

Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

April 24, 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	27
1.1.5	Removing Duplicates	28
1.1.6	Set of all Simple Clauses	28
1.1.7	Experiment: Expressing the Entailments as Locales	32
1.1.8	Entailment to be extended	33
1.2	Partial Annotated Herbrand Interpretation	34
1.2.1	Decided Literals	34
1.2.2	Backtracking	40
1.2.3	Decomposition with respect to the First Decided Literals	40
1.2.4	Negation of a Clause	48
1.2.5	Other	53
1.2.6	Extending Entailments to multisets	55
1.2.7	More Lemmas	56
1.2.8	Negation of annotated clauses	56
1.3	Bridging of total and partial Herbrand interpretation	59
2	Normalisation	63
2.1	Logics	63
2.1.1	Definition and Abstraction	63
2.1.2	Properties of the Abstraction	64
2.1.3	Subformulas and Properties	67
2.1.4	Positions	70
2.2	Semantics over the Syntax	73

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  $a :: \langle 'v \text{ literal} \rangle$ 
  using uminus-lit-swap by auto
```

```
lemma uminus-lit-swap:  $- a = b \longleftrightarrow (a :: 'a \text{ literal}) = - b$ 
  by auto
```

```
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  $L C'$ 
  by (cases L) (auto simp: atm-iff-pos-or-neg-lit)
```

```
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  $L C'$ 
  by (auto simp: atm-iff-pos-or-neg-lit)
```

```
lemma atms-of-uminus[simp]:  $\langle \text{atms-of } (\text{uminus } '\# C) = \text{atms-of } C \rangle$ 
  by (auto simp: atms-of-def image-image)
```

```
lemma distinct-mset-atm-ofD:
   $\langle \text{distinct-mset } (\text{atm-of } '\# \text{ mset } xc) \implies \text{distinct } xc \rangle$ 
  by (induction xc) auto
```

```
lemma atms-of-cong-set-mset:
   $\langle \text{set-mset } D = \text{set-mset } D' \implies \text{atms-of } D = \text{atms-of } D' \rangle$ 
  by (auto simp: atms-of-def)
```

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

$\langle NO-MATCH (Pos\ x)\ l \implies NO-MATCH (Neg\ x)\ l \implies$
 $l \in M \longleftrightarrow (\exists l'. (l = Pos\ l' \wedge Pos\ l' \in M) \vee (l = Neg\ l' \wedge Neg\ l' \in M)) \rangle$
by (*cases l*) *auto*

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*:

$\langle x \neq Neg\ (atm-of\ x) \longleftrightarrow is-pos\ x \rangle$
 $\langle x \neq Pos\ (atm-of\ x) \longleftrightarrow is-neg\ x \rangle$
 $\langle \neg x \neq Neg\ (atm-of\ x) \longleftrightarrow is-neg\ x \rangle$
 $\langle \neg x \neq Pos\ (atm-of\ x) \longleftrightarrow is-pos\ x \rangle$
 $\langle Neg\ (atm-of\ x) \neq x \longleftrightarrow is-pos\ x \rangle$
 $\langle Pos\ (atm-of\ x) \neq x \longleftrightarrow is-neg\ x \rangle$
 $\langle Neg\ (atm-of\ x) \neq \neg x \longleftrightarrow is-neg\ x \rangle$
 $\langle Pos\ (atm-of\ x) \neq \neg x \longleftrightarrow is-pos\ x \rangle$
by (*cases x; auto; fail*) $+$

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*: $is-neg\ (\neg\ L) \longleftrightarrow \neg\ is-neg\ L$

by (*cases L*) *auto*

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 $\{\}$ **unfolding** *consistent-interp-def* **by** *auto*

lemma *consistent-interp-single*[*simp*]:

consistent-interp $\{L\}$ **unfolding** *consistent-interp-def* **by** *auto*

lemma *Ex-consistent-interp*: $\langle Ex\ consistent-interp \rangle$

by (*auto simp: consistent-interp-def*)

lemma *consistent-interp-subset*:

assumes

$A \subseteq B$ **and**

consistent-interp B
shows *consistent-interp A*
using *assms unfolding consistent-interp-def* **by** *auto*

lemma *consistent-interp-change-insert*:
 $a \notin A \implies -a \notin A \implies \text{consistent-interp } (\text{insert } (-a) A) \longleftrightarrow \text{consistent-interp } (\text{insert } a A)$
unfolding *consistent-interp-def* **by** *fastforce*

lemma *consistent-interp-insert-pos[simp]*:
 $a \notin A \implies \text{consistent-interp } (\text{insert } a A) \longleftrightarrow \text{consistent-interp } A \wedge -a \notin A$
unfolding *consistent-interp-def* **by** *auto*

lemma *consistent-interp-insert-not-in*:
 $\text{consistent-interp } A \implies a \notin A \implies -a \notin A \implies \text{consistent-interp } (\text{insert } a A)$
unfolding *consistent-interp-def* **by** *auto*

lemma *consistent-interp-unionD*: $\langle \text{consistent-interp } (I \cup I') \implies \text{consistent-interp } I \rangle$
unfolding *consistent-interp-def* **by** *auto*

lemma *consistent-interp-insert-iff*:
 $\langle \text{consistent-interp } (\text{insert } L C) \longleftrightarrow \text{consistent-interp } C \wedge -L \notin C \rangle$
by (*metis consistent-interp-def consistent-interp-insert-pos insert-absorb*)

lemma (*in -*) *distinct-consistent-distinct-atm*:
 $\langle \text{distinct } M \implies \text{consistent-interp } (\text{set } M) \implies \text{distinct-mset } (\text{atm-of } \# \text{ mset } M) \rangle$
by (*induction M*) (*auto simp: atm-of-eq-atm-of*)

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]*:
 $\text{atms-of } (\text{mset } a) = \text{atm-of } \text{'set } a$
by (*induct a*) *auto*

lemma *atms-of-ms-mset-unfold*:
 $\text{atms-of-ms } (\text{mset } \text{' } b) = (\bigcup x \in b. \text{atm-of } \text{'set } x)$
unfolding *atms-of-ms-def* **by** *simp*

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

lemma *atms-of-ms-empty-set[simp]*:
 $\text{atms-of-ms } \{\} = \{\}$
unfolding *atms-of-ms-def* **by** *auto*

lemma *atms-of-ms-mempty[simp]*:
 $\text{atms-of-ms } \{\{\#\}\} = \{\}$
unfolding *atms-of-ms-def* **by** *auto*

lemma *atms-of-ms-mono*:

$A \subseteq B \implies \text{atms-of-ms } A \subseteq \text{atms-of-ms } B$
unfolding *atms-of-ms-def* **by** *auto*

lemma *atms-of-ms-finite[simp]*:
 $\text{finite } \psi \implies \text{finite } (\text{atms-of-ms } \psi)$
unfolding *atms-of-ms-def* **by** *auto*

lemma *atms-of-ms-union[simp]*:
 $\text{atms-of-ms } (\psi \cup \chi) = \text{atms-of-ms } \psi \cup \text{atms-of-ms } \chi$
unfolding *atms-of-ms-def* **by** *auto*

lemma *atms-of-ms-insert[simp]*:
 $\text{atms-of-ms } (\text{insert } \psi \chi) = \text{atms-of-ms } \psi \cup \text{atms-of-ms } \chi$
unfolding *atms-of-ms-def* **by** *auto*

lemma *atms-of-ms-singleton[simp]*: $\text{atms-of-ms } \{L\} = \text{atms-of-ms } L$
unfolding *atms-of-ms-def* **by** *auto*

lemma *atms-of-atms-of-ms-mono[simp]*:
 $A \in \psi \implies \text{atms-of-ms } A \subseteq \text{atms-of-ms } \psi$
unfolding *atms-of-ms-def* **by** *fastforce*

lemma *atms-of-ms-remove-incl*:
shows $\text{atms-of-ms } (\text{Set.remove } a \psi) \subseteq \text{atms-of-ms } \psi$
unfolding *atms-of-ms-def* **by** *auto*

lemma *atms-of-ms-remove-subset*:
 $\text{atms-of-ms } (\varphi - \psi) \subseteq \text{atms-of-ms } \varphi$
unfolding *atms-of-ms-def* **by** *auto*

lemma *finite-atms-of-ms-remove-subset[simp]*:
 $\text{finite } (\text{atms-of-ms } A) \implies \text{finite } (\text{atms-of-ms } (A - C))$
using *atms-of-ms-remove-subset[of A C]* *finite-subset* **by** *blast*

lemma *atms-of-ms-empty-iff*:
 $\text{atms-of-ms } A = \{\} \iff A = \{\{\#\}\} \vee A = \{\}$
apply (*rule iffI*)
apply (*metis (no-types, lifting) atms-empty-iff-empty atms-of-atms-of-ms-mono insert-absorb singleton-iff singleton-insert-inj-eq' subsetI subset-empty*)
apply (*auto; fail*)
done

lemma *in-implies-atm-of-on-atms-of-ms*:
assumes $L \in \# C$ **and** $C \in N$
shows $\text{atm-of } L \in \text{atms-of-ms } N$
using *atms-of-atms-of-ms-mono[of C N]* *assms* **by** (*simp add: atm-of-lit-in-atms-of subset-iff*)

lemma *in-plus-implies-atm-of-on-atms-of-ms*:
assumes $C + \{\#L\} \in N$
shows $\text{atm-of } L \in \text{atms-of-ms } N$
using *in-implies-atm-of-on-atms-of-ms[of - C + {\#L}]* *assms* **by** *auto*

lemma *in-m-in-literals*:
assumes $\text{add-mset } A D \in \psi$
shows $\text{atm-of } A \in \text{atms-of-ms } \psi$
using *assms* **by** (*auto dest: atms-of-atms-of-ms-mono*)

lemma *atms-of-s-union*[simp]:
 $atms-of-s (Ia \cup Ib) = atms-of-s Ia \cup atms-of-s Ib$
unfolding *atms-of-s-def* **by** *auto*

lemma *atms-of-s-single*[simp]:
 $atms-of-s \{L\} = \{atm-of L\}$
unfolding *atms-of-s-def* **by** *auto*

lemma *atms-of-s-insert*[simp]:
 $atms-of-s (insert L Ib) = \{atm-of L\} \cup atms-of-s Ib$
unfolding *atms-of-s-def* **by** *auto*

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

proof

assume $?P$

then show $?Q$ **unfolding** *atms-of-s-def* **by** (*metis image-iff literal.exhaust-sel*)

next

assume $?Q$

then show $?P$ **unfolding** *atms-of-s-def* **by** *force*

qed

lemma *atm-of-in-atm-of-set-in-uminus*:
 $atm-of L' \in atm-of 'B \implies L' \in B \vee - L' \in B$
using *atms-of-s-def* **by** (*cases L'*) *fastforce+*

lemma *finite-atms-of-s*[simp]:
 $\langle finite M \implies finite (atms-of-s M) \rangle$
by (*auto simp: atms-of-s-def*)

lemma

atms-of-s-empty [simp]:

$\langle atms-of-s \{\} = \{\} \rangle$ **and**

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

$\langle atms-of-s x = \{\} \longleftrightarrow x = \{\} \rangle$

by (*auto simp: atms-of-s-def*)

Totality

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

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

lemma *total-over-set-empty*[simp]:
 $total-over-set I \{\}$
unfolding *total-over-set-def* **by** *auto*

lemma *total-over-m-empty*[simp]:
 $total-over-m I \{\}$
unfolding *total-over-m-def* **by** *auto*

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

unfolding *total-over-set-def* **by** *auto*

lemma *total-over-set-insert*[*iff*]:
 $total-over-set\ I\ (insert\ L\ Ls) \longleftrightarrow ((Pos\ L \in I \vee Neg\ L \in I) \wedge total-over-set\ I\ Ls)$
unfolding *total-over-set-def* **by** *auto*

lemma *total-over-set-union*[*iff*]:
 $total-over-set\ I\ (Ls \cup Ls') \longleftrightarrow (total-over-set\ I\ Ls \wedge total-over-set\ I\ Ls')$
unfolding *total-over-set-def* **by** *auto*

lemma *total-over-m-subset*:
 $A \subseteq B \implies total-over-m\ I\ B \implies total-over-m\ I\ A$
using *atms-of-ms-mono*[*of A*] **unfolding** *total-over-m-def* *total-over-set-def* **by** *auto*

lemma *total-over-m-sum*[*iff*]:
shows $total-over-m\ I\ \{C + D\} \longleftrightarrow (total-over-m\ I\ \{C\} \wedge total-over-m\ I\ \{D\})$
unfolding *total-over-m-def* *total-over-set-def* **by** *auto*

lemma *total-over-m-union*[*iff*]:
 $total-over-m\ I\ (A \cup B) \longleftrightarrow (total-over-m\ I\ A \wedge total-over-m\ I\ B)$
unfolding *total-over-m-def* *total-over-set-def* **by** *auto*

lemma *total-over-m-insert*[*iff*]:
 $total-over-m\ I\ (insert\ a\ A) \longleftrightarrow (total-over-set\ I\ (atms-of\ a) \wedge total-over-m\ I\ A)$
unfolding *total-over-m-def* *total-over-set-def* **by** *fastforce*

lemma *total-over-m-extension*:
fixes $I :: 'v\ literal\ set$ **and** $A :: 'v\ clause-set$
assumes *total*: $total-over-m\ I\ A$
shows $\exists I'. total-over-m\ (I \cup I')\ (A \cup B)$
 $\wedge (\forall x \in I'. atm-of\ x \in atms-of-ms\ B \wedge atm-of\ x \notin atms-of-ms\ A)$
proof –
let $?I' = \{Pos\ v \mid v. v \in atms-of-ms\ B \wedge v \notin atms-of-ms\ A\}$
have $\forall x \in ?I'. atm-of\ x \in atms-of-ms\ B \wedge atm-of\ x \notin atms-of-ms\ A$ **by** *auto*
moreover have $total-over-m\ (I \cup ?I')\ (A \cup B)$
using *total* **unfolding** *total-over-m-def* *total-over-set-def* **by** *auto*
ultimately show *?thesis* **by** *blast*
qed

lemma *total-over-m-consistent-extension*:
fixes $I :: 'v\ literal\ set$ **and** $A :: 'v\ clause-set$
assumes
 $total$: $total-over-m\ I\ A$ **and**
 $cons$: $consistent-interp\ I$
shows $\exists I'. total-over-m\ (I \cup I')\ (A \cup B)$
 $\wedge (\forall x \in I'. atm-of\ x \in atms-of-ms\ B \wedge atm-of\ x \notin atms-of-ms\ A) \wedge consistent-interp\ (I \cup I')$
proof –
let $?I' = \{Pos\ v \mid v. v \in atms-of-ms\ B \wedge v \notin atms-of-ms\ A \wedge Pos\ v \notin I \wedge Neg\ v \notin I\}$
have $\forall x \in ?I'. atm-of\ x \in atms-of-ms\ B \wedge atm-of\ x \notin atms-of-ms\ A$ **by** *auto*
moreover have $total-over-m\ (I \cup ?I')\ (A \cup B)$
using *total* **unfolding** *total-over-m-def* *total-over-set-def* **by** *auto*
moreover have $consistent-interp\ (I \cup ?I')$
using *cons* **unfolding** *consistent-interp-def* **by** (*intro allI*) (*rename-tac L, case-tac L, auto*)
ultimately show *?thesis* **by** *blast*
qed

lemma *total-over-set-atms-of-m[simp]*:
total-over-set Ia (atms-of-s Ia)
unfolding *total-over-set-def atms-of-s-def* **by** (*metis image-iff literal.exhaust-sel*)

lemma *total-over-set-literal-defined*:
assumes *add-mset A D ∈ ψs*
and *total-over-set I (atms-of-ms ψs)*
shows *A ∈ I ∨ −A ∈ I*
using *assms unfolding total-over-set-def* **by** (*metis (no-types) Neg-atm-of-iff in-m-in-literals literal.collapse(1) uminus-Neg uminus-Pos*)

lemma *tot-over-m-remove*:
assumes *total-over-m (I ∪ {L}) {ψ}*
and *L: L ∉# ψ −L ∉# ψ*
shows *total-over-m I {ψ}*
unfolding *total-over-m-def total-over-set-def*

proof
fix *l*
assume *l: l ∈ atms-of-ms {ψ}*
then have *Pos l ∈ I ∨ Neg l ∈ I ∨ l = atm-of L*
using *assms unfolding total-over-m-def total-over-set-def* **by** *auto*
moreover have *atm-of L ∉ atms-of-ms {ψ}*
proof (*rule ccontr*)
assume *¬ ?thesis*
then have *atm-of L ∈ atms-of ψ* **by** *auto*
then have *Pos (atm-of L) ∈# ψ ∨ Neg (atm-of L) ∈# ψ*
using *atm-imp-pos-or-neg-lit* **by** *metis*
then have *L ∈# ψ ∨ − L ∈# ψ* **by** (*cases L*) *auto*
then show *False* **using** *L* **by** *auto*
qed
ultimately show *Pos l ∈ I ∨ Neg l ∈ I* **using** *l* **by** *metis*
qed

lemma *total-union*:
assumes *total-over-m I ψ*
shows *total-over-m (I ∪ I') ψ*
using *assms unfolding total-over-m-def total-over-set-def* **by** *auto*

lemma *total-union-2*:
assumes *total-over-m I ψ*
and *total-over-m I' ψ'*
shows *total-over-m (I ∪ I') (ψ ∪ ψ')*
using *assms unfolding total-over-m-def total-over-set-def* **by** *auto*

lemma *total-over-m-alt-def*: $\langle \text{total-over-m } I \ S \longleftrightarrow \text{atms-of-ms } S \subseteq \text{atms-of-s } I \rangle$
by (*auto simp: total-over-m-def total-over-set-def*)

lemma *total-over-set-alt-def*: $\langle \text{total-over-set } M \ A \longleftrightarrow A \subseteq \text{atms-of-s } M \rangle$
by (*auto simp: total-over-set-def*)

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]*: $\neg I \models \{\#\}$

unfolding *true-cls-def* **by** *auto*

lemma *true-cls-singleton*[*iff*]: $I \models \{\#L\# \} \longleftrightarrow I \models_l L$
unfolding *true-cls-def* **by** (*auto split:if-split-asm*)

lemma *true-cls-add-mset*[*iff*]: $I \models \text{add-mset } a \ D \longleftrightarrow a \in I \vee I \models D$
unfolding *true-cls-def* **by** *auto*

lemma *true-cls-union*[*iff*]: $I \models C + D \longleftrightarrow I \models C \vee I \models D$
unfolding *true-cls-def* **by** *auto*

lemma *true-cls-mono-set-mset*: $\text{set-mset } C \subseteq \text{set-mset } D \implies I \models C \implies I \models D$
unfolding *true-cls-def subset-eq Bex-def* **by** *metis*

lemma *true-cls-mono-leD*[*dest*]: $A \subseteq\# B \implies I \models A \implies I \models B$
unfolding *true-cls-def* **by** *auto*

lemma
assumes $I \models \psi$
shows
true-cls-union-increase[*simp*]: $I \cup I' \models \psi$ **and**
true-cls-union-increase'[*simp*]: $I' \cup I \models \psi$
using *assms* **unfolding** *true-cls-def* **by** *auto*

lemma *true-cls-mono-set-mset-l*:
assumes $A \models \psi$
and $A \subseteq B$
shows $B \models \psi$
using *assms* **unfolding** *true-cls-def* **by** *auto*

lemma *true-cls-replicate-mset*[*iff*]: $I \models \text{replicate-mset } n \ L \longleftrightarrow n \neq 0 \wedge I \models_l L$
by (*induct n*) *auto*

lemma *true-cls-empty-entails*[*iff*]: $\neg \{\} \models N$
by (*auto simp add: true-cls-def*)

lemma *true-cls-not-in-remove*:
assumes $L \notin\# \chi$ **and** $I \cup \{L\} \models \chi$
shows $I \models \chi$
using *assms* **unfolding** *true-cls-def* **by** *auto*

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*]: $I \models_s \{\}$
unfolding *true-clss-def* **by** *blast*

lemma *true-clss-singleton*[*iff*]: $I \models_s \{C\} \longleftrightarrow I \models C$
unfolding *true-clss-def* **by** *blast*

lemma *true-clss-empty-entails-empty*[*iff*]: $\{\} \models_s N \longleftrightarrow N = \{\}$
unfolding *true-clss-def* **by** (*auto simp add: true-cls-def*)

lemma *true-cls-insert-l* [*simp*]:
 $M \models A \implies \text{insert } L \ M \models A$
unfolding *true-cls-def* **by** *auto*

lemma *true-clss-union*[*iff*]: $I \models_s CC \cup DD \longleftrightarrow I \models_s CC \wedge I \models_s DD$
unfolding *true-clss-def* **by** *blast*

lemma *true-clss-insert*[*iff*]: $I \models_s \text{insert } C \ DD \longleftrightarrow I \models C \wedge I \models_s DD$
unfolding *true-clss-def* **by** *blast*

lemma *true-clss-mono*: $DD \subseteq CC \implies I \models_s CC \implies I \models_s DD$
unfolding *true-clss-def* **by** *blast*

lemma *true-clss-union-increase*[*simp*]:
assumes $I \models_s \psi$
shows $I \cup I' \models_s \psi$
using *assms* **unfolding** *true-clss-def* **by** *auto*

lemma *true-clss-union-increase'*[*simp*]:
assumes $I' \models_s \psi$
shows $I \cup I' \models_s \psi$
using *assms* **by** (*auto simp add: true-clss-def*)

lemma *true-clss-commute-l*:
 $(I \cup I' \models_s \psi) \longleftrightarrow (I' \cup I \models_s \psi)$
by (*simp add: Un-commute*)

lemma *model-remove*[*simp*]: $I \models_s N \implies I \models_s \text{Set.remove } a \ N$
by (*simp add: true-clss-def*)

lemma *model-remove-minus*[*simp*]: $I \models_s N \implies I \models_s N - A$
by (*simp add: true-clss-def*)

lemma *notin-vars-union-true-clss-true-clss*:
assumes $\forall x \in I'. \text{atm-of } x \notin \text{atms-of-ms } A$
and $\text{atms-of } L \subseteq \text{atms-of-ms } A$
and $I \cup I' \models L$
shows $I \models L$
using *assms* **unfolding** *true-clss-def true-lit-def Bex-def*
by (*metis Un-iff atm-of-lit-in-atms-of contra-subsetD*)

lemma *notin-vars-union-true-clss-true-clss*:
assumes $\forall x \in I'. \text{atm-of } x \notin \text{atms-of-ms } A$
and $\text{atms-of-ms } L \subseteq \text{atms-of-ms } A$
and $I \cup I' \models_s L$
shows $I \models_s L$
using *assms* **unfolding** *true-clss-def true-lit-def Ball-def*
by (*meson atms-of-atms-of-ms-mono notin-vars-union-true-clss-true-clss subset-trans*)

lemma *true-clss-def-set-mset-eq*:
 $\langle \text{set-mset } A = \text{set-mset } B \implies I \models A \longleftrightarrow I \models B \rangle$
by (*auto simp: true-clss-def*)

lemma *true-clss-add-mset-strict*: $\langle I \models \text{add-mset } L \ C \longleftrightarrow L \in I \vee I \models (\text{removeAll-mset } L \ C) \rangle$
using *true-clss-mono-set-mset*[*of* ($\langle \text{removeAll-mset } L \ C \rangle \ C \ I$)]
apply (*cases* $\langle L \in \# \ C \rangle$)
apply (*auto dest: multi-member-split simp: removeAll-notin*)
apply (*metis (mono-tags, lifting) in-multiset-minus-notin-snd in-replicate-mset true-clss-def true-lit-def*)
done

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 $\{\{\#L\#\}\}$
unfolding *satisfiable-def* **by** *fastforce*

lemma *satisfiable-empty[simp]*: $\langle \text{satisfiable } \{\} \rangle$
by (*auto simp: satisfiable-def Ex-consistent-interp*)

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
using *assms total-over-m-union* **unfolding** *satisfiable-def* **by** *blast*

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)

proof

assume $?B$ **then show** $?sat$ **by** (*auto simp add: satisfiable-def*)

next

assume $?sat$

then obtain I **where**

$I \models_s CC$ **and**

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

$\text{tot: total-over-m } I \ CC$

unfolding *satisfiable-def* **by** *auto*

let $?I = \{P. P \in I \wedge \text{atm-of } P \in \text{atms-of-ms } CC\}$

have $I \models_s CC$

using $I \models_s CC$ *in-implies-atm-of-on-atms-of-ms* **unfolding** *true-clss-def Ball-def true-cls-def*

Bex-def true-lit-def

by *blast*

moreover have $\text{cons: consistent-interp } ?I$

using cons **unfolding** *consistent-interp-def* **by** *auto*

moreover have $\text{total-over-m } ?I \ CC$

using tot **unfolding** *total-over-m-def total-over-set-def* **by** *auto*

moreover

have $\text{atms-CC-incl: atms-of-ms } CC \subseteq \text{atm-of } I$

using tot **unfolding** *total-over-m-def total-over-set-def atms-of-ms-def*

by (*auto simp add: atms-of-def atms-of-s-def[symmetric]*)

have $\text{atm-of } ?I = \text{atms-of-ms } CC$

using atms-CC-incl **unfolding** *atms-of-ms-def* **by** *force*

ultimately show $?B$ **by** *auto*

qed

lemma *satisfiable-carac*:

$(\exists I. \text{consistent-interp } I \wedge I \models_s \varphi) \longleftrightarrow \text{satisfiable } \varphi$ **(is $(\exists I. ?Q \ I) \longleftrightarrow ?S$)**

proof

```

assume ?S
then show  $\exists I. ?Q I$  unfolding satisfiable-def by auto
next
assume  $\exists I. ?Q I$ 
then obtain I where cons: consistent-interp I and  $I \models_s \varphi$  by metis
let  $?I' = \{Pos\ v \mid v. v \notin \text{atms-of-}s\ I \wedge v \in \text{atms-of-}ms\ \varphi\}$ 
have consistent-interp ( $I \cup ?I'$ )
  using cons unfolding consistent-interp-def by (intro allI) (rename-tac L, case-tac L, auto)
moreover have total-over-m ( $I \cup ?I'$ )  $\varphi$ 
  unfolding total-over-m-def total-over-set-def by auto
moreover have  $I \cup ?I' \models_s \varphi$ 
  using I unfolding Ball-def true-clss-def true-cls-def by auto
ultimately show ?S unfolding satisfiable-def by blast
qed

```

```

lemma satisfiable-carac'[simp]: consistent-interp I  $\implies I \models_s \varphi \implies \text{satisfiable } \varphi$ 
using satisfiable-carac by metis

```

```

lemma unsatisfiable-mono:
 $\langle N \subseteq N' \implies \text{unsatisfiable } N \implies \text{unsatisfiable } N' \rangle$ 
by (metis (full-types) satisfiable-decreasing subset-Un-eq)

```

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]:  $I \models_m \{\#\}$ 
unfolding true-cls-mset-def by auto

```

```

lemma true-cls-mset-singleton[iff]:  $I \models_m \{\# C \#\} \longleftrightarrow I \models C$ 
unfolding true-cls-mset-def by (auto split: if-split-asm)

```

```

lemma true-cls-mset-union[iff]:  $I \models_m CC + DD \longleftrightarrow I \models_m CC \wedge I \models_m DD$ 
unfolding true-cls-mset-def by fastforce

```

```

lemma true-cls-mset-add-mset[iff]:  $I \models_m \text{add-mset } C\ CC \longleftrightarrow I \models C \wedge I \models_m CC$ 
unfolding true-cls-mset-def by auto

```

```

lemma true-cls-mset-image-mset[iff]:  $I \models_m \text{image-mset } f\ A \longleftrightarrow (\forall x \in \# A. I \models f\ x)$ 
unfolding true-cls-mset-def by fastforce

```

```

lemma true-cls-mset-mono: set-mset  $DD \subseteq \text{set-mset } CC \implies I \models_m CC \implies I \models_m DD$ 
unfolding true-cls-mset-def subset-iff by auto

```

```

lemma true-clss-set-mset[iff]:  $I \models_s \text{set-mset } CC \longleftrightarrow I \models_m CC$ 
unfolding true-clss-def true-cls-mset-def by auto

```

```

lemma true-cls-mset-increasing-r[simp]:
 $I \models_m CC \implies I \cup J \models_m CC$ 
unfolding true-cls-mset-def by auto

```

```

theorem true-cls-remove-unused:
assumes  $I \models \psi$ 
shows  $\{v \in I. \text{atm-of } v \in \text{atms-of } \psi\} \models \psi$ 
using assms unfolding true-cls-def atms-of-def by auto

```

theorem *true-clss-remove-unused*:
assumes $I \models_s \psi$
shows $\{v \in I. \text{atm-of } v \in \text{atms-of-ms } \psi\} \models_s \psi$
unfolding *true-clss-def atms-of-def Ball-def*
proof (*intro allI impI*)
fix x
assume $x \in \psi$
then have $I \models x$
using *assms unfolding true-clss-def atms-of-def Ball-def* **by** *auto*

then have $\{v \in I. \text{atm-of } v \in \text{atms-of } x\} \models x$
by (*simp only: true-clss-remove-unused[of I]*)
moreover have $\{v \in I. \text{atm-of } v \in \text{atms-of } x\} \subseteq \{v \in I. \text{atm-of } v \in \text{atms-of-ms } \psi\}$
using $\langle x \in \psi \rangle$ **by** (*auto simp add: atms-of-ms-def*)
ultimately show $\{v \in I. \text{atm-of } v \in \text{atms-of-ms } \psi\} \models x$
using *true-clss-mono-set-mset-l* **by** *blast*
qed

A simple application of the previous theorem:

lemma *true-clss-union-decrease*:
assumes $II': I \cup I' \models \psi$
and $H: \forall v \in I'. \text{atm-of } v \notin \text{atms-of } \psi$
shows $I \models \psi$
proof –
let $?I = \{v \in I \cup I'. \text{atm-of } v \in \text{atms-of } \psi\}$
have $?I \models \psi$ **using** *true-clss-remove-unused II'* **by** *blast*
moreover have $?I \subseteq I$ **using** H **by** *auto*
ultimately show *?thesis* **using** *true-clss-mono-set-mset-l* **by** *blast*
qed

lemma *multiset-not-empty*:
assumes $M \neq \{\#\}$
and $x \in\# M$
shows $\exists A. x = \text{Pos } A \vee x = \text{Neg } A$
using *assms literal.exhaust-sel* **by** *blast*

lemma *atms-of-ms-empty*:
fixes $\psi :: 'v \text{ clause-set}$
assumes $\text{atms-of-ms } \psi = \{\}$
shows $\psi = \{\} \vee \psi = \{\{\#\}\}$
using *assms* **by** (*auto simp add: atms-of-ms-def*)

lemma *consistent-interp-disjoint*:
assumes *consI: consistent-interp I*
and *disj: atms-of-s A \cap atms-of-s I = $\{\}$*
and *consA: consistent-interp A*
shows *consistent-interp (A \cup I)*
proof (*rule ccontr*)
assume $\neg ?thesis$
moreover have $\bigwedge L. \neg (L \in A \wedge \neg L \in I)$
using *disj unfolding atms-of-s-def* **by** (*auto simp add: rev-image-eqI*)
ultimately show *False*
using *consA consI unfolding consistent-interp-def* **by** (*metis (full-types) Un-iff literal.exhaust-sel uminus-Neg uminus-Pos*)
qed

lemma *total-remove-unused*:

assumes *total-over-m* $I \psi$
shows *total-over-m* $\{v \in I. \text{atm-of } v \in \text{atms-of-ms } \psi\} \psi$
using *assms* **unfolding** *total-over-m-def total-over-set-def*
by (*metis* (*lifting*) *literal.sel(1,2) mem-Collect-eq*)

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

assumes *insert* $a \models D$
and *atm-of* $a \notin \text{atms-of } D$
shows $M' \models D$
using *assms* **by** (*auto simp add: atm-of-lit-in-atms-of true-cls-def*)

lemma *total-over-set-atm-of*:

fixes $I :: 'v \text{ partial-interp}$ **and** $K :: 'v \text{ set}$
shows *total-over-set* $I K \longleftrightarrow (\forall l \in K. l \in (\text{atm-of } I))$
unfolding *total-over-set-def* **by** (*metis atms-of-s-def in-atms-of-s-decomp*)

lemma *true-cls-mset-true-clss-iff*:

$\langle \text{finite } C \implies I \models_m \text{mset-set } C \longleftrightarrow I \models_s C \rangle$
 $\langle I \models_m D \longleftrightarrow I \models_s \text{set-mset } D \rangle$
by (*auto simp: true-clss-def true-cls-mset-def Ball-def*
dest: multi-member-split)

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
using *assms* **unfolding** *tautology-def total-over-set-def true-cls-def Bex-def*
by (*meson atm-iff-pos-or-neg-lit true-lit-def*)

lemma *tautology-minus[simp]*:

assumes $L \in\# A$ **and** $\neg L \in\# A$
shows *tautology* A
by (*metis assms literal.exhaust tautology-Pos-Neg uminus-Neg uminus-Pos*)

lemma *tautology-exists-Pos-Neg*:

assumes *tautology* ψ
shows $\exists p. \text{Pos } p \in\# \psi \wedge \text{Neg } p \in\# \psi$

proof (*rule ccontr*)

assume $p: \neg (\exists p. \text{Pos } p \in\# \psi \wedge \text{Neg } p \in\# \psi)$

let $?I = \{-L \mid L. L \in\# \psi\}$

have *total-over-set* $?I (\text{atms-of } \psi)$

unfolding *total-over-set-def* **using** *atm-imp-pos-or-neg-lit* **by** *force*

moreover **have** $\neg ?I \models \psi$

unfolding *true-cls-def true-lit-def Bex-def* **apply** *clarify*

using p **by** (*rename-tac x L, case-tac L*) *fastforce+*

ultimately **show** *False* **using** *assms* **unfolding** *tautology-def* **by** *auto*

qed

lemma *tautology-decomp*:
 $\text{tautology } \psi \longleftrightarrow (\exists p. \text{Pos } p \in \# \psi \wedge \text{Neg } p \in \# \psi)$
using *tautology-exists-Pos-Neg* **by** *auto*

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

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

lemma *not-tautology-minus*:
 $\langle \neg \text{tautology } A \implies \neg \text{tautology } (A - B) \rangle$
by (*auto simp: tautology-decomp dest: in-diffD*)

lemma *tautology-false[simp]*: $\neg \text{tautology } \{\#\}$
unfolding *tautology-def* **by** *auto*

lemma *tautology-add-mset*:
 $\text{tautology } (\text{add-mset } a L) \longleftrightarrow \text{tautology } L \vee -a \in \# L$
unfolding *tautology-decomp* **by** (*cases a*) *auto*

lemma *tautology-single[simp]*: $\langle \neg \text{tautology } \{\#L\# \} \rangle$
by (*simp add: tautology-add-mset*)

lemma *tautology-union*:
 $\langle \text{tautology } (A + B) \longleftrightarrow \text{tautology } A \vee \text{tautology } B \vee (\exists a. a \in \# A \wedge -a \in \# B) \rangle$
by (*metis tautology-decomp tautology-minus uminus-Neg uminus-Pos union-iff*)

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

lemma *tautology-uminus[simp]*:
 $\langle \text{tautology } (\text{uminus } \# w) \longleftrightarrow \text{tautology } w \rangle$
by (*auto 5 5 simp: tautology-decomp add-mset-eq-add-mset eq-commute[of <Pos -> <->]
eq-commute[of <Neg -> <->]
simp flip: uminus-lit-swap
dest!: multi-member-split*)

lemma *minus-interp-tautology*:
assumes $\{-L \mid L. L \in \# \chi\} \models \chi$
shows *tautology* χ
proof –
obtain L **where** $L \in \# \chi \wedge -L \in \# \chi$
using *assms unfolding true-cls-def* **by** *auto*
then show *?thesis* **using** *tautology-decomp literal.exhaust uminus-Neg uminus-Pos* **by** *metis*
qed

lemma *remove-literal-in-model-tautology*:
assumes $I \cup \{\text{Pos } P\} \models \varphi$
and $I \cup \{\text{Neg } P\} \models \varphi$
shows $I \models \varphi \vee \text{tautology } \varphi$
using *assms unfolding true-cls-def* **by** *auto*

lemma *tautology-imp-tautology*:
fixes $\chi \chi' :: 'v \text{ clause}$
assumes $\forall I. \text{total-over-m } I \{ \chi \} \longrightarrow I \models \chi \longrightarrow I \models \chi'$ **and** *tautology* χ
shows *tautology* χ' **unfolding** *tautology-def*
proof (*intro allI HOL.impI*)
fix $I :: 'v \text{ literal set}$
assume *totI*: *total-over-set* I (*atms-of* χ')
let $?I' = \{ \text{Pos } v \mid v. v \in \text{atms-of } \chi \wedge v \notin \text{atms-of-s } I \}$
have *totI'*: *total-over-m* $(I \cup ?I')$ $\{ \chi \}$ **unfolding** *total-over-m-def* *total-over-set-def* **by** *auto*
then have $\chi: I \cup ?I' \models \chi$ **using** *assms*(2) **unfolding** *total-over-m-def* *tautology-def* **by** *simp*
then have $I \cup (?I' - I) \models \chi'$ **using** *assms*(1) *totI'* **by** *auto*
moreover have $\bigwedge L. L \in \# \chi' \implies L \notin ?I'$
using *totI* **unfolding** *total-over-set-def* **by** (*auto dest: pos-lit-in-atms-of*)
ultimately show $I \models \chi'$ **unfolding** *true-cls-def* **by** *auto*
qed

lemma *not-tautology-mono*: $\langle D' \subseteq \# D \implies \neg \text{tautology } D \implies \neg \text{tautology } D' \rangle$
by (*meson tautology-imp-tautology true-cls-add-mset true-cls-mono-leD*)

lemma *tautology-decomp'*:
 $\langle \text{tautology } C \longleftrightarrow (\exists L. L \in \# C \wedge - L \in \# C) \rangle$
unfolding *tautology-decomp*
apply *auto*
apply (*case-tac* L)
apply *auto*
done

lemma *consistent-interp-tautology*:
 $\langle \text{consistent-interp } (\text{set } M') \longleftrightarrow \neg \text{tautology } (\text{mset } M') \rangle$
by (*auto simp: consistent-interp-def tautology-decomp lit-in-set-iff-atm*)

lemma *consistent-interp-tautology-mset-set*:
 $\langle \text{finite } x \implies \text{consistent-interp } x \longleftrightarrow \neg \text{tautology } (\text{mset-set } x) \rangle$
using *ex-mset*[*of* $\langle \text{mset-set } x \rangle$]
by (*auto simp: consistent-interp-tautology eq-commute*[*of* $\langle \text{mset } \cdot \rangle$] *mset-set-eq-mset-iff* *mset-set-set*)

lemma *tautology-distinct-atm-iff*:
 $\langle \text{distinct-mset } C \implies \text{tautology } C \longleftrightarrow \neg \text{distinct-mset } (\text{atm-of } \# C) \rangle$
by (*induction* C)
(auto simp: tautology-add-mset atm-of-eq-atm-of
dest: multi-member-split)

lemma *not-tautology-minusD*:
 $\langle \text{tautology } (A - B) \implies \text{tautology } A \rangle$
by (*auto simp: tautology-decomp dest: in-diffD*)

Entailment for clauses and propositions

We also need entailment of clauses by other clauses.

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

definition *true-clss-cl* :: '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-clss-cl-refl[simp]*:
 $A \models_f A$
unfolding *true-clss-cl-def* **by** *auto*

lemma *true-clss-cl-empty-empty[iff]*:
 $\langle \{\} \models_p \{\# \} \longleftrightarrow \text{False} \rangle$
unfolding *true-clss-cl-def consistent-interp-def* **by** *auto*

lemma *true-clss-cl-insert-l[simp]*:
 $a \models_f C \implies \text{insert } a\ A \models_p C$
unfolding *true-clss-cl-def true-clss-cl-def true-clss-def* **by** *fastforce*

lemma *true-clss-cl-empty[iff]*:
 $N \models_{fs} \{\}$
unfolding *true-clss-clss-def* **by** *auto*

lemma *true-prop-true-clause[iff]*:
 $\{\varphi\} \models_p \psi \longleftrightarrow \varphi \models_f \psi$
unfolding *true-clss-cl-def true-clss-cl-def* **by** *auto*

lemma *true-clss-clss-true-clss-cl[iff]*:
 $N \models_{ps} \{\psi\} \longleftrightarrow N \models_p \psi$
unfolding *true-clss-clss-def true-clss-cl-def* **by** *auto*

lemma *true-clss-clss-true-clss-clss[iff]*:
 $\{\chi\} \models_{ps} \psi \longleftrightarrow \chi \models_{fs} \psi$
unfolding *true-clss-clss-def true-clss-clss-def* **by** *auto*

lemma *true-clss-clss-empty[simp]*:
 $N \models_{ps} \{\}$
unfolding *true-clss-clss-def* **by** *auto*

lemma *true-clss-cl-subset*:
 $A \subseteq B \implies A \models_p CC \implies B \models_p CC$
unfolding *true-clss-cl-def total-over-m-union* **by** (*simp add: total-over-m-subset true-clss-mono*)

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-cl-subsetI*: $\langle I \models_p A \implies I \subseteq I' \implies I' \models_p A \rangle$
by (*simp add: true-clss-cl-subset*)

lemma *true-clss-cl-mono-l[simp]*:
 $A \models_p CC \implies A \cup B \models_p CC$
by (*auto intro: true-clss-cl-subset*)

lemma *true-clss-cl-mono-l2[simp]*:
 $B \models_p CC \implies A \cup B \models_p CC$
by (*auto intro: true-clss-cl-subset*)

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

$A \models_p CC \implies A \models_p CC + CC'$
unfolding *true-clss-clss-def total-over-m-union total-over-m-sum* **by** *blast*

lemma *true-clss-clss-mono-r'[simp]*:
 $A \models_p CC' \implies A \models_p CC + CC'$
unfolding *true-clss-clss-def total-over-m-union total-over-m-sum* **by** *blast*

lemma *true-clss-clss-mono-add-mset[simp]*:
 $A \models_p CC \implies A \models_p \text{add-mset } L \ CC$
using *true-clss-clss-mono-r[of A CC add-mset L {\#}]* **by** *simp*

lemma *true-clss-clss-union-l[simp]*:
 $A \models_{ps} CC \implies A \cup B \models_{ps} CC$
unfolding *true-clss-clss-def total-over-m-union* **by** *fastforce*

lemma *true-clss-clss-union-l-r[simp]*:
 $B \models_{ps} CC \implies A \cup B \models_{ps} CC$
unfolding *true-clss-clss-def total-over-m-union* **by** *fastforce*

lemma *true-clss-clss-in[simp]*:
 $CC \in A \implies A \models_p CC$
unfolding *true-clss-clss-def true-clss-def total-over-m-union* **by** *fastforce*

lemma *true-clss-clss-insert-l[simp]*:
 $A \models_p C \implies \text{insert } a \ A \models_p C$
unfolding *true-clss-clss-def true-clss-def* **using** *total-over-m-union*
by (*metis Un-iff insert-is-Un sup commute*)

lemma *true-clss-clss-insert-l[simp]*:
 $A \models_{ps} C \implies \text{insert } a \ A \models_{ps} C$
unfolding *true-clss-clss-def true-clss-clss-def true-clss-def* **by** *blast*

lemma *true-clss-clss-union-and[iff]*:
 $A \models_{ps} C \cup D \longleftrightarrow (A \models_{ps} C \wedge A \models_{ps} D)$

proof
{
 fix $A \ C \ D :: 'a \ \text{clause-set}$
 assume $A: A \models_{ps} C \cup D$
 have $A \models_{ps} C$
 unfolding *true-clss-clss-def true-clss-clss-def insert-def total-over-m-insert*
 proof (*intro allI impI*)
 fix I
 assume
 totAC: total-over-m I (A ∪ C) and
 cons: consistent-interp I and
 I: I ⊨s A
 then have *tot: total-over-m I A and tot': total-over-m I C* **by** *auto*
 obtain I' **where**
 tot': total-over-m (I ∪ I') (A ∪ C ∪ D) and
 cons': consistent-interp (I ∪ I') and
 H: ∀x∈I'. atm-of x ∈ atms-of-ms D ∧ atm-of x ∉ atms-of-ms (A ∪ C)
 using *total-over-m-consistent-extension[OF - cons, of A ∪ C] tot tot'* **by** *blast*
 moreover have $I \cup I' \models_s A$ **using** I **by** *simp*
 ultimately have $I \cup I' \models_s C \cup D$ **using** A **unfolding** *true-clss-clss-def* **by** *auto*
 then have $I \cup I' \models_s C \cup D$ **by** *auto*
 then show $I \models_s C$ **using** *notin-vars-union-true-clss-true-clss[of I'] H* **by** *auto*

```

    qed
  } note H = this
  assume A  $\models_{ps}$  C  $\cup$  D
  then show A  $\models_{ps}$  C  $\wedge$  A  $\models_{ps}$  D using H[of A] Un-commute[of C D] by metis
next
  assume A  $\models_{ps}$  C  $\wedge$  A  $\models_{ps}$  D
  then show A  $\models_{ps}$  C  $\cup$  D
    unfolding true-clss-clss-def by auto
qed

```

```

lemma true-clss-clss-insert[iff]:
  A  $\models_{ps}$  insert L Ls  $\longleftrightarrow$  (A  $\models_p$  L  $\wedge$  A  $\models_{ps}$  Ls)
  using true-clss-clss-union-and[of A {L} Ls] by auto

```

```

lemma true-clss-clss-subset:
  A  $\subseteq$  B  $\implies$  A  $\models_{ps}$  CC  $\implies$  B  $\models_{ps}$  CC
  by (metis subset-Un-eq true-clss-clss-union-l)

```

Better suited as intro rule:

```

lemma true-clss-clss-subsetI:
  A  $\models_{ps}$  CC  $\implies$  A  $\subseteq$  B  $\implies$  B  $\models_{ps}$  CC
  by (metis subset-Un-eq true-clss-clss-union-l)

```

```

lemma union-trus-clss-clss[simp]: A  $\cup$  B  $\models_{ps}$  B
  unfolding true-clss-clss-def by auto

```

```

lemma true-clss-clss-remove[simp]:
  A  $\models_{ps}$  B  $\implies$  A  $\models_{ps}$  B - C
  by (metis Un-Diff-Int true-clss-clss-union-and)

```

```

lemma true-clss-clss-subsetE:
  N  $\models_{ps}$  B  $\implies$  A  $\subseteq$  B  $\implies$  N  $\models_{ps}$  A
  by (metis sup.orderE true-clss-clss-union-and)

```

```

lemma true-clss-clss-in-imp-true-clss-clss:
  assumes N  $\models_{ps}$  U
  and A  $\in$  U
  shows N  $\models_p$  A
  using assms mk-disjoint-insert by fastforce

```

```

lemma all-in-true-clss-clss:  $\forall x \in B. x \in A \implies A \models_{ps} B$ 
  unfolding true-clss-clss-def true-clss-def by auto

```

```

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
  using assms unfolding true-clss-clss-def by auto

```

```

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
proof -
  assume a1: A  $\cup$  C  $\models_{ps}$  D
  assume B  $\models_{ps}$  C
  then have f2:  $\bigwedge M. M \cup B \models_{ps} C$ 
    by (meson true-clss-clss-union-l-r)

```

have $\bigwedge M. C \cup (M \cup A) \models_{ps} D$
 using *a1* by (*simp add: Un-commute sup-left-commute*)
 then show *?thesis*
 using *f2* by (*metis (no-types) Un-commute true-clss-clss-left-right true-clss-clss-union-and*)
 qed

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 $N \models_p D + C$
 unfolding *true-clss-clss-def*
proof (*intro allI impI*)
 fix I
 assume
tot: total-over-m I (N \cup {D + C}) and
consistent-interp I and
 $I \models_s N$
 {
 assume $L: L \in I \vee -L \in I$
 then have *total-over-m I {D + {#- L#}}*
 using *tot* by (*cases L*) *auto*
 then have $I \models D + \{ \#- L \# \}$ using $D \langle I \models_s N \rangle \text{ tot } \langle \text{consistent-interp } I \rangle$
 unfolding *true-clss-clss-def* by *auto*
 moreover
 have *total-over-m I {C + {#L#}}*
 using $L \text{ tot}$ by (*cases L*) *auto*
 then have $I \models C + \{ \#L \# \}$
 using $C \langle I \models_s N \rangle \text{ tot } \langle \text{consistent-interp } I \rangle$ unfolding *true-clss-clss-def* by *auto*
 ultimately have $I \models D + C$ using $\langle \text{consistent-interp } I \rangle \text{ consistent-interp-def}$ by *fastforce*
 }
 moreover {
 assume $L: L \notin I \wedge -L \notin I$
 let $?I' = I \cup \{L\}$
 have *consistent-interp ?I'* using $L \langle \text{consistent-interp } I \rangle$ by *auto*
 moreover have *total-over-m ?I' {add-mset (-L) D}*
 using *tot* unfolding *total-over-m-def total-over-set-def* by (*auto simp add: atms-of-def*)
 moreover have *total-over-m ?I' N* using *tot* using *total-union* by *blast*
 moreover have $?I' \models_s N$ using $\langle I \models_s N \rangle$ using *true-clss-union-increase* by *blast*
 ultimately have $?I' \models \text{add-mset } (-L) \ D$
 using D unfolding *true-clss-clss-def* by *blast*
 then have $?I' \models D$ using L by *auto*
 moreover
 have *total-over-set I (atms-of (D + C))* using *tot* by *auto*
 then have $L \notin \# D \wedge -L \notin \# D$
 using L unfolding *total-over-set-def atms-of-def* by (*cases L*) *force+*
 ultimately have $I \models D + C$ unfolding *true-clss-clss-def* by *auto*
 }
 ultimately show $I \models D + C$ by *blast*
 qed

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

lemma *true-clss-clss-sup-iff-add:* $N \models_p C \cup \# D \longleftrightarrow N \models_p C + D$
 by (*auto simp: true-clss-clss-def*)

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 $N \models_p D \cup\# \ C$

using *true-clss-clss-or-true-clss-clss-or-not-true-clss-clss-or*[*OF assms*]

by (*subst true-clss-clss-sup-iff-add*)

lemma *true-clss-clss-tautology-iff*:

$\langle \{ \} \models_p a \longleftrightarrow \text{tautology } a \rangle$ (**is** $\langle ?A \longleftrightarrow ?B \rangle$)

proof

assume $?A$

then have $H: \langle \text{total-over-set } I \ (\text{atms-of } a) \implies \text{consistent-interp } I \implies I \models a \rangle$ **for** I

by (*auto simp: true-clss-clss-def tautology-decomp add-mset-eq-add-mset*
dest!: multi-member-split)

show $?B$

unfolding *tautology-def*

proof (*intro allI impI*)

fix I

assume *tot*: $\langle \text{total-over-set } I \ (\text{atms-of } a) \rangle$

let $?Iinter = \langle I \cap \text{uminus } 'I \rangle$

let $?I = \langle I - ?Iinter \cup \text{Pos } ' \text{atm-of } ' ?Iinter \rangle$

have $\langle \text{total-over-set } ?I \ (\text{atms-of } a) \rangle$

using *tot* **by** (*force simp: total-over-set-def image-image Clausal-Logic.uminus-lit-swap*
simp: image-iff)

moreover have $\langle \text{consistent-interp } ?I \rangle$

unfolding *consistent-interp-def image-iff*

apply *clarify*

subgoal for L

apply (*cases L*)

apply (*auto simp: consistent-interp-def uminus-lit-swap image-iff*)

apply (*case-tac xa; auto; fail*)**+**

done

done

ultimately have $\langle ?I \models a \rangle$

using $H[?I]$ **by** *fast*

moreover have $\langle ?I \subseteq I \rangle$

apply (*rule*)

subgoal for x **by** (*cases x; auto; rename-tac xb; case-tac xb; auto*)

done

ultimately show $\langle I \models a \rangle$

by (*blast intro: true-clss-mono-set-mset-l*)

qed

next

assume $?B$

then show $\langle ?A \rangle$

by (*auto simp: true-clss-clss-def tautology-decomp add-mset-eq-add-mset*
dest!: multi-member-split)

qed

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

by (*cases C*) *auto*

lemma *true-clss-mono-left*:

$\langle I \models_s A \implies I \subseteq J \implies J \models_s A \rangle$

by (metis sup.orderE true-clss-union-increase')

lemma true-clss-remove-alien:

$\langle I \models N \longleftrightarrow \{L. L \in I \wedge \text{atm-of } L \in \text{atms-of } N\} \models N \rangle$

by (auto simp: true-clss-def dest: multi-member-split)

lemma true-clss-remove-alien:

$\langle I \models_s N \longleftrightarrow \{L. L \in I \wedge \text{atm-of } L \in \text{atms-of-ms } N\} \models_s N \rangle$

by (auto simp: true-clss-def true-clss-def in-implies-atm-of-on-atms-of-ms
dest: multi-member-split)

lemma true-clss-alt-def:

$\langle N \models_p \chi \longleftrightarrow$

$(\forall I. \text{atms-of-s } I = \text{atms-of-ms } (N \cup \{\chi\}) \longrightarrow \text{consistent-interp } I \longrightarrow I \models_s N \longrightarrow I \models \chi) \rangle$

apply (rule iffI)

subgoal

unfolding total-over-set-alt-def true-clss-clss-def total-over-m-alt-def

by auto

subgoal

unfolding total-over-set-alt-def true-clss-clss-def total-over-m-alt-def

apply (intro conjI impI allI)

subgoal for I

using consistent-interp-subset[of $\langle \{L \in I. \text{atm-of } L \in \text{atms-of-ms } (N \cup \{\chi\}) \} I$

$\text{true-clss-mono-left}$ [of $\langle \{L \in I. \text{atm-of } L \in \text{atms-of-ms } N \} N$

$\langle \{L \in I. \text{atm-of } L \in \text{atms-of-ms } (N \cup \{\chi\}) \}$

$\text{true-clss-remove-alien}$ [of $I N$]

by (drule-tac $x = \langle \{L \in I. \text{atm-of } L \in \text{atms-of-ms } (N \cup \{\chi\}) \} \rangle$ in spec)

(auto dest: true-clss-mono-set-mset-l)

done

done

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 $\langle ?A \longleftrightarrow ?B \rangle$)

proof (rule iffI)

assume $?A$

then have H :

$\langle \bigwedge I. \text{atms-of-ms } (N \cup \{\chi\}) \subseteq \text{atms-of-s } I \longrightarrow$

$\text{consistent-interp } I \longrightarrow I \models_s N \longrightarrow I \models \chi \rangle$

unfolding total-over-set-alt-def total-over-m-alt-def true-clss-clss-def by blast

show $?B$

unfolding total-over-set-alt-def total-over-m-alt-def true-clss-clss-def

proof (intro conjI impI allI)

fix $I :: \langle \text{'a literal set} \rangle$

assume

$\text{atms}: \langle \text{atms-of-s } I = \text{atms-of-ms } N \rangle$ and

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

$\langle I \models_s N \rangle$

let $?I1 = \langle I \cup \text{uminus } \langle \{L \in \text{set-mset } \chi. \text{atm-of } L \notin \text{atms-of-s } I \} \rangle$

have $\langle \text{atms-of-ms } (N \cup \{\chi\}) \subseteq \text{atms-of-s } ?I1 \rangle$

by (auto simp add: atms in-image-uminus-uminus atm-iff-pos-or-neg-lit)

moreover have $\langle \text{consistent-interp } ?I1 \rangle$

using cons assms by (auto simp: consistent-interp-def)

(rename-tac x ; case-tac x ; auto; fail)+

moreover have $\langle ?I1 \models_s N \rangle$

```

    using ⟨ $I \models_s N$ ⟩ by auto
  ultimately have ⟨ $?I1 \models \chi$ ⟩
    using  $H[?I1]$  by auto
  then show ⟨ $I \models \chi$ ⟩
    using assms by (auto simp: true-cls-def)
qed
next
assume ?B
show ?A
  unfolding total-over-m-alt-def true-clss-alt-def
proof (intro conjI impI allI)
  fix  $I :: \langle 'a \text{ literal set} \rangle$ 
  assume
    atms: ⟨ $\text{atms-of-s } I = \text{atms-of-ms } (N \cup \{\chi\})$ ⟩ and
    cons: ⟨ $\text{consistent-interp } I$ ⟩ and
    ⟨ $I \models_s N$ ⟩
  let ?I1 = ⟨ $\{L \in I. \text{atm-of } L \in \text{atms-of-ms } N\}$ ⟩
  have ⟨ $\text{atms-of-s } ?I1 = \text{atms-of-ms } N$ ⟩
    using atms by (auto simp add: in-image-uminus-uminus atm-iff-pos-or-neg-lit)
  moreover have ⟨ $\text{consistent-interp } ?I1$ ⟩
    using cons assms by (auto simp: consistent-interp-def)
  moreover have ⟨ $?I1 \models_s N$ ⟩
    using ⟨ $I \models_s N$ ⟩ by (subst (asm) true-clss-remove-alien)
  ultimately have ⟨ $?I1 \models \chi$ ⟩
    using ⟨?B⟩ by auto
  then show ⟨ $I \models \chi$ ⟩
    using assms by (auto simp: true-cls-def)
qed
qed

lemma true-clss-restrict-iff:
  assumes ⟨ $\neg \text{tautology } \chi$ ⟩
  shows ⟨ $N \models_p \chi \longleftrightarrow N \models_p \{\#L \in \# \chi. \text{atm-of } L \in \text{atms-of-ms } N \#\}$ ⟩ (is ⟨ $?A \longleftrightarrow ?B$ ⟩)
  apply (subst true-clss-alt-def2[OF assms])
  apply (subst true-clss-alt-def2)
  subgoal using not-tautology-mono[OF - assms] by (auto dest: not-tautology-minus)
  apply (rule HOL.iff-allI)
  apply (auto 5 5 simp: true-cls-def atms-of-s-def dest!: multi-member-split)
done

```

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-cls:
  assumes ⟨ $N \models_p C$ ⟩
    ⟨ $I \models_s N$ ⟩ and
    cons: ⟨ $\text{consistent-interp } I$ ⟩ and
    tauto: ⟨ $\neg \text{tautology } C$ ⟩
  shows ⟨ $I \models C$ ⟩
proof -
  let ?I = ⟨ $I \cup \text{uminus } \{L \in \text{set-mset } C. \text{atm-of } L \notin \text{atms-of-s } I\}$ ⟩
  let ?I2 = ⟨ $?I \cup \text{Pos } \{L \in \text{atms-of-ms } N. L \notin \text{atms-of-s } ?I\}$ ⟩
  have ⟨ $\text{total-over-m } ?I2 (N \cup \{C\})$ ⟩
    by (auto simp: total-over-m-alt-def atms-of-def in-image-uminus-uminus
      dest!: multi-member-split)
  moreover have ⟨ $\text{consistent-interp } ?I2$ ⟩
    using cons tauto unfolding consistent-interp-def

```

```

  apply (intro allI)
  apply (case-tac L)
  by (auto simp: uminus-lit-swap eq-commute[of ⟨Pos → ⟨− →⟩⟩
    eq-commute[of ⟨Neg → ⟨− →⟩⟩])
  moreover have ⟨?I2 ⊨s N⟩
  using ⟨I ⊨s N⟩ by auto
  ultimately have ⟨?I2 ⊨ C⟩
  using assms(1) unfolding true-clss-cls-def by fast
  then show ?thesis
  using tauto
  by (subst (asm) true-cls-remove-alien)
    (auto simp: true-cls-def in-image-uminus-uminus)
qed

```

1.1.4 Subsumptions

lemma *subsumption-total-over-m*:

```

  assumes  $A \subseteq\# B$ 
  shows  $\text{total-over-m } I \{B\} \implies \text{total-over-m } I \{A\}$ 
  using assms unfolding subset-mset-def total-over-m-def total-over-set-def
  by (auto simp add: mset-subset-eq-exists-conv)

```

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

```

  atms-of (D − replicate-mset (count D L) L − replicate-mset (count D (−L)) (−L))
= atms-of D − {atm-of L}
  by (auto simp: atm-of-eq-atm-of atms-of-def in-diff-count dest: in-diffD)

```

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

```

proof (induct card {Pos v | v. v ∈ atms-of D ∧ v ∉ atms-of C} arbitrary: D
rule: nat-less-induct-case)

```

  case 0 note n = this(1) and H = this(2) and incl = this(3)
  then have atms-of D ⊆ atms-of C by auto
  then have  $\forall I. \text{total-over-m } I \{C\} \longrightarrow \text{total-over-m } I \{D\}$ 
    unfolding total-over-m-def total-over-set-def by auto
  moreover have  $\forall I. I \models C \longrightarrow I \models D$  using incl true-cls-mono-leD by blast
  ultimately show ?case using H by auto

```

next

```

  case (Suc n D) note IH = this(1) and card = this(2) and H = this(3) and incl = this(4)
  let ?atms = {Pos v | v. v ∈ atms-of D ∧ v ∉ atms-of C}
  have finite ?atms by auto
  then obtain L where L: L ∈ ?atms
    using card by (metis (no-types, lifting) Collect-empty-eq card-0-eq mem-Collect-eq
      nat.simps(3))
  let ?D' = D − replicate-mset (count D L) L − replicate-mset (count D (−L)) (−L)
  have atms-of-D: atms-of-ms {D} ⊆ atms-of-ms {?D'} ∪ {atm-of L}
    using atms-of-replicate-mset-replicate-mset-uminus by force

```

```

{
  fix I
  assume total-over-m I {?D'}
  then have tot: total-over-m (I ∪ {L}) {D}

```

```

unfolding total-over-m-def total-over-set-def using atms-of-D by auto

assume IDL: I ⊨ ?D'
then have insert L I ⊨ D unfolding true-cls-def by (fastforce dest: in-diffD)
then have insert L I ⊨ φ using H tot by auto

moreover
  have tot': total-over-m (I ∪ {-L}) {D}
    using tot unfolding total-over-m-def total-over-set-def by auto
  have I ∪ {-L} ⊨ D using IDL unfolding true-cls-def by (force dest: in-diffD)
  then have I ∪ {-L} ⊨ φ using H tot' by auto
ultimately have I ⊨ φ ∨ tautology φ
  using L remove-literal-in-model-tautology by force
} note H' = this

have L ∉# C and -L ∉# C using L atm-iff-pos-or-neg-lit by force+
then have C-in-D': C ⊆# ?D' using C ⊆# D by (auto simp: subseteq-mset-def not-in-iff)
have card {Pos v | v. v ∈ atms-of ?D' ∧ v ∉ atms-of C} <
  card {Pos v | v. v ∈ atms-of D ∧ v ∉ atms-of C}
  using L unfolding atms-of-replicate-mset-replicate-mset-uminus[of D L]
  by (auto intro!: psubset-card-mono)
then show ?case
  using IH C-in-D' H' unfolding card[symmetric] by blast
qed

```

1.1.5 Removing Duplicates

```

lemma tautology-remdups-mset[iff]:
  tautology (remdups-mset C) ⟷ tautology C
  unfolding tautology-decomp by auto

lemma atms-of-remdups-mset[simp]: atms-of (remdups-mset C) = atms-of C
  unfolding atms-of-def by auto

lemma true-cls-remdups-mset[iff]: I ⊨ remdups-mset C ⟷ I ⊨ C
  unfolding true-cls-def by auto

lemma true-clss-cls-remdups-mset[iff]: A ⊨p remdups-mset C ⟷ A ⊨p C
  unfolding true-clss-cls-def total-over-m-def by auto

```

1.1.6 Set of all Simple Clauses

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

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

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

definition *simple-clss :: 'v set ⇒ 'v clause set* **where**
simple-clss atms = {C. atms-of C ⊆ atms ∧ ¬tautology C ∧ distinct-mset C}

```

lemma simple-clss-empty[simp]:
  simple-clss {} = {{#}}
  unfolding simple-clss-def by auto

lemma simple-clss-insert:
  assumes  $l \notin \text{atms}$ 
  shows simple-clss (insert  $l$  atms) =
    ((+) {#Pos  $l$ #}) ‘ (simple-clss atms)
     $\cup$  ((+) {#Neg  $l$ #}) ‘ (simple-clss atms)
     $\cup$  simple-clss atms(is ? $I$  = ? $U$ )
proof (standard; standard)
  fix  $C$ 
  assume  $C \in ?I$ 
  then have
    atms: atms-of  $C \subseteq \text{insert } l \text{ atms}$  and
    taut:  $\neg \text{tautology } C$  and
    dist: distinct-mset  $C$ 
    unfolding simple-clss-def by auto
  have  $H$ :  $\bigwedge x. x \in \# C \implies \text{atm-of } x \in \text{insert } l \text{ atms}$ 
    using atm-of-lit-in-atms-of atms by blast
  consider
    (Add)  $L$  where  $L \in \# C$  and  $L = \text{Neg } l \vee L = \text{Pos } l$ 
    | (No)  $\text{Pos } l \notin \# C$   $\text{Neg } l \notin \# C$ 
    by auto
  then show  $C \in ?U$ 
  proof cases
    case Add
    then have LCL:  $L \notin \# C - \{\#L\# \}$ 
      using dist unfolding distinct-mset-def by (auto simp: not-in-iff)
    have LC:  $-L \notin \# C$ 
      using taut Add by auto
    obtain aa :: 'a where
       $f4$ : ( $aa \in \text{atms-of } (\text{remove1-mset } L \text{ } C) \implies aa \in \text{atms} \implies \text{atms-of } (\text{remove1-mset } L \text{ } C) \subseteq \text{atms}$ 
      by (meson subset-iff)
    obtain ll :: 'a literal where
       $aa \notin \text{atm-of } \text{'set-mset } (\text{remove1-mset } L \text{ } C) \vee aa = \text{atm-of } ll \wedge ll \in \# \text{remove1-mset } L \text{ } C$ 
      by blast
    then have atms-of ( $C - \{\#L\# \}$ )  $\subseteq \text{atms}$ 
      using  $f4$  Add LCL LC H unfolding atms-of-def by (metis H in-diffD insertE literal.exhaust-sel uminus-Neg uminus-Pos)
    moreover have  $\neg \text{tautology } (C - \{\#L\# \})$ 
      using taut by (metis Add(1) insert-DiffM tautology-add-mset)
    moreover have distinct-mset ( $C - \{\#L\# \}$ )
      using dist by auto
    ultimately have ( $C - \{\#L\# \}$ )  $\in \text{simple-clss atms}$ 
      using Add unfolding simple-clss-def by auto
    moreover have  $C = \{\#L\# \} + (C - \{\#L\# \})$ 
      using Add by (auto simp: multiset-eq-iff)
    ultimately show ?thesis using Add by blast
  next
  case No
  then have  $C \in \text{simple-clss atms}$ 
    using taut atms dist unfolding simple-clss-def
    by (auto simp: atm-iff-pos-or-neg-lit split: if-split-asm dest!: H)
  then show ?thesis by blast
qed

```

```

next
fix C
assume C ∈ ?U
then consider
  (Add) L C' where C = {#L#} + C' and C' ∈ simple-clss atms and
    L = Pos l ∨ L = Neg l
  | (No) C ∈ simple-clss atms
  by auto
then show C ∈ ?I
proof cases
  case No
  then show ?thesis unfolding simple-clss-def by auto
next
case (Add L C') note C' = this(1) and C = this(2) and L = this(3)
then have
  atms: atms-of C' ⊆ atms and
  taut: ¬tautology C' and
  dist: distinct-mset C'
  unfolding simple-clss-def by auto
have atms-of C ⊆ insert l atms
  using atms C' L by auto
moreover have ¬tautology C
  using taut C' L assms atms by (metis union-mset-add-mset-left add.left-neutral
    neg-lit-in-atms-of pos-lit-in-atms-of subsetCE tautology-add-mset
    uminus-Neg uminus-Pos)
moreover have distinct-mset C
  using dist C' L by (metis union-mset-add-mset-left add.left-neutral assms atms
    distinct-mset-add-mset neg-lit-in-atms-of pos-lit-in-atms-of subsetCE)
ultimately show ?thesis unfolding simple-clss-def by blast
qed
qed

```

```

lemma simple-clss-finite:
  fixes atms :: 'v set
  assumes finite atms
  shows finite (simple-clss atms)
  using assms by (induction rule: finite-induct) (auto simp: simple-clss-insert)

```

```

lemma simple-clssE:
  assumes
    x ∈ simple-clss atms
  shows atms-of x ⊆ atms ∧ ¬tautology x ∧ distinct-mset x
  using assms unfolding simple-clss-def by auto

```

```

lemma cls-in-simple-clss:
  shows {#} ∈ simple-clss s
  unfolding simple-clss-def by auto

```

```

lemma simple-clss-card:
  fixes atms :: 'v set
  assumes finite atms
  shows card (simple-clss atms) ≤ (3::nat) ^ (card atms)
  using assms
proof (induct atms rule: finite-induct)
  case empty
  then show ?case by auto

```

```

next
case (insert l C) note fin = this(1) and l = this(2) and IH = this(3)
have notin:
   $\wedge C'. \text{add-mset } (\text{Pos } l) \ C' \notin \text{simple-clss } C$ 
   $\wedge C'. \text{add-mset } (\text{Neg } l) \ C' \notin \text{simple-clss } C$ 
  using l unfolding simple-clss-def by auto
have H:  $\wedge C' D. \{\# \text{Pos } l\} + C' = \{\# \text{Neg } l\} + D \implies D \in \text{simple-clss } C \implies \text{False}$ 
proof -
  fix C' D
  assume C'D:  $\{\# \text{Pos } l\} + C' = \{\# \text{Neg } l\} + D$  and D:  $D \in \text{simple-clss } C$ 
  then have Pos l  $\in \#$  D
    by (auto simp: add-mset-eq-add-mset-ne)
  then have l  $\in$  atms-of D
    by (simp add: atm-iff-pos-or-neg-lit)
  then show False using D l unfolding simple-clss-def by auto
qed
let ?P = ((+)  $\{\# \text{Pos } l\}$ ) ' (simple-clss C)
let ?N = ((+)  $\{\# \text{Neg } l\}$ ) ' (simple-clss C)
let ?O = simple-clss C
have card (?P  $\cup$  ?N  $\cup$  ?O) = card (?P  $\cup$  ?N) + card ?O
  apply (subst card-Un-disjoint)
  using l fin by (auto simp: simple-clss-finite notin)
moreover have card (?P  $\cup$  ?N) = card ?P + card ?N
  apply (subst card-Un-disjoint)
  using l fin H by (auto simp: simple-clss-finite notin)
moreover
  have card ?P = card ?O
    using inj-on-iff-eq-card[of ?O (+)  $\{\# \text{Pos } l\}$ ]
    by (auto simp: fin simple-clss-finite inj-on-def)
  moreover have card ?N = card ?O
    using inj-on-iff-eq-card[of ?O (+)  $\{\# \text{Neg } l\}$ ]
    by (auto simp: fin simple-clss-finite inj-on-def)
  moreover have  $(3::\text{nat}) \wedge \text{card } (\text{insert } l \ C) = 3 \wedge (\text{card } C) + 3 \wedge (\text{card } C) + 3 \wedge (\text{card } C)$ 
    using l by (simp add: fin mult-2-right numeral-3-eq-3)
  ultimately show ?case using IH l by (auto simp: simple-clss-insert)
qed

lemma simple-clss-mono:
  assumes incl:  $\text{atms} \subseteq \text{atms}'$ 
  shows simple-clss  $\text{atms} \subseteq \text{simple-clss } \text{atms}'$ 
  using assms unfolding simple-clss-def by auto

lemma distinct-mset-not-tautology-implies-in-simple-clss:
  assumes distinct-mset  $\chi$  and  $\neg \text{tautology } \chi$ 
  shows  $\chi \in \text{simple-clss } (\text{atms-of } \chi)$ 
  using assms unfolding simple-clss-def by auto

lemma simplified-in-simple-clss:
  assumes distinct-mset-set  $\psi$  and  $\forall \chi \in \psi. \neg \text{tautology } \chi$ 
  shows  $\psi \subseteq \text{simple-clss } (\text{atms-of-ms } \psi)$ 
  using assms unfolding simple-clss-def
  by (auto simp: distinct-mset-set-def atms-of-ms-def)

lemma simple-clss-element-mono:
   $\langle x \in \text{simple-clss } A \implies y \subseteq \# x \implies y \in \text{simple-clss } A \rangle$ 
  by (auto simp: simple-clss-def atms-of-def intro: distinct-mset-mono)

```

dest: not-tautology-mono)

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]:
   $I \models_{es} \{\}$ 
  unfolding entails-def by auto

lemma entails-single[iff]:
   $I \models_{es} \{a\} \longleftrightarrow I \models_e a$ 
  unfolding entails-def by auto

lemma entails-insert-l[simp]:
   $M \models_{es} A \implies \text{insert } L \ M \models_{es} A$ 
  unfolding entails-def by (metis Un-commute entail-union insert-is-Un)

lemma entails-union[iff]:  $I \models_{es} CC \cup DD \longleftrightarrow I \models_{es} CC \wedge I \models_{es} DD$ 
  unfolding entails-def by blast

lemma entails-insert[iff]:  $I \models_{es} \text{insert } C \ DD \longleftrightarrow I \models_e C \wedge I \models_{es} DD$ 
  unfolding entails-def by blast

lemma entails-insert-mono:  $DD \subseteq CC \implies I \models_{es} CC \implies I \models_{es} DD$ 
  unfolding entails-def by blast

lemma entails-union-increase[simp]:
  assumes  $I \models_{es} \psi$ 
  shows  $I \cup I' \models_{es} \psi$ 
  using assms unfolding entails-def by auto

lemma true-clss-commute-l:
   $I \cup I' \models_{es} \psi \longleftrightarrow I' \cup I \models_{es} \psi$ 
  by (simp add: Un-commute)

lemma entails-remove[simp]:  $I \models_{es} N \implies I \models_{es} \text{Set.remove } a \ N$ 
  by (simp add: entails-def)

lemma entails-remove-minus[simp]:  $I \models_{es} N \implies I \models_{es} N - A$ 
  by (simp add: entails-def)

end

interpretation true-cls: entail true-cls
  by standard (auto simp add: true-cls-def)

```


1.1.8 Entailment to be extended

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

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

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

where

$I \models_{\text{sext}} N \iff (\forall J. I \subseteq J \longrightarrow \text{consistent-interp } J \longrightarrow \text{total-over-m } J \ N \longrightarrow J \models_s N)$

lemma *true-clss-imp-true-clss-ext*:

$I \models_s N \implies I \models_{\text{sext}} N$

unfolding *true-clss-ext-def* **by** (metis sup.orderE true-clss-union-increase')

lemma *true-clss-ext-decrease-right-remove-r*:

assumes $I \models_{\text{sext}} N$

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

unfolding *true-clss-ext-def*

proof (intro allI impI)

fix J

assume

$I \subseteq J$ **and**

cons: consistent-interp J **and**

tot: total-over-m J ($N - \{C\}$)

let $?J = J \cup \{Pos \ (atm\text{-}of \ P) \mid P. P \in \# \ C \wedge atm\text{-}of \ P \notin atm\text{-}of \ ' \ J\}$

have $I \subseteq ?J$ **using** $\langle I \subseteq J \rangle$ **by** auto

moreover have consistent-interp $?J$

using *cons* **unfolding** consistent-interp-def **apply** (intro allI)

by (rename-tac L , case-tac L) (fastforce simp add: image-iff)+

moreover have total-over-m $?J \ N$

using *tot* **unfolding** total-over-m-def total-over-set-def atms-of-ms-def

apply clarify

apply (rename-tac $l \ a$, case-tac $a \in N - \{C\}$)

apply (auto; fail)

using atms-of-s-def atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set

by (fastforce simp: atms-of-def)

ultimately have $?J \models_s N$

using *assms* **unfolding** true-clss-ext-def **by** blast

then have $?J \models_s N - \{C\}$ **by** auto

have $\{v \in ?J. atm\text{-}of \ v \in atms\text{-}of\text{-}ms \ (N - \{C\})\} \subseteq J$

using *tot* **unfolding** total-over-m-def total-over-set-def

by (auto intro!: rev-image-eqI)

then show $J \models_s N - \{C\}$

using true-clss-remove-unused[OF $\langle ?J \models_s N - \{C\} \rangle$] **unfolding** true-clss-def

by (meson true-clss-mono-set-mset-l)

qed

lemma *consistent-true-clss-ext-satisfiable*:

assumes consistent-interp I **and** $I \models_{\text{sext}} A$

shows satisfiable A

by (metis Un-empty-left assms satisfiable-carac subset-Un-eq sup.left-idem

total-over-m-consistent-extension total-over-m-empty true-clss-ext-def)

```

lemma not-consistent-true-clss-ext:
  assumes  $\neg \text{consistent-interp } I$ 
  shows  $I \models_{\text{sext}} A$ 
  by (meson assms consistent-interp-subset true-clss-ext-def)

lemma inj-on-Pos:  $\langle \text{inj-on Pos } A \rangle$  and
  inj-on-Neg:  $\langle \text{inj-on Neg } A \rangle$ 
  by (auto simp: inj-on-def)

lemma inj-on-uminus-lit:  $\langle \text{inj-on uminus } A \rangle$  for  $A :: \langle 'a \text{ literal set} \rangle$ 
  by (auto simp: inj-on-def)

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

```

```

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
     $\langle \bigwedge L \text{ m xs. } P \text{ xs} \implies P (\text{Propagated } L \text{ m} \# \text{ xs}) \rangle$ 
  shows  $\langle P \text{ xs} \rangle$ 
  using assms apply (induction xs, simp)
  by (rename-tac a xs, case-tac a) auto

```

```

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

```

```

lemma is-propedE:  $\langle \text{is-proped } L \implies (\bigwedge K \text{ C. } L = \text{Propagated } K \text{ C} \implies P) \implies P \rangle$ 
  by (cases L) auto

```

```

lemma is-decided-no-proped-iff:  $\langle \text{is-decided } L \longleftrightarrow \neg \text{is-proped } L \rangle$ 
  by (cases L) auto

```

lemma *not-is-decidedE*:
 $\langle \neg \text{is-decided } E \implies (\bigwedge K \ C. \ E = \text{Propagated } K \ C \implies \text{thesis}) \implies \text{thesis} \rangle$
by (cases E) auto

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

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

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

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

lemma *lits-of-l-empty[simp]*:
 $\langle \text{lits-of } \{\} = \{\} \rangle$
unfolding *lits-of-def* **by** auto

lemma *lits-of-insert[simp]*:
 $\langle \text{lits-of } (\text{insert } L \ Ls) = \text{insert } (\text{lit-of } L) (\text{lits-of } Ls) \rangle$
unfolding *lits-of-def* **by** auto

lemma *lits-of-l-Un[simp]*:
 $\langle \text{lits-of } (l \cup l') = \text{lits-of } l \cup \text{lits-of } l' \rangle$
unfolding *lits-of-def* **by** auto

lemma *finite-lits-of-def[simp]*:
 $\langle \text{finite } (\text{lits-of-l } L) \rangle$
unfolding *lits-of-def* **by** auto

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

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

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

lemma *atms-of-ms-lambda-lit-of-is-atm-of-lit-of[simp]*:
 $\langle \text{atms-of-ms } (\text{unmark-l } M') = \text{atm-of } ' \text{lits-of-l } M' \rangle$
unfolding *atms-of-ms-def lits-of-def* **by** auto

lemma *lits-of-l-empty-is-empty[iff]*:
 $\langle \text{lits-of-l } M = \{\} \longleftrightarrow M = [] \rangle$
by (induct M) (auto simp: *lits-of-def*)

lemma *in-unmark-l-in-lits-of-l-iff*: $\langle \{\# L \# \} \in \text{unmark-l } M \longleftrightarrow L \in \text{lits-of-l } M \rangle$
by (induction M) auto

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

Entailment

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

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

lemma $true-annot-empty-model[simp]$:

$\langle \neg[] \models_a \psi \rangle$

unfolding $true-annot-def\ true-cls-def$ **by** $simp$

lemma $true-annot-empty[simp]$:

$\langle \neg I \models_a \{\#\} \rangle$

unfolding $true-annot-def\ true-cls-def$ **by** $simp$

lemma $empty-true-annots-def[iff]$:

$\langle [] \models_{as} \psi \longleftrightarrow \psi = \{\} \rangle$

unfolding $true-annots-def$ **by** $auto$

lemma $true-annots-empty[simp]$:

$\langle I \models_{as} \{\} \rangle$

unfolding $true-annots-def$ **by** $auto$

lemma $true-annots-single-true-annot[iff]$:

$\langle I \models_{as} \{C\} \longleftrightarrow I \models_a C \rangle$

unfolding $true-annots-def$ **by** $auto$

lemma $true-annot-insert-l[simp]$:

$\langle M \models_a A \Longrightarrow L \# M \models_a A \rangle$

unfolding $true-annot-def$ **by** $auto$

lemma $true-annots-insert-l\ [simp]$:

$\langle M \models_{as} A \Longrightarrow L \# M \models_{as} A \rangle$

unfolding $true-annots-def$ **by** $auto$

lemma $true-annots-union[iff]$:

$\langle M \models_{as} A \cup B \longleftrightarrow (M \models_{as} A \wedge M \models_{as} B) \rangle$

unfolding $true-annots-def$ **by** $auto$

lemma $true-annots-insert[iff]$:

$\langle M \models_{as} insert\ a\ A \longleftrightarrow (M \models_a a \wedge M \models_{as} A) \rangle$

unfolding $true-annots-def$ **by** $auto$

lemma $true-annot-append-l$:

$\langle M \models_a A \Longrightarrow M' @ M \models_a A \rangle$

unfolding $true-annot-def$ **by** $auto$

lemma $true-annots-append-l$:

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

unfolding $true-annots-def$ **by** $(auto\ simp:\ true-annot-append-l)$

Link between \models_{as} and \models_s :

lemma $true-annots-true-cl$:

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

unfolding $true-annots-def\ Ball-def\ true-annot-def\ true-clss-def$ **by** $auto$

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

$\langle a \in \text{lits-of-l } M \longleftrightarrow M \models_a \{\#a\# \} \rangle$
unfolding *true-annot-def lits-of-def* **by** *auto*

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

$\langle L \# M \models_a A \implies \text{lit-of } L \notin \# A \implies M \models_a A \rangle$
unfolding *true-annot-def true-clss-def* **by** *auto*

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

$\langle I \models_s \text{unmark-l } MLs \implies \text{lits-of-l } MLs \subseteq I \rangle$
unfolding *true-clss-def lits-of-def* **by** *auto*

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

$\langle MLs \models_a \psi \implies \text{set } (\text{map unmark } MLs) \models_p \psi \rangle$
unfolding *true-annot-def true-clss-clss-def true-clss-def*
by (*auto dest: true-clss-singleton-lit-of-implies-incl*)

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

$\langle MLs \models_{as} \psi \implies \text{set } (\text{map unmark } MLs) \models_{ps} \psi \rangle$
by (*auto*
dest: true-clss-singleton-lit-of-implies-incl
simp add: true-clss-def true-annots-def true-annot-def lits-of-def true-clss-def
true-clss-clss-def)

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

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

proof –

have *: $\langle \text{lit-of 'Decided ' set } M = \text{set } M \rangle$ **unfolding** *lits-of-def* **by** *force*
show ?thesis **by** (*simp add: true-annots-true-clss * lits-of-def*)

qed

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

unfolding *true-annot-def lits-of-def* **by** *auto*

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

$\langle A \models_{as} \Psi \implies \text{unmark-l } A \models_{ps} \Psi \rangle$
unfolding *true-clss-clss-def true-annots-def true-clss-def*
by (*auto dest!: true-clss-singleton-lit-of-implies-incl*
simp: lits-of-def true-annot-def true-clss-def)

lemma *true-annot-commute*:

$\langle M @ M' \models_a D \longleftrightarrow M' @ M \models_a D \rangle$
unfolding *true-annot-def* **by** (*simp add: Un-commute*)

lemma *true-annots-commute*:

$\langle M @ M' \models_{as} D \longleftrightarrow M' @ M \models_{as} D \rangle$
unfolding *true-annots-def* **by** (*auto simp: true-annot-commute*)

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

$\langle \text{set } I \subseteq \text{set } I' \implies I \models_a N \implies I' \models_a N \rangle$
using *true-clss-mono-set-mset-l* **unfolding** *true-annot-def lits-of-def*
by (*metis (no-types) Un-commute Un-upper1 image-Un sup.orderE*)

lemma *true-annots-mono*:

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

unfolding *true-annots-def* **by** *auto*

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]*:

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

unfolding *defined-lit-def* **by** *auto*

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

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

shows

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

using *assms* **by** (*metis* (*full-types*) *lit-of.elims*)

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

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

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

using *assms* **by** (*cases* *x*) *auto*

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

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

unfolding *defined-lit-def* **by** (*auto* *simp* *add: lits-of-def rev-image-eqI dest!: literal-is-lit-of-decided*)

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

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

by (*simp* *add: true-annots-true-cls*)

lemma *defined-lit-map*:

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

unfolding *defined-lit-def* **apply** (*rule iffI*)

using *image-iff* **apply** *fastforce*

by (*fastforce* *simp* *add: atm-of-eq-atm-of dest: atm-imp-decided-or-proped*)

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

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

unfolding *defined-lit-def* **by** *auto*

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

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

unfolding *lits-of-def* **by** (*metis* *lits-of-def true-annot-iff-decided-or-true-lit true-lit-def*)

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 $\langle \text{consistent-interp } (\text{insert } L \ (\text{lits-of-l } Ls)) \rangle$
using *assms* **unfolding** *consistent-interp-def* **by** (auto simp: *Decided-Propagated-in-iff-in-lits-of-l*)

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

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

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

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

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

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

lemma *in-lits-of-l-defined-litD*: $\langle L\text{-max} \in \text{lits-of-l } M \implies \text{defined-lit } M \ L\text{-max} \rangle$
by (auto simp: *Decided-Propagated-in-iff-in-lits-of-l*)

lemma *undefined-notin*: $\langle \text{undefined-lit } M \ (\text{lit-of } x) \implies x \notin \text{set } M \rangle$ **for** $x \ M$
by (metis *in-lits-of-l-defined-litD insert-iff lits-of-insert mk-disjoint-insert*)

lemma *uminus-lits-of-l-definedD*:
 $\langle -x \in \text{lits-of-l } M \implies \text{defined-lit } M \ x \rangle$
by (simp add: *Decided-Propagated-in-iff-in-lits-of-l*)

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

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

lemma *defined-lit-mono*:
 $\langle \text{defined-lit } M2 \ L \implies \text{set } M2 \subseteq \text{set } M3 \implies \text{defined-lit } M3 \ L \rangle$
by (auto simp: *Decided-Propagated-in-iff-in-lits-of-l*)

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

by (auto simp: Decided-Propagated-in-iff-in-lits-of-l lits-of-def)

1.2.2 Backtracking

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

lemma *backtrack-split-fst-not-decided*: $\langle a \in \text{set } (\text{fst } (\text{backtrack-split } l)) \implies \neg \text{is-decided } a \rangle$
 by (induct l rule: ann-lit-list-induct) auto

lemma *backtrack-split-snd-hd-decided*:
 $\langle \text{snd } (\text{backtrack-split } l) \neq [] \implies \text{is-decided } (\text{hd } (\text{snd } (\text{backtrack-split } l))) \rangle$
 by (induct l rule: ann-lit-list-induct) auto

lemma *backtrack-split-list-eq[simp]*:
 $\langle \text{fst } (\text{backtrack-split } l) @ (\text{snd } (\text{backtrack-split } l)) = l \rangle$
 by (induct l rule: ann-lit-list-induct) auto

lemma *backtrack-snd-empty-not-decided*:
 $\langle \text{backtrack-split } M = (M'', []) \implies \forall l \in \text{set } M. \neg \text{is-decided } l \rangle$
 by (metis append-Nil2 backtrack-split-fst-not-decided backtrack-split-list-eq snd-conv)

lemma *backtrack-split-some-is-decided-then-snd-has-hd*:
 $\langle \exists l \in \text{set } M. \text{is-decided } l \implies \exists M' L' M''. \text{backtrack-split } M = (M', L' \# M'') \rangle$
 by (metis backtrack-snd-empty-not-decided list.exhaust prod.collapse)

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*:
 $\langle \text{backtrack-split } M = (\text{takeWhile } (\text{Not } o \text{ is-decided}) \ M, \text{dropWhile } (\text{Not } o \text{ is-decided}) \ M) \rangle$
 by (induction M rule: ann-lit-list-induct) auto

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**
 $\langle \text{get-all-ann-decomposition } (\text{Decided } L \ \# \ Ls) =$
 $(\text{Decided } L \ \# \ Ls, []) \ \# \ \text{get-all-ann-decomposition } Ls \rangle$ |
 $\langle \text{get-all-ann-decomposition } (\text{Propagated } L \ P \ \# \ Ls) =$
 $(\text{apsnd } ((\#) (\text{Propagated } L \ P)) (\text{hd } (\text{get-all-ann-decomposition } Ls)))$
 $\ \# \ \text{tl } (\text{get-all-ann-decomposition } Ls) \rangle$ |
 $\langle \text{get-all-ann-decomposition } [] = [([], [])] \rangle$

value $\langle \text{get-all-ann-decomposition } [\text{Propagated } A5 \ B5, \text{Decided } C4, \text{Propagated } A3 \ B3],$

Propagated A2 B2, Decided C1, Propagated A0 B0⟩

Now we can prove several simple properties about the function.

lemma *get-all-ann-decomposition-never-empty*[iff]:
 ⟨*get-all-ann-decomposition* $M = [] \longleftrightarrow \text{False}$ ⟩
by (induct M , simp) (rename-tac a xs , case-tac a , auto)

lemma *get-all-ann-decomposition-never-empty-sym*[iff]:
 ⟨ $[] = \text{get-all-ann-decomposition } M \longleftrightarrow \text{False}$ ⟩
using *get-all-ann-decomposition-never-empty*[of M] **by** *presburger*

lemma *get-all-ann-decomposition-decomp*:
 ⟨ $\text{hd } (\text{get-all-ann-decomposition } S) = (a, c) \implies S = c @ a$ ⟩
proof (induct S arbitrary: a c)
 case Nil
 then show ?case **by** simp
next
 case (Cons x A)
 then show ?case **by** (cases x ; cases ⟨ $\text{hd } (\text{get-all-ann-decomposition } A)$ ⟩) auto
qed

lemma *get-all-ann-decomposition-backtrack-split*:
 ⟨ $\text{backtrack-split } S = (M, M') \longleftrightarrow \text{hd } (\text{get-all-ann-decomposition } S) = (M', M)$ ⟩
proof (induction S arbitrary: M M')
 case Nil
 then show ?case **by** auto
next
 case (Cons a S)
 then show ?case **using** *backtrack-split-takeWhile-dropWhile* **by** (cases a) force+
qed

lemma *get-all-ann-decomposition-Nil-backtrack-split-snd-Nil*:
 ⟨ $\text{get-all-ann-decomposition } S = [([], A)] \implies \text{snd } (\text{backtrack-split } S) = []$ ⟩
by (simp add: *get-all-ann-decomposition-backtrack-split sndI*)

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

lemma *get-all-ann-decomposition-length-1-fst-empty-or-length-1*:
assumes ⟨ $\text{get-all-ann-decomposition } M = (a, b) \# []$ ⟩
shows ⟨ $a = [] \vee (\text{length } a = 1 \wedge \text{is-decided } (\text{hd } a) \wedge \text{hd } a \in \text{set } M)$ ⟩
using *assms*
proof (induct M arbitrary: a b rule: *ann-lit-list-induct*)
 case Nil then show ?case **by** simp
next
 case (Decided L mark)
 then show ?case **by** simp
next
 case (Propagated L mark M)
 then show ?case **by** (cases ⟨ $\text{get-all-ann-decomposition } M$ ⟩) force+
qed

lemma *get-all-ann-decomposition-fst-empty-or-hd-in-M*:
assumes ⟨ $\text{get-all-ann-decomposition } M = (a, b) \# l$ ⟩
shows ⟨ $a = [] \vee (\text{is-decided } (\text{hd } a) \wedge \text{hd } a \in \text{set } M)$ ⟩
using *assms*

```

proof (induct M arbitrary: a b rule: ann-lit-list-induct)
  case Nil
  then show ?case by auto
next
  case (Decided L ann xs)
  then show ?case by auto
next
  case (Propagated L m xs) note IH = this(1) and d = this(2)
  then show ?case
    using IH[of ⟨fst (hd (get-all-ann-decomposition xs))⟩ ⟨snd (hd (get-all-ann-decomposition xs))⟩]
    by (cases ⟨get-all-ann-decomposition xs⟩; cases a) auto
qed

```

```

lemma get-all-ann-decomposition-snd-not-decided:
  assumes ⟨(a, b) ∈ set (get-all-ann-decomposition M)⟩
  and ⟨L ∈ set b⟩
  shows ⟨¬is-decided L⟩
  using assms apply (induct M arbitrary: a b rule: ann-lit-list-induct, simp)
  by (rename-tac L' xs a b, case-tac ⟨get-all-ann-decomposition xs⟩; fastforce)+

```

```

lemma tl-get-all-ann-decomposition-skip-some:
  assumes ⟨x ∈ set (tl (get-all-ann-decomposition M1))⟩
  shows ⟨x ∈ set (tl (get-all-ann-decomposition (M0 @ M1)))⟩
  using assms
  by (induct M0 rule: ann-lit-list-induct)
    (auto simp add: list.set-sel(2))

```

```

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

```

```

proof (induction M0 rule: ann-lit-list-induct)
  case Nil
  then show ?case by auto
next
  case (Decided L M0)
  then show ?case by auto
next
  case (Propagated L C M0) note xy = this(1)[OF this(2-)] and hd = this(2)
  then show ?case
    by (cases ⟨get-all-ann-decomposition (M0 @ Decided K # M1)⟩)
      (auto dest!: get-all-ann-decomposition-decomp
        arg-cong[of ⟨get-all-ann-decomposition -⟩ - hd])
qed

```

```

lemma in-get-all-ann-decomposition-in-get-all-ann-decomposition-prepend:
  ⟨(a, b) ∈ set (get-all-ann-decomposition M') ⟹
  ∃ b'. (a, b' @ b) ∈ set (get-all-ann-decomposition (M @ M'))⟩
apply (induction M rule: ann-lit-list-induct)
  apply (metis append-Nil)
  apply auto[]
  by (rename-tac L' m xs, case-tac ⟨get-all-ann-decomposition (xs @ M')⟩) auto

```

```

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

```

```

using assms
proof (induct M arbitrary: a b rule: ann-lit-list-induct)
  case Nil then show ?case by simp
next
  case (Decided l M)
  then show ?case by auto
next
  case (Propagated l mark M)
  then show ?case by (cases (get-all-ann-decomposition M)) force+
qed

lemma get-all-ann-decomposition-remove-undecided-length:
  assumes  $\forall l \in \text{set } M'. \neg \text{is-decided } l$ 
  shows  $\langle \text{length (get-all-ann-decomposition (M' @ M''))} = \text{length (get-all-ann-decomposition M'')} \rangle$ 
  using assms by (induct M' arbitrary: M'' rule: ann-lit-list-induct) auto

lemma get-all-ann-decomposition-not-is-decided-length:
  assumes  $\forall l \in \text{set } M'. \neg \text{is-decided } l$ 
  shows  $\langle 1 + \text{length (get-all-ann-decomposition (Propagated (-L) P \# M))} = \text{length (get-all-ann-decomposition (M' @ Decided L \# M))} \rangle$ 
  using assms get-all-ann-decomposition-remove-undecided-length by fastforce

lemma get-all-ann-decomposition-last-choice:
  assumes  $\langle \text{tl (get-all-ann-decomposition (M' @ Decided L \# M))} \neq [] \rangle$ 
  and  $\forall l \in \text{set } M'. \neg \text{is-decided } l$ 
  and  $\langle \text{hd (tl (get-all-ann-decomposition (M' @ Decided L \# M)))} = (M0', M0) \rangle$ 
  shows  $\langle \text{hd (get-all-ann-decomposition (Propagated (-L) P \# M))} = (M0', \text{Propagated } (-L) P \# M0) \rangle$ 
  using assms by (induct M' rule: ann-lit-list-induct) auto

lemma get-all-ann-decomposition-except-last-choice-equal:
  assumes  $\forall l \in \text{set } M'. \neg \text{is-decided } l$ 
  shows  $\langle \text{tl (get-all-ann-decomposition (Propagated (-L) P \# M))} = \text{tl (tl (get-all-ann-decomposition (M' @ Decided L \# M)))} \rangle$ 
  using assms by (induct M' rule: ann-lit-list-induct) auto

lemma get-all-ann-decomposition-hd-hd:
  assumes  $\langle \text{get-all-ann-decomposition } Ls = (M, C) \# (M0, M0') \# l \rangle$ 
  shows  $\langle \text{tl } M = M0' @ M0 \wedge \text{is-decided (hd } M) \rangle$ 
  using assms
proof (induct Ls arbitrary: M C M0 M0' l)
  case Nil
  then show ?case by simp
next
  case (Cons a Ls M C M0 M0' l)
  note IH = this(1) and g = this(2)
  { fix L ann level
    assume a:  $\langle a = \text{Decided } L \rangle$ 
    have  $\langle Ls = M0' @ M0 \rangle$ 
    using g a by (force intro: get-all-ann-decomposition-decomp)
    then have  $\langle \text{tl } M = M0' @ M0 \wedge \text{is-decided (hd } M) \rangle$  using g a by auto
  }
moreover {
  fix L P
  assume a:  $\langle a = \text{Propagated } L P \rangle$ 
  have  $\langle \text{tl } M = M0' @ M0 \wedge \text{is-decided (hd } M) \rangle$ 
  using IH Cons.premis unfolding a by (cases (get-all-ann-decomposition Ls)) auto
}

```

```

}
ultimately show ?case by (cases a) auto
qed

```

```

lemma get-all-ann-decomposition-exists-prepend[dest]:
  assumes  $\langle (a, b) \in \text{set } (\text{get-all-ann-decomposition } M) \rangle$ 
  shows  $\langle \exists c. M = c @ b @ a \rangle$ 
  using assms apply (induct M rule: ann-lit-list-induct)
  apply simp
  by (rename-tac L' xs, case-tac  $\langle \text{get-all-ann-decomposition } xs \rangle$ ;
    auto dest!: arg-cong[of  $\langle \text{get-all-ann-decomposition } - \rangle - \text{hd}$ ]
    get-all-ann-decomposition-decomp)+

```

```

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  $\langle \text{set } a \subseteq \text{set } M \rangle$ 
  using assms get-all-ann-decomposition-exists-prepend by fastforce+

```

```

lemma get-all-ann-decomposition-exists-prepend':
  assumes  $\langle (a, b) \in \text{set } (\text{get-all-ann-decomposition } M) \rangle$ 
  obtains c where  $\langle M = c @ b @ a \rangle$ 
  using assms apply (induct M rule: ann-lit-list-induct)
  apply auto[1]
  by (rename-tac L' xs, case-tac  $\langle \text{hd } (\text{get-all-ann-decomposition } xs) \rangle$ ,
    auto dest!: get-all-ann-decomposition-decomp simp add: list.set-sel(2))+

```

```

lemma union-in-get-all-ann-decomposition-is-subset:
  assumes  $\langle (a, b) \in \text{set } (\text{get-all-ann-decomposition } M) \rangle$ 
  shows  $\langle \text{set } a \cup \text{set } b \subseteq \text{set } M \rangle$ 
  using assms by force

```

```

lemma Decided-cons-in-get-all-ann-decomposition-append-Decided-cons:
   $\langle \exists c''. (\text{Decided } K \# c, c'') \in \text{set } (\text{get-all-ann-decomposition } (c' @ \text{Decided } K \# c)) \rangle$ 
  apply (induction c' rule: ann-lit-list-induct)
  apply auto[2]
  apply (rename-tac L xs,
    case-tac  $\langle \text{hd } (\text{get-all-ann-decomposition } (xs @ \text{Decided } K \# c)) \rangle$ )
  apply (case-tac  $\langle \text{get-all-ann-decomposition } (xs @ \text{Decided } K \# c) \rangle$ )
  by auto

```

```

lemma fst-get-all-ann-decomposition-prepend-not-decided:
  assumes  $\langle \forall m \in \text{set } MS. \neg \text{is-decided } m \rangle$ 
  shows  $\langle \text{set } (\text{map fst } (\text{get-all-ann-decomposition } M))$ 
     $= \text{set } (\text{map fst } (\text{get-all-ann-decomposition } (MS @ M))) \rangle$ 
  using assms apply (induction MS rule: ann-lit-list-induct)
  apply auto[2]
  by (rename-tac L m xs; case-tac  $\langle \text{get-all-ann-decomposition } (xs @ M) \rangle$ ) simp-all

```

```

lemma no-decision-get-all-ann-decomposition:
   $\langle \forall l \in \text{set } M. \neg \text{is-decided } l \implies \text{get-all-ann-decomposition } M = [([], M)] \rangle$ 
  by (induction M rule: ann-lit-list-induct) auto

```

Entailment of the Propagated by the Decided Literal

```

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

```

```

  (is (?M M = ?U M ∪ ?Ls M))
proof (induct M rule: ann-lit-list-induct)
  case Nil
  then show ?case by simp
next
  case (Decided L M) note IH = this(1)
  then have ⟨Decided L ∈ ?Ls (Decided L # M)⟩ by auto
  moreover have ⟨?U (Decided L # M) = ?U M⟩ by auto
  moreover have ⟨?M M = ?U M ∪ ?Ls M⟩ using IH by auto
  ultimately show ?case by auto
next
  case (Propagated L m M)
  then show ?case by (cases ⟨(get-all-ann-decomposition M)⟩) auto
qed

```

definition *all-decomposition-implies* :: ⟨'a clause set
 $\Rightarrow ((\text{'a}, \text{'m}) \text{ ann-lits} \times (\text{'a}, \text{'m}) \text{ ann-lits}) \text{ list} \Rightarrow \text{bool})$ where
 $\langle \text{all-decomposition-implies } N \ S \longleftrightarrow (\forall (Ls, \text{seen}) \in \text{set } S. \text{ unmark-l } Ls \cup N \models_{ps} \text{ unmark-l seen}) \rangle$

lemma *all-decomposition-implies-empty*[iff]:
 $\langle \text{all-decomposition-implies } N \ [] \rangle$ **unfolding** *all-decomposition-implies-def* by auto

lemma *all-decomposition-implies-single*[iff]:
 $\langle \text{all-decomposition-implies } N \ [(Ls, \text{seen})] \longleftrightarrow \text{ unmark-l } Ls \cup N \models_{ps} \text{ unmark-l seen} \rangle$
unfolding *all-decomposition-implies-def* by auto

lemma *all-decomposition-implies-append*[iff]:
 $\langle \text{all-decomposition-implies } N \ (S \ @ \ S') \longleftrightarrow (\text{all-decomposition-implies } N \ S \wedge \text{all-decomposition-implies } N \ S') \rangle$
unfolding *all-decomposition-implies-def* by auto

lemma *all-decomposition-implies-cons-pair*[iff]:
 $\langle \text{all-decomposition-implies } N \ ((Ls, \text{seen}) \ # \ S') \longleftrightarrow (\text{all-decomposition-implies } N \ [(Ls, \text{seen})] \wedge \text{all-decomposition-implies } N \ S') \rangle$
unfolding *all-decomposition-implies-def* by auto

lemma *all-decomposition-implies-cons-single*[iff]:
 $\langle \text{all-decomposition-implies } N \ (l \ # \ S') \longleftrightarrow (\text{unmark-l } (\text{fst } l) \cup N \models_{ps} \text{ unmark-l } (\text{snd } l) \wedge \text{all-decomposition-implies } N \ S') \rangle$
unfolding *all-decomposition-implies-def* by auto

lemma *all-decomposition-implies-trail-is-implied*:
assumes $\langle \text{all-decomposition-implies } N \ (\text{get-all-ann-decomposition } M) \rangle$
shows $\langle N \cup \{\text{unmark } L \mid L. \text{ is-decided } L \wedge L \in \text{set } M\} \models_{ps} \text{ unmark } ' \bigcup (\text{set } ' \text{ snd } ' \text{ set } (\text{get-all-ann-decomposition } M)) \rangle$
using *assms*
proof (induct ⟨length (get-all-ann-decomposition M)⟩ arbitrary: M)
 case 0
 then show ?case by auto
next
 case (Suc n) note IH = this(1) and length = this(2) and decomp = this(3)
consider
 (le1) ⟨length (get-all-ann-decomposition M) ≤ 1⟩

```

| (gt1) ⟨length (get-all-ann-decomposition M) > 1⟩
by arith
then show ?case
proof cases
case le1
then obtain a b where g: ⟨get-all-ann-decomposition M = (a, b) # []⟩
  by (cases ⟨get-all-ann-decomposition M⟩) auto
moreover {
  assume ⟨a = []⟩
  then have ?thesis using Suc.prem1 g by auto
}
moreover {
  assume l: ⟨length a = 1⟩ and m: ⟨is-decided (hd a)⟩ and hd: ⟨hd a ∈ set M⟩
  then have ⟨unmark (hd a) ∈ {unmark L | L. is-decided L ∧ L ∈ set M}⟩ by auto
  then have H: ⟨unmark-l a ∪ N ⊆ N ∪ {unmark L | L. is-decided L ∧ L ∈ set M}⟩
    using l by (cases a) auto
  have f1: ⟨unmark-l a ∪ N ⊨ps unmark-l b⟩
    using decomp unfolding all-decomposition-implies-def g by simp
  have ?thesis
    apply (rule true-clss-clss-subset) using f1 H g by auto
}
ultimately show ?thesis
  using get-all-ann-decomposition-length-1-fst-empty-or-length-1 by blast
next
case gt1
then obtain Ls0 seen0 M' where
  Ls0: ⟨get-all-ann-decomposition M = (Ls0, seen0) # get-all-ann-decomposition M'⟩ and
  length': ⟨length (get-all-ann-decomposition M') = n⟩ and
  M'-in-M: ⟨set M' ⊆ set M⟩
  using length by (induct M rule: ann-lit-list-induct) (auto simp: subset-insertI2)
let ?d = ⟨⋃ (set 'snd ' set (get-all-ann-decomposition M'))⟩
let ?unM = ⟨{unmark L | L. is-decided L ∧ L ∈ set M}⟩
let ?unM' = ⟨{unmark L | L. is-decided L ∧ L ∈ set M'}⟩
{
  assume ⟨n = 0⟩
  then have ⟨get-all-ann-decomposition M' = []⟩ using length' by auto
  then have ?thesis using Suc.prem1 unfolding all-decomposition-implies-def Ls0 by auto
}
moreover {
  assume n: ⟨n > 0⟩
  then obtain Ls1 seen1 l where
    Ls1: ⟨get-all-ann-decomposition M' = (Ls1, seen1) # l⟩
    using length' by (induct M' rule: ann-lit-list-induct) auto

  have ⟨all-decomposition-implies N (get-all-ann-decomposition M')⟩
    using decomp unfolding Ls0 by auto
  then have N: ⟨N ∪ ?unM' ⊨ps unmark-s ?d⟩
    using IH length' by auto
  have l: ⟨N ∪ ?unM' ⊆ N ∪ ?unM⟩
    using M'-in-M by auto
  from true-clss-clss-subset[OF this N]
  have ΨN: ⟨N ∪ ?unM ⊨ps unmark-s ?d⟩ by auto
  have ⟨is-decided (hd Ls0)⟩ and LS: ⟨tl Ls0 = seen1 @ Ls1⟩
    using get-all-ann-decomposition-hd-hd[of M] unfolding Ls0 Ls1 by auto

  have LSM: ⟨seen1 @ Ls1 = M'⟩ using get-all-ann-decomposition-decomp[of M] Ls1 by auto

```

```

have M': ⟨set M' = ?d ∪ {L | L. is-decided L ∧ L ∈ set M'}⟩
  using get-all-ann-decomposition-snd-union by auto

{
  assume ⟨Ls0 ≠ []⟩
  then have ⟨hd Ls0 ∈ set M⟩
    using get-all-ann-decomposition-fst-empty-or-hd-in-M Ls0 by blast
  then have ⟨N ∪ ?unM ⊨ps unmark (hd Ls0)⟩
    using ⟨is-decided (hd Ls0)⟩ by (metis (mono-tags, lifting) UnCI mem-Collect-eq
      true-clss-clss-in)
} note hd-Ls0 = this

have l: ⟨unmark ' (?d ∪ {L | L. is-decided L ∧ L ∈ set M'}) = unmark-s ?d ∪ ?unM'⟩
  by auto
have ⟨N ∪ ?unM' ⊨ps unmark ' (?d ∪ {L | L. is-decided L ∧ L ∈ set M'})⟩
  unfolding l using N by (auto simp: all-in-true-clss-clss)
then have t: ⟨N ∪ ?unM' ⊨ps unmark-l (tl Ls0)⟩
  using M' unfolding LS LSM by auto
then have ⟨N ∪ ?unM ⊨ps unmark-l (tl Ls0)⟩
  using M'-in-M true-clss-clss-subset[OF - t, of ⟨N ∪ ?unM⟩] by auto
then have ⟨N ∪ ?unM ⊨ps unmark-l Ls0⟩
  using hd-Ls0 by (cases Ls0) auto

moreover have ⟨unmark-l Ls0 ∪ N ⊨ps unmark-l seen0⟩
  using decomp unfolding Ls0 by simp
moreover have ⟨ $\bigwedge M$  Ma. (M::'a clause set) ∪ Ma ⊨ps M⟩
  by (simp add: all-in-true-clss-clss)
ultimately have Ψ: ⟨N ∪ ?unM ⊨ps unmark-l seen0⟩
  by (meson true-clss-clss-left-right true-clss-clss-union-and true-clss-clss-union-l-r)

moreover have ⟨unmark ' (set seen0 ∪ ?d) = unmark-l seen0 ∪ unmark-s ?d⟩
  by auto
ultimately have ?thesis using ΨN unfolding Ls0 by simp
}
qed

```

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

```

assumes ⟨all-decomposition-implies N (get-all-ann-decomposition M)⟩
shows ⟨N ∪ {unmark L | L. is-decided L ∧ L ∈ set M} ⊨ps unmark-l M⟩
  (is ⟨?I ⊨ps ?A⟩)

```

proof –

```

have ⟨?I ⊨ps unmark-s {L | L. is-decided L ∧ L ∈ set M}⟩
  by (auto intro: all-in-true-clss-clss)
moreover have ⟨?I ⊨ps unmark '  $\bigcup$  (set ' snd ' set (get-all-ann-decomposition M))⟩
  using all-decomposition-implies-trail-is-implied assms by blast
ultimately have ⟨N ∪ {unmark m | m. is-decided m ∧ m ∈ set M}
  ⊨ps unmark '  $\bigcup$  (set ' snd ' set (get-all-ann-decomposition M))
  ∪ unmark ' {m | m. is-decided m ∧ m ∈ set M}⟩
  by blast
then show ?thesis
  by (metis (no-types) get-all-ann-decomposition-snd-union[of M] image-Un)

```

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

$\langle \text{all-decomposition-implies } N \ M \implies \text{all-decomposition-implies } (\text{insert } C \ N) \ M \rangle$
unfolding *all-decomposition-implies-def* **by** *auto*

lemma *all-decomposition-implies-union*:

$\langle \text{all-decomposition-implies } N \ M \implies \text{all-decomposition-implies } (N \cup N') \ M \rangle$
unfolding *all-decomposition-implies-def sup.assoc[symmetric]* **by** *(auto intro: true-clss-clss-union-l)*

lemma *all-decomposition-implies-mono*:

$\langle N \subseteq N' \implies \text{all-decomposition-implies } N \ M \implies \text{all-decomposition-implies } N' \ M \rangle$
by *(metis all-decomposition-implies-union le-iff-sup)*

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

$\langle \text{all-decomposition-implies } I \ (\text{get-all-ann-decomposition } (M' @ M)) \implies$
 $\text{all-decomposition-implies } I \ (\text{get-all-ann-decomposition } M) \rangle$
apply *(induction M' arbitrary: M rule: ann-lit-list-induct)*
subgoal by *auto*
subgoal by *auto*
subgoal for $L \ C \ M' \ M$
by *(cases (get-all-ann-decomposition (M' @ M)) auto)*
done

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]*: $\langle \text{finite } (CNot \ C) \rangle$

by *(auto simp: CNot-def)*

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

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

unfolding *CNot-def* **by** *force*

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]: $\langle CNot \ (A + B) = CNot \ A \cup CNot \ B \rangle$

unfolding *CNot-def* **by** *auto*

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

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

unfolding *CNot-def* **by** *(auto simp add: multiset-eqI)*

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** $\langle -L \in \text{lits-of-l } M \rangle$

using *assms* **by** *(auto simp: true-annots-def true-annot-def CNot-def)*

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

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

unfolding *CNot-def* **by** *auto*

lemma *Ball-CNot-Ball-mset[simp]*:
 $\langle (\forall x \in CNot\ D. P\ x) \longleftrightarrow (\forall L \in \#\ D. P\ \{\# - L\# \}) \rangle$
unfolding *CNot-def* **by** *auto*

lemma *consistent-CNot-not*:
assumes $\langle consistent_interp\ I \rangle$
shows $\langle I \models_s CNot\ \varphi \implies \neg I \models \varphi \rangle$
using *assms* **unfolding** *consistent-interp-def true-clss-def true-cl-def* **by** *auto*

lemma *total-not-true-cl-true-clss-CNot*:
assumes $\langle total_over_m\ I\ \{\varphi\} \rangle$ **and** $\langle \neg I \models \varphi \rangle$
shows $\langle I \models_s CNot\ \varphi \rangle$
using *assms* **unfolding** *total-over-m-def total-over-set-def true-clss-def true-cl-def CNot-def*
apply *clarify*
by $(rename_tac\ x\ L, case_tac\ L)\ (force\ intro: pos_lit_in_atms_of\ neg_lit_in_atms_of) +$

lemma *total-not-CNot*:
assumes $\langle total_over_m\ I\ \{\varphi\} \rangle$ **and** $\langle \neg I \models_s CNot\ \varphi \rangle$
shows $\langle I \models \varphi \rangle$
using *assms* *total-not-true-cl-true-clss-CNot* **by** *auto*

lemma *atms-of-ms-CNot-atms-of[simp]*:
 $\langle atms_of_ms\ (CNot\ C) = atms_of\ C \rangle$
unfolding *atms-of-ms-def atms-of-def CNot-def* **by** *fastforce*

lemma *true-clss-clss-contradiction-true-clss-cl-false*:
 $\langle C \in D \implies D \models_{ps} CNot\ C \implies D \models_p \{\#\} \rangle$
unfolding *true-clss-clss-def true-clss-cl-def total-over-m-def*
by $(metis\ Un_commute\ atms_of_empty\ atms_of_ms_CNot_atms_of\ atms_of_ms_insert\ atms_of_ms_union\ consistent_CNot_not\ insert_absorb\ sup_bot.left_neutral\ true_clss_def)$

lemma *true-annots-CNot-all-atms-defined*:
assumes $\langle M \models_{as} CNot\ T \rangle$ **and** $a1: \langle L \in \#\ T \rangle$
shows $\langle atm_of\ L \in atm_of\ ' lits_of_l\ M \rangle$
by $(metis\ assms\ atm_of_uminus\ image_eqI\ in_CNot_implies_uminus(1)\ true_annot_singleton)$

lemma *true-annots-CNot-all-uminus-atms-defined*:
assumes $\langle M \models_{as} CNot\ T \rangle$ **and** $a1: \langle \neg L \in \#\ T \rangle$
shows $\langle atm_of\ L \in atm_of\ ' lits_of_l\ M \rangle$
by $(metis\ assms\ atm_of_uminus\ image_eqI\ in_CNot_implies_uminus(1)\ true_annot_singleton)$

lemma *true-clss-clss-false-left-right*:
assumes $\langle \{\{\#L\#\} \cup B \models_p \{\#\} \rangle$
shows $\langle B \models_{ps} CNot\ \{\#L\#\} \rangle$
unfolding *true-clss-clss-def true-clss-cl-def*
proof $(intro\ allI\ impI)$
fix *I*
assume
 $tot: \langle total_over_m\ I\ (B \cup CNot\ \{\#L\#\}) \rangle$ **and**
 $cons: \langle consistent_interp\ I \rangle$ **and**
 $I: \langle I \models_s B \rangle$
have $\langle total_over_m\ I\ (\{\{\#L\#\} \cup B) \rangle$ **using** *tot* **by** *auto*
then have $\langle \neg I \models_s insert\ \{\#L\#\}\ B \rangle$
using *assms cons* **unfolding** *true-clss-cl-def* **by** *simp*
then show $\langle I \models_s CNot\ \{\#L\#\} \rangle$
using *tot I* **by** $(cases\ L)\ auto$

qed

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

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

unfolding *CNot-def true-annots-true-clss true-clss-def* **by** *auto*

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

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

by (*auto simp: CNot-def true-clss-def*)

lemma *true-annots-CNot-definedD*:

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

unfolding *true-annots-true-clss-def-iff-negation-in-model*

by (*auto simp: Decided-Propagated-in-iff-in-lits-of-l*)

lemma *true-annot-CNot-diff*:

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

by (*auto simp: true-annots-true-clss-def-iff-negation-in-model dest: in-diffD*)

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

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

by (*induction n*) *auto*

lemma *consistent-CNot-not-tautology*:

$\langle \text{consistent-interp}\ M \implies M \models_s CNot\ D \implies \neg \text{tautology}\ D \rangle$

by (*metis atms-of-ms-CNot-atms-of consistent-CNot-not satisfiable-carac' satisfiable-def tautology-def total-over-m-def*)

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

by *simp*

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

$\langle \text{total-over-m}\ I\ (CNot\ C) = \text{total-over-set}\ I\ (\text{atms-of}\ C) \rangle$

unfolding *total-over-m-def total-over-set-def* **by** *auto*

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

assumes

CC-L: $\langle A \models_p \text{add-mset}\ L\ CC \rangle$ **and**

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

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

unfolding *true-clss-clss-def true-clss-clss-def CNot-def total-over-m-def*

proof (*intro allI impI*)

fix *I*

assume

tot: $\langle \text{total-over-set}\ I\ (\text{atms-of-ms}\ (A \cup \{\#L\})) \rangle$ **and**

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

I: $\langle I \models_s A \rangle$

let *?I* = $\langle I \cup \{Pos\ P \mid P \in \text{atms-of}\ CC \wedge P \notin \text{atm-of}\ 'I\} \rangle$

have *cons'*: $\langle \text{consistent-interp}\ ?I \rangle$

using *cons* **unfolding** *consistent-interp-def*

by (*auto simp: uminus-lit-swap atms-of-def rev-image-eqI*)

have *I'*: $\langle ?I \models_s A \rangle$

using *I* *true-clss-union-increase* **by** *blast*

have *tot-CNot*: $\langle \text{total-over-m}\ ?I\ (A \cup CNot\ CC) \rangle$

using *tot* *atms-of-s-def* **by** (*fastforce simp: total-over-m-def total-over-set-def*)

then have $\text{tot-}I\text{-}A\text{-}CC\text{-}L$: $\langle \text{total-over-}m \text{ ?}I (A \cup \{\text{add-mset } L \text{ } CC\}) \rangle$
 using tot **unfolding** $\text{total-over-}m\text{-def}$ $\text{total-over-set-atm-of}$ **by** auto
 then have $\langle ?I \models \text{add-mset } L \text{ } CC \rangle$ using $CC\text{-}L$ $\text{cons}' I'$ **unfolding** $\text{true-clss-clss-def}$ **by** blast
 moreover
 have $\langle ?I \models s \text{ } CNot \text{ } CC \rangle$ using $CNot\text{-}CC$ $\text{cons}' I'$ $\text{tot-}CNot$ **unfolding** $\text{true-clss-clss-def}$ **by** auto
 then have $\langle \neg A \models p \text{ } CC \rangle$
 by $(\text{metis } (\text{no-types, lifting}) I' \text{ atms-of-}ms\text{-}CNot\text{-atms-of-}ms \text{ atms-of-}ms\text{-union } \text{cons}'$
 $\text{consistent-}CNot\text{-not } \text{tot-}CNot \text{ total-over-}m\text{-def } \text{true-clss-clss-def})$
 then have $\langle \neg ?I \models CC \rangle$ using $\langle ?I \models s \text{ } CNot \text{ } CC \rangle$ cons' $\text{consistent-}CNot\text{-not}$ **by** blast
 ultimately have $\langle ?I \models \{\#L\# \} \rangle$ **by** blast
 then show $\langle I \models \{\#L\# \} \rangle$
 by $(\text{metis } (\text{no-types, lifting}) \text{ atms-of-}ms\text{-union } \text{cons}' \text{ consistent-}CNot\text{-not } \text{tot } \text{total-not-}CNot$
 $\text{total-over-}m\text{-def } \text{total-over-set-union } \text{true-clss-union-increase})$
 qed

lemma $\text{true-annot-}CNot\text{-lit-of-notin-skip}$:

assumes LM : $\langle L \# M \models_{as} CNot \text{ } A \rangle$ and LA : $\langle \text{lit-of } L \notin \# \text{ } A \rangle \langle \neg \text{lit-of } L \notin \# \text{ } A \rangle$
 shows $\langle M \models_{as} CNot \text{ } A \rangle$
 using LM **unfolding** $\text{true-annot-}CNot\text{-def}$ Ball-def
proof (intro allI impI)
 fix l
 assume H : $\langle \forall x. x \in CNot \text{ } A \longrightarrow L \# M \models_a x \rangle$ and l : $\langle l \in CNot \text{ } A \rangle$
 then have $\langle L \# M \models_a l \rangle$ **by** auto
 then show $\langle M \models_a l \rangle$ using LA l **by** $(\text{cases } L) (\text{auto simp: } CNot\text{-def})$
 qed

lemma $\text{true-clss-clss-union-false-true-clss-clss-cnot}$:

$\langle A \cup \{B\} \models_{ps} \{\{\#\}\} \rangle \longleftrightarrow A \models_{ps} CNot \text{ } B$
 using $\text{total-not-}CNot$ $\text{consistent-}CNot\text{-not}$ **unfolding** $\text{total-over-}m\text{-def}$ $\text{true-clss-clss-def}$
by fastforce

lemma $\text{true-annot-remove-hd-if-notin-vars}$:

assumes $\langle a \# M' \models_a D \rangle$ and $\langle \text{atm-of } (\text{lit-of } a) \notin \text{atms-of } D \rangle$
 shows $\langle M' \models_a D \rangle$
 using assms $\text{true-clss-remove-hd-if-notin-vars}$ **unfolding** true-annot-def **by** auto

lemma $\text{true-annot-remove-if-notin-vars}$:

assumes $\langle M @ M' \models_a D \rangle$ and $\langle \forall x \in \text{atms-of } D. x \notin \text{atm-of } \text{' lits-of-}l \text{ } M \rangle$
 shows $\langle M' \models_a D \rangle$
 using assms **by** $(\text{induct } M) (\text{auto dest: } \text{true-annot-remove-hd-if-notin-vars})$

lemma $\text{true-annot-}CNot\text{-remove-if-notin-vars}$:

assumes $\langle M @ M' \models_{as} D \rangle$ and $\langle \forall x \in \text{atms-of-}ms \text{ } D. x \notin \text{atm-of } \text{' lits-of-}l \text{ } M \rangle$
 shows $\langle M' \models_{as} D \rangle$ **unfolding** $\text{true-annot-}CNot\text{-def}$
 using assms **unfolding** $\text{true-annot-}CNot\text{-def}$ $\text{atms-of-}ms\text{-def}$
by $(\text{force dest: } \text{true-annot-}CNot\text{-remove-if-notin-vars})$

lemma $\text{all-variables-defined-not-imply-cnot}$:

assumes
 $\langle \forall s \in \text{atms-of-}ms \text{ } \{B\}. s \in \text{atm-of } \text{' lits-of-}l \text{ } A \rangle$ and
 $\langle \neg A \models_a B \rangle$
 shows $\langle A \models_{as} CNot \text{ } B \rangle$
unfolding true-annot-def $\text{true-annot-}CNot\text{-def}$ Ball-def $CNot\text{-def}$ true-lit-def
proof $(\text{clarify, rule ccontr})$
 fix L

assume $LB: \langle L \in \# B \rangle$ **and** $L\text{-false}: \langle \neg \text{ lits-of-l } A \models \{\#\} \rangle$ **and** $L\text{-A}: \langle \neg L \notin \text{ lits-of-l } A \rangle$
then have $\langle \text{atm-of } L \in \text{atm-of } \text{' lits-of-l } A \rangle$
using $\text{assms}(1)$ **by** $(\text{simp add: atm-of-lit-in-atms-of lits-of-def})$
then have $\langle L \in \text{ lits-of-l } A \vee \neg L \in \text{ lits-of-l } A \rangle$
using $\text{atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set}$ **by** metis
then have $\langle L \in \text{ lits-of-l } A \rangle$ **using** $L\text{-A}$ **by** auto
then show False
using LB $\text{assms}(2)$ **unfolding** $\text{true-annot-def true-lit-def true-clss-def Bex-def}$
by blast
qed

lemma $C\text{Not-union-mset}[\text{simp}]$:
 $\langle C\text{Not } (A \cup \# B) = C\text{Not } A \cup C\text{Not } B \rangle$
unfolding $C\text{Not-def}$ **by** auto

lemma $\text{true-clss-clss-true-clss-clss-true-clss-clss}$:
assumes
 $\langle A \models_{ps} \text{unmark-l } M \rangle$ **and** $\langle M \models_{as} D \rangle$
shows $\langle A \models_{ps} D \rangle$
by $(\text{meson assms total-over-m-union true-annots-true-clss true-annots-true-clss-clss}$
 $\text{true-clss-clss-def true-clss-clss-left-right true-clss-clss-union-and}$
 $\text{true-clss-clss-union-l-r})$

lemma $\text{true-clss-clss-CNot-true-clss-clss-unsatisfiable}$:
assumes $\langle A \models_{ps} C\text{Not } D \rangle$ **and** $\langle A \models_p D \rangle$
shows $\langle \text{unsatisfiable } A \rangle$
using $\text{assms}(2)$ **unfolding** $\text{true-clss-clss-def true-clss-clss-def satisfiable-def}$
by $(\text{metis (full-types) Un-absorb Un-empty-right assms}(1) \text{atms-of-empty}$
 $\text{atms-of-ms-empty-set total-over-m-def total-over-m-insert total-over-m-union}$
 $\text{true-clss-clss-def true-clss-clss-generalise-true-clss-clss}$
 $\text{true-clss-clss-true-clss-clss true-clss-clss-union-false-true-clss-clss-cnot})$

lemma $\text{true-clss-clss-neg}$:
 $\langle N \models_p I \iff N \cup (\lambda L. \{\# - L\# \}) \text{' set-mset } I \models_p \{\#\} \rangle$
proof –
have $[\text{simp}]: \langle (\lambda L. \{\# - L\# \}) \text{' set-mset } I = C\text{Not } I \rangle$ **for** I
by $(\text{auto simp: CNot-def})$
have $[\text{iff}]: \langle \text{total-over-m } Ia ((\lambda L. \{\# - L\# \}) \text{' set-mset } I) \iff$
 $\text{total-over-set } Ia (\text{atms-of } I) \rangle$ **for** Ia
by $(\text{auto simp: total-over-m-def}$
 $\text{total-over-set-def atms-of-ms-def atms-of-def})$
show $?thesis$
by $(\text{auto simp: true-clss-clss-def consistent-CNot-not}$
 $\text{total-not-CNot})$
qed

lemma $\text{all-decomposition-imply-conflict-DECO-clause}$:
assumes $\langle \text{all-decomposition-imply } N (\text{get-all-ann-decomposition } M) \rangle$ **and**
 $\langle M \models_{as} C\text{Not } C \rangle$ **and**
 $\langle C \in N \rangle$
shows $\langle N \models_p (\text{uminus o lit-of}) \text{' \# (filter-mset is-decided (mset } M)) \rangle$
 $(\text{is } \langle ?I \models_p ?A \rangle)$
proof –
have $\langle \{\text{unmark } m \mid m. \text{is-decided } m \wedge m \in \text{set } M\} =$
 $\text{unmark-s } \{L \in \text{set } M. \text{is-decided } L\} \rangle$
by auto

```

have  $\langle N \cup \text{unmark-s } \{L \in \text{set } M. \text{is-decided } L\} \models_p \{\#\} \rangle$ 
  by (metis (mono-tags, lifting) UnCI
     $\langle \{\text{unmark } m \mid m. \text{is-decided } m \wedge m \in \text{set } M\} = \text{unmark-s } \{L \in \text{set } M. \text{is-decided } L\} \rangle$ 
    all-decomposition-implies-propagated-lits-are-implied assms
    true-clss-clss-contradiction-true-clss-clb-false true-clss-clss-true-clss-clb-true-clss-clss)
then show ?thesis
  apply (subst true-clss-clb-neg)
  by (auto simp: image-image)
qed

```

1.2.5 Other

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

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

```

 $\langle \text{no-dup } [] \rangle$ 
by (auto simp: no-dup-def)

```

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

```

 $\langle \text{no-dup } (L \# M) \longleftrightarrow \text{undefined-lit } M (\text{lits-of } L) \wedge \text{no-dup } M \rangle$ 
by (auto simp: no-dup-def defined-lit-map)

```

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

```

 $\langle \text{no-dup } (M @ L \# M') \longleftrightarrow \text{undefined-lit } (M @ M') (\text{lits-of } L) \wedge \text{no-dup } (M @ M') \rangle$ 
by (auto simp: no-dup-def defined-lit-map)

```

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

```

 $\langle \text{no-dup } (M0 @ M @ L \# M') \longleftrightarrow \text{undefined-lit } (M0 @ M @ M') (\text{lits-of } L) \wedge \text{no-dup } (M0 @ M @ M') \rangle$ 
by (auto simp: no-dup-def defined-lit-map)

```

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

```

 $\langle \text{no-dup } (\text{rev } M) \longleftrightarrow \text{no-dup } M \rangle$ 
by (auto simp: rev-map[symmetric] no-dup-def)

```

lemma *no-dup-appendD*:

```

 $\langle \text{no-dup } (a @ b) \implies \text{no-dup } b \rangle$ 
by (auto simp: no-dup-def)

```

lemma *no-dup-appendD1*:

```

 $\langle \text{no-dup } (a @ b) \implies \text{no-dup } a \rangle$ 
by (auto simp: no-dup-def)

```

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

```

assumes  $\langle \text{no-dup } M \rangle$ 
shows  $\langle \text{length } M = \text{card } (\text{atm-of } (\text{lits-of-l } M)) \rangle$ 
using assms unfolding lits-of-def by (induct M) (auto simp add: image-image no-dup-def)

```

lemma *distinct-consistent-interp*:

```

 $\langle \text{no-dup } M \implies \text{consistent-interp } (\text{lits-of-l } M) \rangle$ 

```

proof (*induct M*)

```

case Nil
show ?case by auto

```

next

```

case (Cons L M)
then have a1:  $\langle \text{consistent-interp } (\text{lits-of-l } M) \rangle$  by auto

```

have $\langle \text{undefined-lit } M \text{ (lit-of } L) \rangle$
using *Cons.prem*s **by** *auto*
then show *?case*
using *a1* **by** *simp*
qed

lemma *same-mset-no-dup-iff*:
 $\langle \text{mset } M = \text{mset } M' \implies \text{no-dup } M \longleftrightarrow \text{no-dup } M' \rangle$
by (*auto simp: no-dup-def same-mset-distinct-iff*)

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 $\langle \text{no-dup } (a @ b) \rangle$
using *assms* **by** (*force simp: no-dup-def*)

lemma *true-annots-lit-of-notin-skip*:
assumes $\langle L \# M \models_{as} \text{CNot } A \rangle$
and $\langle \neg \text{lit-of } L \notin \# A \rangle$
and $\langle \text{no-dup } (L \# M) \rangle$
shows $\langle M \models_{as} \text{CNot } A \rangle$
proof –
have $\langle \forall l \in \# A. \neg l \in \text{lits-of-l } (L \# M) \rangle$
using *assms(1) in-CNot-implies-uminus(2)* **by** *blast*
moreover {
have $\langle \text{undefined-lit } M \text{ (lit-of } L) \rangle$
using *assms(3)* **by** *force*
then have $\langle \neg \text{lit-of } L \notin \text{lits-of-l } M \rangle$
by (*simp add: Decided-Propagated-in-iff-in-lits-of-l*) }
ultimately have $\langle \forall l \in \# A. \neg l \in \text{lits-of-l } M \rangle$
using *assms(2)* **by** (*metis insert-iff list.simps(15) lits-of-insert uminus-of-uminus-id*)
then show *?thesis* **by** (*auto simp add: true-annots-def*)
qed

lemma *no-dup-imp-distinct*: $\langle \text{no-dup } M \implies \text{distinct } M \rangle$
by (*induction M*) (*auto simp: defined-lit-map*)

lemma *no-dup-tlD*: $\langle \text{no-dup } a \implies \text{no-dup } (\text{tl } a) \rangle$
unfolding *no-dup-def* **by** (*cases a*) *auto*

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

lemma *no-dup-consistentD*:
 $\langle \text{no-dup } M \implies L \in \text{lits-of-l } M \implies \neg L \notin \text{lits-of-l } M \rangle$
using *consistent-interp-def distinct-consistent-interp* **by** *blast*

lemma *no-dup-not-tautology*: $\langle \text{no-dup } M \implies \neg \text{tautology } (\text{image-mset lit-of } (\text{mset } M)) \rangle$
by (*induction M*) (*auto simp: tautology-add-mset uminus-lit-swap defined-lit-def*
dest: atm-imp-decided-or-proped)

lemma *no-dup-distinct*: $\langle \text{no-dup } M \implies \text{distinct-mset } (\text{image-mset lit-of } (\text{mset } M)) \rangle$
by (*induction M*) (*auto simp: uminus-lit-swap defined-lit-def*)

dest: atm-imp-decided-or-proped)

lemma *no-dup-not-tautology-uminus*: $\langle \text{no-dup } M \implies \neg \text{tautology } \{\# \text{lit-of } L. L \in \# \text{ mset } M \# \} \rangle$
by (*induction* *M*) (*auto simp: tautology-add-mset uminus-lit-swap defined-lit-def*
dest: atm-imp-decided-or-proped)

lemma *no-dup-distinct-uminus*: $\langle \text{no-dup } M \implies \text{distinct-mset } \{\# \text{lit-of } L. L \in \# \text{ mset } M \# \} \rangle$
by (*induction* *M*) (*auto simp: uminus-lit-swap defined-lit-def*
dest: atm-imp-decided-or-proped)

lemma *no-dup-map-lit-of*: $\langle \text{no-dup } M \implies \text{distinct } (\text{map lit-of } M) \rangle$
apply (*induction* *M*)
apply (*auto simp: dest: no-dup-imp-distinct*)
by (*meson distinct.simps(2) no-dup-cons no-dup-imp-distinct*)

lemma *no-dup-alt-def*:
 $\langle \text{no-dup } M \iff \text{distinct-mset } \{\# \text{atm-of } (\text{lit-of } x). x \in \# \text{ mset } M \# \} \rangle$
by (*auto simp: no-dup-def simp flip: distinct-mset-mset-distinct*)

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 $\langle \text{undefined-lit } M L \rangle$
using *assms* **by** (*auto simp add: atm-lit-of-set-lits-of-l no-dup-def*
defined-lit-map)

lemma *no-dup-uminus-append-in-atm-notin*:
assumes $\langle \text{no-dup } (M @ M') \rangle$ **and** $\langle \neg L \in \text{lits-of-l } M' \rangle$
shows $\langle \text{undefined-lit } M L \rangle$
using *Decided-Propagated-in-iff-in-lits-of-l assms defined-lit-no-dupD(1)* **by** *blast*

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*: $\langle N \models_{psm} B \implies A \subseteq \# B \implies N \models_{psm} A \rangle$
using *set-mset-mono true-clss-clss-subsetE* **by** *blast*

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*: $\langle \text{atms-of-mm } U = \text{set-mset } (\bigcup \# (\text{image-mset } (\text{image-mset } \text{atm-of}) U)) \rangle$
unfolding *atms-of-ms-def* **by** (*auto simp: atms-of-def*)

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-cls-cong-set-mset*:
 $\langle N \models_{pm} D \Longrightarrow \text{set-mset } D = \text{set-mset } D' \Longrightarrow N \models_{pm} D' \rangle$
by (*auto simp add: true-clss-cls-def true-cls-def atms-of-cong-set-mset[of D D']*)

1.2.7 More Lemmas

lemma *no-dup-cannot-not-lit-and-uminus*:
 $\langle \text{no-dup } M \Longrightarrow - \text{lit-of } xa = \text{lit-of } x \Longrightarrow x \in \text{set } M \Longrightarrow xa \notin \text{set } M \rangle$
by (*metis atm-of-uminus distinct-map inj-on-eq-iff uminus-not-id' no-dup-def*)

lemma *atms-of-ms-single-atm-of[simp]*:
 $\langle \text{atms-of-ms } \{\text{unmark } L \mid L. P L\} = \text{atm-of } ' \{\text{lit-of } L \mid L. P L\} \rangle$
unfolding *atms-of-ms-def* **by** *force*

lemma *true-cls-mset-restrict*:
 $\langle \{L \in I. \text{atm-of } L \in \text{atms-of-mm } N\} \models_m N \longleftrightarrow I \models_m N \rangle$
by (*auto simp: true-cls-mset-def true-cls-def*
dest!: multi-member-split)

lemma *true-clss-restrict*:
 $\langle \{L \in I. \text{atm-of } L \in \text{atms-of-mm } N\} \models_{sm} N \longleftrightarrow I \models_{sm} N \rangle$
by (*auto simp: true-clss-def true-cls-def*
dest!: multi-member-split)

lemma *total-over-m-atms-incl*:
assumes $\langle \text{total-over-m } M (\text{set-mset } N) \rangle$
shows
 $\langle x \in \text{atms-of-mm } N \Longrightarrow x \in \text{atms-of-s } M \rangle$
by (*meson assms contra-subsetD total-over-m-alt-def*)

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** $\langle ?A \longleftrightarrow ?B \rangle$)
apply (*subst true-clss-alt-def2[OF assms]*)
apply (*subst true-clss-alt-def2*)
subgoal using *not-tautology-mono[OF - assms]* **by** (*auto dest: not-tautology-minus*)
apply (*rule HOL.iff-allI*)
apply (*auto 5 5 simp: true-cls-def atms-of-s-def dest!: multi-member-split*)
done

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]*: $\langle \text{negate-ann-lits } [] = \{\# \} \rangle$
by (*auto simp: negate-ann-lits-def*)

lemma *entails-CNot-negate-ann-lits*:
 $\langle M \models_{as} CNot D \longleftrightarrow \text{set-mset } D \subseteq \text{set-mset } (\text{negate-ann-lits } M) \rangle$
by (*auto simp: true-annots-true-cls-def-iff-negation-in-model*
negate-ann-lits-def lits-of-def uminus-lit-swap
dest!: multi-member-split)

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*:
 $\langle pNeg (\text{add-mset } A C) = \text{add-mset } (-A) (pNeg C) \rangle$
 $\langle pNeg (C + D) = pNeg C + pNeg D \rangle$
by (*auto simp: pNeg-def*)

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

lemma *negate-ann-lits-pNeg-lit-of*: $\langle \text{negate-ann-lits} = pNeg \circ \text{image-mset lit-of} \circ \text{mset} \rangle$
by (*intro ext*) (*auto simp: negate-ann-lits-def pNeg-def*)

lemma *negate-ann-lits-empty-iff*: $\langle \text{negate-ann-lits } M \neq \{\# \} \longleftrightarrow M \neq [] \rangle$
by (*auto simp: negate-ann-lits-def*)

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

lemma *tautology-pNeg[simp]*:
 $\langle \text{tautology } (pNeg C) \longleftrightarrow \text{tautology } C \rangle$
by (*auto 5 5 simp: tautology-decomp pNeg-def*
uminus-lit-swap add-mset-eq-add-mset eq-commute[of \langle Neg \rightarrow \langle - \rightarrow \rangle \rangle eq-commute[of \langle Pos \rightarrow \langle - \rightarrow \rangle \rangle]
dest!: multi-member-split)

lemma *pNeg-convolution[simp]*:
 $\langle pNeg (pNeg C) = C \rangle$
by (*auto simp: pNeg-def*)

lemma *pNeg-minus[simp]*: $\langle pNeg (A - B) = pNeg A - pNeg B \rangle$
unfolding *pNeg-def*
by (*subst image-mset-minus-inj-on*) (*auto simp: inj-on-def*)

lemma *pNeg-empty[simp]*: $\langle pNeg \{\# \} = \{\# \} \rangle$
unfolding *pNeg-def*
by (*auto simp: inj-on-def*)

lemma *pNeg-replicate-mset[simp]*: $\langle pNeg (\text{replicate-mset } n L) = \text{replicate-mset } n (-L) \rangle$
unfolding *pNeg-def* **by** *auto*

lemma *distinct-mset-pNeg-iff[iff]*: $\langle \text{distinct-mset } (pNeg x) \longleftrightarrow \text{distinct-mset } x \rangle$
unfolding *pNeg-def*
by (*rule distinct-image-mset-inj*) (*auto simp: inj-on-def*)

lemma *pNeg-simple-clss-iff*[simp]:

$\langle pNeg\ M \in simple-clss\ N \longleftrightarrow M \in simple-clss\ N \rangle$

by (auto simp: simple-clss-def)

lemma *atms-of-ms-pNeg*[simp]: $\langle atms-of-ms\ (pNeg\ 'N) = atms-of-ms\ N \rangle$

unfolding *atms-of-ms-def* *pNeg-def* **by** (auto simp: image-image atms-of-def)

definition *DECO-clause* :: $\langle ('v, 'a)\ ann-lits \Rightarrow 'v\ clause \rangle$ **where**

$\langle DECO-clause\ M = (uminus\ o\ lit-of)\ ' \# (filter-mset\ is-decided\ (mset\ M)) \rangle$

lemma

DECO-clause-cons-Decide[simp]:

$\langle DECO-clause\ (Decided\ L\ \# M) = add-mset\ (-L)\ (DECO-clause\ M) \rangle$ **and**

DECO-clause-cons-Proped[simp]:

$\langle DECO-clause\ (Propagated\ L\ C\ \# M) = DECO-clause\ M \rangle$

by (auto simp: DECO-clause-def)

lemma *no-dup-distinct-mset*[intro!]:

assumes *n-d*: $\langle no-dup\ M \rangle$

shows $\langle distinct-mset\ (negate-ann-lits\ M) \rangle$

unfolding *negate-ann-lits-def* *no-dup-def*

proof (*subst distinct-image-mset-inj*)

show $\langle inj-on\ (\lambda L. -\ lit-of\ L)\ (set-mset\ (mset\ M)) \rangle$

unfolding *inj-on-def* *Ball-def*

proof (*intro allI impI, rule ccontr*)

fix *L L'*

assume

L: $\langle L \in \# mset\ M \rangle$ **and**

L': $\langle L' \in \# mset\ M \rangle$ **and**

lit: $\langle -\ lit-of\ L = -\ lit-of\ L' \rangle$ **and**

LL': $\langle L \neq L' \rangle$

have $\langle atm-of\ (lit-of\ L) = atm-of\ (lit-of\ L') \rangle$

using *lit* **by** *auto*

moreover have $\langle atm-of\ (lit-of\ L) \in \# (\lambda l. atm-of\ (lit-of\ l))\ ' \# mset\ M \rangle$

using *L* **by** *auto*

moreover have $\langle atm-of\ (lit-of\ L') \in \# (\lambda l. atm-of\ (lit-of\ l))\ ' \# mset\ M \rangle$

using *L'* **by** *auto*

ultimately show *False*

using *assms LL' L L'* **unfolding** *distinct-mset-mset-distinct*[*symmetric*] *mset-map no-dup-def*

apply $-$ **apply** (*rule distinct-image-mset-not-equal*[*of L L' (λl. atm-of (lit-of l))*])

by *auto*

qed

next

show $\langle distinct-mset\ (mset\ M) \rangle$

using *no-dup-imp-distinct*[*OF n-d*] **by** *simp*

qed

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

unfolding *negate-ann-lits-def* *lits-of-def* **by** (auto simp: *uminus-lit-swap*)

lemma *negate-ann-lits-cons*[simp]:

$\langle negate-ann-lits\ (L\ \# M) = add-mset\ (-\ lit-of\ L)\ (negate-ann-lits\ M) \rangle$

by (auto simp: *negate-ann-lits-def*)

lemma *uminus-simple-clss-iff*[simp]:
 $\langle \text{uminus } \# M \in \text{simple-clss } N \longleftrightarrow M \in \text{simple-clss } N \rangle$
by (*metis pNeg-simple-clss-iff pNeg-def*)

lemma *pNeg-mono*: $\langle C \subseteq \# C' \implies \text{pNeg } C \subseteq \# \text{pNeg } C' \rangle$
by (*auto simp: image-mset-subseteq-mono pNeg-def*)

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**
 $\langle \text{partial-model-of } I = \text{Pos } \langle I \cup \text{Neg } \langle \{x. x \notin I\} \rangle \rangle$

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*:
 $\langle \text{total-over-set } (\text{partial-model-of } I) \text{ UNIV} \rangle$
unfolding *total-over-set-def*
by (*auto simp: partial-model-of-def*)

lemma *consistent-interp-partial-model-of*:
 $\langle \text{consistent-interp } (\text{partial-model-of } I) \rangle$
unfolding *consistent-interp-def*
by (*auto simp: partial-model-of-def*)

lemma *consistent-interp-alt-def*:
 $\langle \text{consistent-interp } I \longleftrightarrow (\forall L. \neg(\text{Pos } L \in I \wedge \text{Neg } L \in I)) \rangle$
unfolding *consistent-interp-def*
by (*metis literal.exhaust uminus-Neg uminus-of-uminus-id*)

context

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

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

begin

lemma *partial-implies-total-true-cls-total-model-of*:
assumes $\langle \text{Partial-Herbrand-Interpretation.true-cls } I \ C \rangle$
shows $\langle \text{Herbrand-Interpretation.true-cls } (\text{total-model-of } I) \ C \rangle$
using *assms cons* **apply** –
unfolding *total-model-of-def Partial-Herbrand-Interpretation.true-cls-def*
Herbrand-Interpretation.true-cls-def consistent-interp-alt-def
by (*rule bexE, assumption*)
(case-tac x; auto)

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

```

  ⟨total-over-set I (atms-of C)⟩
shows ⟨Partial-Herbrand-Interpretation.true-cls I C⟩
using assms cons
unfolding total-model-of-def Partial-Herbrand-Interpretation.true-cls-def
  Herbrand-Interpretation.true-cls-def consistent-interp-alt-def
  total-over-m-def total-over-set-def
by (auto simp: atms-of-def dest: multi-member-split)

lemma partial-implies-total-true-clss-total-model-of:
assumes ⟨Partial-Herbrand-Interpretation.true-clss I C⟩
shows ⟨Herbrand-Interpretation.true-clss (total-model-of I) C⟩
using assms cons
unfolding Partial-Herbrand-Interpretation.true-clss-def
  Herbrand-Interpretation.true-clss-def
by (auto simp: partial-implies-total-true-cls-total-model-of)

lemma total-implies-partial-true-clss-total-model-of:
assumes ⟨Herbrand-Interpretation.true-clss (total-model-of I) C⟩ and
  ⟨total-over-m I C⟩
shows ⟨Partial-Herbrand-Interpretation.true-clss I C⟩
using assms cons mk-disjoint-insert
unfolding Partial-Herbrand-Interpretation.true-clss-def
  Herbrand-Interpretation.true-clss-def
  total-over-set-def
by (fastforce simp: total-implies-partial-true-cls-total-model-of
  dest: multi-member-split)

end

lemma total-implies-partial-true-cls-partial-model-of:
assumes ⟨Herbrand-Interpretation.true-cls I C⟩
shows ⟨Partial-Herbrand-Interpretation.true-cls (partial-model-of I) C⟩
using assms apply –
unfolding partial-model-of-def Partial-Herbrand-Interpretation.true-cls-def
  Herbrand-Interpretation.true-cls-def consistent-interp-alt-def
by (rule bexE, assumption)
  (case-tac x; auto)

lemma total-implies-partial-true-clss-partial-model-of:
assumes ⟨Herbrand-Interpretation.true-clss I C⟩
shows ⟨Partial-Herbrand-Interpretation.true-clss (partial-model-of I) C⟩
using assms
unfolding Partial-Herbrand-Interpretation.true-clss-def
  Herbrand-Interpretation.true-clss-def
by (auto simp: total-implies-partial-true-cls-partial-model-of)

lemma partial-total-satisfiable-iff:
  ⟨Partial-Herbrand-Interpretation.satisfiable N ⟷ Herbrand-Interpretation.satisfiable N⟩
by (meson consistent-interp-partial-model-of partial-implies-total-true-clss-total-model-of
  satisfiable-carac total-implies-partial-true-clss-partial-model-of)

end
theory Prop-Logic
imports Main

```

begin

Chapter 2

Normalisation

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

2.1 Logics

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

2.1.1 Definition and Abstraction

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

datatype *'v propo* =
 FT | *FF* | *FVar 'v* | *FNot 'v propo* | *FAnd 'v propo 'v propo* | *FOR 'v propo 'v propo*
 | *FImp 'v propo 'v propo* | *FEq 'v propo 'v propo*

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

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

datatype *'v connective* = *CT* | *CF* | *CVar 'v* | *CNot* | *CAnd* | *COr* | *CImp* | *CEq*

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

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

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

lemma *propo-induct-arity*[*case-names nullary unary binary*]:

fixes $\varphi\ \psi :: 'v\ propo$
 assumes *nullary*: $\bigwedge \varphi\ x. \varphi = FF \vee \varphi = FT \vee \varphi = FVar\ x \implies P\ \varphi$
 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 $P\ \psi$
 apply (*induct rule: propo.induct*)
 using *assms* **by** *metis+*

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
using assms by (cases c) (auto simp: binary-connectives-def)
```

```
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
using assms by (cases c, auto simp add: binary-connectives-def)
```

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
using assms by induction (auto simp: binary-connectives-def)
```

2.1.2 Properties of the Abstraction

First we can define simplification rules.

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



```

wf-conn CT l  $\implies$  conn CT l = FT
wf-conn CF l  $\implies$  conn CF l = FF
wf-conn (CVar x) l  $\implies$  conn (CVar x) l = FVar x
apply (simp-all add: wf-conn.simps)
unfolding binary-connectives-def by simp-all

```

```

lemma wf-conn-list-decomp[simp]:
  wf-conn CT l  $\longleftrightarrow$  l = []
  wf-conn CF l  $\longleftrightarrow$  l = []
  wf-conn (CVar x) l  $\longleftrightarrow$  l = []
  wf-conn CNot ( $\xi$  @  $\varphi$  #  $\xi'$ )  $\longleftrightarrow$   $\xi$  = []  $\wedge$   $\xi'$  = []
apply (simp-all add: wf-conn.simps)
  unfolding binary-connectives-def apply simp-all
by (metis append-Nil append-is-Nil-conv list.distinct(1) list.sel(3) tl-append2)

```

```

lemma wf-conn-list:
  wf-conn c l  $\implies$  conn c l = FT  $\longleftrightarrow$  (c = CT  $\wedge$  l = [])
  wf-conn c l  $\implies$  conn c l = FF  $\longleftrightarrow$  (c = CF  $\wedge$  l = [])
  wf-conn c l  $\implies$  conn c l = FVar x  $\longleftrightarrow$  (c = CVar x  $\wedge$  l = [])
  wf-conn c l  $\implies$  conn c l = FAnd a b  $\longleftrightarrow$  (c = CAnd  $\wedge$  l = a # b # [])
  wf-conn c l  $\implies$  conn c l = FOr a b  $\longleftrightarrow$  (c = COr  $\wedge$  l = a # b # [])
  wf-conn c l  $\implies$  conn c l = FEq a b  $\longleftrightarrow$  (c = CEq  $\wedge$  l = a # b # [])
  wf-conn c l  $\implies$  conn c l = FImp a b  $\longleftrightarrow$  (c = CImp  $\wedge$  l = a # b # [])
  wf-conn c l  $\implies$  conn c l = FNot a  $\longleftrightarrow$  (c = CNot  $\wedge$  l = a # [])
apply (induct l rule: wf-conn.induct)
unfolding binary-connectives-def by auto

```

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 # [])
apply (induct l, auto)
by (rename-tac l, case-tac l, auto)

```

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-connectives
  shows length l = 2  $\longleftrightarrow$  wf-conn c l
proof
  assume length l = 2
  then show wf-conn c l using wf-conn-binary list-length2-decomp using conn by metis
next
  assume wf-conn c l
  then show length l = 2 (is ?P l)
  proof (cases rule: wf-conn.induct)
    case wf-conn-nullary
    then show ?P [] using conn binary-connectives-def
    using connective.distinct(11) connective.distinct(13) connective.distinct(9) by blast
  next
  fix  $\psi$  :: 'v propo
  case wf-conn-unary
  then show ?P [ $\psi$ ] using conn binary-connectives-def
  using connective.distinct by blast

```

```

next
  fix  $\psi \psi' :: 'v \text{ propo}$ 
  show  $?P [\psi, \psi']$  by auto
qed
qed

```

```

lemma wf-conn-not-list-length[iff]:
  fixes  $l :: 'v \text{ propo list}$ 
  shows  $\text{wf-conn } CNot\ l \longleftrightarrow \text{length } l = 1$ 
  apply auto
  apply (metis append-Nil connective.distinct(5,17,27) length-Cons list.size(3) wf-conn.simps
    wf-conn-list-decomp(4))
  by (simp add: length-Suc-conv wf-conn.simps)

```

Decomposing the Not into an element is moreover very useful.

```

lemma wf-conn-Not-decomp:
  fixes  $l :: 'v \text{ propo list}$  and  $a :: 'v$ 
  assumes  $\text{corr}: \text{wf-conn } CNot\ l$ 
  shows  $\exists a. l = [a]$ 
  by (metis (no-types, lifting) One-nat-def Suc-length-conv corr length-0-conv
    wf-conn-not-list-length)

```

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:
   $\text{length } l = \text{length } l' \implies \text{wf-conn } c\ l \longleftrightarrow \text{wf-conn } c\ l'$ 
proof -
  {
    fix  $l\ l'$ 
    have  $\text{length } l = \text{length } l' \implies \text{wf-conn } c\ l \implies \text{wf-conn } c\ l'$ 
      apply (cases  $c\ l$  rule: wf-conn.induct, auto)
      by (metis wf-conn-bin-list-length)
  }
  then show  $\text{length } l = \text{length } l' \implies \text{wf-conn } c\ l = \text{wf-conn } c\ l'$  by metis
qed

```

```

lemma wf-conn-no-arity-change-helper:
   $\text{length } (\xi @ \varphi \# \xi') = \text{length } (\xi @ \varphi' \# \xi')$ 
  by auto

```

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

```

lemma conn-inj-not:
  assumes  $\text{correct}: \text{wf-conn } c\ l$ 
  and  $\text{conn}: \text{conn } c\ l = FNot\ \psi$ 
  shows  $c = CNot$  and  $l = [\psi]$ 
  apply (cases  $c\ l$  rule: wf-conn.cases)
  using correct conn unfolding binary-connectives-def apply auto
  apply (cases  $c\ l$  rule: wf-conn.cases)
  using correct conn unfolding binary-connectives-def by auto

```

```

lemma conn-inj:
  fixes  $c\ ca :: 'v \text{ connective}$  and  $l\ \psi s :: 'v \text{ propo list}$ 
  assumes  $\text{corr}: \text{wf-conn } ca\ l$ 
  and  $\text{corr}': \text{wf-conn } c\ \psi s$ 

```

```

and eq: conn ca l = conn c  $\psi$ s
shows ca = c  $\wedge$   $\psi$ s = l
using corr
proof (cases ca l rule: wf-conn.cases)
case (wf-conn-nullary v)
then show ca = c  $\wedge$   $\psi$ s = l using assms
by (metis conn.simps(1) conn.simps(2) conn.simps(3) wf-conn-list(1-3))
next
case (wf-conn-unary  $\psi'$ )
then have *: FNot  $\psi'$  = conn c  $\psi$ s using conn-inj-not eq assms by auto
then have c = ca by (metis conn-inj-not(1) corr' wf-conn-unary(2))
moreover have  $\psi$ s = l using * conn-inj-not(2) corr' wf-conn-unary(1) by force
ultimately show ca = c  $\wedge$   $\psi$ s = l by auto
next
case (wf-conn-binary  $\psi'$   $\psi''$ )
then show ca = c  $\wedge$   $\psi$ s = l
using eq corr' unfolding binary-connectives-def apply (cases ca, auto simp add: wf-conn-list)
using wf-conn-list(4-7) corr' by metis+
qed

```

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 $\varphi \preceq \psi \Rightarrow \varphi \preceq \psi$
apply (induct rule: subformula.induct)
using subformula-into-subformula wf-conn-unary subformula-refl list.set-intros(1) subformula-refl
by (fastforce intro: subformula-into-subformula)+

lemma subformula-in-binary-conn:
assumes conn: $c \in \text{binary-connectives}$
shows $f \preceq \text{conn } c \ [f, g]$
and $g \preceq \text{conn } c \ [f, g]$
proof –
have *a*: wf-conn $c \ (f \# [g])$ **using** conn wf-conn-binary binary-connectives-def **by** auto
moreover **have** *b*: $f \preceq f$ **using** subformula-refl **by** auto
ultimately **show** $f \preceq \text{conn } c \ [f, g]$
by (metis append-Nil in-set-conv-decomp subformula-into-subformula)
next
have *a*: wf-conn $c \ ([f] @ [g])$ **using** conn wf-conn-binary binary-connectives-def **by** auto
moreover **have** *b*: $g \preceq g$ **using** subformula-refl **by** auto
ultimately **show** $g \preceq \text{conn } c \ [f, g]$ **using** subformula-into-subformula **by** force
qed

lemma subformula-trans:

$\psi \preceq \psi' \implies \varphi \preceq \psi \implies \varphi \preceq \psi'$
apply (induct ψ' rule: subformula.inducts)
by (auto simp: subformula-into-subformula)

lemma subformula-leaf:
fixes $\varphi \psi :: 'v \text{ propo}$
assumes incl: $\varphi \preceq \psi$
and simple: $\psi = FT \vee \psi = FF \vee \psi = FVar x$
shows $\varphi = \psi$
using incl simple
by (induct rule: subformula.induct, auto simp: wf-conn-list)

lemma subformula-not-incl-eq:
assumes $\varphi \preceq \text{conn } c \ l$
and wf-conn $c \ l$
and $\forall \psi. \psi \in \text{set } l \longrightarrow \neg \varphi \preceq \psi$
shows $\varphi = \text{conn } c \ l$
using assms **apply** (induction conn $c \ l$ rule: subformula.induct, auto)
using conn-inj **by** blast

lemma wf-subformula-conn-cases:
 $\text{wf-conn } c \ l \implies \varphi \preceq \text{conn } c \ l \longleftrightarrow (\varphi = \text{conn } c \ l \vee (\exists \psi. \psi \in \text{set } l \wedge \varphi \preceq \psi))$
apply standard
using subformula-not-incl-eq **apply** metis
by (auto simp add: subformula-into-subformula)

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

proof –

have wf-conn CAnd $[\psi, \psi']$ **by** (simp add: binary-connectives-def)
then have $\varphi \preceq \text{conn } CAnd \ [\psi, \psi'] \longleftrightarrow$
 $(\varphi = \text{conn } CAnd \ [\psi, \psi'] \vee (\exists \psi''. \psi'' \in \text{set } [\psi, \psi'] \wedge \varphi \preceq \psi''))$
using wf-subformula-conn-cases **by** metis
then show ?P FAnd **by** auto

next

have wf-conn COr $[\psi, \psi']$ **by** (simp add: binary-connectives-def)
then have $\varphi \preceq \text{conn } COr \ [\psi, \psi'] \longleftrightarrow$
 $(\varphi = \text{conn } COr \ [\psi, \psi'] \vee (\exists \psi''. \psi'' \in \text{set } [\psi, \psi'] \wedge \varphi \preceq \psi''))$
using wf-subformula-conn-cases **by** metis
then show ?P FOr **by** auto

next

have wf-conn CEq $[\psi, \psi']$ **by** (simp add: binary-connectives-def)
then have $\varphi \preceq \text{conn } CEq \ [\psi, \psi'] \longleftrightarrow$
 $(\varphi = \text{conn } CEq \ [\psi, \psi'] \vee (\exists \psi''. \psi'' \in \text{set } [\psi, \psi'] \wedge \varphi \preceq \psi''))$
using wf-subformula-conn-cases **by** metis
then show ?P FEq **by** auto

next

have wf-conn CImp $[\psi, \psi']$ **by** (simp add: binary-connectives-def)
then have $\varphi \preceq \text{conn } CImp \ [\psi, \psi'] \longleftrightarrow$
 $(\varphi = \text{conn } CImp \ [\psi, \psi'] \vee (\exists \psi''. \psi'' \in \text{set } [\psi, \psi'] \wedge \varphi \preceq \psi''))$
using wf-subformula-conn-cases **by** metis
then show ?P FImp **by** auto

qed

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

wf-conn *CNot* [φ]
wf-conn *CT* []
wf-conn *CF* []
wf-conn (*CVar* x) []
wf-conn *CAnd* [φ , ψ]
wf-conn *COr* [φ , ψ]
wf-conn *CImp* [φ , ψ]
wf-conn *CEq* [φ , ψ]
using *wf-conn.intros* **unfolding** *binary-connectives-def* **by** *fastforce+*

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

by (*cases* φ) *force+*

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

assumes *wf*: *wf-conn* $c l$

shows $\varphi \preceq \text{conn } c l \longleftrightarrow (\varphi = \text{conn } c l \vee (\exists \psi \in \text{set } l. \varphi \preceq \psi))$ (**is** $?A \longleftrightarrow ?B$)

proof (*rule iffI*)

{
 fix ξ
 have $\varphi \preceq \xi \implies \xi = \text{conn } c l \implies \text{wf-conn } c l \implies \forall x::'a \text{ propo} \in \text{set } l. \neg \varphi \preceq x \implies \varphi = \text{conn } c l$
 apply (*induct rule: subformula.induct*)
 apply *simp*
 using *conn-inj* **by** *blast*
}
moreover assume $?A$
ultimately show $?B$ **using** *wf* **by** *metis*

next

assume $?B$

then show $\varphi \preceq \text{conn } c l$ **using** *wf* *wf-subformula-conn-cases* **by** *blast*

qed

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

$\varphi \preceq FT \longleftrightarrow \varphi = FT$
 $\varphi \preceq FF \longleftrightarrow \varphi = FF$
 $\varphi \preceq FVar x \longleftrightarrow \varphi = FVar x$
apply *auto*
using *subformula-leaf* **by** *metis* +

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**

vars-of-prop *FT* = {} |
vars-of-prop *FF* = {} |
vars-of-prop (*FVar* x) = { x } |
vars-of-prop (*FNot* φ) = *vars-of-prop* φ |
vars-of-prop (*FAnd* $\varphi \psi$) = *vars-of-prop* $\varphi \cup \text{vars-of-prop } \psi$ |
vars-of-prop (*FOr* $\varphi \psi$) = *vars-of-prop* $\varphi \cup \text{vars-of-prop } \psi$ |
vars-of-prop (*FImp* $\varphi \psi$) = *vars-of-prop* $\varphi \cup \text{vars-of-prop } \psi$ |
vars-of-prop (*FEq* $\varphi \psi$) = *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 *corr*: *wf-conn* $c l$ **and** *incl*: $\psi \in \text{set } l$

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

proof (*cases c rule: connective-cases-arity-2*)

```

case nullary
then have False using corr incl by auto
then show vars-of-prop  $\psi \subseteq \text{vars-of-prop } (\text{conn } c \ l)$  by blast
next
case binary note c = this
then obtain a b where ab:  $l = [a, b]$ 
  using wf-conn-bin-list-length list-length2-decomp corr by metis
then have  $\psi = a \vee \psi = b$  using incl by auto
then show vars-of-prop  $\psi \subseteq \text{vars-of-prop } (\text{conn } c \ l)$ 
  using ab c unfolding binary-connectives-def by auto
next
case unary note c = this
fix  $\varphi :: 'v \text{ propo}$ 
have  $l = [\psi]$  using corr c incl split-list by force
then show vars-of-prop  $\psi \subseteq \text{vars-of-prop } (\text{conn } c \ l)$  using c by auto
qed

```

The set of variables is compatible with the subformula order.

lemma *subformula-vars-of-prop*:

```

 $\varphi \preceq \psi \implies \text{vars-of-prop } \varphi \subseteq \text{vars-of-prop } \psi$ 
apply (induct rule: subformula.induct)
apply simp
using vars-of-prop-incl-conn by blast

```

2.1.4 Positions

Instead of 1 or 2 we use L or R

datatype *sign* = $L \mid R$

We use *nil* instead of ε .

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

```

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

```

lemma *finite-pos*: *finite* (*pos* φ)

by (*induct* φ , *auto*)

lemma *finite-inj-comp-set*:

```

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

```

proof (*induct* s *rule*: *finite-induct*)

show *card* $\{f \ p \mid p. p \in \{\}\} = \text{card } \{\}$ **by** *auto*

next

```

fix x ::  $'v$  and s ::  $'v \text{ set}$ 
assume f: finite s and notin:  $x \notin s$ 
and IH: card  $\{f \ p \mid p. p \in s\} = \text{card } s$ 

```

have f' : *finite* $\{f\ p \mid p. p \in \text{insert } x\ s\}$ **using** f **by** *auto*
have *notin'*: $f\ x \notin \{f\ p \mid p. p \in s\}$ **using** *notin inj injD* **by** *fastforce*
have $\{f\ p \mid p. p \in \text{insert } x\ s\} = \text{insert } (f\ x)\ \{f\ p \mid p. p \in s\}$ **by** *auto*
then have $\text{card } \{f\ p \mid p. p \in \text{insert } x\ s\} = 1 + \text{card } \{f\ p \mid p. p \in s\}$
using *finite card-insert-disjoint f' notin'* **by** *auto*
moreover have $\dots = \text{card } (\text{insert } x\ s)$ **using** *notin f IH* **by** *auto*
finally show $\text{card } \{f\ p \mid p. p \in \text{insert } x\ s\} = \text{card } (\text{insert } x\ s)$.
qed

lemma *cons-inject*:

inj ((#) s)
by (*meson injI list.inject*)

lemma *finite-insert-nil-cons*:

finite s \implies card (insert [] {L # p | p. p \in s}) = 1 + card {L # p | p. p \in s}
using *card-insert-disjoint* **by** *auto*

lemma *cord-not[simp]*:

card (pos (FNot φ)) = 1 + card (pos φ)
by (*simp add: cons-inject finite-inj-comp-set finite-pos*)

lemma *card-seperate*:

assumes *finite s1 and 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** $\text{card } (?L \cup ?R) = \text{card } ?L + \text{card } ?R$)

proof –

have *finite ?L* **using** *assms* **by** *auto*
moreover have *finite ?R* **using** *assms* **by** *auto*
moreover have $?L \cap ?R = \{\}$ **by** *blast*
ultimately show *?thesis* **using** *assms card-Un-disjoint* **by** *blast*

qed

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

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

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

unfolding *prop-size-def* **apply** (*induct φ , auto simp add: cons-inject finite-inj-comp-set finite-pos*)

proof –

fix $\varphi1\ \varphi2 :: 'v\ \text{propo}$
assume *IH1: card (vars-of-prop $\varphi1$) \leq card (pos $\varphi1$)*
and *IH2: card (vars-of-prop $\varphi2$) \leq card (pos $\varphi2$)*
let $?L = \{L \# p \mid p. p \in \text{pos } \varphi1\}$
let $?R = \{R \# p \mid p. p \in \text{pos } \varphi2\}$
have $\text{card } (?L \cup ?R) = \text{card } ?L + \text{card } ?R$
using *card-seperate finite-pos* **by** *blast*
moreover have $\dots = \text{card } (\text{pos } \varphi1) + \text{card } (\text{pos } \varphi2)$
by (*simp add: cons-inject finite-inj-comp-set finite-pos*)
moreover have $\dots \geq \text{card } (\text{vars-of-prop } \varphi1) + \text{card } (\text{vars-of-prop } \varphi2)$ **using** *IH1 IH2* **by** *arith*
then have $\dots \geq \text{card } (\text{vars-of-prop } \varphi1 \cup \text{vars-of-prop } \varphi2)$ **using** *card-Un-le le-trans* **by** *blast*
ultimately
show $\text{card } (\text{vars-of-prop } \varphi1 \cup \text{vars-of-prop } \varphi2) \leq \text{Suc } (\text{card } (?L \cup ?R))$
 $\text{card } (\text{vars-of-prop } \varphi1 \cup \text{vars-of-prop } \varphi2) \leq \text{Suc } (\text{card } (?L \cup ?R))$
 $\text{card } (\text{vars-of-prop } \varphi1 \cup \text{vars-of-prop } \varphi2) \leq \text{Suc } (\text{card } (?L \cup ?R))$

```

      card (vars-of-prop  $\varphi 1 \cup$  vars-of-prop  $\varphi 2$ )  $\leq$  Suc (card (?L  $\cup$  ?R))
    by auto
  qed

```

```

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:  $c \in$  binary-connectives  $\vee c = CNot \implies$  wf-conn c ( $\varphi \# l$ )  $\implies$  path-to p  $\varphi$   $\varphi' \implies$ 
    path-to (L#p) (conn c ( $\varphi \# l$ ))  $\varphi'$  |
  path-to-r:  $c \in$  binary-connectives  $\implies$  wf-conn c ( $\psi \# \varphi \# []$ )  $\implies$  path-to p  $\varphi$   $\varphi' \implies$ 
    path-to (R#p) (conn c ( $\psi \# \varphi \# []$ ))  $\varphi'$ 

```

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

lemma path-to-subformula:

```

  path-to p  $\varphi$   $\varphi' \implies \varphi' \preceq \varphi$ 
apply (induct rule: path-to.induct)
apply simp
apply (metis list.set-intros(1) subformula-into-subformula)
using subformula-trans subformula-in-binary-conn(2) by metis

```

lemma subformula-path-exists:

```

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

```

proof (induct rule: subformula.induct)

```

  case subformula-refl
  have path-to []  $\varphi'$   $\varphi'$  by auto
  then show  $\exists p. \text{path-to } p \varphi' \varphi'$  by metis

```

next

```

  case (subformula-into-subformula  $\psi$  l c)
  note wf = this(2) and IH = this(4) and  $\psi = \text{this}(1)$ 
  then obtain p where p: path-to p  $\psi$   $\varphi'$  by metis

```

```

  {
    fix x :: 'v
    assume  $c = CT \vee c = CF \vee c = CVar x$ 
    then have False using subformula-into-subformula by auto
    then have  $\exists p. \text{path-to } p (\text{conn } c \ l) \varphi'$  by blast
  }

```

moreover {

```

  assume c:  $c = CNot$ 
  then have  $l = [\psi]$  using wf  $\psi$  wf-conn-Not-decomp by fastforce
  then have path-to (L # p) (conn c l)  $\varphi'$  by (metis c wf-conn-unary p path-to-l)
  then have  $\exists p. \text{path-to } p (\text{conn } c \ l) \varphi'$  by blast

```

}

moreover {

```

  assume c:  $c \in$  binary-connectives
  obtain a b where ab:  $[a, b] = l$  using subformula-into-subformula c wf-conn-bin-list-length
    list-length2-decomp by metis
  then have  $a = \psi \vee b = \psi$  using  $\psi$  by auto
  then have path-to (L # p) (conn c l)  $\varphi' \vee$  path-to (R # p) (conn c l)  $\varphi'$  using c path-to-l
    path-to-r p ab by (metis wf-conn-binary)
  then have  $\exists p. \text{path-to } p (\text{conn } c \ l) \varphi'$  by blast

```

}

ultimately show $\exists p. \text{path-to } p (\text{conn } c \ l) \varphi'$ **using** connective-cases-arity **by** metis

qed


```

fun replace-at :: sign list  $\Rightarrow$  'v propo  $\Rightarrow$  'v propo  $\Rightarrow$  'v propo where
replace-at [] -  $\psi = \psi$  |
replace-at (L # l) (FAnd  $\varphi \varphi'$ )  $\psi = FAnd$  (replace-at l  $\varphi \psi$ )  $\varphi'$  |
replace-at (R # l) (FAnd  $\varphi \varphi'$ )  $\psi = FAnd$   $\varphi$  (replace-at l  $\varphi' \psi$ ) |
replace-at (L # l) (FOr  $\varphi \varphi'$ )  $\psi = FOr$  (replace-at l  $\varphi \psi$ )  $\varphi'$  |
replace-at (R # l) (FOr  $\varphi \varphi'$ )  $\psi = FOr$   $\varphi$  (replace-at l  $\varphi' \psi$ ) |
replace-at (L # l) (FEq  $\varphi \varphi'$ )  $\psi = FEq$  (replace-at l  $\varphi \psi$ )  $\varphi'$  |
replace-at (R # l) (FEq  $\varphi \varphi'$ )  $\psi = FEq$   $\varphi$  (replace-at l  $\varphi' \psi$ ) |
replace-at (L # l) (FImp  $\varphi \varphi'$ )  $\psi = FImp$  (replace-at l  $\varphi \psi$ )  $\varphi'$  |
replace-at (R # l) (FImp  $\varphi \varphi'$ )  $\psi = FImp$   $\varphi$  (replace-at l  $\varphi' \psi$ ) |
replace-at (L # l) (FNot  $\varphi$ )  $\psi = FNot$  (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
 $\mathcal{A} \models FT = True$  |
 $\mathcal{A} \models FF = False$  |
 $\mathcal{A} \models FVar\ v = (\mathcal{A}\ v)$  |
 $\mathcal{A} \models FNot\ \varphi = (\neg(\mathcal{A} \models \varphi))$  |
 $\mathcal{A} \models FAnd\ \varphi_1\ \varphi_2 = (\mathcal{A} \models \varphi_1 \wedge \mathcal{A} \models \varphi_2)$  |
 $\mathcal{A} \models FOr\ \varphi_1\ \varphi_2 = (\mathcal{A} \models \varphi_1 \vee \mathcal{A} \models \varphi_2)$  |
 $\mathcal{A} \models FImp\ \varphi_1\ \varphi_2 = (\mathcal{A} \models \varphi_1 \longrightarrow \mathcal{A} \models \varphi_2)$  |
 $\mathcal{A} \models FEq\ \varphi_1\ \varphi_2 = (\mathcal{A} \models \varphi_1 \longleftrightarrow \mathcal{A} \models \varphi_2)$ 

```

```

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

```

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

theorem *deduction-theorem*:

$\varphi \models_f \psi \longleftrightarrow (\forall A. A \models FImp\ \varphi\ \psi)$

proof

```

assume H:  $\varphi \models_f \psi$ 
{
  fix A
  have  $A \models FImp\ \varphi\ \psi$ 
  proof (cases  $A \models \varphi$ )
    case True
    then have  $A \models \psi$  using H unfolding evalf-def by metis
    then show  $A \models FImp\ \varphi\ \psi$  by auto
  next
    case False
    then show  $A \models FImp\ \varphi\ \psi$  by auto
  qed
}
then show  $\forall A. A \models FImp\ \varphi\ \psi$  by blast
next
assume A:  $\forall A. A \models FImp\ \varphi\ \psi$ 
show  $\varphi \models_f \psi$ 
proof (rule ccontr)
  assume  $\neg \varphi \models_f \psi$ 
  then obtain A where  $A \models \varphi$  and  $\neg A \models \psi$  using evalf-def by metis

```

```

    then have  $\neg A \models FImp\ \varphi\ \psi$  by auto
    then show False using A by blast
qed

```

A shorter proof:

```

lemma  $\varphi \models_f \psi \longleftrightarrow (\forall A. A \models FImp\ \varphi\ \psi)$ 
  by (simp add: evalf-def)

```

```

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

```

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

```

lemma same-over-set-eval:
  assumes same-over-set A B (vars-of-prop  $\varphi$ )
  shows  $A \models \varphi \longleftrightarrow B \models \varphi$ 
  using assms unfolding same-over-set-def by (induct  $\varphi$ , auto)
end

```