies A \cup B \models_p CC$$

$\langle \text{proof} \rangle$

lemma *true-clss-cls-mono-r[simp]*:

$$A \models_p CC \implies A \models_p CC + CC'$$

$\langle \text{proof} \rangle$

lemma *true-clss-cls-mono-r'[simp]*:

$$A \models_p CC' \implies A \models_p CC + CC'$$

$\langle \text{proof} \rangle$

lemma *true-clss-cls-mono-add-mset[simp]*:

$$A \models_p CC \implies A \models_p \text{add-mset } L \ CC$$

$\langle \text{proof} \rangle$

lemma *true-clss-clss-union-l[simp]*:

$$A \models_{ps} CC \implies A \cup B \models_{ps} CC$$

$\langle \text{proof} \rangle$

lemma *true-clss-clss-union-l-r[simp]*:

$$B \models_{ps} CC \implies A \cup B \models_{ps} CC$$

$\langle \text{proof} \rangle$

lemma *true-clss-cls-in[simp]*:

$$CC \in A \implies A \models_p CC$$

$\langle \text{proof} \rangle$

lemma *true-clss-cls-insert-l[simp]*:

$$A \models_p C \implies \text{insert } a \ A \models_p C$$

$\langle \text{proof} \rangle$

lemma *true-clss-clss-insert-l[simp]*:

$$A \models_{ps} C \implies \text{insert } a \ A \models_{ps} C$$

$\langle \text{proof} \rangle$

lemma *true-clss-clss-union-and[iff]*:

$$A \models_{ps} C \cup D \longleftrightarrow (A \models_{ps} C \wedge A \models_{ps} D)$$

$\langle \text{proof} \rangle$

lemma *true-clss-clss-insert[iff]*:

$$A \models_{ps} \text{insert } L \ Ls \longleftrightarrow (A \models_p L \wedge A \models_{ps} Ls)$$

$\langle \text{proof} \rangle$

lemma *true-clss-clss-subset*:

$A \subseteq B \implies A \models_{ps} CC \implies B \models_{ps} CC$
 $\langle proof \rangle$

Better suited as intro rule:

lemma *true-clss-clss-subsetI*:

$A \models_{ps} CC \implies A \subseteq B \implies B \models_{ps} CC$
 $\langle proof \rangle$

lemma *union-trus-clss-clss[simp]*: $A \cup B \models_{ps} B$

$\langle proof \rangle$

lemma *true-clss-clss-remove[simp]*:

$A \models_{ps} B \implies A \models_{ps} B - C$
 $\langle proof \rangle$

lemma *true-clss-clss-subsetE*:

$N \models_{ps} B \implies A \subseteq B \implies N \models_{ps} A$
 $\langle proof \rangle$

lemma *true-clss-clss-in-imp-true-clss-clss*:

assumes $N \models_{ps} U$
and $A \in U$
shows $N \models_p A$
 $\langle proof \rangle$

lemma *all-in-true-clss-clss*: $\forall x \in B. x \in A \implies A \models_{ps} B$

$\langle proof \rangle$

lemma *true-clss-clss-left-right*:

assumes $A \models_{ps} B$
and $A \cup B \models_{ps} M$
shows $A \models_{ps} M \cup B$
 $\langle proof \rangle$

lemma *true-clss-clss-generalise-true-clss-clss*:

$A \cup C \models_{ps} D \implies B \models_{ps} C \implies A \cup B \models_{ps} D$
 $\langle proof \rangle$

lemma *true-clss-clss-or-true-clss-clss-or-not-true-clss-clss-or*:

assumes $D: N \models_p \text{add-mset } (-L) \ D$
and $C: N \models_p \text{add-mset } L \ C$
shows $N \models_p D + C$
 $\langle proof \rangle$

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

$\langle proof \rangle$

lemma *true-clss-clss-sup-iff-add*: $N \models_p C \cup \# D \longleftrightarrow N \models_p C + D$

$\langle proof \rangle$

lemma *true-clss-clss-union-mset-true-clss-clss-or-not-true-clss-clss-or*:

assumes
 $D: N \models_p \text{add-mset } (-L) \ D$ **and**
 $C: N \models_p \text{add-mset } L \ C$

shows $N \models_p D \cup \# C$
 $\langle \text{proof} \rangle$

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

lemma *true-clss-mset-empty-iff[simp]*: $\langle \{ \} \models_m C \longleftrightarrow C = \{ \# \} \rangle$
 $\langle \text{proof} \rangle$

lemma *true-clss-mono-left*:
 $\langle I \models_s A \implies I \subseteq J \implies J \models_s A \rangle$
 $\langle \text{proof} \rangle$

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

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

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

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

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

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

lemma *true-clss-cls-true-clss-true-clss*:
assumes $\langle N \models_p C \rangle$
 $\langle I \models_s N \rangle$ **and**
 $\text{cons: } \langle \text{consistent-interp } I \rangle$ **and**
 $\text{tauto: } \langle \neg \text{tautology } C \rangle$
shows $\langle I \models C \rangle$
 $\langle \text{proof} \rangle$

1.1.4 Subsumptions

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

lemma *atms-of-replicate-mset-replicate-mset-uminus[simp]*:

$$\text{atms-of } (D - \text{replicate-mset } (\text{count } D \ L) \ L - \text{replicate-mset } (\text{count } D \ (-L)) \ (-L))$$

$$= \text{atms-of } D - \{\text{atm-of } L\}$$
 $\langle \text{proof} \rangle$

lemma *subsumption-chained*:

assumes

$\forall I. \text{total-over-m } I \ \{D\} \longrightarrow I \models D \longrightarrow I \models \varphi$ **and**

$C \subseteq\# D$

shows $(\forall I. \text{total-over-m } I \ \{C\} \longrightarrow I \models C \longrightarrow I \models \varphi) \vee \text{tautology } \varphi$

$\langle \text{proof} \rangle$

1.1.5 Removing Duplicates

lemma *tautology-remdups-mset[iff]*:

$\text{tautology } (\text{remdups-mset } C) \longleftrightarrow \text{tautology } C$

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma *true-clss-cl-remdups-mset[iff]*: $A \models_p \text{remdups-mset } C \longleftrightarrow A \models_p C$

$\langle \text{proof} \rangle$

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* :: $'v \text{ set} \Rightarrow 'v \text{ clause set}$ **where**

$\text{simple-clss } \text{atms} = \{C. \text{atms-of } C \subseteq \text{atms} \wedge \neg \text{tautology } C \wedge \text{distinct-mset } C\}$

lemma *simple-clss-empty[simp]*:

$\text{simple-clss } \{\} = \{\{\#\}\}$

$\langle \text{proof} \rangle$

lemma *simple-clss-insert*:

assumes $l \notin \text{atms}$

shows $\text{simple-clss } (\text{insert } l \ \text{atms}) =$

$((+) \ \{\#\text{Pos } l\# \}) \ ' (\text{simple-clss } \text{atms})$

$\cup ((+) \ \{\#\text{Neg } l\# \}) \ ' (\text{simple-clss } \text{atms})$

$\cup \text{simple-clss } \text{atms}(\text{is } ?I = ?U)$

$\langle \text{proof} \rangle$

lemma *simple-clss-finite*:

```

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

```

```

lemma simple-clsE:
assumes
   $x \in \text{simple-cls } atms$ 
shows  $atms\text{-of } x \subseteq atms \wedge \neg \text{tautology } x \wedge \text{distinct-mset } x$ 
⟨proof⟩

```

```

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

```

```

lemma simple-cls-card:
fixes atms :: 'v set
assumes finite atms
shows  $\text{card } (\text{simple-cls } atms) \leq (\mathcal{P}::nat) \wedge (\text{card } atms)$ 
⟨proof⟩

```

```

lemma simple-cls-mono:
assumes  $\text{incl}: atms \subseteq atms'$ 
shows  $\text{simple-cls } atms \subseteq \text{simple-cls } atms'$ 
⟨proof⟩

```

```

lemma distinct-mset-not-tautology-implies-in-simple-cls:
assumes  $\text{distinct-mset } \chi$  and  $\neg \text{tautology } \chi$ 
shows  $\chi \in \text{simple-cls } (atms\text{-of } \chi)$ 
⟨proof⟩

```

```

lemma simplified-in-simple-cls:
assumes  $\text{distinct-mset-set } \psi$  and  $\forall \chi \in \psi. \neg \text{tautology } \chi$ 
shows  $\psi \subseteq \text{simple-cls } (atms\text{-of-} ms \ \psi)$ 
⟨proof⟩

```

```

lemma simple-cls-element-mono:
 $\langle x \in \text{simple-cls } A \implies y \subseteq \# \ x \implies y \in \text{simple-cls } A \rangle$ 
⟨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]:  $I \neq \{\} \implies \text{insert } L \ I \models_e x \longleftrightarrow \{L\} \models_e x \vee I \models_e x$ 
  assumes entail-union[simp]:  $I \models_e A \implies I \cup I' \models_e A$ 
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]:

```

$I \models_{es} \{a\} \longleftrightarrow I \models_e a$
 $\langle proof \rangle$

lemma *entails-insert-l[simp]*:
 $M \models_{es} A \implies insert\ L\ M \models_{es} A$
 $\langle proof \rangle$

lemma *entails-union[iff]*: $I \models_{es} CC \cup DD \longleftrightarrow I \models_{es} CC \wedge I \models_{es} DD$
 $\langle proof \rangle$

lemma *entails-insert[iff]*: $I \models_{es} insert\ C\ DD \longleftrightarrow I \models_e C \wedge I \models_{es} DD$
 $\langle proof \rangle$

lemma *entails-insert-mono*: $DD \subseteq CC \implies I \models_{es} CC \implies I \models_{es} DD$
 $\langle proof \rangle$

lemma *entails-union-increase[simp]*:
assumes $I \models_{es} \psi$
shows $I \cup I' \models_{es} \psi$
 $\langle proof \rangle$

lemma *true-clss-commute-l*:
 $I \cup I' \models_{es} \psi \longleftrightarrow I' \cup I \models_{es} \psi$
 $\langle proof \rangle$

lemma *entails-remove[simp]*: $I \models_{es} N \implies I \models_{es} Set.remove\ a\ N$
 $\langle proof \rangle$

lemma *entails-remove-minus[simp]*: $I \models_{es} N \implies I \models_{es} N - A$
 $\langle proof \rangle$

end

interpretation *true-clss*: *entail true-clss*
 $\langle proof \rangle$

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 these 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 \Rightarrow 'a clause set \Rightarrow bool (**infix** \models_{sext} 49)

where

$I \models_{sext} N \longleftrightarrow (\forall J. I \subseteq J \longrightarrow consistent_interp\ J \longrightarrow total_over_m\ J\ N \longrightarrow J \models_s N)$

lemma *true-clss-imp-true-clss-ext*:
 $I \models_s N \implies I \models_{sext} N$
 $\langle proof \rangle$

lemma *true-clss-ext-decrease-right-remove-r*:
assumes $I \models_{sext} N$

shows $I \models_{\text{sext}} N - \{C\}$
 $\langle \text{proof} \rangle$

lemma *consistent-true-clss-ext-satisfiable*:
assumes *consistent-interp* I **and** $I \models_{\text{sext}} A$
shows *satisfiable* A
 $\langle \text{proof} \rangle$

lemma *not-consistent-true-clss-ext*:
assumes $\neg \text{consistent-interp } I$
shows $I \models_{\text{sext}} A$
 $\langle \text{proof} \rangle$

lemma *inj-on-Pos*: $\langle \text{inj-on Pos } A \rangle$ **and**
inj-on-Neg: $\langle \text{inj-on Neg } A \rangle$
 $\langle \text{proof} \rangle$

lemma *inj-on-uminus-lit*: $\langle \text{inj-on uminus } A \rangle$ **for** $A :: \langle 'a \text{ literal set} \rangle$
 $\langle \text{proof} \rangle$

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 $\langle 'v, 'w, 'mark \rangle \text{ annotated-lit} =$
is-decided: *Decided* (*lit-dec*: $'v$) |
is-proped: *Propagated* (*lit-prop*: $'w$) (*mark-of*: $'mark$)

type-synonym $\langle 'v, 'w, 'mark \rangle \text{ annotated-lits} = \langle \langle 'v, 'w, 'mark \rangle \text{ annotated-lit list} \rangle$

type-synonym $\langle 'v, 'mark \rangle \text{ ann-lit} = \langle \langle 'v \text{ literal}, 'v \text{ literal}, 'mark \rangle \text{ annotated-lit} \rangle$

lemma *ann-lit-list-induct*[*case-names Nil Decided Propagated*]:
assumes
 $\langle P [] \rangle$ **and**
 $\langle \bigwedge L \text{ xs. } P \text{ xs} \implies P (\text{Decided } L \# \text{ xs}) \rangle$ **and**
 $\langle \bigwedge L m \text{ xs. } P \text{ xs} \implies P (\text{Propagated } L m \# \text{ xs}) \rangle$
shows $\langle P \text{ xs} \rangle$
 $\langle \text{proof} \rangle$

lemma *is-decided-ex-Decided*:
 $\langle \text{is-decided } L \implies (\bigwedge K. L = \text{Decided } K \implies P) \implies P \rangle$
 $\langle \text{proof} \rangle$

lemma *is-propedE*: $\langle is-proped\ L \implies (\bigwedge K\ C.\ L = Propagated\ K\ C \implies P) \implies P \rangle$
 $\langle proof \rangle$

lemma *is-decided-no-proped-iff*: $\langle is-decided\ L \longleftrightarrow \neg is-proped\ L \rangle$
 $\langle proof \rangle$

lemma *not-is-decidedE*:
 $\langle \neg is-decided\ E \implies (\bigwedge K\ C.\ E = Propagated\ K\ C \implies thesis) \implies thesis \rangle$
 $\langle proof \rangle$

type-synonym $\langle 'v, 'm \rangle\ ann-lits = \langle 'v, 'm \rangle\ ann-lit\ list$

fun *lit-of* :: $\langle ('a, 'a, 'mark)\ annotated-lit \Rightarrow 'a \rangle$ **where**
 $\langle lit-of\ (Decided\ L) = L \mid$
 $\langle lit-of\ (Propagated\ L\ -) = L \rangle$

definition *lits-of* :: $\langle ('a, 'b)\ ann-lit\ set \Rightarrow 'a\ literal\ set \rangle$ **where**
 $\langle lits-of\ Ls = lit-of\ 'Ls \rangle$

abbreviation *lits-of-l* :: $\langle ('a, 'b)\ ann-lits \Rightarrow 'a\ literal\ set \rangle$ **where**
 $\langle lits-of-l\ Ls \equiv lits-of\ (set\ Ls) \rangle$

lemma *lits-of-l-empty[simp]*:
 $\langle lits-of\ \{\} = \{\} \rangle$
 $\langle proof \rangle$

lemma *lits-of-insert[simp]*:
 $\langle lits-of\ (insert\ L\ Ls) = insert\ (lit-of\ L)\ (lits-of\ Ls) \rangle$
 $\langle proof \rangle$

lemma *lits-of-l-Un[simp]*:
 $\langle lits-of\ (l \cup l') = lits-of\ l \cup lits-of\ l' \rangle$
 $\langle proof \rangle$

lemma *finite-lits-of-def[simp]*:
 $\langle finite\ (lits-of-l\ L) \rangle$
 $\langle proof \rangle$

abbreviation *unmark* **where**
 $\langle unmark \equiv (\lambda a.\ \{\#lit-of\ a\#\}) \rangle$

abbreviation *unmark-s* **where**
 $\langle unmark-s\ M \equiv unmark\ 'M \rangle$

abbreviation *unmark-l* **where**
 $\langle unmark-l\ M \equiv unmark-s\ (set\ M) \rangle$

lemma *atms-of-ms-lambda-lit-of-is-atm-of-lit-of[simp]*:
 $\langle atms-of-ms\ (unmark-l\ M') = atm-of\ 'lits-of-l\ M' \rangle$
 $\langle proof \rangle$

lemma *lits-of-l-empty-is-empty[iff]*:
 $\langle lits-of-l\ M = \{\} \longleftrightarrow M = [] \rangle$
 $\langle proof \rangle$

lemma *in-unmark-l-in-lits-of-l-iff*: $\langle \{ \#L\# \} \in \text{unmark-l } M \longleftrightarrow L \in \text{lits-of-l } M \rangle$
 $\langle \text{proof} \rangle$

lemma *atm-lit-of-set-lits-of-l*:
 $\langle \lambda l. \text{atm-of } (\text{lit-of } l) \text{ ' set } xs = \text{atm-of ' lits-of-l } xs \rangle$
 $\langle \text{proof} \rangle$

Entailment

definition *true-annot* :: $\langle ('a, 'm) \text{ ann-lits} \Rightarrow 'a \text{ clause} \Rightarrow \text{bool} \rangle$ (**infix** \models_a 49) **where**
 $\langle I \models_a C \longleftrightarrow (\text{lits-of-l } I) \models C \rangle$

definition *true-annots* :: $\langle ('a, 'm) \text{ ann-lits} \Rightarrow 'a \text{ clause-set} \Rightarrow \text{bool} \rangle$ (**infix** \models_{as} 49) **where**
 $\langle I \models_{as} CC \longleftrightarrow (\forall C \in CC. I \models_a C) \rangle$

lemma *true-annot-empty-model[simp]*:
 $\langle \neg[] \models_a \psi \rangle$
 $\langle \text{proof} \rangle$

lemma *true-annot-empty[simp]*:
 $\langle \neg I \models_a \{ \# \} \rangle$
 $\langle \text{proof} \rangle$

lemma *empty-true-annots-def[iff]*:
 $\langle [] \models_{as} \psi \longleftrightarrow \psi = \{ \} \rangle$
 $\langle \text{proof} \rangle$

lemma *true-annots-empty[simp]*:
 $\langle I \models_{as} \{ \} \rangle$
 $\langle \text{proof} \rangle$

lemma *true-annots-single-true-annot[iff]*:
 $\langle I \models_{as} \{ C \} \longleftrightarrow I \models_a C \rangle$
 $\langle \text{proof} \rangle$

lemma *true-annot-insert-l[simp]*:
 $\langle M \models_a A \Longrightarrow L \# M \models_a A \rangle$
 $\langle \text{proof} \rangle$

lemma *true-annots-insert-l [simp]*:
 $\langle M \models_{as} A \Longrightarrow L \# M \models_{as} A \rangle$
 $\langle \text{proof} \rangle$

lemma *true-annots-union[iff]*:
 $\langle M \models_{as} A \cup B \longleftrightarrow (M \models_{as} A \wedge M \models_{as} B) \rangle$
 $\langle \text{proof} \rangle$

lemma *true-annots-insert[iff]*:
 $\langle M \models_{as} \text{insert } a \ A \longleftrightarrow (M \models_a a \wedge M \models_{as} A) \rangle$
 $\langle \text{proof} \rangle$

lemma *true-annot-append-l*:
 $\langle M \models_a A \Longrightarrow M' @ M \models_a A \rangle$
 $\langle \text{proof} \rangle$

lemma *true-annots-append-l*:

$\langle M \models_{as} A \implies M' @ M \models_{as} A \rangle$
 $\langle proof \rangle$

Link between \models_{as} and \models_s :

lemma *true-annots-true-cls*:

$\langle I \models_{as} CC \longleftrightarrow lits-of-l\ I \models_s CC \rangle$
 $\langle proof \rangle$

lemma *in-lit-of-true-annot*:

$\langle a \in lits-of-l\ M \longleftrightarrow M \models_a \{ \#a\# \} \rangle$
 $\langle proof \rangle$

lemma *true-annot-lit-of-notin-skip*:

$\langle L \# M \models_a A \implies lit-of\ L \notin \# A \implies M \models_a A \rangle$
 $\langle proof \rangle$

lemma *true-clss-singleton-lit-of-implies-incl*:

$\langle I \models_s unmark-l\ MLs \implies lits-of-l\ MLs \subseteq I \rangle$
 $\langle proof \rangle$

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

$\langle MLs \models_a \psi \implies set\ (map\ unmark\ MLs) \models_p \psi \rangle$
 $\langle proof \rangle$

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

$\langle MLs \models_{as} \psi \implies set\ (map\ unmark\ MLs) \models_{ps} \psi \rangle$
 $\langle proof \rangle$

lemma *true-annots-decided-true-cls*[*iff*]:

$\langle map\ Decided\ M \models_{as} N \longleftrightarrow set\ M \models_s N \rangle$
 $\langle proof \rangle$

lemma *true-annot-singleton*[*iff*]: $\langle M \models_a \{ \#L\# \} \longleftrightarrow L \in lits-of-l\ M \rangle$

$\langle proof \rangle$

lemma *true-annots-true-clss-clss*:

$\langle A \models_{as} \Psi \implies unmark-l\ A \models_{ps} \Psi \rangle$
 $\langle proof \rangle$

lemma *true-annot-commute*:

$\langle M @ M' \models_a D \longleftrightarrow M' @ M \models_a D \rangle$
 $\langle proof \rangle$

lemma *true-annots-commute*:

$\langle M @ M' \models_{as} D \longleftrightarrow M' @ M \models_{as} D \rangle$
 $\langle proof \rangle$

lemma *true-annot-mono*[*dest*]:

$\langle set\ I \subseteq set\ I' \implies I \models_a N \implies I' \models_a N \rangle$
 $\langle proof \rangle$

lemma *true-annots-mono*:

$\langle set\ I \subseteq set\ I' \implies I \models_{as} N \implies I' \models_{as} N \rangle$
 $\langle proof \rangle$

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* :: $\langle ('a \text{ literal}, 'a \text{ literal}, 'm) \text{ annotated-lits} \Rightarrow 'a \text{ literal} \Rightarrow \text{bool} \rangle$

where

$\langle \text{defined-lit } I \ L \longleftrightarrow (Decided \ L \in \text{set } I) \vee (\exists P. \text{Propagated } L \ P \in \text{set } I) \vee (Decided \ (-L) \in \text{set } I) \vee (\exists P. \text{Propagated } (-L) \ P \in \text{set } I) \rangle$

abbreviation *undefined-lit* :: $\langle ('a \text{ literal}, 'a \text{ literal}, 'm) \text{ annotated-lits} \Rightarrow 'a \text{ literal} \Rightarrow \text{bool} \rangle$

where $\langle \text{undefined-lit } I \ L \equiv \neg \text{defined-lit } I \ L \rangle$

lemma *defined-lit-rev[simp]*:

$\langle \text{defined-lit } (\text{rev } M) \ L \longleftrightarrow \text{defined-lit } M \ L \rangle$
 $\langle \text{proof} \rangle$

lemma *atm-imp-decided-or-proped*:

assumes $\langle x \in \text{set } I \rangle$

shows

$\langle (Decided \ (- \text{lit-of } x) \in \text{set } I) \vee (Decided \ (\text{lit-of } x) \in \text{set } I) \vee (\exists l. \text{Propagated } (- \text{lit-of } x) \ l \in \text{set } I) \vee (\exists l. \text{Propagated } (\text{lit-of } x) \ l \in \text{set } I) \rangle$
 $\langle \text{proof} \rangle$

lemma *literal-is-lit-of-decided*:

assumes $\langle L = \text{lit-of } x \rangle$

shows $\langle (x = Decided \ L) \vee (\exists l'. x = \text{Propagated } L \ l') \rangle$
 $\langle \text{proof} \rangle$

lemma *true-annot-iff-decided-or-true-lit*:

$\langle \text{defined-lit } I \ L \longleftrightarrow (\text{lits-of-l } I \models L \vee \text{lits-of-l } I \models -L) \rangle$
 $\langle \text{proof} \rangle$

lemma *consistent-inter-true-annots-satisfiable*:

$\langle \text{consistent-interp } (\text{lits-of-l } I) \implies I \models_{\text{as}} N \implies \text{satisfiable } N \rangle$
 $\langle \text{proof} \rangle$

lemma *defined-lit-map*:

$\langle \text{defined-lit } Ls \ L \longleftrightarrow \text{atm-of } L \in (\lambda l. \text{atm-of } (\text{lit-of } l)) \text{ ` set } Ls \rangle$
 $\langle \text{proof} \rangle$

lemma *defined-lit-uminus[iff]*:

$\langle \text{defined-lit } I \ (-L) \longleftrightarrow \text{defined-lit } I \ L \rangle$
 $\langle \text{proof} \rangle$

lemma *Decided-Propagated-in-iff-in-lits-of-l*:

$\langle \text{defined-lit } I \ L \longleftrightarrow (L \in \text{lits-of-l } I \vee -L \in \text{lits-of-l } I) \rangle$
 $\langle \text{proof} \rangle$

lemma *consistent-add-undefined-lit-consistent[simp]*:

assumes

$\langle \text{consistent-interp } (\text{lits-of-l } Ls) \rangle$ **and**
 $\langle \text{undefined-lit } Ls \ L \rangle$

shows $\langle \text{consistent-interp } (\text{insert } L \text{ (lits-of-l } Ls)) \rangle$
 $\langle \text{proof} \rangle$

lemma *decided-empty[simp]*:
 $\langle \neg \text{defined-lit } [] \ L \rangle$
 $\langle \text{proof} \rangle$

lemma *undefined-lit-single[iff]*:
 $\langle \text{defined-lit } [L] \ K \longleftrightarrow \text{atm-of } (\text{lit-of } L) = \text{atm-of } K \rangle$
 $\langle \text{proof} \rangle$

lemma *undefined-lit-cons[iff]*:
 $\langle \text{undefined-lit } (L \# M) \ K \longleftrightarrow \text{atm-of } (\text{lit-of } L) \neq \text{atm-of } K \wedge \text{undefined-lit } M \ K \rangle$
 $\langle \text{proof} \rangle$

lemma *undefined-lit-append[iff]*:
 $\langle \text{undefined-lit } (M @ M') \ K \longleftrightarrow \text{undefined-lit } M \ K \wedge \text{undefined-lit } M' \ K \rangle$
 $\langle \text{proof} \rangle$

lemma *defined-lit-cons*:
 $\langle \text{defined-lit } (L \# M) \ K \longleftrightarrow \text{atm-of } (\text{lit-of } L) = \text{atm-of } K \vee \text{defined-lit } M \ K \rangle$
 $\langle \text{proof} \rangle$

lemma *defined-lit-append*:
 $\langle \text{defined-lit } (M @ M') \ K \longleftrightarrow \text{defined-lit } M \ K \vee \text{defined-lit } M' \ K \rangle$
 $\langle \text{proof} \rangle$

lemma *in-lits-of-l-defined-litD*: $\langle L\text{-max} \in \text{lits-of-l } M \implies \text{defined-lit } M \ L\text{-max} \rangle$
 $\langle \text{proof} \rangle$

lemma *undefined-notin*: $\langle \text{undefined-lit } M \ (\text{lit-of } x) \implies x \notin \text{set } M \rangle$ **for** $x \ M$
 $\langle \text{proof} \rangle$

lemma *uminus-lits-of-l-definedD*:
 $\langle -x \in \text{lits-of-l } M \implies \text{defined-lit } M \ x \rangle$
 $\langle \text{proof} \rangle$

lemma *defined-lit-Neg-Pos-iff*:
 $\langle \text{defined-lit } M \ (\text{Neg } A) \longleftrightarrow \text{defined-lit } M \ (\text{Pos } A) \rangle$
 $\langle \text{proof} \rangle$

lemma *defined-lit-Pos-atm-iff[simp]*:
 $\langle \text{defined-lit } M1 \ (\text{Pos } (\text{atm-of } x)) \longleftrightarrow \text{defined-lit } M1 \ x \rangle$
 $\langle \text{proof} \rangle$

lemma *defined-lit-mono*:
 $\langle \text{defined-lit } M2 \ L \implies \text{set } M2 \subseteq \text{set } M3 \implies \text{defined-lit } M3 \ L \rangle$
 $\langle \text{proof} \rangle$

lemma *defined-lit-nth*:
 $\langle n < \text{length } M2 \implies \text{defined-lit } M2 \ (\text{lit-of } (M2 ! n)) \rangle$
 $\langle \text{proof} \rangle$

1.2.2 Backtracking

fun *backtrack-split* :: $\langle ('a, 'v, 'm) \text{ annotated-lits} \rangle$

$\Rightarrow ('a, 'v, 'm) \text{ annotated-lits} \times ('a, 'v, 'm) \text{ annotated-lits} \rangle$ **where**
 $\langle \text{backtrack-split } [] = ([], []) \rangle \mid$
 $\langle \text{backtrack-split } (\text{Propagated } L \ P \ \# \ \text{mlits}) = \text{apfst } ((\#) (\text{Propagated } L \ P)) (\text{backtrack-split } \text{mlits}) \rangle \mid$
 $\langle \text{backtrack-split } (\text{Decided } L \ \# \ \text{mlits}) = ([], \text{Decided } L \ \# \ \text{mlits}) \rangle$

lemma *backtrack-split-fst-not-decided*: $\langle a \in \text{set } (\text{fst } (\text{backtrack-split } l)) \Rightarrow \neg \text{is-decided } a \rangle$
 $\langle \text{proof} \rangle$

lemma *backtrack-split-snd-hd-decided*:
 $\langle \text{snd } (\text{backtrack-split } l) \neq [] \Rightarrow \text{is-decided } (\text{hd } (\text{snd } (\text{backtrack-split } l))) \rangle$
 $\langle \text{proof} \rangle$

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

lemma *backtrack-split-snd-empty-not-decided*:
 $\langle \text{backtrack-split } M = (M'', []) \Rightarrow \forall l \in \text{set } M. \neg \text{is-decided } l \rangle$
 $\langle \text{proof} \rangle$

lemma *backtrack-split-some-is-decided-then-snd-has-hd*:
 $\langle \exists l \in \text{set } M. \text{is-decided } l \Rightarrow \exists M' \ L' \ M''. \text{backtrack-split } M = (M'', L' \ \# \ M') \rangle$
 $\langle \text{proof} \rangle$

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*:
 $\langle \text{backtrack-split } M = (\text{takeWhile } (\text{Not } o \text{ is-decided}) \ M, \text{dropWhile } (\text{Not } o \text{ is-decided}) \ M) \rangle$
 $\langle \text{proof} \rangle$

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* :: $\langle ('a, 'b, 'm) \text{ annotated-lits} \Rightarrow (('a, 'b, 'm) \text{ annotated-lits} \times ('a, 'b, 'm) \text{ annotated-lits}) \text{ list} \rangle$ **where**
 $\langle \text{get-all-ann-decomposition } (\text{Decided } L \ \# \ Ls) =$
 $\quad (\text{Decided } L \ \# \ Ls, []) \ \# \ \text{get-all-ann-decomposition } Ls \rangle \mid$
 $\langle \text{get-all-ann-decomposition } (\text{Propagated } L \ P \ \# \ Ls) =$
 $\quad (\text{apsnd } ((\#) (\text{Propagated } L \ P)) (\text{hd } (\text{get-all-ann-decomposition } Ls)))$
 $\quad \ \# \ \text{tl } (\text{get-all-ann-decomposition } Ls) \rangle \mid$
 $\langle \text{get-all-ann-decomposition } [] = [([], [])] \rangle$

value $\langle \text{get-all-ann-decomposition } [\text{Propagated } A5 \ B5, \text{Decided } C4, \text{Propagated } A3 \ B3,$
 $\quad \text{Propagated } A2 \ B2, \text{Decided } C1, \text{Propagated } A0 \ B0] \rangle$

Now we can prove several simple properties about the function.

lemma *get-all-ann-decomposition-never-empty[iff]*:
 $\langle \text{get-all-ann-decomposition } M = [] \iff \text{False} \rangle$

$\langle \text{proof} \rangle$

lemma *get-all-ann-decomposition-never-empty-sym*[iff]:

$\langle [] = \text{get-all-ann-decomposition } M \longleftrightarrow \text{False} \rangle$

$\langle \text{proof} \rangle$

lemma *get-all-ann-decomposition-decomp*:

$\langle \text{hd } (\text{get-all-ann-decomposition } S) = (a, c) \implies S = c @ a \rangle$

$\langle \text{proof} \rangle$

lemma *get-all-ann-decomposition-backtrack-split*:

$\langle \text{backtrack-split } S = (M, M') \longleftrightarrow \text{hd } (\text{get-all-ann-decomposition } S) = (M', M) \rangle$

$\langle \text{proof} \rangle$

lemma *get-all-ann-decomposition-Nil-backtrack-split-snd-Nil*:

$\langle \text{get-all-ann-decomposition } S = [([], A)] \implies \text{snd } (\text{backtrack-split } S) = [] \rangle$

$\langle \text{proof} \rangle$

This function 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 $\langle \text{get-all-ann-decomposition } M = (a, b) \# [] \rangle$

shows $\langle a = [] \vee (\text{length } a = 1 \wedge \text{is-decided } (\text{hd } a) \wedge \text{hd } a \in \text{set } M) \rangle$

$\langle \text{proof} \rangle$

lemma *get-all-ann-decomposition-fst-empty-or-hd-in-M*:

assumes $\langle \text{get-all-ann-decomposition } M = (a, b) \# l \rangle$

shows $\langle a = [] \vee (\text{is-decided } (\text{hd } a) \wedge \text{hd } a \in \text{set } M) \rangle$

$\langle \text{proof} \rangle$

lemma *get-all-ann-decomposition-snd-not-decided*:

assumes $\langle (a, b) \in \text{set } (\text{get-all-ann-decomposition } M) \rangle$

and $\langle L \in \text{set } b \rangle$

shows $\langle \neg \text{is-decided } L \rangle$

$\langle \text{proof} \rangle$

lemma *tl-get-all-ann-decomposition-skip-some*:

assumes $\langle x \in \text{set } (\text{tl } (\text{get-all-ann-decomposition } M1)) \rangle$

shows $\langle x \in \text{set } (\text{tl } (\text{get-all-ann-decomposition } (M0 @ M1))) \rangle$

$\langle \text{proof} \rangle$

lemma *hd-get-all-ann-decomposition-skip-some*:

assumes $\langle (x, y) = \text{hd } (\text{get-all-ann-decomposition } M1) \rangle$

shows $\langle (x, y) \in \text{set } (\text{get-all-ann-decomposition } (M0 @ \text{Decided } K \# M1)) \rangle$

$\langle \text{proof} \rangle$

lemma *in-get-all-ann-decomposition-in-get-all-ann-decomposition-prepend*:

$\langle (a, b) \in \text{set } (\text{get-all-ann-decomposition } M') \implies$

$\exists b'. (a, b' @ b) \in \text{set } (\text{get-all-ann-decomposition } (M @ M')) \rangle$

$\langle \text{proof} \rangle$

lemma *in-get-all-ann-decomposition-decided-or-empty*:

assumes $\langle (a, b) \in \text{set } (\text{get-all-ann-decomposition } M) \rangle$

shows $\langle a = [] \vee (\text{is-decided } (\text{hd } a)) \rangle$

$\langle \text{proof} \rangle$

lemma *get-all-ann-decomposition-remove-undecided-length:*

assumes $\langle \forall l \in \text{set } M'. \neg \text{is-decided } l \rangle$

shows $\langle \text{length } (\text{get-all-ann-decomposition } (M' @ M'')) = \text{length } (\text{get-all-ann-decomposition } M'') \rangle$

$\langle \text{proof} \rangle$

lemma *get-all-ann-decomposition-not-is-decided-length:*

assumes $\langle \forall l \in \text{set } M'. \neg \text{is-decided } l \rangle$

shows $\langle 1 + \text{length } (\text{get-all-ann-decomposition } (\text{Propagated } (-L) P \# M))$

$= \text{length } (\text{get-all-ann-decomposition } (M' @ \text{Decided } L \# M)) \rangle$

$\langle \text{proof} \rangle$

lemma *get-all-ann-decomposition-last-choice:*

assumes $\langle \text{tl } (\text{get-all-ann-decomposition } (M' @ \text{Decided } L \# M)) \neq [] \rangle$

and $\langle \forall l \in \text{set } M'. \neg \text{is-decided } l \rangle$

and $\langle \text{hd } (\text{tl } (\text{get-all-ann-decomposition } (M' @ \text{Decided } L \# M))) = (M0', M0) \rangle$

shows $\langle \text{hd } (\text{get-all-ann-decomposition } (\text{Propagated } (-L) P \# M)) = (M0', \text{Propagated } (-L) P \# M0) \rangle$

$\langle \text{proof} \rangle$

lemma *get-all-ann-decomposition-except-last-choice-equal:*

assumes $\langle \forall l \in \text{set } M'. \neg \text{is-decided } l \rangle$

shows $\langle \text{tl } (\text{get-all-ann-decomposition } (\text{Propagated } (-L) P \# M))$

$= \text{tl } (\text{tl } (\text{get-all-ann-decomposition } (M' @ \text{Decided } L \# M))) \rangle$

$\langle \text{proof} \rangle$

lemma *get-all-ann-decomposition-hd-hd:*

assumes $\langle \text{get-all-ann-decomposition } Ls = (M, C) \# (M0, M0') \# l \rangle$

shows $\langle \text{tl } M = M0' @ M0 \wedge \text{is-decided } (\text{hd } M) \rangle$

$\langle \text{proof} \rangle$

lemma *get-all-ann-decomposition-exists-prepend[dest]:*

assumes $\langle (a, b) \in \text{set } (\text{get-all-ann-decomposition } M) \rangle$

shows $\langle \exists c. M = c @ b @ a \rangle$

$\langle \text{proof} \rangle$

lemma *get-all-ann-decomposition-incl:*

assumes $\langle (a, b) \in \text{set } (\text{get-all-ann-decomposition } M) \rangle$

shows $\langle \text{set } b \subseteq \text{set } M \rangle$ **and** $\langle \text{set } a \subseteq \text{set } M \rangle$

$\langle \text{proof} \rangle$

lemma *get-all-ann-decomposition-exists-prepend':*

assumes $\langle (a, b) \in \text{set } (\text{get-all-ann-decomposition } M) \rangle$

obtains c **where** $\langle M = c @ b @ a \rangle$

$\langle \text{proof} \rangle$

lemma *union-in-get-all-ann-decomposition-is-subset:*

assumes $\langle (a, b) \in \text{set } (\text{get-all-ann-decomposition } M) \rangle$

shows $\langle \text{set } a \cup \text{set } b \subseteq \text{set } M \rangle$

$\langle \text{proof} \rangle$

lemma *Decided-cons-in-get-all-ann-decomposition-append-Decided-cons:*

$\langle \exists c''. (\text{Decided } K \# c, c'') \in \text{set } (\text{get-all-ann-decomposition } (c' @ \text{Decided } K \# c)) \rangle$

$\langle \text{proof} \rangle$

lemma *fst-get-all-ann-decomposition-prepend-not-decided:*

assumes $\langle \forall m \in \text{set } MS. \neg \text{is-decided } m \rangle$

shows $\langle \text{set } (\text{map fst } (\text{get-all-ann-decomposition } M))$
 $= \text{set } (\text{map fst } (\text{get-all-ann-decomposition } (MS @ M))) \rangle$
 $\langle \text{proof} \rangle$

lemma *no-decision-get-all-ann-decomposition*:

$\langle \forall l \in \text{set } M. \neg \text{is-decided } l \implies \text{get-all-ann-decomposition } M = [([], M)] \rangle$
 $\langle \text{proof} \rangle$

Entailment of the Propagated by the Decided Literal

lemma *get-all-ann-decomposition-snd-union*:

$\langle \text{set } M = \bigcup (\text{set } ' \text{snd } ' \text{set } (\text{get-all-ann-decomposition } M)) \cup \{L \mid L. \text{is-decided } L \wedge L \in \text{set } M\} \rangle$
 $\langle \text{is } (?M \text{ } M = ?U \text{ } M \cup ?Ls \text{ } M) \rangle$
 $\langle \text{proof} \rangle$

definition *all-decomposition-implies* :: $\langle 'a \text{ clause set}$

$\Rightarrow ((('a, 'm) \text{ ann-lits} \times ('a, 'm) \text{ ann-lits}) \text{ list} \Rightarrow \text{bool})$ **where**

$\langle \text{all-decomposition-implies } N \text{ } S \longleftrightarrow (\forall (Ls, \text{seen}) \in \text{set } S. \text{unmark-l } Ls \cup N \models_{ps} \text{unmark-l } \text{seen}) \rangle$

lemma *all-decomposition-implies-empty*[iff]:

$\langle \text{all-decomposition-implies } N [] \rangle \langle \text{proof} \rangle$

lemma *all-decomposition-implies-single*[iff]:

$\langle \text{all-decomposition-implies } N [(Ls, \text{seen})] \longleftrightarrow \text{unmark-l } Ls \cup N \models_{ps} \text{unmark-l } \text{seen} \rangle$
 $\langle \text{proof} \rangle$

lemma *all-decomposition-implies-append*[iff]:

$\langle \text{all-decomposition-implies } N (S @ S') \longleftrightarrow (\text{all-decomposition-implies } N \text{ } S \wedge \text{all-decomposition-implies } N \text{ } S') \rangle$
 $\langle \text{proof} \rangle$

lemma *all-decomposition-implies-cons-pair*[iff]:

$\langle \text{all-decomposition-implies } N ((Ls, \text{seen}) \# S') \longleftrightarrow (\text{all-decomposition-implies } N [(Ls, \text{seen})] \wedge \text{all-decomposition-implies } N \text{ } S') \rangle$
 $\langle \text{proof} \rangle$

lemma *all-decomposition-implies-cons-single*[iff]:

$\langle \text{all-decomposition-implies } N (l \# S') \longleftrightarrow (\text{unmark-l } (\text{fst } l) \cup N \models_{ps} \text{unmark-l } (\text{snd } l) \wedge \text{all-decomposition-implies } N \text{ } S') \rangle$
 $\langle \text{proof} \rangle$

lemma *all-decomposition-implies-trail-is-implied*:

assumes $\langle \text{all-decomposition-implies } N (\text{get-all-ann-decomposition } M) \rangle$

shows $\langle N \cup \{\text{unmark } L \mid L. \text{is-decided } L \wedge L \in \text{set } M\}$

$\models_{ps} \text{unmark } ' \bigcup (\text{set } ' \text{snd } ' \text{set } (\text{get-all-ann-decomposition } M)) \rangle$

$\langle \text{proof} \rangle$

lemma *all-decomposition-implies-propagated-lits-are-implied*:

assumes $\langle \text{all-decomposition-implies } N (\text{get-all-ann-decomposition } M) \rangle$

shows $\langle N \cup \{\text{unmark } L \mid L. \text{is-decided } L \wedge L \in \text{set } M\} \models_{ps} \text{unmark-l } M \rangle$

$\langle \text{is } (?I \models_{ps} ?A) \rangle$

$\langle \text{proof} \rangle$

lemma *all-decomposition-implies-insert-single*:

$\langle \text{all-decomposition-implies } N \ M \implies \text{all-decomposition-implies } (\text{insert } C \ N) \ M \rangle$
 $\langle \text{proof} \rangle$

lemma *all-decomposition-implies-union*:

$\langle \text{all-decomposition-implies } N \ M \implies \text{all-decomposition-implies } (N \cup N') \ M \rangle$
 $\langle \text{proof} \rangle$

lemma *all-decomposition-implies-mono*:

$\langle N \subseteq N' \implies \text{all-decomposition-implies } N \ M \implies \text{all-decomposition-implies } N' \ M \rangle$
 $\langle \text{proof} \rangle$

lemma *all-decomposition-implies-mono-right*:

$\langle \text{all-decomposition-implies } I \ (\text{get-all-ann-decomposition } (M' @ M)) \implies$
 $\text{all-decomposition-implies } I \ (\text{get-all-ann-decomposition } M) \rangle$
 $\langle \text{proof} \rangle$

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* :: $\langle 'v \text{ clause} \Rightarrow 'v \text{ clause-set} \rangle$ **where**

$\langle CNot \ \psi = \{ \{ \# - L \# \} \mid L. L \in \# \ \psi \} \rangle$

lemma *finite-CNot[simp]*: $\langle \text{finite } (CNot \ C) \rangle$

$\langle \text{proof} \rangle$

lemma *in-CNot-uminus[iff]*:

shows $\langle \{ \# L \# \} \in CNot \ \psi \longleftrightarrow -L \in \# \ \psi \rangle$
 $\langle \text{proof} \rangle$

lemma

shows

CNot-add-mset[simp]: $\langle CNot \ (\text{add-mset } L \ \psi) = \text{insert } \{ \# - L \# \} \ (CNot \ \psi) \rangle$ **and**

CNot-empty[simp]: $\langle CNot \ \{ \# \} = \{ \} \rangle$ **and**

CNot-plus[simp]: $\langle CNot \ (A + B) = CNot \ A \cup CNot \ B \rangle$

$\langle \text{proof} \rangle$

lemma *CNot-eq-empty[iff]*:

$\langle CNot \ D = \{ \} \longleftrightarrow D = \{ \# \} \rangle$
 $\langle \text{proof} \rangle$

lemma *in-CNot-implies-uminus*:

assumes $\langle L \in \# \ D \rangle$ **and** $\langle M \models_{as} CNot \ D \rangle$

shows $\langle M \models_a \{ \# - L \# \} \rangle$ **and** $\langle -L \in \text{lits-of-l } M \rangle$

$\langle \text{proof} \rangle$

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

$\langle CNot \ (\text{remdups-mset } A) = CNot \ A \rangle$
 $\langle \text{proof} \rangle$

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

$\langle (\forall x \in CNot \ D. P \ x) \longleftrightarrow (\forall L \in \# \ D. P \ \{ \# - L \# \}) \rangle$
 $\langle \text{proof} \rangle$

lemma *consistent-CNot-not*:

assumes $\langle \text{consistent-interp } I \rangle$
shows $\langle I \models_s \text{CNot } \varphi \implies \neg I \models \varphi \rangle$
 $\langle \text{proof} \rangle$

lemma *total-not-true-cls-true-cls-CNot*:

assumes $\langle \text{total-over-}m \ I \ \{\varphi\} \rangle$ **and** $\langle \neg I \models \varphi \rangle$
shows $\langle I \models_s \text{CNot } \varphi \rangle$
 $\langle \text{proof} \rangle$

lemma *total-not-CNot*:

assumes $\langle \text{total-over-}m \ I \ \{\varphi\} \rangle$ **and** $\langle \neg I \models_s \text{CNot } \varphi \rangle$
shows $\langle I \models \varphi \rangle$
 $\langle \text{proof} \rangle$

lemma *atms-of-ms-CNot-atms-of[simp]*:

$\langle \text{atms-of-ms } (\text{CNot } C) = \text{atms-of } C \rangle$
 $\langle \text{proof} \rangle$

lemma *true-cls-cls-contradiction-true-cls-cls-false*:

$\langle C \in D \implies D \models_{ps} \text{CNot } C \implies D \models_p \{\#\} \rangle$
 $\langle \text{proof} \rangle$

lemma *true-annots-CNot-all-atms-defined*:

assumes $\langle M \models_{as} \text{CNot } T \rangle$ **and** $a1: \langle L \in\# \ T \rangle$
shows $\langle \text{atm-of } L \in \text{atm-of } ' \text{ lits-of-}l \ M \rangle$
 $\langle \text{proof} \rangle$

lemma *true-annots-CNot-all-uminus-atms-defined*:

assumes $\langle M \models_{as} \text{CNot } T \rangle$ **and** $a1: \langle \neg L \in\# \ T \rangle$
shows $\langle \text{atm-of } L \in \text{atm-of } ' \text{ lits-of-}l \ M \rangle$
 $\langle \text{proof} \rangle$

lemma *true-cls-cls-false-left-right*:

assumes $\langle \{\{\#L\#\}\} \cup B \models_p \{\#\} \rangle$
shows $\langle B \models_{ps} \text{CNot } \{\#L\#\} \rangle$
 $\langle \text{proof} \rangle$

lemma *true-annots-true-cls-def-iff-negation-in-model*:

$\langle M \models_{as} \text{CNot } C \longleftrightarrow (\forall L \in\# \ C. \neg L \in \text{lits-of-}l \ M) \rangle$
 $\langle \text{proof} \rangle$

lemma *true-cls-def-iff-negation-in-model*:

$\langle M \models_s \text{CNot } C \longleftrightarrow (\forall l \in\# \ C. \neg l \in M) \rangle$
 $\langle \text{proof} \rangle$

lemma *true-annots-CNot-definedD*:

$\langle M \models_{as} \text{CNot } C \implies \forall L \in\# \ C. \text{defined-lit } M \ L \rangle$
 $\langle \text{proof} \rangle$

lemma *true-annot-CNot-diff*:

$\langle I \models_{as} \text{CNot } C \implies I \models_{as} \text{CNot } (C - C') \rangle$
 $\langle \text{proof} \rangle$

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

$\langle CNot (mset (replicate n L)) = (if\ n = 0\ then\ \{\}\ else\ \{\{\#-L\#\}\}) \rangle$
 $\langle proof \rangle$

lemma *consistent-CNot-not-tautology*:

$\langle consistent_interp\ M \implies M \models_s CNot\ D \implies \neg tautology\ D \rangle$
 $\langle proof \rangle$

lemma *atms-of-ms-CNot-atms-of-ms*: $\langle atms_of_ms\ (CNot\ CC) = atms_of_ms\ \{CC\} \rangle$

$\langle proof \rangle$

lemma *total-over-m-CNot-toal-over-m[simp]*:

$\langle total_over_m\ I\ (CNot\ C) = total_over_set\ I\ (atms_of\ C) \rangle$
 $\langle proof \rangle$

lemma *true-clss-clss-plus-CNot*:

assumes

$CC-L$: $\langle A \models_p add_mset\ L\ CC \rangle$ **and**

$CNot-CC$: $\langle A \models_{ps} CNot\ CC \rangle$

shows $\langle A \models_p \{\#L\#\} \rangle$

$\langle proof \rangle$

lemma *true-annots-CNot-lit-of-notin-skip*:

assumes LM : $\langle L \# M \models_{as} CNot\ A \rangle$ **and** LA : $\langle lit_of\ L \notin \# A \rangle \langle \neg lit_of\ L \notin \# A \rangle$

shows $\langle M \models_{as} CNot\ A \rangle$

$\langle proof \rangle$

lemma *true-clss-clss-union-false-true-clss-clss-cnot*:

$\langle A \cup \{B\} \models_{ps} \{\{\#\}\} \longleftrightarrow A \models_{ps} CNot\ B \rangle$

$\langle proof \rangle$

lemma *true-annot-remove-hd-if-notin-vars*:

assumes $\langle a \# M' \models_a D \rangle$ **and** $\langle atm_of\ (lit_of\ a) \notin atms_of\ D \rangle$

shows $\langle M' \models_a D \rangle$

$\langle proof \rangle$

lemma *true-annot-remove-if-notin-vars*:

assumes $\langle M @ M' \models_a D \rangle$ **and** $\langle \forall x \in atms_of\ D. x \notin atm_of\ ' lits_of_l\ M \rangle$

shows $\langle M' \models_a D \rangle$

$\langle proof \rangle$

lemma *true-annots-remove-if-notin-vars*:

assumes $\langle M @ M' \models_{as} D \rangle$ **and** $\langle \forall x \in atms_of_ms\ D. x \notin atm_of\ ' lits_of_l\ M \rangle$

shows $\langle M' \models_{as} D \rangle$ $\langle proof \rangle$

lemma *all-variables-defined-not-imply-cnot*:

assumes

$\langle \forall s \in atms_of_ms\ \{B\}. s \in atm_of\ ' lits_of_l\ A \rangle$ **and**

$\langle \neg A \models_a B \rangle$

shows $\langle A \models_{as} CNot\ B \rangle$

$\langle proof \rangle$

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

$\langle CNot\ (A \cup \# B) = CNot\ A \cup CNot\ B \rangle$

$\langle proof \rangle$

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

assumes
 $\langle A \models_{ps} \text{unmark-}l\ M \rangle$ **and** $\langle M \models_{as} D \rangle$
shows $\langle A \models_{ps} D \rangle$
 $\langle \text{proof} \rangle$

lemma *true-clss-clss-CNot-true-clss-clss-unsatisfiable:*

assumes $\langle A \models_{ps} C\text{Not}\ D \rangle$ **and** $\langle A \models_p D \rangle$
shows $\langle \text{unsatisfiable}\ A \rangle$
 $\langle \text{proof} \rangle$

lemma *true-clss-clss-neg:*

$\langle N \models_p I \longleftrightarrow N \cup (\lambda L. \{\# - L\# \})\ \text{'set-mset}\ I \models_p \{\#\} \rangle$
 $\langle \text{proof} \rangle$

lemma *all-decomposition-implies-conflict-DECO-clause:*

assumes $\langle \text{all-decomposition-implies}\ N\ (\text{get-all-ann-decomposition}\ M) \rangle$ **and**
 $\langle M \models_{as} C\text{Not}\ C \rangle$ **and**
 $\langle C \in N \rangle$
shows $\langle N \models_p (\text{uminus}\ o\ \text{lit-of})\ \text{'\# (filter-mset is-decided (mset}\ M)) \rangle$
 $\langle \text{is}\ \langle ?I \models_p\ ?A \rangle \rangle$
 $\langle \text{proof} \rangle$

1.2.5 Other

definition $\langle \text{no-dup}\ L \equiv \text{distinct}\ (\text{map}\ (\lambda l. \text{atm-of}\ (\text{lit-of}\ l))\ L) \rangle$

lemma *no-dup-nil[simp]:*

$\langle \text{no-dup}\ [] \rangle$
 $\langle \text{proof} \rangle$

lemma *no-dup-cons[simp]:*

$\langle \text{no-dup}\ (L \# M) \longleftrightarrow \text{undefined-lit}\ M\ (\text{lit-of}\ L) \wedge \text{no-dup}\ M \rangle$
 $\langle \text{proof} \rangle$

lemma *no-dup-append-cons[iff]:*

$\langle \text{no-dup}\ (M @ L \# M') \longleftrightarrow \text{undefined-lit}\ (M @ M')\ (\text{lit-of}\ L) \wedge \text{no-dup}\ (M @ M') \rangle$
 $\langle \text{proof} \rangle$

lemma *no-dup-append-append-cons[iff]:*

$\langle \text{no-dup}\ (M0 @ M @ L \# M') \longleftrightarrow \text{undefined-lit}\ (M0 @ M @ M')\ (\text{lit-of}\ L) \wedge \text{no-dup}\ (M0 @ M @ M') \rangle$
 $\langle \text{proof} \rangle$

lemma *no-dup-rev[simp]:*

$\langle \text{no-dup}\ (\text{rev}\ M) \longleftrightarrow \text{no-dup}\ M \rangle$
 $\langle \text{proof} \rangle$

lemma *no-dup-appendD:*

$\langle \text{no-dup}\ (a @ b) \implies \text{no-dup}\ b \rangle$
 $\langle \text{proof} \rangle$

lemma *no-dup-appendD1:*

$\langle \text{no-dup}\ (a @ b) \implies \text{no-dup}\ a \rangle$
 $\langle \text{proof} \rangle$

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

assumes $\langle \text{no-dup } M \rangle$
shows $\langle \text{length } M = \text{card } (\text{atm-of } \text{' } \text{ lits-of-l } M) \rangle$
 $\langle \text{proof} \rangle$

lemma *distinct-consistent-interp*:
 $\langle \text{no-dup } M \implies \text{consistent-interp } (\text{lits-of-l } M) \rangle$
 $\langle \text{proof} \rangle$

lemma *same-mset-no-dup-iff*:
 $\langle \text{mset } M = \text{mset } M' \implies \text{no-dup } M \longleftrightarrow \text{no-dup } M' \rangle$
 $\langle \text{proof} \rangle$

lemma *distinct-get-all-ann-decomposition-no-dup*:
assumes $\langle (a, b) \in \text{set } (\text{get-all-ann-decomposition } M) \rangle$
and $\langle \text{no-dup } M \rangle$
shows $\langle \text{no-dup } (a @ b) \rangle$
 $\langle \text{proof} \rangle$

lemma *true-annots-lit-of-notin-skip*:
assumes $\langle L \# M \models_{\text{as}} \text{CNot } A \rangle$
and $\langle \neg \text{lit-of } L \notin \# A \rangle$
and $\langle \text{no-dup } (L \# M) \rangle$
shows $\langle M \models_{\text{as}} \text{CNot } A \rangle$
 $\langle \text{proof} \rangle$

lemma *no-dup-imp-distinct*: $\langle \text{no-dup } M \implies \text{distinct } M \rangle$
 $\langle \text{proof} \rangle$

lemma *no-dup-tlD*: $\langle \text{no-dup } a \implies \text{no-dup } (\text{tl } a) \rangle$
 $\langle \text{proof} \rangle$

lemma *defined-lit-no-dupD*:
 $\langle \text{defined-lit } M1 \ L \implies \text{no-dup } (M2 @ M1) \implies \text{undefined-lit } M2 \ L \rangle$
 $\langle \text{defined-lit } M1 \ L \implies \text{no-dup } (M2' @ M2 @ M1) \implies \text{undefined-lit } M2' \ L \rangle$
 $\langle \text{defined-lit } M1 \ L \implies \text{no-dup } (M2' @ M2 @ M1) \implies \text{undefined-lit } M2 \ L \rangle$
 $\langle \text{proof} \rangle$

lemma *no-dup-consistentD*:
 $\langle \text{no-dup } M \implies L \in \text{lits-of-l } M \implies \neg L \notin \text{lits-of-l } M \rangle$
 $\langle \text{proof} \rangle$

lemma *no-dup-not-tautology*: $\langle \text{no-dup } M \implies \neg \text{tautology } (\text{image-mset lit-of } (\text{mset } M)) \rangle$
 $\langle \text{proof} \rangle$

lemma *no-dup-distinct*: $\langle \text{no-dup } M \implies \text{distinct-mset } (\text{image-mset lit-of } (\text{mset } M)) \rangle$
 $\langle \text{proof} \rangle$

lemma *no-dup-not-tautology-uminus*: $\langle \text{no-dup } M \implies \neg \text{tautology } \{ \# \neg \text{lit-of } L. L \in \# \text{mset } M \# \} \rangle$
 $\langle \text{proof} \rangle$

lemma *no-dup-distinct-uminus*: $\langle \text{no-dup } M \implies \text{distinct-mset } \{ \# \neg \text{lit-of } L. L \in \# \text{mset } M \# \} \rangle$
 $\langle \text{proof} \rangle$

lemma *no-dup-map-lit-of*: $\langle \text{no-dup } M \implies \text{distinct } (\text{map lit-of } M) \rangle$
 $\langle \text{proof} \rangle$

lemma *no-dup-alt-def*:

$\langle \text{no-dup } M \longleftrightarrow \text{distinct-mset } \{\# \text{atm-of } (\text{lit-of } x). x \in \# \text{ mset } M \# \} \rangle$
 $\langle \text{proof} \rangle$

lemma *no-dup-append-in-atm-notin*:

assumes $\langle \text{no-dup } (M @ M') \rangle$ **and** $\langle L \in \text{lits-of-l } M' \rangle$
shows $\langle \text{undefined-lit } M L \rangle$
 $\langle \text{proof} \rangle$

lemma *no-dup-uminus-append-in-atm-notin*:

assumes $\langle \text{no-dup } (M @ M') \rangle$ **and** $\langle -L \in \text{lits-of-l } M' \rangle$
shows $\langle \text{undefined-lit } M L \rangle$
 $\langle \text{proof} \rangle$

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** \models_{asm} 50) **where**

$\langle I \models_{asm} C \equiv I \models_{as} (\text{set-mset } C) \rangle$

abbreviation *true-clss-clss-m* :: $\langle 'v \text{ clause multiset} \Rightarrow 'v \text{ clause multiset} \Rightarrow \text{bool} \rangle$ (**infix** \models_{psm} 50)

where

$\langle I \models_{psm} C \equiv \text{set-mset } I \models_{ps} (\text{set-mset } C) \rangle$

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

lemma *true-clss-clssm-subsetE*: $\langle N \models_{psm} B \Longrightarrow A \subseteq \# B \Longrightarrow N \models_{psm} A \rangle$

$\langle \text{proof} \rangle$

abbreviation *true-clss-clss-m* :: $\langle 'a \text{ clause multiset} \Rightarrow 'a \text{ clause} \Rightarrow \text{bool} \rangle$ (**infix** \models_{pm} 50) **where**

$\langle I \models_{pm} C \equiv \text{set-mset } I \models_p C \rangle$

abbreviation *distinct-mset-mset* :: $\langle 'a \text{ multiset multiset} \Rightarrow \text{bool} \rangle$ **where**

$\langle \text{distinct-mset-mset } \Sigma \equiv \text{distinct-mset-set } (\text{set-mset } \Sigma) \rangle$

abbreviation *all-decomposition-implies-m* **where**

$\langle \text{all-decomposition-implies-m } A B \equiv \text{all-decomposition-implies } (\text{set-mset } A) B \rangle$

abbreviation *atms-of-mm* :: $\langle 'a \text{ clause multiset} \Rightarrow 'a \text{ set} \rangle$ **where**

$\langle \text{atms-of-mm } U \equiv \text{atms-of-ms } (\text{set-mset } U) \rangle$

Other definition using $\bigcup \#$

lemma *atms-of-mm-alt-def*: $\langle \text{atms-of-mm } U = \text{set-mset } (\bigcup \# (\text{image-mset } (\text{image-mset } \text{atm-of}) U)) \rangle$

$\langle \text{proof} \rangle$

abbreviation *true-clss-m* :: $\langle 'a \text{ partial-interp} \Rightarrow 'a \text{ clause multiset} \Rightarrow \text{bool} \rangle$ (**infix** \models_{sm} 50) **where**

$\langle I \models_{sm} C \equiv I \models_s \text{set-mset } C \rangle$

abbreviation *true-clss-ext-m* (**infix** \models_{sextm} 49) **where**

$\langle I \models_{sextm} C \equiv I \models_{sext} \text{set-mset } C \rangle$

lemma *true-clss-clss-cong-set-mset*:

$\langle N \models_{pm} D \Longrightarrow \text{set-mset } D = \text{set-mset } D' \Longrightarrow N \models_{pm} D' \rangle$

$\langle \text{proof} \rangle$

1.2.7 More Lemmas

lemma *no-dup-cannot-not-lit-and-uminus*:

$\langle \text{no-dup } M \implies - \text{ lit-of } xa = \text{ lit-of } x \implies x \in \text{set } M \implies xa \notin \text{set } M \rangle$
 $\langle \text{proof} \rangle$

lemma *atms-of-ms-single-atm-of[simp]*:

$\langle \text{atms-of-ms } \{\text{unmark } L \mid L. P L\} = \text{atm-of } \{ \text{lit-of } L \mid L. P L \} \rangle$
 $\langle \text{proof} \rangle$

lemma *true-cls-mset-restrict*:

$\langle \{L \in I. \text{atm-of } L \in \text{atms-of-mm } N\} \models_m N \longleftrightarrow I \models_m N \rangle$
 $\langle \text{proof} \rangle$

lemma *true-clss-restrict*:

$\langle \{L \in I. \text{atm-of } L \in \text{atms-of-mm } N\} \models_{sm} N \longleftrightarrow I \models_{sm} N \rangle$
 $\langle \text{proof} \rangle$

lemma *total-over-m-atms-incl*:

assumes $\langle \text{total-over-m } M \text{ (set-mset } N) \rangle$
shows
 $\langle x \in \text{atms-of-mm } N \implies x \in \text{atms-of-s } M \rangle$
 $\langle \text{proof} \rangle$

lemma *true-clss-restrict-iff*:

assumes $\langle \neg \text{tautology } \chi \rangle$
shows $\langle N \models_p \chi \longleftrightarrow N \models_p \{\#L \in \# \chi. \text{atm-of } L \in \text{atms-of-ms } N \# \} \rangle$ (**is** $\langle ?A \longleftrightarrow ?B \rangle$)
 $\langle \text{proof} \rangle$

1.2.8 Negation of annotated clauses

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

lemma *negate-ann-lits-empty[simp]*: $\langle \text{negate-ann-lits } [] = \{\# \} \rangle$
 $\langle \text{proof} \rangle$

lemma *entails-CNot-negate-ann-lits*:

$\langle M \models_{as} C\text{Not } D \longleftrightarrow \text{set-mset } D \subseteq \text{set-mset } (\text{negate-ann-lits } M) \rangle$
 $\langle \text{proof} \rangle$

Pointwise negation of a clause:

definition *pNeg* :: $\langle 'v \text{ clause} \Rightarrow 'v \text{ clause} \rangle$ **where**

$\langle pNeg C = \{\# - D. D \in \# C \# \} \rangle$

lemma *pNeg-simps*:

$\langle pNeg (\text{add-mset } A C) = \text{add-mset } (-A) (pNeg C) \rangle$
 $\langle pNeg (C + D) = pNeg C + pNeg D \rangle$
 $\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma *negate-ann-lits-pNeg-lit-of*: $\langle \text{negate-ann-lits} = pNeg \circ \text{image-mset lit-of} \circ \text{mset} \rangle$

$\langle \text{proof} \rangle$

lemma *negate-ann-lits-empty-iff*: $\langle \text{negate-ann-lits } M \neq \{\#\} \longleftrightarrow M \neq [] \rangle$
 $\langle \text{proof} \rangle$

lemma *atms-of-negate-ann-lits[simp]*: $\langle \text{atms-of } (\text{negate-ann-lits } M) = \text{atm-of } '(\text{lits-of-l } M) \rangle$
 $\langle \text{proof} \rangle$

lemma *tautology-pNeg[simp]*:
 $\langle \text{tautology } (p\text{Neg } C) \longleftrightarrow \text{tautology } C \rangle$
 $\langle \text{proof} \rangle$

lemma *pNeg-convolution[simp]*:
 $\langle p\text{Neg } (p\text{Neg } C) = C \rangle$
 $\langle \text{proof} \rangle$

lemma *pNeg-minus[simp]*: $\langle p\text{Neg } (A - B) = p\text{Neg } A - p\text{Neg } B \rangle$
 $\langle \text{proof} \rangle$

lemma *pNeg-empty[simp]*: $\langle p\text{Neg } \{\#\} = \{\#\} \rangle$
 $\langle \text{proof} \rangle$

lemma *pNeg-replicate-mset[simp]*: $\langle p\text{Neg } (\text{replicate-mset } n \ L) = \text{replicate-mset } n \ (-L) \rangle$
 $\langle \text{proof} \rangle$

lemma *distinct-mset-pNeg-iff[iff]*: $\langle \text{distinct-mset } (p\text{Neg } x) \longleftrightarrow \text{distinct-mset } x \rangle$
 $\langle \text{proof} \rangle$

lemma *pNeg-simple-clss-iff[simp]*:
 $\langle p\text{Neg } M \in \text{simple-clss } N \longleftrightarrow M \in \text{simple-clss } N \rangle$
 $\langle \text{proof} \rangle$

lemma *atms-of-ms-pNeg[simp]*: $\langle \text{atms-of-ms } (p\text{Neg } 'N) = \text{atms-of-ms } N \rangle$
 $\langle \text{proof} \rangle$

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

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

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

lemma *in-negate-trial-iff*: $\langle L \in \# \text{ negate-ann-lits } M \longleftrightarrow - \ L \in \text{ lits-of-l } M \rangle$
 $\langle \text{proof} \rangle$

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

lemma *uminus-simple-clss-iff[simp]*:
 $\langle \text{uminus } \# M \in \text{simple-clss } N \longleftrightarrow M \in \text{simple-clss } N \rangle$
 $\langle \text{proof} \rangle$

lemma *pNeg-mono*: $\langle C \subseteq \# C' \implies \text{pNeg } C \subseteq \# \text{pNeg } C' \rangle$
 $\langle \text{proof} \rangle$

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

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

lemma *total-over-set-partial-model-of*:
 $\langle \text{total-over-set } (\text{partial-model-of } I) \text{ UNIV} \rangle$
 $\langle \text{proof} \rangle$

lemma *consistent-interp-partial-model-of*:
 $\langle \text{consistent-interp } (\text{partial-model-of } I) \rangle$
 $\langle \text{proof} \rangle$

lemma *consistent-interp-alt-def*:
 $\langle \text{consistent-interp } I \longleftrightarrow (\forall L. \neg(\text{Pos } L \in I \wedge \text{Neg } L \in I)) \rangle$
 $\langle \text{proof} \rangle$

context

fixes $I :: \langle 'a \text{ partial-interp} \rangle$

assumes *cons*: $\langle \text{consistent-interp } I \rangle$

begin

lemma *partial-implies-total-true-clss-total-model-of*:
assumes $\langle \text{Partial-Herbrand-Interpretation.true-clss } I \ C \rangle$
shows $\langle \text{Herbrand-Interpretation.true-clss } (\text{total-model-of } I) \ C \rangle$
 $\langle \text{proof} \rangle$

lemma *total-implies-partial-true-clss-total-model-of*:
assumes $\langle \text{Herbrand-Interpretation.true-clss } (\text{total-model-of } I) \ C \rangle$ **and**
 $\langle \text{total-over-set } I \ (\text{atms-of } C) \rangle$
shows $\langle \text{Partial-Herbrand-Interpretation.true-clss } I \ C \rangle$
 $\langle \text{proof} \rangle$

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

```

shows  $\langle \text{Herbrand-Interpretation.true-clss } (\text{total-model-of } I) \ C \rangle$ 
 $\langle \text{proof} \rangle$ 

lemma total-implies-partial-true-clss-total-model-of:
assumes  $\langle \text{Herbrand-Interpretation.true-clss } (\text{total-model-of } I) \ C \rangle$  and
 $\langle \text{total-over-m } I \ C \rangle$ 
shows  $\langle \text{Partial-Herbrand-Interpretation.true-clss } I \ C \rangle$ 
 $\langle \text{proof} \rangle$ 

end

lemma total-implies-partial-true-clss-partial-model-of:
assumes  $\langle \text{Herbrand-Interpretation.true-clss } I \ C \rangle$ 
shows  $\langle \text{Partial-Herbrand-Interpretation.true-clss } (\text{partial-model-of } I) \ C \rangle$ 
 $\langle \text{proof} \rangle$ 

lemma total-implies-partial-true-clss-partial-model-of:
assumes  $\langle \text{Herbrand-Interpretation.true-clss } I \ C \rangle$ 
shows  $\langle \text{Partial-Herbrand-Interpretation.true-clss } (\text{partial-model-of } I) \ C \rangle$ 
 $\langle \text{proof} \rangle$ 

lemma partial-total-satisfiable-iff:
 $\langle \text{Partial-Herbrand-Interpretation.satisfiable } N \longleftrightarrow \text{Herbrand-Interpretation.satisfiable } N \rangle$ 
 $\langle \text{proof} \rangle$ 

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 $'v \text{ propo} =$
 $FT \mid FF \mid FVar\ 'v \mid FNot\ 'v \text{ propo} \mid FAnd\ 'v \text{ propo}\ 'v \text{ propo} \mid FOr\ 'v \text{ propo}\ 'v \text{ propo}$
 $\mid FImp\ 'v \text{ propo}\ 'v \text{ propo} \mid FEq\ 'v \text{ propo}\ 'v \text{ propo}$

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 $'v \text{ connective} = CT \mid CF \mid CVar\ 'v \mid CNot \mid CAnd \mid COr \mid CImp \mid CEq$

abbreviation $nullary\text{-}connective \equiv \{CF\} \cup \{CT\} \cup \{CVar\ x \mid x. True\}$

definition $binary\text{-}connectives \equiv \{CAnd, COr, CImp, CEq\}$

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

lemma $propo\text{-}induct\text{-}arity[case\text{-}names\ nullary\ unary\ binary]:$

fixes $\varphi\ \psi :: 'v \text{ propo}$
assumes $nullary: \bigwedge \varphi\ x. \varphi = FF \vee \varphi = FT \vee \varphi = FVar\ x \implies P\ \varphi$
and $unary: \bigwedge \psi. P\ \psi \implies P\ (FNot\ \psi)$
and $binary: \bigwedge \varphi\ \psi1\ \psi2. P\ \psi1 \implies P\ \psi2 \implies \varphi = FAnd\ \psi1\ \psi2 \vee \varphi = FOr\ \psi1\ \psi2 \vee \varphi = FImp\ \psi1\ \psi2$
 $\vee \varphi = FEq\ \psi1\ \psi2 \implies P\ \varphi$
shows $P\ \psi$
 $\langle proof \rangle$

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

```
fun conn :: 'v connective  $\Rightarrow$  'v propo list  $\Rightarrow$  'v propo where
conn CT [] = FT |
conn CF [] = FF |
conn (CVar v) [] = FVar v |
conn CNot [ $\varphi$ ] = FNot  $\varphi$  |
conn CAnd ( $\varphi$  # [ $\psi$ ]) = FAnd  $\varphi$   $\psi$  |
conn COr ( $\varphi$  # [ $\psi$ ]) = FOr  $\varphi$   $\psi$  |
conn CImp ( $\varphi$  # [ $\psi$ ]) = FImp  $\varphi$   $\psi$  |
conn CEq ( $\varphi$  # [ $\psi$ ]) = FEq  $\varphi$   $\psi$  |
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:  $\bigwedge x. c = CT \vee c = CF \vee c = CVar x \implies P$ 
and binary:  $c \in \text{binary-connectives} \implies P$ 
and unary:  $c = CNot \implies P$ 
shows P
<proof>
```

```
lemma connective-cases-arity-2[case-names nullary unary binary]:
assumes nullary:  $c \in \text{nullary-connective} \implies P$ 
and unary:  $c = CNot \implies P$ 
and binary:  $c \in \text{binary-connectives} \implies P$ 
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]:  $(c = CT \vee c = CF \vee c = CVar v) \implies \text{wf-conn } c []$  |
wf-conn-unary[simp]:  $c = CNot \implies \text{wf-conn } c [\psi]$  |
wf-conn-binary[simp]:  $c \in \text{binary-connectives} \implies \text{wf-conn } c (\psi \# \psi' \# [])$ 
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
 $\bigwedge v. c = CT \implies P []$  and
 $\bigwedge v. c = CF \implies P []$  and
 $\bigwedge v. c = CVar v \implies P []$  and
 $\bigwedge \psi. c = CNot \implies P [\psi]$  and
 $\bigwedge \psi \psi'. c = COr \implies P [\psi, \psi']$  and
 $\bigwedge \psi \psi'. c = CAnd \implies P [\psi, \psi']$  and
 $\bigwedge \psi \psi'. c = CImp \implies P [\psi, \psi']$  and
 $\bigwedge \psi \psi'. c = CEq \implies P [\psi, \psi']$ 
shows P x
<proof>
```

2.1.2 Properties of the Abstraction

First we can define simplification rules.

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

$wf\text{-}conn\ CT\ l \implies conn\ CT\ l = FT$
 $wf\text{-}conn\ CF\ l \implies conn\ CF\ l = FF$
 $wf\text{-}conn\ (CVar\ x)\ l \implies conn\ (CVar\ x)\ l = FVar\ x$
 $\langle proof \rangle$

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

$wf\text{-}conn\ CT\ l \longleftrightarrow l = []$
 $wf\text{-}conn\ CF\ l \longleftrightarrow l = []$
 $wf\text{-}conn\ (CVar\ x)\ l \longleftrightarrow l = []$
 $wf\text{-}conn\ CNot\ (\xi\ @\ \varphi\ \# \ \xi') \longleftrightarrow \xi = [] \wedge \xi' = []$
 $\langle proof \rangle$

lemma *wf-conn-list*:

$wf\text{-}conn\ c\ l \implies conn\ c\ l = FT \longleftrightarrow (c = CT \wedge l = [])$
 $wf\text{-}conn\ c\ l \implies conn\ c\ l = FF \longleftrightarrow (c = CF \wedge l = [])$
 $wf\text{-}conn\ c\ l \implies conn\ c\ l = FVar\ x \longleftrightarrow (c = CVar\ x \wedge l = [])$
 $wf\text{-}conn\ c\ l \implies conn\ c\ l = FAnd\ a\ b \longleftrightarrow (c = CAnd \wedge l = a\ \# \ b\ \# \ [])$
 $wf\text{-}conn\ c\ l \implies conn\ c\ l = FOr\ a\ b \longleftrightarrow (c = COr \wedge l = a\ \# \ b\ \# \ [])$
 $wf\text{-}conn\ c\ l \implies conn\ c\ l = FEq\ a\ b \longleftrightarrow (c = CEq \wedge l = a\ \# \ b\ \# \ [])$
 $wf\text{-}conn\ c\ l \implies conn\ c\ l = FImp\ a\ b \longleftrightarrow (c = CImp \wedge l = a\ \# \ b\ \# \ [])$
 $wf\text{-}conn\ c\ l \implies conn\ c\ l = FNot\ a \longleftrightarrow (c = CNot \wedge l = a\ \# \ [])$
 $\langle proof \rangle$

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

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

$\langle proof \rangle$

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 $length\ l = 2 \longleftrightarrow wf\text{-}conn\ c\ l$

$\langle proof \rangle$

lemma *wf-conn-not-list-length[iff]*:

fixes $l :: 'v\ propo\ list$
shows $wf\text{-}conn\ CNot\ l \longleftrightarrow length\ l = 1$

$\langle proof \rangle$

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 $\exists\ a.\ l = [a]$

$\langle proof \rangle$

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*:

$length\ l = length\ l' \implies wf\text{-}conn\ c\ l \longleftrightarrow wf\text{-}conn\ c\ l'$

$\langle proof \rangle$

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

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

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

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

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* $\varphi \preceq \psi \Rightarrow \varphi \preceq \psi$
 $\langle \text{proof} \rangle$

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

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

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

lemma *subformula-not-incl-eq*:

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

lemma *wf-subformula-conn-cases*:

$\text{wf-conn } c \ l \implies \varphi \preceq \text{conn } c \ l \longleftrightarrow (\varphi = \text{conn } c \ l \vee (\exists \psi. \psi \in \text{set } l \wedge \varphi \preceq \psi))$
 $\langle \text{proof} \rangle$

lemma *subformula-decomp-explicit[simp]*:

$\varphi \preceq \text{FAnd } \psi \ \psi' \longleftrightarrow (\varphi = \text{FAnd } \psi \ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi') \text{ (is ?P FAnd)}$
 $\varphi \preceq \text{FOr } \psi \ \psi' \longleftrightarrow (\varphi = \text{FOr } \psi \ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi')$
 $\varphi \preceq \text{FEq } \psi \ \psi' \longleftrightarrow (\varphi = \text{FEq } \psi \ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi')$
 $\varphi \preceq \text{FImp } \psi \ \psi' \longleftrightarrow (\varphi = \text{FImp } \psi \ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi')$
 $\langle \text{proof} \rangle$

lemma *wf-conn-helper-facts[iff]*:

$\text{wf-conn } \text{CNot } [\varphi]$
 $\text{wf-conn } \text{CT } []$
 $\text{wf-conn } \text{CF } []$
 $\text{wf-conn } (\text{CVar } x) []$
 $\text{wf-conn } \text{CAnd } [\varphi, \psi]$
 $\text{wf-conn } \text{COr } [\varphi, \psi]$
 $\text{wf-conn } \text{CImp } [\varphi, \psi]$
 $\text{wf-conn } \text{CEq } [\varphi, \psi]$
 $\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma *subformula-conn-decomp[simp]*:

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

lemma *subformula-leaf-explicit[simp]*:

$\varphi \preceq \text{FT} \longleftrightarrow \varphi = \text{FT}$
 $\varphi \preceq \text{FF} \longleftrightarrow \varphi = \text{FF}$
 $\varphi \preceq \text{FVar } x \longleftrightarrow \varphi = \text{FVar } x$
 $\langle \text{proof} \rangle$

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

primrec *vars-of-prop*:: $'v \text{ propo} \Rightarrow 'v \text{ set}$ **where**

$\text{vars-of-prop } \text{FT} = \{\}$ |
 $\text{vars-of-prop } \text{FF} = \{\}$ |
 $\text{vars-of-prop } (\text{FVar } x) = \{x\}$ |
 $\text{vars-of-prop } (\text{FNot } \varphi) = \text{vars-of-prop } \varphi$ |
 $\text{vars-of-prop } (\text{FAnd } \varphi \ \psi) = \text{vars-of-prop } \varphi \cup \text{vars-of-prop } \psi$ |
 $\text{vars-of-prop } (\text{FOr } \varphi \ \psi) = \text{vars-of-prop } \varphi \cup \text{vars-of-prop } \psi$ |
 $\text{vars-of-prop } (\text{FImp } \varphi \ \psi) = \text{vars-of-prop } \varphi \cup \text{vars-of-prop } \psi$ |
 $\text{vars-of-prop } (\text{FEq } \varphi \ \psi) = \text{vars-of-prop } \varphi \cup \text{vars-of-prop } \psi$

lemma *vars-of-prop-incl-conn*:

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

shows $\text{vars-of-prop } \psi \subseteq \text{vars-of-prop } (\text{conn } c \ l)$
 $\langle \text{proof} \rangle$

The set of variables is compatible with the subformula order.

lemma *subformula-vars-of-prop*:

$\varphi \preceq \psi \implies \text{vars-of-prop } \varphi \subseteq \text{vars-of-prop } \psi$
 $\langle \text{proof} \rangle$

2.1.4 Positions

Instead of 1 or 2 we use L or R

datatype $\text{sign} = L \mid R$

We use nil instead of ε .

fun $\text{pos} :: 'v \text{ propo} \Rightarrow \text{sign list set}$ **where**

$\text{pos } FF = \{\Box\} \mid$
 $\text{pos } FT = \{\Box\} \mid$
 $\text{pos } (FVar \ x) = \{\Box\} \mid$
 $\text{pos } (FAnd \ \varphi \ \psi) = \{\Box\} \cup \{L \ \# \ p \mid p. p \in \text{pos } \varphi\} \cup \{R \ \# \ p \mid p. p \in \text{pos } \psi\} \mid$
 $\text{pos } (FOr \ \varphi \ \psi) = \{\Box\} \cup \{L \ \# \ p \mid p. p \in \text{pos } \varphi\} \cup \{R \ \# \ p \mid p. p \in \text{pos } \psi\} \mid$
 $\text{pos } (FEq \ \varphi \ \psi) = \{\Box\} \cup \{L \ \# \ p \mid p. p \in \text{pos } \varphi\} \cup \{R \ \# \ p \mid p. p \in \text{pos } \psi\} \mid$
 $\text{pos } (FImp \ \varphi \ \psi) = \{\Box\} \cup \{L \ \# \ p \mid p. p \in \text{pos } \varphi\} \cup \{R \ \# \ p \mid p. p \in \text{pos } \psi\} \mid$
 $\text{pos } (FNot \ \varphi) = \{\Box\} \cup \{L \ \# \ p \mid p. p \in \text{pos } \varphi\}$

lemma *finite-pos*: $\text{finite } (\text{pos } \varphi)$

$\langle \text{proof} \rangle$

lemma *finite-inj-comp-set*:

fixes $s :: 'v \text{ set}$
assumes $\text{finite}: \text{finite } s$
and $\text{inj}: \text{inj } f$
shows $\text{card } (\{f \ p \mid p. p \in s\}) = \text{card } s$
 $\langle \text{proof} \rangle$

lemma *cons-inject*:

$\text{inj } ((\#) \ s)$
 $\langle \text{proof} \rangle$

lemma *finite-insert-nil-cons*:

$\text{finite } s \implies \text{card } (\text{insert } \Box \ \{L \ \# \ p \mid p. p \in s\}) = 1 + \text{card } \{L \ \# \ p \mid p. p \in s\}$
 $\langle \text{proof} \rangle$

lemma *card-not[simp]*:

$\text{card } (\text{pos } (FNot \ \varphi)) = 1 + \text{card } (\text{pos } \varphi)$
 $\langle \text{proof} \rangle$

lemma *card-seperate*:

assumes $\text{finite } s1$ **and** $\text{finite } s2$
shows $\text{card } (\{L \ \# \ p \mid p. p \in s1\} \cup \{R \ \# \ p \mid p. p \in s2\}) = \text{card } (\{L \ \# \ p \mid p. p \in s1\})$
 $+ \text{card } (\{R \ \# \ p \mid p. p \in s2\})$ (**is** $\text{card } (?L \cup ?R) = \text{card } ?L + \text{card } ?R$)
 $\langle \text{proof} \rangle$

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

lemma *prop-size-vars-of-prop*:

fixes $\varphi :: 'v \text{ propo}$

shows $\text{card } (\text{vars-of-prop } \varphi) \leq \text{prop-size } \varphi$

$\langle \text{proof} \rangle$

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* [] $\varphi \varphi$ |

path-to-l: $c \in \text{binary-connectives} \vee c = \text{CNot} \implies \text{wf-conn } c (\varphi \# l) \implies \text{path-to } p \varphi \varphi' \implies$

path-to (*L* # *p*) (*conn* *c* ($\varphi \# l$)) φ' |

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

path-to (*R* # *p*) (*conn* *c* ($\psi \# \varphi \# []$)) 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* $\varphi \varphi' \implies \varphi' \preceq \varphi$

$\langle \text{proof} \rangle$

lemma *subformula-path-exists*:

fixes $\varphi \varphi' :: 'v \text{ propo}$

shows $\varphi' \preceq \varphi \implies \exists p. \text{path-to } p \varphi \varphi'$

$\langle \text{proof} \rangle$

fun *replace-at* :: *sign list* \Rightarrow *'v propo* \Rightarrow *'v propo* \Rightarrow *'v propo* **where**

replace-at [] - $\psi = \psi$ |

replace-at (*L* # *l*) (*FAnd* $\varphi \varphi'$) $\psi = \text{FAnd } (\text{replace-at } l \varphi \psi) \varphi'$ |

replace-at (*R* # *l*) (*FAnd* $\varphi \varphi'$) $\psi = \text{FAnd } \varphi (\text{replace-at } l \varphi' \psi)$ |

replace-at (*L* # *l*) (*FOr* $\varphi \varphi'$) $\psi = \text{FOr } (\text{replace-at } l \varphi \psi) \varphi'$ |

replace-at (*R* # *l*) (*FOr* $\varphi \varphi'$) $\psi = \text{FOr } \varphi (\text{replace-at } l \varphi' \psi)$ |

replace-at (*L* # *l*) (*FEq* $\varphi \varphi'$) $\psi = \text{FEq } (\text{replace-at } l \varphi \psi) \varphi'$ |

replace-at (*R* # *l*) (*FEq* $\varphi \varphi'$) $\psi = \text{FEq } \varphi (\text{replace-at } l \varphi' \psi)$ |

replace-at (*L* # *l*) (*FImp* $\varphi \varphi'$) $\psi = \text{FImp } (\text{replace-at } l \varphi \psi) \varphi'$ |

replace-at (*R* # *l*) (*FImp* $\varphi \varphi'$) $\psi = \text{FImp } \varphi (\text{replace-at } l \varphi' \psi)$ |

replace-at (*L* # *l*) (*FNot* φ) $\psi = \text{FNot } (\text{replace-at } l \varphi \psi)$

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**

$\mathcal{A} \models \text{FT} = \text{True}$ |

$\mathcal{A} \models \text{FF} = \text{False}$ |

$\mathcal{A} \models \text{FVar } v = (\mathcal{A} \ v)$ |

$\mathcal{A} \models \text{FNot } \varphi = (\neg(\mathcal{A} \models \varphi))$ |

$\mathcal{A} \models \text{FAnd } \varphi_1 \varphi_2 = (\mathcal{A} \models \varphi_1 \wedge \mathcal{A} \models \varphi_2)$ |

$\mathcal{A} \models \text{FOr } \varphi_1 \varphi_2 = (\mathcal{A} \models \varphi_1 \vee \mathcal{A} \models \varphi_2)$ |

$\mathcal{A} \models \text{FImp } \varphi_1 \varphi_2 = (\mathcal{A} \models \varphi_1 \longrightarrow \mathcal{A} \models \varphi_2)$ |

$\mathcal{A} \models \text{FEq } \varphi_1 \varphi_2 = (\mathcal{A} \models \varphi_1 \longleftrightarrow \mathcal{A} \models \varphi_2)$

definition *evalf* (**infix** \models_f 50) **where**

evalf $\varphi \psi = (\forall A. A \models \varphi \longrightarrow A \models \psi)$

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

theorem *deduction-theorem*:

$\varphi \models^f \psi \longleftrightarrow (\forall A. A \models FImp \varphi \psi)$
 $\langle proof \rangle$

A shorter proof:

lemma $\varphi \models^f \psi \longleftrightarrow (\forall A. A \models FImp \varphi \psi)$
 $\langle proof \rangle$

definition *same-over-set*:: $(v \Rightarrow bool) \Rightarrow (v \Rightarrow bool) \Rightarrow v \text{ set} \Rightarrow bool$ **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 $A \models \varphi \longleftrightarrow B \models \varphi$
 $\langle proof \rangle$

end