

Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

March 31, 2016

Contents

1	Resolution	1
1.1	Simplification Rules	1
1.2	Unconstrained Resolution	3
1.2.1	Subsumption	3
1.3	Inference Rule	4
1.4	Lemma about the simplified state	19
1.5	Resolution and Invariants	21
1.5.1	Invariants	22
1.5.2	well-foundedness if the relation	27
2	Superposition	42
2.1	We can now define the rules of the calculus	49

theory *Prop-Resolution*

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

begin

1 Resolution

1.1 Simplification Rules

inductive *simplify* :: '*v* clauses \Rightarrow '*v* clauses \Rightarrow bool **for** *N* :: '*v* clause set **where**

tautology-deletion:

$(A + \{\#Pos\ P\# \} + \{\#Neg\ P\# \}) \in N \Longrightarrow simplify\ N\ (N - \{A + \{\#Pos\ P\# \} + \{\#Neg\ P\# \}\})$

condensation:

$(A + \{\#L\# \} + \{\#L\# \}) \in N \Longrightarrow simplify\ N\ (N - \{A + \{\#L\# \} + \{\#L\# \}\} \cup \{A + \{\#L\# \}\})$

subsumption:

$A \in N \Longrightarrow A \subset\# B \Longrightarrow B \in N \Longrightarrow simplify\ N\ (N - \{B\})$

lemma *simplify-preserves-un-sat'*:

fixes *N N'* :: '*v* clauses

assumes *simplify N N'*

and *total-over-m I N*

shows $I \models_s N' \longrightarrow I \models_s N$

using *assms*

proof (*induct rule: simplify.induct*)

case (*tautology-deletion A P*)

then have $I \models A + \{\#Pos\ P\# \} + \{\#Neg\ P\# \}$

by (*metis total-over-m-def total-over-set-literal-defined true-cls-singleton true-cls-union*)

```

      true-lit-def uminus-Neg union-commute)
    then show ?case by (metis Un-Diff-cancel2 true-clss-singleton true-clss-union)
  next
    case (condensation A P)
    then show ?case by (metis Diff-insert-absorb Set.set-insert insertE true-clss-union true-clss-def
      true-clss-singleton true-clss-union)
  next
    case (subsumption A B)
    have  $A \neq B$  using subsumption.hyps(2) by auto
    then have  $I \models N - \{B\} \implies I \models A$  using  $\langle A \in N \rangle$  by (simp add: true-clss-def)
    moreover have  $I \models A \implies I \models B$  using  $\langle A < \# B \rangle$  by auto
    ultimately show ?case by (metis insert-Diff-single true-clss-insert)
qed

```

```

lemma simplify-preserves-un-sat:
  fixes  $N N' :: 'v \text{ clauses}$ 
  assumes simplify  $N N'$ 
  and total-over-m  $I N$ 
  shows  $I \models_s N \longrightarrow I \models_s N'$ 
  using assms apply (induct rule: simplify.induct)
  using true-clss-def by fastforce+

```

```

lemma simplify-preserves-un-sat'':
  fixes  $N N' :: 'v \text{ clauses}$ 
  assumes simplify  $N N'$ 
  and total-over-m  $I N'$ 
  shows  $I \models_s N \longrightarrow I \models_s N'$ 
  using assms apply (induct rule: simplify.induct)
  using true-clss-def by fastforce+

```

```

lemma simplify-preserves-un-sat-eq:
  fixes  $N N' :: 'v \text{ clauses}$ 
  assumes simplify  $N N'$ 
  and total-over-m  $I N$ 
  shows  $I \models_s N \longleftrightarrow I \models_s N'$ 
  using simplify-preserves-un-sat simplify-preserves-un-sat' assms by blast

```

```

lemma simplify-preserves-finite:
  assumes simplify  $\psi \psi'$ 
  shows finite  $\psi \longleftrightarrow$  finite  $\psi'$ 
  using assms by (induct rule: simplify.induct, auto simp add: remove-def)

```

```

lemma rtranclp-simplify-preserves-finite:
  assumes rtranclp simplify  $\psi \psi'$ 
  shows finite  $\psi \longleftrightarrow$  finite  $\psi'$ 
  using assms by (induct rule: rtranclp-induct) (auto simp add: simplify-preserves-finite)

```

```

lemma simplify-atms-of-ms:
  assumes simplify  $\psi \psi'$ 
  shows atms-of-ms  $\psi' \subseteq$  atms-of-ms  $\psi$ 
  using assms unfolding atms-of-ms-def
proof (induct rule: simplify.induct)
  case (tautology-deletion A P)
  then show ?case by auto
next

```

```

case (condensation A P)
moreover have A + {#P#} + {#P#} ∈ ψ ⇒ ∃ x ∈ ψ. atm-of P ∈ atm-of ‘ set-mset x
  by (metis Un-iff atms-of-def atms-of-plus atms-of-singleton insert-iff)
ultimately show ?case by (auto simp add: atms-of-def)
next
case (subsumption A P)
then show ?case by auto
qed

lemma rtranclp-simplify-atms-of-ms:
  assumes rtranclp simplify ψ ψ'
  shows atms-of-ms ψ' ⊆ atms-of-ms ψ
  using assms apply (induct rule: rtranclp-induct)
  apply (fastforce intro: simplify-atms-of-ms)
  using simplify-atms-of-ms by blast

lemma factoring-imp-simplify:
  assumes {#L#} + {#L#} + C ∈ N
  shows ∃ N'. simplify N N'
proof -
  have C + {#L#} + {#L#} ∈ N using assms by (simp add: add.commute union-lcomm)
  from condensation[OF this] show ?thesis by blast
qed

```

1.2 Unconstrained Resolution

```

type-synonym 'v uncon-state = 'v clauses
inductive uncon-res :: 'v uncon-state ⇒ 'v uncon-state ⇒ bool where
resolution:
  {#Pos p#} + C ∈ N ⇒ {#Neg p#} + D ∈ N ⇒ ({#Pos p#} + C, {#Neg p#} + D) ∉
already-used
  ⇒ uncon-res (N) (N ∪ {C + D}) |
factoring: {#L#} + {#L#} + C ∈ N ⇒ uncon-res N (N ∪ {C + {#L#}})

lemma uncon-res-increasing:
  assumes uncon-res S S' and ψ ∈ S
  shows ψ ∈ S'
  using assms by (induct rule: uncon-res.induct) auto

lemma rtranclp-uncon-inference-increasing:
  assumes rtranclp uncon-res S S' and ψ ∈ S
  shows ψ ∈ S'
  using assms by (induct rule: rtranclp-induct) (auto simp add: uncon-res-increasing)

```

1.2.1 Subsumption

```

definition subsumes :: 'a literal multiset ⇒ 'a literal multiset ⇒ bool where
subsumes χ χ' ⟷
  (∀ I. total-over-m I {χ'} ⟶ total-over-m I {χ})
  ∧ (∀ I. total-over-m I {χ} ⟶ I ⊨ χ ⟶ I ⊨ χ')

lemma subsumes-refl[simp]:
  subsumes χ χ
  unfolding subsumes-def by auto

```

lemma *subsumes-subsumption*:

assumes *subsumes* $D \chi$
and $C \subset\# D$ **and** $\neg \text{tautology } \chi$
shows *subsumes* $C \chi$ **unfolding** *subsumes-def*
using *assms* *subsumption-total-over-m* *subsumption-chained* **unfolding** *subsumes-def*
by (*blast intro!*: *subset-mset.less-imp-le*)

lemma *subsumes-tautology*:

assumes *subsumes* $(C + \{\#Pos\ P\# \} + \{\#Neg\ P\# \}) \chi$
shows *tautology* χ
using *assms* **unfolding** *subsumes-def* **by** (*simp add*: *tautology-def*)

1.3 Inference Rule

type-synonym *'v state* = *'v clauses* \times (*'v clause* \times *'v clause*) *set*

inductive *inference-clause* :: *'v state* \Rightarrow *'v clause* \times (*'v clause* \times *'v clause*) *set* \Rightarrow *bool*

(**infix** \Rightarrow_{Res} 100) **where**

resolution:

$\{\#Pos\ p\#\} + C \in N \implies \{\#Neg\ p\#\} + D \in N \implies (\{\#Pos\ p\#\} + C, \{\#Neg\ p\#\} + D) \notin$
already-used
 $\implies \text{inference-clause } (N, \text{already-used}) (C + D, \text{already-used} \cup \{(\{\#Pos\ p\#\} + C, \{\#Neg\ p\#\} + D)\}) \mid$

factoring: $\{\#L\#\} + \{\#L\#\} + C \in N \implies \text{inference-clause } (N, \text{already-used}) (C + \{\#L\#\}, \text{already-used})$

inductive *inference* :: *'v state* \Rightarrow *'v state* \Rightarrow *bool* **where**

inference-step: *inference-clause* S (*clause*, *already-used*)

$\implies \text{inference } S (\text{fst } S \cup \{\text{clause}\}, \text{already-used})$

abbreviation *already-used-inv*

:: *'a literal multiset set* \times (*'a literal multiset* \times *'a literal multiset*) *set* \Rightarrow *bool* **where**

already-used-inv state \equiv

$(\forall (A, B) \in \text{snd state}. \exists p. \text{Pos } p \in\# A \wedge \text{Neg } p \in\# B \wedge$
 $((\exists \chi \in \text{fst state}. \text{subsumes } \chi ((A - \{\#Pos\ p\#\}) + (B - \{\#Neg\ p\#\})))$
 $\vee \text{tautology } ((A - \{\#Pos\ p\#\}) + (B - \{\#Neg\ p\#\}))))$

lemma *inference-clause-preserves-already-used-inv*:

assumes *inference-clause* $S S'$
and *already-used-inv* S
shows *already-used-inv* $(\text{fst } S \cup \{\text{fst } S'\}, \text{snd } S')$
using *assms* **apply** (*induct rule*: *inference-clause.induct*)
by *fastforce*+

lemma *inference-preserves-already-used-inv*:

assumes *inference* $S S'$
and *already-used-inv* S
shows *already-used-inv* S'
using *assms*

proof (*induct rule*: *inference.induct*)

case (*inference-step* S *clause* *already-used*)

then show ?*case*

using *inference-clause-preserves-already-used-inv*[*of* S (*clause*, *already-used*)] **by** *simp*

qed

lemma *rtranclp-inference-preserves-already-used-inv*:

assumes *rtranclp inference* $S S'$

```

and already-used-inv S
shows already-used-inv S'
using assms apply (induct rule: rtrancp-induct, simp)
using inference-preserves-already-used-inv unfolding tautology-def by fast

lemma subsumes-condensation:
  assumes subsumes (C + {#L#} + {#L#}) D
  shows subsumes (C + {#L#}) D
  using assms unfolding subsumes-def by simp

lemma simplify-preserves-already-used-inv:
  assumes simplify N N'
  and already-used-inv (N, already-used)
  shows already-used-inv (N', already-used)
  using assms
proof (induct rule: simplify.induct)
  case (condensation C L)
  then show ?case
    using subsumes-condensation by simp fast
next
{
  fix a:: 'a and A :: 'a set and P
  have  $(\exists x \in \text{Set.remove } a \ A. P \ x) \longleftrightarrow (\exists x \in A. x \neq a \wedge P \ x)$  by auto
} note ex-member-remove = this
{
  fix a a0 :: 'v clause and A :: 'v clauses and y
  assume a ∈ A and a0 ⊂# a
  then have  $(\exists x \in A. \text{subsumes } x \ y) \longleftrightarrow (\text{subsumes } a \ y \vee (\exists x \in A. x \neq a \wedge \text{subsumes } x \ y))$ 
    by auto
} note tt2 = this
case (subsumption A B) note A = this(1) and AB = this(2) and B = this(3) and inv = this(4)
show ?case
proof (standard, standard)
  fix x a b
  assume x:  $x \in \text{snd } (N - \{B\}, \text{already-used})$  and [simp]:  $x = (a, b)$ 
  obtain p where p:  $\text{Pos } p \in\# a \wedge \text{Neg } p \in\# b$  and
    q:  $(\exists \chi \in N. \text{subsumes } \chi (a - \{\# \text{Pos } p\# \} + (b - \{\# \text{Neg } p\# \})))$ 
     $\vee \text{tautology } (a - \{\# \text{Pos } p\# \} + (b - \{\# \text{Neg } p\# \}))$ 
  using inv x by fastforce
  consider (taut)  $\text{tautology } (a - \{\# \text{Pos } p\# \} + (b - \{\# \text{Neg } p\# \})) \mid$ 
     $(\chi) \chi$  where  $\chi \in N$   $\text{subsumes } \chi (a - \{\# \text{Pos } p\# \} + (b - \{\# \text{Neg } p\# \}))$ 
     $\neg \text{tautology } (a - \{\# \text{Pos } p\# \} + (b - \{\# \text{Neg } p\# \}))$ 
  using q by auto
  then show
     $\exists p. \text{Pos } p \in\# a \wedge \text{Neg } p \in\# b$ 
     $\wedge ((\exists \chi \in \text{fst } (N - \{B\}, \text{already-used}). \text{subsumes } \chi (a - \{\# \text{Pos } p\# \} + (b - \{\# \text{Neg } p\# \})))$ 
     $\vee \text{tautology } (a - \{\# \text{Pos } p\# \} + (b - \{\# \text{Neg } p\# \})))$ 
  proof cases
    case taut
    then show ?thesis using p by auto
  next
    case  $\chi$  note H = this
    show ?thesis using p A AB B subsumes-subsumption[OF - AB H(3)] H(1,2) by auto
  qed
qed

```

next
case (*tautology-deletion* $C\ P$)
then show ?*case* **apply** *clarify*
proof –
 fix $a\ b$
 assume $C + \{\#Pos\ P\# \} + \{\#Neg\ P\# \} \in N$
 assume *already-used-inv* (N , *already-used*)
 and $(a, b) \in \text{snd } (N - \{C + \{\#Pos\ P\# \} + \{\#Neg\ P\# \}\}, \text{already-used})$
 then obtain p **where**
 $Pos\ p \in \# a \wedge Neg\ p \in \# b \wedge$
 $((\exists \chi \in \text{fst } (N \cup \{C + \{\#Pos\ P\# \} + \{\#Neg\ P\# \}\}, \text{already-used}).$
 $\text{subsumes } \chi (a - \{\#Pos\ p\# \} + (b - \{\#Neg\ p\# \})))$
 $\vee \text{tautology } (a - \{\#Pos\ p\# \} + (b - \{\#Neg\ p\# \})))$
 by *fastforce*
moreover have *tautology* ($C + \{\#Pos\ P\# \} + \{\#Neg\ P\# \}$) **by** *auto*
ultimately show
 $\exists p. Pos\ p \in \# a \wedge Neg\ p \in \# b$
 $\wedge ((\exists \chi \in \text{fst } (N - \{C + \{\#Pos\ P\# \} + \{\#Neg\ P\# \}\}, \text{already-used}).$
 $\text{subsumes } \chi (a - \{\#Pos\ p\# \} + (b - \{\#Neg\ p\# \})))$
 $\vee \text{tautology } (a - \{\#Pos\ p\# \} + (b - \{\#Neg\ p\# \})))$
 by (*metis* (*no-types*) *Diff-iff Un-insert-right empty-iff fst-conv insertE subsumes-tautology*
 sup-bot.right-neutral)
qed
qed

lemma
factoring-satisfiable: $I \models \{\#L\# \} + \{\#L\# \} + C \longleftrightarrow I \models \{\#L\# \} + C$ **and**
resolution-satisfiable:
 consistent-interp $I \implies I \models \{\#Pos\ p\# \} + C \implies I \models \{\#Neg\ p\# \} + D \implies I \models C + D$ **and**
 factoring-same-vars: $\text{atms-of } (\{\#L\# \} + \{\#L\# \} + C) = \text{atms-of } (\{\#L\# \} + C)$
unfolding *true-cls-def consistent-interp-def* **by** (*fastforce split: if-split-asm*) +

lemma *inference-increasing*:
assumes *inference* $S\ S'$ **and** $\psi \in \text{fst } S$
shows $\psi \in \text{fst } S'$
using *assms* **by** (*induct rule: inference.induct, auto*)

lemma *rtranclp-inference-increasing*:
assumes *rtranclp inference* $S\ S'$ **and** $\psi \in \text{fst } S$
shows $\psi \in \text{fst } S'$
using *assms* **by** (*induct rule: rtranclp-induct, auto simp add: inference-increasing*)

lemma *inference-clause-already-used-increasing*:
assumes *inference-clause* $S\ S'$
shows $\text{snd } S \subseteq \text{snd } S'$
using *assms* **by** (*induct rule: inference-clause.induct, auto*)

lemma *inference-already-used-increasing*:
assumes *inference* $S\ S'$
shows $\text{snd } S \subseteq \text{snd } S'$
using *assms* **apply** (*induct rule: inference.induct*)
using *inference-clause-already-used-increasing* **by** *fastforce*

```

lemma inference-clause-preserves-un-sat:
  fixes  $N\ N' :: 'v\ clauses$ 
  assumes inference-clause  $T\ T'$ 
  and total-over-m  $I\ (fst\ T)$ 
  and consistent: consistent-interp  $I$ 
  shows  $I \models_s fst\ T \longleftrightarrow I \models_s fst\ T \cup \{fst\ T'\}$ 
  using assms apply (induct rule: inference-clause.induct)
  unfolding consistent-interp-def true-clss-def by auto force+

lemma inference-preserves-un-sat:
  fixes  $N\ N' :: 'v\ clauses$ 
  assumes inference  $T\ T'$ 
  and total-over-m  $I\ (fst\ T)$ 
  and consistent: consistent-interp  $I$ 
  shows  $I \models_s fst\ T \longleftrightarrow I \models_s fst\ T'$ 
  using assms apply (induct rule: inference.induct)
  using inference-clause-preserves-un-sat by fastforce

lemma inference-clause-preserves-atms-of-ms:
  assumes inference-clause  $S\ S'$ 
  shows atms-of-ms  $(fst\ (fst\ S \cup \{fst\ S'\},\ snd\ S')) \subseteq \textit{atms-of-ms}\ (fst\ S)$ 
  using assms apply (induct rule: inference-clause.induct)
  apply auto
  apply (metis Set.set-insert UnCI atms-of-ms-insert atms-of-plus)
  apply (metis Set.set-insert UnCI atms-of-ms-insert atms-of-plus)
  apply (simp add: in-m-in-literals union-assoc)
  unfolding atms-of-ms-def using assms by fastforce

lemma inference-preserves-atms-of-ms:
  fixes  $N\ N' :: 'v\ clauses$ 
  assumes inference  $T\ T'$ 
  shows atms-of-ms  $(fst\ T') \subseteq \textit{atms-of-ms}\ (fst\ T)$ 
  using assms apply (induct rule: inference.induct)
  using inference-clause-preserves-atms-of-ms by fastforce

lemma inference-preserves-total:
  fixes  $N\ N' :: 'v\ clauses$ 
  assumes inference  $(N,\ \textit{already-used})\ (N',\ \textit{already-used'})$ 
  shows total-over-m  $I\ N \implies \textit{total-over-m}\ I\ N'$ 
  using assms inference-preserves-atms-of-ms unfolding total-over-m-def total-over-set-def
  by fastforce

lemma rtranclp-inference-preserves-total:
  assumes rtranclp inference  $T\ T'$ 
  shows total-over-m  $I\ (fst\ T) \implies \textit{total-over-m}\ I\ (fst\ T')$ 
  using assms by (induct rule: rtranclp-induct, auto simp add: inference-preserves-total)

lemma rtranclp-inference-preserves-un-sat:
  assumes rtranclp inference  $N\ N'$ 
  and total-over-m  $I\ (fst\ N)$ 
  and consistent: consistent-interp  $I$ 
  shows  $I \models_s fst\ N \longleftrightarrow I \models_s fst\ N'$ 
  using assms apply (induct rule: rtranclp-induct)

```

apply (*simp add: inference-preserves-un-sat*)
using *inference-preserves-un-sat rtranclp-inference-preserves-total* **by** *blast*

lemma *inference-preserves-finite*:
assumes *inference ψ ψ' and finite (fst ψ)*
shows *finite (fst ψ')*
using *assms by (induct rule: inference.induct, auto simp add: simplify-preserves-finite)*

lemma *inference-clause-preserves-finite-snd*:
assumes *inference-clause ψ ψ' and finite (snd ψ)*
shows *finite (snd ψ')*
using *assms by (induct rule: inference-clause.induct, auto)*

lemma *inference-preserves-finite-snd*:
assumes *inference ψ ψ' and finite (snd ψ)*
shows *finite (snd ψ')*
using *assms inference-clause-preserves-finite-snd by (induct rule: inference.induct, fastforce)*

lemma *rtranclp-inference-preserves-finite*:
assumes *rtranclp inference ψ ψ' and finite (fst ψ)*
shows *finite (fst ψ')*
using *assms by (induct rule: rtranclp-induct)*
(auto simp add: simplify-preserves-finite inference-preserves-finite)

lemma *consistent-interp-insert*:
assumes *consistent-interp I*
and *atm-of $P \notin \text{atm-of } I$*
shows *consistent-interp (insert P I)*
proof –
have *P : insert P I = I \cup { P }* **by** *auto*
show *?thesis unfolding P*
apply (*rule consistent-interp-disjoint*)
using *assms by (auto simp: image-iff)*
qed

lemma *simplify-clause-preserves-sat*:
assumes *simp: simplify ψ ψ'*
and *satisfiable ψ'*
shows *satisfiable ψ*
using *assms*
proof *induction*
case (*tautology-deletion A P*) **note** *AP = this(1) and sat = this(2)*
let *?A' = A + {#Pos P#} + {#Neg P#}*
let *? ψ' = ψ - { $?A'$ }*
obtain *I where*
I: I \models_s ? ψ' and
cons: consistent-interp I and
tot: total-over-m I ? ψ'
using *sat unfolding satisfiable-def by auto*
{ assume *Pos P \in I \vee Neg P \in I*
then have *I \models ?A' by auto*
then have *I \models_s ψ using I by (metis insert-Diff tautology-deletion.hyps true-clss-insert)*


```

    then have ?case using cons tot by auto
  }
  moreover {
    assume Pos: Pos P  $\notin$  I and Neg: Neg P  $\notin$  I
    then have consistent-interp (I  $\cup$  {Pos P}) using cons by simp
    moreover have I'A: I  $\cup$  {Pos P}  $\models$  ?A' by auto
    have {Pos P}  $\cup$  I  $\models_s \psi - \{A + \{\#Pos\ P\#\} + \{\#Neg\ P\#\}\}$ 
      using  $\langle I \models_s \psi - \{A + \{\#Pos\ P\#\} + \{\#Neg\ P\#\}\} \rangle$  true-clss-union-increase' by blast
    then have I  $\cup$  {Pos P}  $\models_s \psi$ 
      by (metis (no-types) Un-empty-right Un-insert-left Un-insert-right I'A insert-Diff
        sup-bot.left-neutral tautology-deletion.hyps true-clss-insert)
    ultimately have ?case using satisfiable-carac' by blast
  }
  ultimately show ?case by blast
next
case (condensation A L) note AL = this(1) and sat = this(2)
have f3: simplify  $\psi$  ( $\psi - \{A + \{\#L\#\} + \{\#L\#\}\} \cup \{A + \{\#L\#\}\}$ )
  using AL simplify.condensation by blast
obtain LL :: 'a literal multiset set  $\Rightarrow$  'a literal set where
  f4: LL ( $\psi - \{A + \{\#L\#\} + \{\#L\#\}\} \cup \{A + \{\#L\#\}\}$ )  $\models_s \psi - \{A + \{\#L\#\} + \{\#L\#\}\} \cup \{A$ 
 $+ \{\#L\#\}$ 
     $\wedge$  consistent-interp (LL ( $\psi - \{A + \{\#L\#\} + \{\#L\#\}\} \cup \{A + \{\#L\#\}\}$ ))
     $\wedge$  total-over-m (LL ( $\psi - \{A + \{\#L\#\} + \{\#L\#\}\}$ 
       $\cup \{A + \{\#L\#\}\}$ )) ( $\psi - \{A + \{\#L\#\} + \{\#L\#\}\} \cup \{A + \{\#L\#\}\}$ )
  using sat by (meson satisfiable-def)
have f5: insert (A + {#L#} + {#L#}) ( $\psi - \{A + \{\#L\#\} + \{\#L\#\}\}$ ) =  $\psi$ 
  using AL by fastforce
have atms-of (A + {#L#} + {#L#}) = atms-of ({#L#} + A)
  by simp
then show ?case
  using f5 f4 f3 by (metis (no-types) add.commute satisfiable-def simplify-preserves-un-sat'
    total-over-m-insert total-over-m-union)
next
case (subsumption A B) note A = this(1) and AB = this(2) and B = this(3) and sat = this(4)
let ? $\psi'$  =  $\psi - \{B\}$ 
obtain I where I: I  $\models_s$  ? $\psi'$  and cons: consistent-interp I and tot: total-over-m I ? $\psi'$ 
  using sat unfolding satisfiable-def by auto
have I  $\models$  A using A I by (metis AB Diff-iff subset-mset.less-irrefl singletonD true-clss-def)
then have I  $\models$  B using AB subset-mset.less-imp-le true-clss-mono-leD by blast
then have I  $\models_s \psi$  using I by (metis insert-Diff-single true-clss-insert)
then show ?case using cons satisfiable-carac' by blast
qed

```

lemma simplify-preserves-unsat:

```

  assumes inference  $\psi$   $\psi'$ 
  shows satisfiable (fst  $\psi'$ )  $\longrightarrow$  satisfiable (fst  $\psi$ )
  using assms apply (induct rule: inference.induct)
  using satisfiable-decreasing by (metis fst-conv)+

```

lemma inference-preserves-unsat:

```

  assumes inference** S S'
  shows satisfiable (fst S')  $\longrightarrow$  satisfiable (fst S)
  using assms apply (induct rule: rtranclp-induct)
  apply simp-all
  using simplify-preserves-unsat by blast

```

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

fun sem-tree-size :: 'v sem-tree \Rightarrow nat **where**

sem-tree-size Leaf = 0 |

sem-tree-size (Node - ag ad) = 1 + sem-tree-size ag + sem-tree-size ad

lemma sem-tree-size[case-names bigger]:

($\bigwedge xs:: 'v \text{ sem-tree. } (\bigwedge ys:: 'v \text{ sem-tree. } \text{sem-tree-size } ys < \text{sem-tree-size } xs \Rightarrow P \text{ } ys) \Rightarrow P \text{ } xs$)
 $\Rightarrow P \text{ } xs$

by (fact Nat.measure-induct-rule)

fun partial-interps :: 'v sem-tree \Rightarrow 'v interp \Rightarrow 'v clauses \Rightarrow bool **where**

partial-interps Leaf I ψ = ($\exists \chi. \neg I \models \chi \wedge \chi \in \psi \wedge \text{total-over-m } I \{ \chi \}$) |

partial-interps (Node v ag ad) I $\psi \longleftrightarrow$

(partial-interps ag (I \cup {Pos v}) $\psi \wedge$ partial-interps ad (I \cup {Neg v}) ψ)

lemma simplify-preserve-partial-leaf:

simplify N N' \Rightarrow partial-interps Leaf I N \Rightarrow partial-interps Leaf I N'

apply (induct rule: simplify.induct)

using union-lcomm **apply** auto[1]

apply (simp, metis atms-of-plus total-over-set-union true-cls-union)

apply simp

by (metis atms-of-ms-singleton mset-le-exists-conv subset-mset-def true-cls-mono-leD
total-over-m-def total-over-m-sum)

lemma simplify-preserve-partial-tree:

assumes simplify N N'

and partial-interps t I N

shows partial-interps t I N'

using assms **apply** (induct t arbitrary: I, simp)

using simplify-preserve-partial-leaf **by** metis

lemma inference-preserve-partial-tree:

assumes inference S S'

and partial-interps t I (fst S)

shows partial-interps t I (fst S')

using assms **apply** (induct t arbitrary: I, simp-all)

by (meson inference-increasing)

lemma rtranclp-inference-preserve-partial-tree:

assumes rtranclp inference N N'

and partial-interps t I (fst N)

shows partial-interps t I (fst N')

using assms **apply** (induct rule: rtranclp-induct, auto)

using inference-preserve-partial-tree **by** force

function build-sem-tree :: 'v :: linorder set \Rightarrow 'v clauses \Rightarrow 'v sem-tree **where**

build-sem-tree atms ψ =

```

    (if atms = {}  $\vee$   $\neg$  finite atms
    then Leaf
    else Node (Min atms) (build-sem-tree (Set.remove (Min atms) atms)  $\psi$ )
      (build-sem-tree (Set.remove (Min atms) atms)  $\psi$ ))
  by auto
termination
  apply (relation measure ( $\lambda(A, -).$  card A), simp-all)
  apply (metis Min-in card-Diff1-less remove-def)+
done
declare build-sem-tree.induct[case-names tree]

lemma unsatisfiable-empty[simp]:
   $\neg$ unsatisfiable {}
  unfolding satisfiable-def apply auto
  using consistent-interp-def unfolding total-over-m-def total-over-set-def atms-of-ms-def by blast

lemma partial-interps-build-sem-tree-atms-general:
  fixes  $\psi :: 'v :: \text{linorder}$  clauses and  $p :: 'v$  literal list
  assumes unsat: unsatisfiable  $\psi$  and finite  $\psi$  and consistent-interp I
  and finite atms
  and atms-of-ms  $\psi = \text{atms} \cup \text{atms-of-s } I$  and  $\text{atms} \cap \text{atms-of-s } I = \{\}$ 
  shows partial-interps (build-sem-tree atms  $\psi$ ) I  $\psi$ 
  using assms
proof (induct arbitrary: I rule: build-sem-tree.induct)
  case (1 atms  $\psi$  Ia) note IH1 = this(1) and IH2 = this(2) and unsat = this(3) and finite = this(4)
  and cons = this(5) and f = this(6) and un = this(7) and disj = this(8)
  {
    assume atms: atms = {}
    then have atmsIa: atms-of-ms  $\psi = \text{atms-of-s } Ia$  using un by auto
    then have total-over-m Ia  $\psi$  unfolding total-over-m-def atmsIa by auto
    then have  $\chi: \exists \chi \in \psi. \neg Ia \models \chi$ 
      using unsat cons unfolding true-clss-def satisfiable-def by auto
    then have build-sem-tree atms  $\psi = \text{Leaf}$  using atms by auto
    moreover
      have tot:  $\bigwedge \chi. \chi \in \psi \implies \text{total-over-m } Ia \ \{\chi\}$ 
      unfolding total-over-m-def total-over-set-def atms-of-ms-def atms-of-s-def
      using atmsIa atms-of-ms-def by fastforce
    have partial-interps Leaf Ia  $\psi$ 
      using  $\chi$  tot by (auto simp add: total-over-m-def total-over-set-def atms-of-ms-def)

    ultimately have ?case by metis
  }
moreover {
  assume atms: atms  $\neq \{\}$ 
  have build-sem-tree atms  $\psi = \text{Node } (\text{Min atms}) (\text{build-sem-tree } (\text{Set.remove } (\text{Min atms}) \text{ atms}) \psi)$ 
    (build-sem-tree (Set.remove (Min atms) atms)  $\psi$ )
    using build-sem-tree.simps[of atms  $\psi$ ] f atms by metis

  have consistent-interp (Ia  $\cup \{\text{Pos } (\text{Min atms})\}$ ) unfolding consistent-interp-def
    by (metis Int-iff Min-in Un-iff atm-of-uminus atms cons consistent-interp-def disj empty-iff
      f in-atms-of-s-decomp insert-iff literal.distinct(1) literal.exhaust-sel literal.sel(2)
      uminus-Neg uminus-Pos)
  moreover have atms-of-ms  $\psi = \text{Set.remove } (\text{Min atms}) \text{ atms} \cup \text{atms-of-s } (Ia \cup \{\text{Pos } (\text{Min atms})\})$ 
    using Min-in atms f un by fastforce
  moreover have disj':  $\text{Set.remove } (\text{Min atms}) \text{ atms} \cap \text{atms-of-s } (Ia \cup \{\text{Pos } (\text{Min atms})\}) = \{\}$ 

```

by simp (metis disj disjoint-iff-not-equal member-remove)
 moreover have finite (Set.remove (Min atms) atms) using f by (simp add: remove-def)
 ultimately have subtree1: partial-interps (build-sem-tree (Set.remove (Min atms) atms) ψ)
 (Ia \cup {Pos (Min atms)}) ψ
 using IH1[of Ia \cup {Pos (Min (atms))}] atms f unsat finite by metis

 have consistent-interp (Ia \cup {Neg (Min atms)}) unfolding consistent-interp-def
 by (metis Int-iff Min-in Un-iff atm-of-uminus atms cons consistent-interp-def disj empty-iff
 f in-atms-of-s-decomp insert-iff literal.distinct(1) literal.exhaust-sel literal.sel(2)
 uminus-Neg)
 moreover have atms-of-ms ψ = Set.remove (Min atms) atms \cup atms-of-s (Ia \cup {Neg (Min atms)})
 using \langle atms-of-ms ψ = Set.remove (Min atms) atms \cup atms-of-s (Ia \cup {Pos (Min atms)}) \rangle by
 blast

 moreover have disj': Set.remove (Min atms) atms \cap atms-of-s (Ia \cup {Neg (Min atms)}) = {}
 using disj by auto
 moreover have finite (Set.remove (Min atms) atms) using f by (simp add: remove-def)
 ultimately have subtree2: partial-interps (build-sem-tree (Set.remove (Min atms) atms) ψ)
 (Ia \cup {Neg (Min atms)}) ψ
 using IH2[of Ia \cup {Neg (Min (atms))}] atms f unsat finite by metis

 then have ?case
 using IH1 subtree1 subtree2 f local.finite unsat atms by simp
 }
 ultimately show ?case by metis
qed

lemma partial-interps-build-sem-tree-atms:

fixes $\psi :: 'v :: \text{linorder clauses}$ and $p :: 'v \text{ literal list}$
 assumes unsat: unsatisfiable ψ and finite: finite ψ
 shows partial-interps (build-sem-tree (atms-of-ms ψ) ψ) {} ψ
 proof –
 have consistent-interp {} unfolding consistent-interp-def by auto
 moreover have atms-of-ms ψ = atms-of-ms $\psi \cup$ atms-of-s {} unfolding atms-of-s-def by auto
 moreover have atms-of-ms $\psi \cap$ atms-of-s {} = {} unfolding atms-of-s-def by auto
 moreover have finite (atms-of-ms ψ) unfolding atms-of-ms-def using finite by simp
 ultimately show partial-interps (build-sem-tree (atms-of-ms ψ) ψ) {} ψ
 using partial-interps-build-sem-tree-atms-general[of ψ {} atms-of-ms ψ] assms by metis
 qed

lemma can-decrease-count:

fixes $\psi'' :: 'v \text{ clauses} \times ('v \text{ clause} \times 'v \text{ clause} \times 'v) \text{ set}$
 assumes count $\chi \ L = n$
 and $L \in \# \chi$ and $\chi \in \text{fst } \psi$
 shows $\exists \psi' \chi'. \text{inference}^{**} \psi \psi' \wedge \chi' \in \text{fst } \psi' \wedge (\forall L. L \in \# \chi \longleftrightarrow L \in \# \chi')$
 $\wedge \text{count } \chi' \ L = 1$
 $\wedge (\forall \varphi. \varphi \in \text{fst } \psi \longrightarrow \varphi \in \text{fst } \psi')$
 $\wedge (I \models \chi \longleftrightarrow I \models \chi')$
 $\wedge (\forall I'. \text{total-over-}m \ I' \{\chi\} \longrightarrow \text{total-over-}m \ I' \{\chi'\})$
 using assms
 proof (induct n arbitrary: $\chi \psi$)
 case 0
 then show ?case by (simp add: not-in-iff[symmetric])
 next

```

case (Suc n  $\chi$ )
note IH = this(1) and count = this(2) and L = this(3) and  $\chi$  = this(4)
{
  assume n = 0
  then have inference**  $\psi$   $\psi$ 
  and  $\chi \in \text{fst } \psi$ 
  and  $\forall L. (L \in \# \chi) \longleftrightarrow (L \in \# \chi)$ 
  and count  $\chi$  L = (1::nat)
  and  $\forall \varphi. \varphi \in \text{fst } \psi \longrightarrow \varphi \in \text{fst } \psi$ 
    by (auto simp add: count L  $\chi$ )
  then have ?case by metis
}
moreover {
  assume n > 0
  then have  $\exists C. \chi = C + \{\#L, L\# \}$ 
    by (smt L Suc-eq-plus1-left add.left-commute add-diff-cancel-left' add-diff-cancel-right'
      count-greater-zero-iff count-single local.count multi-member-split plus-multiset.rep-eq)
  then obtain C where  $\chi = C + \{\#L, L\# \}$  by metis
  let ? $\chi'$  = C + {#L#}
  let ? $\psi'$  = (fst  $\psi \cup \{?\chi'\}$ , snd  $\psi$ )
  have  $\varphi: \forall \varphi \in \text{fst } \psi. (\varphi \in \text{fst } \psi \vee \varphi \neq ?\chi') \longleftrightarrow \varphi \in \text{fst } ?\psi'$  unfolding C by auto
  have inf: inference  $\psi$  ? $\psi'$ 
    using C factoring  $\chi$  prod.collapse union-commute inference-step by metis
  moreover have count': count ? $\chi'$  L = n using C count by auto
  moreover have L $\chi'$ : L  $\in \#$  ? $\chi'$  by auto
  moreover have  $\chi' \psi'$ : ? $\chi' \in \text{fst } ?\psi'$  by auto
  ultimately obtain  $\psi''$  and  $\chi''$ 
  where
    inference** ? $\psi'$   $\psi''$  and
     $\alpha: \chi'' \in \text{fst } \psi''$  and
     $\forall La. (La \in \# ?\chi') \longleftrightarrow (La \in \# \chi'')$  and
     $\beta: \text{count } \chi''$  L = (1::nat) and
     $\varphi': \forall \varphi. \varphi \in \text{fst } ?\psi' \longrightarrow \varphi \in \text{fst } \psi''$  and
     $I\chi: I \models ?\chi' \longleftrightarrow I \models \chi''$  and
    tot:  $\forall I'. \text{total-over-m } I' \{?\chi'\} \longrightarrow \text{total-over-m } I' \{\chi''\}$ 
    using IH[of ? $\chi'$  ? $\psi'$ ] count' L $\chi'$   $\chi' \psi'$  by blast

  then have inference**  $\psi$   $\psi''$ 
  and  $\forall La. (La \in \# \chi) \longleftrightarrow (La \in \# \chi'')$ 
  using inf unfolding C by auto
  moreover have  $\forall \varphi. \varphi \in \text{fst } \psi \longrightarrow \varphi \in \text{fst } \psi''$  using  $\varphi \varphi'$  by metis
  moreover have  $I \models \chi \longleftrightarrow I \models \chi''$  using I $\chi$  unfolding true-cls-def C by auto
  moreover have  $\forall I'. \text{total-over-m } I' \{\chi\} \longrightarrow \text{total-over-m } I' \{\chi''\}$ 
    using tot unfolding C total-over-m-def by auto
  ultimately have ?case using  $\varphi \varphi' \alpha \beta$  by metis
}
ultimately show ?case by auto
qed

```

lemma can-decrease-tree-size:

```

fixes  $\psi :: 'v$  state and tree :: 'v sem-tree
assumes finite (fst  $\psi$ ) and already-used-inv  $\psi$ 
and partial-interps tree I (fst  $\psi$ )
shows  $\exists (\text{tree}' :: 'v \text{ sem-tree}) \psi'. \text{inference** } \psi \psi' \wedge \text{partial-interps tree}' I (\text{fst } \psi')$ 
   $\wedge (\text{sem-tree-size tree}' < \text{sem-tree-size tree} \vee \text{sem-tree-size tree} = 0)$ 

```

```

using assms
proof (induct arbitrary: I rule: sem-tree-size)
  case (bigger xs I) note IH = this(1) and finite = this(2) and a-u-i = this(3) and part = this(4)

  {
    assume sem-tree-size xs = 0
    then have ?case using part by blast
  }

  moreover {
    assume sn0: sem-tree-size xs > 0
    obtain ag ad v where xs: xs = Node v ag ad using sn0 by (cases xs, auto)
    {
      assume sem-tree-size ag = 0 and sem-tree-size ad = 0
      then have ag: ag = Leaf and ad: ad = Leaf by (cases ag, auto) (cases ad, auto)

      then obtain  $\chi$   $\chi'$  where
         $\chi$ :  $\neg I \cup \{Pos\ v\} \models \chi$  and
        tot $\chi$ : total-over-m ( $I \cup \{Pos\ v\}$ )  $\{\chi\}$  and
         $\chi\psi$ :  $\chi \in fst\ \psi$  and
         $\chi'$ :  $\neg I \cup \{Neg\ v\} \models \chi'$  and
        tot $\chi'$ : total-over-m ( $I \cup \{Neg\ v\}$ )  $\{\chi'\}$  and
         $\chi'\psi$ :  $\chi' \in fst\ \psi$ 
        using part unfolding xs by auto
      have Posv:  $\neg Pos\ v \in \# \chi$  using  $\chi$  unfolding true-cls-def true-lit-def by auto
      have Negv:  $\neg Neg\ v \in \# \chi'$  using  $\chi'$  unfolding true-cls-def true-lit-def by auto
      {
        assume Neg $\chi$ :  $\neg Neg\ v \in \# \chi$ 
        have  $\neg I \models \chi$  using  $\chi$  Posv unfolding true-cls-def true-lit-def by auto
        moreover have total-over-m I  $\{\chi\}$ 
          using Posv Neg $\chi$  atm-imp-pos-or-neg-lit tot $\chi$  unfolding total-over-m-def total-over-set-def
          by fastforce
        ultimately have partial-interps Leaf I (fst  $\psi$ )
          and sem-tree-size Leaf < sem-tree-size xs
          and inference**  $\psi\ \psi$ 
          unfolding xs by (auto simp add:  $\chi\psi$ )
      }
    }
    moreover {
      assume Pos $\chi$ :  $\neg Pos\ v \in \# \chi'$ 
      then have I $\chi$ :  $\neg I \models \chi'$  using  $\chi'$  Posv unfolding true-cls-def true-lit-def by auto
      moreover have total-over-m I  $\{\chi'\}$ 
        using Negv Pos $\chi$  atm-imp-pos-or-neg-lit tot $\chi'$ 
        unfolding total-over-m-def total-over-set-def by fastforce
      ultimately have partial-interps Leaf I (fst  $\psi$ ) and
        sem-tree-size Leaf < sem-tree-size xs and
        inference**  $\psi\ \psi$ 
        using  $\chi'\psi$  I $\chi$  unfolding xs by auto
    }
  }
  moreover {
    assume neg: Neg v  $\in \# \chi$  and pos: Pos v  $\in \# \chi'$ 
    then obtain  $\psi'$   $\chi2$  where inf: rtrnclp inference  $\psi\ \psi'$  and  $\chi2incl$ :  $\chi2 \in fst\ \psi'$ 
    and  $\chi\chi2incl$ :  $\forall L. L \in \# \chi \longleftrightarrow L \in \# \chi2$ 
    and count $\chi2$ : count  $\chi2$  (Neg v) = 1
    and  $\varphi$ :  $\forall \varphi::'v\ literal\ multiset. \varphi \in fst\ \psi \longrightarrow \varphi \in fst\ \psi'$ 
    and I $\chi$ :  $I \models \chi \longleftrightarrow I \models \chi2$ 
  }

```

and $\text{tot-imp}\chi: \forall I'. \text{total-over-m } I' \{ \chi \} \longrightarrow \text{total-over-m } I' \{ \chi^2 \}$
using $\text{can-decrease-count}[\text{of } \chi \text{ Neg } v \text{ count } \chi \text{ (Neg } v) \psi I] \chi \psi \chi' \psi$ **by** *auto*

have $\chi' \in \text{fst } \psi'$ **by** (*simp add: $\chi' \psi \varphi$*)
with *pos*
obtain $\psi'' \chi^2'$ **where**
inf': *inference** $\psi' \psi''$*
and $\chi^2' \text{-incl}: \chi^2' \in \text{fst } \psi''$
and $\chi' \chi^2 \text{-incl}: \forall L::'v \text{ literal. } (L \in \# \chi') = (L \in \# \chi^2')$
and $\text{count}\chi^2': \text{count } \chi^2' (\text{Pos } v) = (1::\text{nat})$
and $\varphi': \forall \varphi::'v \text{ literal multiset. } \varphi \in \text{fst } \psi' \longrightarrow \varphi \in \text{fst } \psi''$
and $I\chi': I \models \chi' \longleftrightarrow I \models \chi^2'$
and $\text{tot-imp}\chi': \forall I'. \text{total-over-m } I' \{ \chi' \} \longrightarrow \text{total-over-m } I' \{ \chi^2' \}$
using $\text{can-decrease-count}[\text{of } \chi' \text{ Pos } v \text{ count } \chi' (\text{Pos } v) \psi' I]$ **by** *auto*

obtain C **where** $\chi^2: \chi^2 = C + \{ \# \text{Neg } v \# \}$ **and** $\text{neg}C: \text{Neg } v \notin \# C$ **and** $\text{pos}C: \text{Pos } v \notin \# C$
proof –
have $\bigwedge m. \text{Suc } 0 - \text{count } m (\text{Neg } v) = \text{count } (\chi^2 - m) (\text{Neg } v)$
by (*simp add: count χ^2*)
then show *?thesis*
using *that by (metis (no-types) One-nat-def Posv Suc-inject Suc-pred $\chi \chi^2 \text{-incl}$ count-diff count-single insert-DiffM2 mem-Collect-eq multi-member-skip neg not-gr0 set-mset-def union-commute)*
qed

obtain C' **where**
 $\chi^2': \chi^2' = C' + \{ \# \text{Pos } v \# \}$ **and**
 $\text{pos}C': \text{Pos } v \notin \# C'$ **and**
 $\text{neg}C': \text{Neg } v \notin \# C'$
proof –
assume $a1: \bigwedge C'. \llbracket \chi^2' = C' + \{ \# \text{Pos } v \# \}; \text{Pos } v \notin \# C'; \text{Neg } v \notin \# C' \rrbracket \Longrightarrow \text{thesis}$
have $f2: \bigwedge n. (n::\text{nat}) - n = 0$
by *simp*
have $\text{Neg } v \notin \# \chi^2' - \{ \# \text{Pos } v \# \}$
using $\text{Negv } \chi' \chi^2 \text{-incl}$ **by** (*auto simp: not-in-iff*)
have $\text{count } \{ \# \text{Pos } v \# \} (\text{Pos } v) = 1$
by *simp*
then show *?thesis*
by (*metis $\chi' \chi^2 \text{-incl}$ $\langle \text{Neg } v \notin \# \chi^2' - \{ \# \text{Pos } v \# \} \rangle$ a1 count χ^2' count-diff f2 insert-DiffM2 less-numeral-extra(3) mem-Collect-eq pos set-mset-def*)
qed

have *already-used-inv ψ'*
using *rtranclp-inference-preserves-already-used-inv[of $\psi \psi']$ a-u-i inf* **by** *blast*
then have *a-u-i- ψ'' : already-used-inv ψ''*
using *rtranclp-inference-preserves-already-used-inv a-u-i inf'* **unfolding** *tautology-def*
by *simp*

have $\text{tot}C: \text{total-over-m } I \{ C \}$
using $\text{tot-imp}\chi \text{ tot}\chi \text{ tot-over-m-remove}[\text{of } I \text{ Pos } v C] \text{ neg}C \text{ pos}C$ **unfolding** χ^2
by (*metis total-over-m-sum uminus-Neg uminus-of-uminus-id*)
have $\text{tot}C': \text{total-over-m } I \{ C' \}$
using $\text{tot-imp}\chi' \text{ tot}\chi' \text{ total-over-m-sum tot-over-m-remove}[\text{of } I \text{ Neg } v C'] \text{ neg}C' \text{ pos}C'$
unfolding χ^2' **by** (*metis total-over-m-sum uminus-Neg*)
have $\neg I \models C + C'$

```

using  $\chi \ I\chi \ \chi' \ I\chi'$  unfolding  $\chi^2 \ \chi^{2'}$  true-cls-def by auto
then have part-I- $\psi'''$ : partial-interps Leaf I (fst  $\psi'' \cup \{C + C'\}$ )
  using totC totC' by simp
  (metis  $\lhd I \models C + C'$ ) atms-of-ms-singleton total-over-m-def total-over-m-sum)
{
  assume ( $\{\#Pos \ v\# \} + C', \{\#Neg \ v\# \} + C \notin \text{snd } \psi''$ 
then have inf'': inference  $\psi''$  (fst  $\psi'' \cup \{C + C'\}$ , snd  $\psi'' \cup \{(\chi^{2'}, \chi^2)\}$ )
  using add.commute  $\varphi' \ \chi^{2'} \text{incl } \langle \chi^{2'} \in \text{fst } \psi'' \rangle$  unfolding  $\chi^2 \ \chi^{2'}$ 
  by (metis prod.collapse inference-step resolution)
have inference**  $\psi$  (fst  $\psi'' \cup \{C + C'\}$ , snd  $\psi'' \cup \{(\chi^{2'}, \chi^2)\}$ )
  using inf inf' inf'' rtranclp-trans by auto
moreover have sem-tree-size Leaf < sem-tree-size xs unfolding xs by auto
ultimately have ?case using part-I- $\psi'''$  by (metis fst-conv)
}
moreover {
  assume a: ( $\{\#Pos \ v\# \} + C', \{\#Neg \ v\# \} + C \in \text{snd } \psi''$ 
then have ( $\exists \chi \in \text{fst } \psi''$ . ( $\forall I$ . total-over-m I  $\{C+C'\} \longrightarrow \text{total-over-m } I \ \{\chi\}$ )
     $\wedge (\forall I$ . total-over-m I  $\{\chi\} \longrightarrow I \models \chi \longrightarrow I \models C' + C)$ )
     $\vee \text{tautology } (C' + C)$ 
proof –
  obtain p where p: Pos p  $\in \# (\{\#Pos \ v\# \} + C')$  and
  n: Neg p  $\in \# (\{\#Neg \ v\# \} + C)$  and
  decomp: ( $\exists \chi \in \text{fst } \psi''$ .
    ( $\forall I$ . total-over-m I  $\{(\{\#Pos \ v\# \} + C') - \{\#Pos \ p\# \}$ 
       $+ ((\{\#Neg \ v\# \} + C) - \{\#Neg \ p\# \})\}$ 
       $\longrightarrow \text{total-over-m } I \ \{\chi\}$ )
     $\wedge (\forall I$ . total-over-m I  $\{\chi\} \longrightarrow I \models \chi$ 
       $\longrightarrow I \models (\{\#Pos \ v\# \} + C') - \{\#Pos \ p\# \} + ((\{\#Neg \ v\# \} + C) - \{\#Neg \ p\# \})$ 
    )
     $\vee \text{tautology } ((\{\#Pos \ v\# \} + C') - \{\#Pos \ p\# \} + ((\{\#Neg \ v\# \} + C) - \{\#Neg \ p\# \}))$ )
  using a by (blast intro: allE[OF a-u-i- $\psi''$ ][unfolding subsumes-def Ball-def],
    of ( $\{\#Pos \ v\# \} + C', \{\#Neg \ v\# \} + C$ ))
  { assume  $p \neq v$ 
    then have Pos p  $\in \# C' \wedge \text{Neg } p \in \# C$  using p n by force
    then have ?thesis unfolding Bex-def by auto
  }
moreover {
  assume  $p = v$ 
  then have ?thesis using decomp by (metis add.commute add-diff-cancel-left)
  }
ultimately show ?thesis by auto
qed
moreover {
  assume  $\exists \chi \in \text{fst } \psi''$ . ( $\forall I$ . total-over-m I  $\{C+C'\} \longrightarrow \text{total-over-m } I \ \{\chi\}$ )
     $\wedge (\forall I$ . total-over-m I  $\{\chi\} \longrightarrow I \models \chi \longrightarrow I \models C' + C)$ 
then obtain  $\vartheta$  where  $\vartheta$ :  $\vartheta \in \text{fst } \psi''$  and
    tot- $\vartheta$ -CC':  $\forall I$ . total-over-m I  $\{C+C'\} \longrightarrow \text{total-over-m } I \ \{\vartheta\}$  and
     $\vartheta$ -inv:  $\forall I$ . total-over-m I  $\{\vartheta\} \longrightarrow I \models \vartheta \longrightarrow I \models C' + C$  by blast
have partial-interps Leaf I (fst  $\psi''$ )
  using tot- $\vartheta$ -CC'  $\vartheta$ -inv totC totC'  $\lhd I \models C + C'$  total-over-m-sum by fastforce
moreover have sem-tree-size Leaf < sem-tree-size xs unfolding xs by auto
ultimately have ?case by (metis inf inf' rtranclp-trans)
}
moreover {
  assume tautCC': tautology  $(C' + C)$ 

```



```

    have total-over-m I {C'+C} using totC totC' total-over-m-sum by auto
    then have ¬tautology (C' + C)
      using ⟨¬ I ⊨ C + C'⟩ unfolding add.commute[of C C'] total-over-m-def
      unfolding tautology-def by auto
    then have False using tautCC' unfolding tautology-def by auto
  }
  ultimately have ?case by auto
}
ultimately have ?case by auto
}
ultimately have ?case using part by (metis (no-types) sem-tree-size.simps(1))
}
moreover {
  assume size-ag: sem-tree-size ag > 0
  have sem-tree-size ag < sem-tree-size xs unfolding xs by auto
  moreover have partial-interps ag (I ∪ {Pos v}) (fst ψ)
    and partad: partial-interps ad (I ∪ {Neg v}) (fst ψ)
    using part partial-interps.simps(2) unfolding xs by metis+
  moreover have sem-tree-size ag < sem-tree-size xs ⟶ finite (fst ψ) ⟶ already-used-inv ψ
    ⟶ ( partial-interps ag (I ∪ {Pos v}) (fst ψ) ⟶
      (∃ tree' ψ'. inference** ψ ψ' ∧ partial-interps tree' (I ∪ {Pos v}) (fst ψ')
        ∧ (sem-tree-size tree' < sem-tree-size ag ∨ sem-tree-size ag = 0)))
    using IH by auto
  ultimately obtain ψ' :: 'v state and tree' :: 'v sem-tree where
    inf: inference** ψ ψ'
    and part: partial-interps tree' (I ∪ {Pos v}) (fst ψ')
    and size: sem-tree-size tree' < sem-tree-size ag ∨ sem-tree-size ag = 0
    using finite part rtranclp.rtrancl-refl a-u-i by blast

  have partial-interps ad (I ∪ {Neg v}) (fst ψ')
    using rtranclp-inference-preserve-partial-tree inf partad by metis
  then have partial-interps (Node v tree' ad) I (fst ψ') using part by auto
  then have ?case using inf size size-ag part unfolding xs by fastforce
}
moreover {
  assume size-ad: sem-tree-size ad > 0
  have sem-tree-size ad < sem-tree-size xs unfolding xs by auto
  moreover have partag: partial-interps ag (I ∪ {Pos v}) (fst ψ) and
    partial-interps ad (I ∪ {Neg v}) (fst ψ)
    using part partial-interps.simps(2) unfolding xs by metis+
  moreover have sem-tree-size ad < sem-tree-size xs ⟶ finite (fst ψ) ⟶ already-used-inv ψ
    ⟶ ( partial-interps ad (I ∪ {Neg v}) (fst ψ)
      ⟶ (∃ tree' ψ'. inference** ψ ψ' ∧ partial-interps tree' (I ∪ {Neg v}) (fst ψ')
        ∧ (sem-tree-size tree' < sem-tree-size ad ∨ sem-tree-size ad = 0)))
    using IH by auto
  ultimately obtain ψ' :: 'v state and tree' :: 'v sem-tree where
    inf: inference** ψ ψ'
    and part: partial-interps tree' (I ∪ {Neg v}) (fst ψ')
    and size: sem-tree-size tree' < sem-tree-size ad ∨ sem-tree-size ad = 0
    using finite part rtranclp.rtrancl-refl a-u-i by blast

  have partial-interps ag (I ∪ {Pos v}) (fst ψ')
    using rtranclp-inference-preserve-partial-tree inf partag by metis
  then have partial-interps (Node v ag tree') I (fst ψ') using part by auto
  then have ?case using inf size size-ad unfolding xs by fastforce
}

```

```

}
ultimately have ?case by auto
}
ultimately show ?case by auto
qed

```

lemma *inference-completeness-inv*:

fixes $\psi :: 'v :: \text{linorder state}$

assumes

unsat: $\neg \text{satisfiable (fst } \psi)$ **and**

finite: $\text{finite (fst } \psi)$ **and**

a-u-v: *already-used-inv* ψ

shows $\exists \psi'. (\text{inference}^{**} \psi \psi' \wedge \{\#\} \in \text{fst } \psi')$

proof –

obtain *tree* **where** *partial-interps* *tree* $\{\}$ (*fst* ψ)

using *partial-interps-build-sem-tree-atms* *assms* **by** *metis*

then show *?thesis*

using *unsat finite a-u-v*

proof (*induct tree arbitrary: ψ rule: sem-tree-size*)

case (*bigger tree* ψ) **note** $H = \text{this}$

{

fix χ

assume *tree*: *tree* = *Leaf*

obtain χ **where** $\chi: \neg \{\} \models \chi$ **and** *tot* χ : *total-over-m* $\{\} \{\chi\}$ **and** $\chi\psi: \chi \in \text{fst } \psi$

using *H unfolding tree* **by** *auto*

moreover have $\{\#\} = \chi$

using *tot* χ *unfolding total-over-m-def total-over-set-def* **by** *fastforce*

moreover have *inference*^{**} $\psi \psi$ **by** *auto*

ultimately have *?case* **by** *metis*

}

moreover {

fix $v \text{ tree1 tree2}$

assume *tree*: *tree* = *Node v tree1 tree2*

obtain

tree' ψ' **where** *inf*: *inference*^{**} $\psi \psi'$ **and**

part': *partial-interps* *tree'* $\{\}$ (*fst* ψ') **and**

decrease: *sem-tree-size* *tree'* < *sem-tree-size* *tree* \vee *sem-tree-size* *tree* = 0

using *can-decrease-tree-size*[*of* ψ] *H*(2,4,5) **unfolding** *tautology-def* **by** *meson*

have *sem-tree-size* *tree'* < *sem-tree-size* *tree* **using** *decrease* **unfolding** *tree* **by** *auto*

moreover have *finite* (*fst* ψ') **using** *rtranclp-inference-preserves-finite inf H*(4) **by** *metis*

moreover have *unsatisfiable* (*fst* ψ')

using *inference-preserves-unsat inf bigger.premis*(2) **by** *blast*

moreover have *already-used-inv* ψ'

using *H*(5) *inf rtranclp-inference-preserves-already-used-inv*[*of* $\psi \psi'$] **by** *auto*

ultimately have *?case* **using** *inf rtranclp-trans part'* *H*(1) **by** *fastforce*

}

ultimately show *?case* **by** (*cases tree, auto*)

qed

qed

lemma *inference-completeness*:

fixes $\psi :: 'v :: \text{linorder state}$

assumes *unsat*: $\neg \text{satisfiable (fst } \psi)$

and *finite*: *finite* (*fst* ψ)

and *snd* $\psi = \{\}$

shows $\exists \psi'. (\text{rtrancp inference } \psi \ \psi' \wedge \{\#\} \in \text{fst } \psi')$
proof –
 have *already-used-inv* ψ **unfolding** *assms* **by** *auto*
 then show *?thesis* **using** *assms inference-completeness-inv* **by** *blast*
qed

lemma *inference-soundness*:
 fixes $\psi :: 'v :: \text{linorder state}$
 assumes *rtrancp inference* $\psi \ \psi'$ **and** $\{\#\} \in \text{fst } \psi'$
 shows *unsatisfiable* (*fst* ψ)
 using *assms* **by** (*meson rtrancp-inference-preserves-un-sat satisfiable-def true-cls-empty true-cls-def*)

lemma *inference-soundness-and-completeness*:
 fixes $\psi :: 'v :: \text{linorder state}$
 assumes *finite*: *finite* (*fst* ψ)
 and *snd* $\psi = \{\}$
 shows $(\exists \psi'. (\text{inference}^{**} \ \psi \ \psi' \wedge \{\#\} \in \text{fst } \psi')) \longleftrightarrow \text{unsatisfiable} (\text{fst } \psi)$
 using *assms inference-completeness inference-soundness* **by** *metis*

1.4 Lemma about the simplified state

abbreviation *simplified* $\psi \equiv (\text{no-step simplify } \psi)$

lemma *simplified-count*:
 assumes *simp*: *simplified* ψ **and** $\chi: \chi \in \psi$
 shows *count* $\chi \ L \leq 1$
proof –
 {
 let $? \chi' = \chi - \{\#L, L\# \}$
 assume *count* $\chi \ L \geq 2$
 then have *f1*: *count* $(\chi - \{\#L, L\# \} + \{\#L, L\# \}) \ L = \text{count } \chi \ L$
 by *simp*
 then have $L \in \# \ \chi - \{\#L\# \}$
 by (*metis* (*no-types*) *add.left-neutral add-diff-cancel-left' count-union diff-diff-add diff-single-trivial insert-DiffM mem-Collect-eq multi-member-this not-gr0 set-mset-def*)
 then have $\chi': ? \chi' + \{\#L\# \} + \{\#L\# \} = \chi$
 using *f1* **by** (*metis diff-diff-add diff-single-eq-union in-diffD*)

 have $\exists \psi'. \text{simplify } \psi \ \psi'$
 by (*metis* (*no-types, hide-lams*) $\chi \ \chi'$ *add.commute factoring-imp-simplify union-assoc*)
 then have *False* **using** *simp* **by** *auto*
 }
 then show *?thesis* **by** *arith*
qed

lemma *simplified-no-both*:
 assumes *simp*: *simplified* ψ **and** $\chi: \chi \in \psi$
 shows $\neg (L \in \# \ \chi \wedge -L \in \# \ \chi)$
proof (*rule ccontr*)
 assume $\neg \neg (L \in \# \ \chi \wedge -L \in \# \ \chi)$
 then have $L \in \# \ \chi \wedge -L \in \# \ \chi$ **by** *metis*
 then obtain χ' **where** $\chi = \chi' + \{\# \text{Pos } (\text{atm-of } L)\# \} + \{\# \text{Neg } (\text{atm-of } L)\# \}$
 by (*metis Neg-atm-of-iff Pos-atm-of-iff diff-union-swap insert-DiffM2 uminus-Neg uminus-Pos*)
 then show *False* **using** χ *simp tautology-deletion* **by** *fastforce*
qed

```

lemma simplified-not-tautology:
  assumes simplified  $\{\psi\}$ 
  shows  $\sim \text{tautology } \psi$ 
proof (rule ccontr)
  assume  $\sim ?thesis$ 
  then obtain  $p$  where  $Pos\ p \in \# \psi \wedge Neg\ p \in \# \psi$  using tautology-decomp by metis
  then obtain  $\chi$  where  $\psi = \chi + \{\#Pos\ p\# \} + \{\#Neg\ p\# \}$ 
    by (metis insert-noteq-member literal.distinct(1) multi-member-split)
  then have  $\sim \text{simplified } \{\psi\}$  by (auto intro: tautology-deletion)
  then show False using assms by auto
qed

```

```

lemma simplified-remove:
  assumes simplified  $\{\psi\}$ 
  shows simplified  $\{\psi - \{\#l\#\}\}$ 
proof (rule ccontr)
  assume  $ns: \neg \text{simplified } \{\psi - \{\#l\#\}\}$ 
  {
    assume  $\neg l \in \# \psi$ 
    then have  $\psi - \{\#l\#\} = \psi$  by simp
    then have False using ns assms by auto
  }
  moreover {
    assume  $l\psi: l \in \# \psi$ 
    have  $A: \bigwedge A. A \in \{\psi - \{\#l\#\}\} \longleftrightarrow A + \{\#l\#\} \in \{\psi\}$  by (auto simp add: lψ)
    obtain  $l'$  where  $l': \text{simplify } \{\psi - \{\#l\#\}\} \ l'$  using ns by metis
    then have  $\exists l'. \text{simplify } \{\psi\} \ l'$ 
    proof (induction rule: simplify.induct)
      case (tautology-deletion A P)
      have  $\{\#Neg\ P\# \} + (\{\#Pos\ P\# \} + (A + \{\#l\#\})) \in \{\psi\}$ 
        by (metis (no-types) A add.commute tautology-deletion.hyps union-lcomm)
      then show ?thesis
        by (metis simplify.tautology-deletion[of A + \{\#l\#\} P \{\psi\}] add.commute)
    next
      case (condensation A L)
      have  $A + \{\#L\#\} + \{\#L\#\} + \{\#l\#\} \in \{\psi\}$ 
        using A condensation.hyps by blast
      then have  $\{\#L, L\#\} + (A + \{\#l\#\}) \in \{\psi\}$ 
        by (metis (no-types) union-assoc union-commute)
      then show ?case
        using factoring-imp-simplify by blast
    next
      case (subsumption A B)
      then show ?case by blast
    qed
  }
  then have False using assms(1) by blast
}
ultimately show False by auto
qed

```

```

lemma in-simplified-simplified:
  assumes simp: simplified  $\psi$  and incl:  $\psi' \subseteq \psi$ 
  shows simplified  $\psi'$ 

```

```

proof (rule ccontr)
  assume  $\neg$  ?thesis
  then obtain  $\psi''$  where simplify  $\psi' \psi''$  by metis
  then have  $\exists l'. \text{ simplify } \psi l'$ 
  proof (induction rule: simplify.induct)
    case (tautology-deletion  $A P$ )
      then show ?thesis using simplify.tautology-deletion[of  $A P \psi$ ] incl by blast
    next
      case (condensation  $A L$ )
        then show ?case using simplify.condensation[of  $A L \psi$ ] incl by blast
      next
        case (subsumption  $A B$ )
          then show ?case using simplify.subsumption[of  $A \psi B$ ] incl by auto
        qed
  then show False using assms(1) by blast
qed

lemma simplified-in:
  assumes simplified  $\psi$ 
  and  $N \in \psi$ 
  shows simplified  $\{N\}$ 
  using assms by (metis Set.set-insert empty-subsetI in-simplified-simplified insert-mono)

lemma subsumes-imp-formula:
  assumes  $\psi \leq \# \varphi$ 
  shows  $\{\psi\} \models_p \varphi$ 
  unfolding true-clss-cls-def apply auto
  using assms true-cls-mono-leD by blast

lemma simplified-imp-distinct-mset-tauto:
  assumes simp: simplified  $\psi'$ 
  shows distinct-mset-set  $\psi'$  and  $\forall \chi \in \psi'. \neg \text{tautology } \chi$ 
proof –
  show  $\forall \chi \in \psi'. \neg \text{tautology } \chi$ 
    using simp by (auto simp add: simplified-in simplified-not-tautology)

  show distinct-mset-set  $\psi'$ 
  proof (rule ccontr)
    assume  $\neg ?thesis$ 
    then obtain  $\chi$  where  $\chi \in \psi'$  and  $\neg \text{distinct-mset } \chi$  unfolding distinct-mset-set-def by auto
    then obtain  $L$  where  $\text{count } \chi L \geq 2$ 
      unfolding distinct-mset-def
      by (meson count-greater-eq-one-iff le-antisym simp simplified-count)
    then show False by (metis Suc-1  $\langle \chi \in \psi' \rangle$  not-less-eq-eq simp simplified-count)
    qed
  qed

lemma simplified-no-more-full1-simplified:
  assumes simplified  $\psi$ 
  shows  $\neg \text{full1 simplify } \psi \psi'$ 
  using assms unfolding full1-def by (meson tranclpD)

```

1.5 Resolution and Invariants

inductive *resolution* :: $'v \text{ state} \Rightarrow 'v \text{ state} \Rightarrow \text{bool}$ **where**
full1-simp: *full1 simplify* $N N' \Longrightarrow \text{resolution } (N, \text{already-used}) (N', \text{already-used}) \mid$

inferring: inference $(N, \text{already-used}) (N', \text{already-used}') \implies \text{simplified } N$
 $\implies \text{full simplify } N' N'' \implies \text{resolution } (N, \text{already-used}) (N'', \text{already-used}')$

1.5.1 Invariants

lemma *resolution-finite:*

assumes *resolution* $\psi \psi'$ **and** *finite* $(fst \psi)$
shows *finite* $(fst \psi')$
using *assms* **by** $(induct \text{ rule: resolution.induct})$
 $(auto \text{ simp add: full1-def full-def rtranclp-simplify-preserves-finite})$
 $dest: \text{trancpl-into-rtranclp inference-preserves-finite})$

lemma *rtranclp-resolution-finite:*

assumes *resolution*** $\psi \psi'$ **and** *finite* $(fst \psi)$
shows *finite* $(fst \psi')$
using *assms* **by** $(induct \text{ rule: rtranclp-induct, auto simp add: resolution-finite})$

lemma *resolution-finite-snd:*

assumes *resolution* $\psi \psi'$ **and** *finite* $(snd \psi)$
shows *finite* $(snd \psi')$
using *assms* **apply** $(induct \text{ rule: resolution.induct, auto simp add: inference-preserves-finite-snd})$
using *inference-preserves-finite-snd snd-conv* **by** *metis*

lemma *rtranclp-resolution-finite-snd:*

assumes *resolution*** $\psi \psi'$ **and** *finite* $(snd \psi)$
shows *finite* $(snd \psi')$
using *assms* **by** $(induct \text{ rule: rtranclp-induct, auto simp add: resolution-finite-snd})$

lemma *resolution-always-simplified:*

assumes *resolution* $\psi \psi'$
shows *simplified* $(fst \psi')$
using *assms* **by** $(induct \text{ rule: resolution.induct})$
 $(auto \text{ simp add: full1-def full-def})$

lemma *trancpl-resolution-always-simplified:*

assumes *trancpl resolution* $\psi \psi'$
shows *simplified* $(fst \psi')$
using *assms* **by** $(induct \text{ rule: trancpl.induct, auto simp add: resolution-always-simplified})$

lemma *resolution-atms-of:*

assumes *resolution* $\psi \psi'$ **and** *finite* $(fst \psi)$
shows *atms-of-ms* $(fst \psi') \subseteq \text{atms-of-ms } (fst \psi)$
using *assms* **apply** $(induct \text{ rule: resolution.induct})$
apply $(simp \text{ add: rtranclp-simplify-atms-of-ms trancpl-into-rtranclp full1-def })$
by $(metis \text{ (no-types, lifting) contra-subsetD fst-conv full-def})$
 $\text{inference-preserves-atms-of-ms rtranclp-simplify-atms-of-ms subsetI})$

lemma *rtranclp-resolution-atms-of:*

assumes *resolution*** $\psi \psi'$ **and** *finite* $(fst \psi)$
shows *atms-of-ms* $(fst \psi') \subseteq \text{atms-of-ms } (fst \psi)$
using *assms* **apply** $(induct \text{ rule: rtranclp-induct})$
using *resolution-atms-of rtranclp-resolution-finite* **by** *blast+*

lemma *resolution-include:*

assumes *res: resolution* $\psi \psi'$ **and** *finite: finite* $(fst \psi)$
shows $\text{fst } \psi' \subseteq \text{simple-clss } (\text{atms-of-ms } (fst \psi))$

proof –

have *finite'*: *finite* (*fst* ψ') **using** *local.finite res resolution-finite* **by** *blast*
have *simplified* (*fst* ψ') **using** *res finite' resolution-always-simplified* **by** *blast*
then have *fst* $\psi' \subseteq \text{simple-clss}$ (*atms-of-ms* (*fst* ψ'))
using *simplified-in-simple-clss finite' simplified-imp-distinct-mset-tauto*[*of fst* ψ'] **by** *auto*
moreover have *atms-of-ms* (*fst* ψ') \subseteq *atms-of-ms* (*fst* ψ)
using *res finite resolution-atms-of*[*of* ψ ψ'] **by** *auto*
ultimately show *?thesis* **by** (*meson atms-of-ms-finite local.finite order.trans rev-finite-subset*
simple-clss-mono)

qed

lemma *rtranclp-resolution-include*:

assumes *res*: *trancpl resolution* ψ ψ' **and** *finite*: *finite* (*fst* ψ)
shows *fst* $\psi' \subseteq \text{simple-clss}$ (*atms-of-ms* (*fst* ψ))
using *assms apply* (*induct rule: trancpl.induct*)
apply (*simp add: resolution-include*)
by (*meson simple-clss-mono order-class.le-trans resolution-include*
rtranclp-resolution-atms-of rtranclp-resolution-finite trancpl-into-rtranclp)

abbreviation *already-used-all-simple*

$:: ('a \text{ literal multiset} \times 'a \text{ literal multiset}) \text{ set} \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$ **where**

already-used-all-simple *already-used* *vars* \equiv

$(\forall (A, B) \in \text{already-used}. \text{simplified } \{A\} \wedge \text{simplified } \{B\} \wedge \text{atms-of } A \subseteq \text{vars} \wedge \text{atms-of } B \subseteq \text{vars})$

lemma *already-used-all-simple-vars-incl*:

assumes *vars* \subseteq *vars'*
shows *already-used-all-simple* *a vars* \implies *already-used-all-simple* *a vars'*
using *assms* **by** *fast*

lemma *inference-clause-preserves-already-used-all-simple*:

assumes *inference-clause* *S S'*
and *already-used-all-simple* (*snd* *S*) *vars*
and *simplified* (*fst* *S*)
and *atms-of-ms* (*fst* *S*) \subseteq *vars*
shows *already-used-all-simple* (*snd* (*fst* $S \cup \{\text{fst } S'\}$, *snd* S')) *vars*
using *assms*

proof (*induct rule: inference-clause.induct*)

case (*factoring* *L C N* *already-used*)

then show *?case* **by** (*simp add: simplified-in factoring-imp-simplify*)

next

case (*resolution* *P C N D* *already-used*) **note** *H = this*

show *?case* **apply** *clarify*

proof –

fix *A B v*

assume $(A, B) \in \text{snd} (\text{fst } (N, \text{already-used}))$

$\cup \{\text{fst } (C + D, \text{already-used} \cup \{(\{\#Pos P\# \} + C, \{\#Neg P\# \} + D)\}),$
 $\text{snd } (C + D, \text{already-used} \cup \{(\{\#Pos P\# \} + C, \{\#Neg P\# \} + D)\})\}$

then have $(A, B) \in \text{already-used} \vee (A, B) = (\{\#Pos P\# \} + C, \{\#Neg P\# \} + D)$ **by** *auto*

moreover {

assume $(A, B) \in \text{already-used}$

then have $\text{simplified } \{A\} \wedge \text{simplified } \{B\} \wedge \text{atms-of } A \subseteq \text{vars} \wedge \text{atms-of } B \subseteq \text{vars}$

using *H(4)* **by** *auto*

}

moreover {

assume *eq*: $(A, B) = (\{\#Pos P\# \} + C, \{\#Neg P\# \} + D)$

```

    then have simplified {A} using simplified-in H(1,5) by auto
    moreover have simplified {B} using eq simplified-in H(2,5) by auto
    moreover have atms-of A  $\subseteq$  atms-of-ms N
      using eq H(1)
      using atms-of-atms-of-ms-mono[of A N] by auto
    moreover have atms-of B  $\subseteq$  atms-of-ms N
      using eq H(2) atms-of-atms-of-ms-mono[of B N] by auto
    ultimately have simplified {A}  $\wedge$  simplified {B}  $\wedge$  atms-of A  $\subseteq$  vars  $\wedge$  atms-of B  $\subseteq$  vars
      using H(6) by auto
  }
  ultimately show simplified {A}  $\wedge$  simplified {B}  $\wedge$  atms-of A  $\subseteq$  vars  $\wedge$  atms-of B  $\subseteq$  vars
    by fast
qed

```

```

lemma inference-preserves-already-used-all-simple:
  assumes inference S S'
  and already-used-all-simple (snd S) vars
  and simplified (fst S)
  and atms-of-ms (fst S)  $\subseteq$  vars
  shows already-used-all-simple (snd S') vars
  using assms
proof (induct rule: inference.induct)
  case (inference-step S clause already-used)
  then show ?case
    using inference-clause-preserves-already-used-all-simple[of S (clause, already-used) vars]
    by auto
qed

```

```

lemma already-used-all-simple-inv:
  assumes resolution S S'
  and already-used-all-simple (snd S) vars
  and atms-of-ms (fst S)  $\subseteq$  vars
  shows already-used-all-simple (snd S') vars
  using assms
proof (induct rule: resolution.induct)
  case (full1-simp N N')
  then show ?case by simp
next
  case (inferring N already-used N' already-used' N'')
  then show already-used-all-simple (snd (N'', already-used')) vars
    using inference-preserves-already-used-all-simple[of (N, already-used)] by simp
qed

```

```

lemma rtrancplp-already-used-all-simple-inv:
  assumes resolution** S S'
  and already-used-all-simple (snd S) vars
  and atms-of-ms (fst S)  $\subseteq$  vars
  and finite (fst S)
  shows already-used-all-simple (snd S') vars
  using assms
proof (induct rule: rtrancplp-induct)
  case base
  then show ?case by simp
next

```


case (*step* $S' S''$) **note** $\text{infstar} = \text{this}(1)$ **and** $\text{IH} = \text{this}(3)$ **and** $\text{res} = \text{this}(2)$ **and**
 $\text{already} = \text{this}(4)$ **and** $\text{atms} = \text{this}(5)$ **and** $\text{finite} = \text{this}(6)$
have *already-used-all-simple* ($\text{snd } S'$) *vars* **using** IH *already* *atms* *finite* **by** *simp*
moreover **have** $\text{atms-of-ms } (\text{fst } S') \subseteq \text{atms-of-ms } (\text{fst } S)$
by (*simp* *add*: *infstar* *local.finite* *rtrancpl-resolution-atms-of*)
then **have** $\text{atms-of-ms } (\text{fst } S') \subseteq \text{vars}$ **using** *atms* **by** *auto*
ultimately **show** *?case*
using *already-used-all-simple-inv*[*OF res*] **by** *simp*
qed

lemma *inference-clause-simplified-already-used-subset*:
assumes *inference-clause* $S S'$
and *simplified* ($\text{fst } S$)
shows $\text{snd } S \subset \text{snd } S'$
using *assms* **apply** (*induct* *rule*: *inference-clause.induct*, *auto*)
using *factoring-imp-simplify* **by** *blast*

lemma *inference-simplified-already-used-subset*:
assumes *inference* $S S'$
and *simplified* ($\text{fst } S$)
shows $\text{snd } S \subset \text{snd } S'$
using *assms* **apply** (*induct* *rule*: *inference.induct*)
by (*metis* *inference-clause-simplified-already-used-subset* *snd-conv*)

lemma *resolution-simplified-already-used-subset*:
assumes *resolution* $S S'$
and *simplified* ($\text{fst } S$)
shows $\text{snd } S \subset \text{snd } S'$
using *assms* **apply** (*induct* *rule*: *resolution.induct*, *simp-all* *add*: *full1-def*)
apply (*meson* *trancplD*)
by (*metis* *inference-simplified-already-used-subset* *fst-conv* *snd-conv*)

lemma *trancpl-resolution-simplified-already-used-subset*:
assumes *trancpl* *resolution* $S S'$
and *simplified* ($\text{fst } S$)
shows $\text{snd } S \subset \text{snd } S'$
using *assms* **apply** (*induct* *rule*: *trancpl.induct*)
using *resolution-simplified-already-used-subset* **apply** *metis*
by (*meson* *trancpl-resolution-always-simplified* *resolution-simplified-already-used-subset* *less-trans*)

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

lemma *already-used-all-simple-in-already-used-top*:
assumes *already-used-all-simple* s *vars* **and** *finite* *vars*
shows $s \subseteq \text{already-used-top vars}$
proof
fix x
assume $x-s: x \in s$
obtain $A B$ **where** $x: x = (A, B)$ **by** (*cases* x , *auto*)
then **have** *simplified* $\{A\}$ **and** $\text{atms-of } A \subseteq \text{vars}$ **using** *assms*(1) $x-s$ **by** *fastforce*+
then **have** $A: A \in \text{simple-clss vars}$
using *simple-clss-mono*[*of* *atms-of* A *vars*] x *assms*(2)
simplified-imp-distinct-mset-tauto[*of* $\{A\}$]
distinct-mset-not-tautology-implies-in-simple-clss **by** *fast*

moreover have *simplified* $\{B\}$ **and** *atms-of* $B \subseteq \text{vars}$ **using** *assms*(1) $x-s$ x **by** *fast+*
then have $B: B \in \text{simple-clss vars}$
using *simplified-imp-distinct-mset-tauto*[of $\{B\}$]
distinct-mset-not-tautology-implies-in-simple-clss
simple-clss-mono[of *atms-of* B *vars*] x *assms*(2) **by** *fast*
ultimately show $x \in \text{simple-clss vars} \times \text{simple-clss vars}$
unfolding x **by** *auto*
qed

lemma *already-used-top-finite*:
assumes *finite vars*
shows *finite* (*already-used-top vars*)
using *simple-clss-finite assms* **by** *auto*

lemma *already-used-top-increasing*:
assumes $\text{var} \subseteq \text{var}'$ **and** *finite var'*
shows *already-used-top* $\text{var} \subseteq \text{already-used-top var}'$
using *assms simple-clss-mono* **by** *auto*

lemma *already-used-all-simple-finite*:
fixes $s :: ('a \text{ literal multiset} \times 'a \text{ literal multiset}) \text{ set}$ **and** $\text{vars} :: 'a \text{ set}$
assumes *already-used-all-simple* s *vars* **and** *finite vars*
shows *finite s*
using *assms already-used-all-simple-in-already-used-top*[*OF* *assms*(1)]
rev-finite-subset[*OF* *already-used-top-finite*[of *vars*]] **by** *auto*

abbreviation *card-simple vars* $\psi \equiv \text{card} (\text{already-used-top vars} - \psi)$

lemma *resolution-card-simple-decreasing*:
assumes *res: resolution* $\psi \psi'$
and *a-u-s: already-used-all-simple* (*snd* ψ) *vars*
and *finite-v: finite vars*
and *finite-fst: finite* (*fst* ψ)
and *finite-snd: finite* (*snd* ψ)
and *simp: simplified* (*fst* ψ)
and *atms-of-ms* (*fst* ψ) $\subseteq \text{vars}$
shows *card-simple vars* (*snd* ψ') $<$ *card-simple vars* (*snd* ψ)

proof –

let $?vars = \text{vars}$
let $?top = \text{simple-clss } ?vars \times \text{simple-clss } ?vars$
have 1: *card-simple vars* (*snd* ψ) $= \text{card } ?top - \text{card} (\text{snd } \psi)$
using *card-Diff-subset finite-snd already-used-all-simple-in-already-used-top*[*OF* *a-u-s*]
finite-v **by** *metis*
have *a-u-s': already-used-all-simple* (*snd* ψ') *vars*
using *already-used-all-simple-inv res a-u-s assms*(7) **by** *blast*
have *f: finite* (*snd* ψ') **using** *already-used-all-simple-finite a-u-s' finite-v* **by** *auto*
have 2: *card-simple vars* (*snd* ψ') $= \text{card } ?top - \text{card} (\text{snd } \psi')$
using *card-Diff-subset*[*OF* *f*] *already-used-all-simple-in-already-used-top*[*OF* *a-u-s' finite-v*]
by *auto*
have *card* (*already-used-top vars*) $\geq \text{card} (\text{snd } \psi')$
using *already-used-all-simple-in-already-used-top*[*OF* *a-u-s' finite-v*]
card-mono[of *already-used-top vars snd* ψ'] *already-used-top-finite*[*OF* *finite-v*] **by** *metis*
then show *?thesis*
using *psubset-card-mono*[*OF* *f resolution-simplified-already-used-subset*[*OF* *res simp*]]
unfolding 1 2 **by** *linarith*

qed

lemma *trancpl-resolution-card-simple-decreasing:*

assumes *trancpl resolution $\psi \ \psi'$ and finite-fst: finite (fst ψ)*
and *already-used-all-simple (snd ψ) vars*
and *atms-of-ms (fst ψ) \subseteq vars*
and *finite-v: finite vars*
and *finite-snd: finite (snd ψ)*
and *simplified (fst ψ)*
shows *card-simple vars (snd ψ') < card-simple vars (snd ψ)*
using *assms*

proof (*induct rule: trancpl-induct*)

case (*base ψ'*)

then show *?case by (simp add: resolution-card-simple-decreasing)*

next

case (*step $\psi' \ \psi''$*) **note** *res = this(1) and res' = this(2) and a-u-s = this(5) and*
atms = this(6) and f-v = this(7) and f-fst = this(4) and H = this

then have *card-simple vars (snd ψ') < card-simple vars (snd ψ) by auto*

moreover have *a-u-s': already-used-all-simple (snd ψ') vars*

using *rtrancpl-already-used-all-simple-inv[OF trancpl-into-rtrancpl[OF res] a-u-s atms f-fst] .*
have *finite (fst ψ')*

by (*meson finite-fst res rtrancpl-resolution-finite trancpl-into-rtrancpl*)

moreover have *finite (snd ψ') using already-used-all-simple-finite[OF a-u-s' f-v] .*

moreover have *simplified (fst ψ') using res trancpl-resolution-always-simplified by blast*

moreover have *atms-of-ms (fst ψ') \subseteq vars*

by (*meson atms f-fst order.trans res rtrancpl-resolution-atms-of trancpl-into-rtrancpl*)

ultimately show *?case*

using *resolution-card-simple-decreasing[OF res' a-u-s' f-v] f-v*
less-trans[of card-simple vars (snd ψ'') card-simple vars (snd ψ')
card-simple vars (snd ψ)]

by *blast*

qed

lemma *trancpl-resolution-card-simple-decreasing-2:*

assumes *trancpl resolution $\psi \ \psi'$*

and *finite-fst: finite (fst ψ)*

and *empty-snd: snd ψ = {}*

and *simplified (fst ψ)*

shows *card-simple (atms-of-ms (fst ψ)) (snd ψ') < card-simple (atms-of-ms (fst ψ)) (snd ψ)*

proof –

let *?vars = (atms-of-ms (fst ψ))*

have *already-used-all-simple (snd ψ) ?vars unfolding empty-snd by auto*

moreover have *atms-of-ms (fst ψ) \subseteq ?vars by auto*

moreover have *finite-v: finite ?vars using finite-fst by auto*

moreover have *finite-snd: finite (snd ψ) unfolding empty-snd by auto*

ultimately show *?thesis*

using *assms(1,2,4) trancpl-resolution-card-simple-decreasing[of $\psi \ \psi'$] by presburger*

qed

1.5.2 well-foundness if the relation

lemma *wf-simplified-resolution:*

assumes *f-vars: finite vars*

shows *wf {(y:: 'v:: linorder state, x). (atms-of-ms (fst x) \subseteq vars \wedge simplified (fst x))}*

$\wedge \text{finite } (\text{snd } x) \wedge \text{finite } (\text{fst } x) \wedge \text{already-used-all-simple } (\text{snd } x) \text{ vars} \wedge \text{resolution } x y\}$
proof –
{
 fix $a b :: 'v::\text{linorder state}$
 assume $(b, a) \in \{(y, x). (\text{atms-of-ms } (\text{fst } x) \subseteq \text{vars} \wedge \text{simplified } (\text{fst } x) \wedge \text{finite } (\text{snd } x) \wedge \text{finite } (\text{fst } x) \wedge \text{already-used-all-simple } (\text{snd } x) \text{ vars}) \wedge \text{resolution } x y\}$
 then have
 $\text{atms-of-ms } (\text{fst } a) \subseteq \text{vars}$ **and**
 $\text{simp: simplified } (\text{fst } a)$ **and**
 $\text{finite } (\text{snd } a)$ **and**
 $\text{finite } (\text{fst } a)$ **and**
 $a\text{-u-v: already-used-all-simple } (\text{snd } a) \text{ vars}$ **and**
 $\text{res: resolution } a b$ **by auto**
 have $\text{finite } (\text{already-used-top vars})$ **using** $f\text{-vars already-used-top-finite}$ **by blast**
 moreover have $\text{already-used-top vars} \subseteq \text{already-used-top vars}$ **by auto**
 moreover have $\text{snd } b \subseteq \text{already-used-top vars}$
 using $\text{already-used-all-simple-in-already-used-top}[of \text{snd } b \text{ vars}]$
 $a\text{-u-v already-used-all-simple-inv}[OF \text{res}] \langle \text{finite } (\text{fst } a) \rangle \langle \text{atms-of-ms } (\text{fst } a) \subseteq \text{vars} \rangle f\text{-vars}$
 by presburger
 moreover have $\text{snd } a \subset \text{snd } b$ **using** $\text{resolution-simplified-already-used-subset}[OF \text{res simp}]$.
 ultimately have $\text{finite } (\text{already-used-top vars}) \wedge \text{already-used-top vars} \subseteq \text{already-used-top vars}$
 $\wedge \text{snd } b \subseteq \text{already-used-top vars} \wedge \text{snd } a \subset \text{snd } b$ **by metis**
}
then show $?thesis$ **using** $\text{wf-bounded-set}[of \{(y:: 'v:: \text{linorder state}, x). (\text{atms-of-ms } (\text{fst } x) \subseteq \text{vars} \wedge \text{simplified } (\text{fst } x) \wedge \text{finite } (\text{snd } x) \wedge \text{finite } (\text{fst } x) \wedge \text{already-used-all-simple } (\text{snd } x) \text{ vars}) \wedge \text{resolution } x y\} \lambda\text{-. already-used-top vars snd}]$ **by auto**
qed

lemma *wf-simplified-resolution'*:

assumes $f\text{-vars: finite vars}$
shows $\text{wf } \{(y:: 'v:: \text{linorder state}, x). (\text{atms-of-ms } (\text{fst } x) \subseteq \text{vars} \wedge \neg \text{simplified } (\text{fst } x) \wedge \text{finite } (\text{snd } x) \wedge \text{finite } (\text{fst } x) \wedge \text{already-used-all-simple } (\text{snd } x) \text{ vars}) \wedge \text{resolution } x y\}$
unfolding wf-def
apply $(\text{simp add: resolution-always-simplified})$
by $(\text{metis } (\text{mono-tags, hide-lams}) \text{fst-conv resolution-always-simplified})$

lemma *wf-resolution*:

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

proof –

have $\text{Domain } ?R \text{ Int Range } ?S = \{\}$ **using** $\text{resolution-always-simplified}$ **by auto blast**
then show $\text{wf } (?R \cup ?S)$
 using $\text{wf-simplified-resolution}[OF f\text{-vars}] \text{wf-simplified-resolution}'[OF f\text{-vars}] \text{wf-Un}[of ?R ?S]$
 by fast

qed

lemma *rtrancp-simplify-already-used-inv*:

assumes $\text{simplify}^{**} S S'$
and $\text{already-used-inv } (S, N)$
shows $\text{already-used-inv } (S', N)$
using assms **apply** induction

```

using simplify-preserves-already-used-inv by fast+

lemma full1-simplify-already-used-inv:
  assumes full1 simplify S S'
  and already-used-inv (S, N)
  shows already-used-inv (S', N)
  using assms tranclp-into-rtranclp[of simplify S S'] rtranclp-simplify-already-used-inv
  unfolding full1-def by fast

lemma full-simplify-already-used-inv:
  assumes full simplify S S'
  and already-used-inv (S, N)
  shows already-used-inv (S', N)
  using assms rtranclp-simplify-already-used-inv unfolding full-def by fast
lemma resolution-already-used-inv:
  assumes resolution S S'
  and already-used-inv S
  shows already-used-inv S'
  using assms
proof induction
  case (full1-simp N N' already-used)
  then show ?case using full1-simplify-already-used-inv by fast
next
  case (inferring N already-used N' already-used' N'') note inf = this(1) and full = this(3) and
    a-u-v = this(4)
  then show ?case
    using inference-preserves-already-used-inv[OF inf a-u-v] full-simplify-already-used-inv full
    by fast
qed

lemma rtranclp-resolution-already-used-inv:
  assumes resolution** S S'
  and already-used-inv S
  shows already-used-inv S'
  using assms apply induction
  using resolution-already-used-inv by fast+

lemma rtanclp-simplify-preserves-unsat:
  assumes simplify**  $\psi$   $\psi'$ 
  shows satisfiable  $\psi' \longrightarrow$  satisfiable  $\psi$ 
  using assms apply induction
  using simplify-clause-preserves-sat by blast+

lemma full1-simplify-preserves-unsat:
  assumes full1 simplify  $\psi$   $\psi'$ 
  shows satisfiable  $\psi' \longrightarrow$  satisfiable  $\psi$ 
  using assms rtanclp-simplify-preserves-unsat[of  $\psi$   $\psi'$ ] tranclp-into-rtranclp
  unfolding full1-def by metis

lemma full-simplify-preserves-unsat:
  assumes full simplify  $\psi$   $\psi'$ 
  shows satisfiable  $\psi' \longrightarrow$  satisfiable  $\psi$ 
  using assms rtanclp-simplify-preserves-unsat[of  $\psi$   $\psi'$ ] unfolding full-def by metis

lemma resolution-preserves-unsat:

```

assumes *resolution* $\psi \psi'$
shows *satisfiable* $(fst \psi') \longrightarrow satisfiable (fst \psi)$
using *assms* **apply** (*induct rule: resolution.induct*)
using *full1-simplify-preserves-unsat* **apply** (*metis fst-conv*)
using *full-simplify-preserves-unsat* *simplify-preserves-unsat* **by** *fastforce*

lemma *rtrancp-resolution-preserves-unsat*:
assumes *resolution*** $\psi \psi'$
shows *satisfiable* $(fst \psi') \longrightarrow satisfiable (fst \psi)$
using *assms* **apply** *induction*
using *resolution-preserves-unsat* **by** *fast+*

lemma *rtrancp-simplify-preserve-partial-tree*:
assumes *simplify*** $N N'$
and *partial-interps* $t I N$
shows *partial-interps* $t I N'$
using *assms* **apply** (*induction, simp*)
using *simplify-preserve-partial-tree* **by** *metis*

lemma *full1-simplify-preserve-partial-tree*:
assumes *full1 simplify* $N N'$
and *partial-interps* $t I N$
shows *partial-interps* $t I N'$
using *assms* *rtrancp-simplify-preserve-partial-tree*[*of* $N N' t I$] *trancp-into-rtrancp*
unfolding *full1-def* **by** *fast*

lemma *full-simplify-preserve-partial-tree*:
assumes *full simplify* $N N'$
and *partial-interps* $t I N$
shows *partial-interps* $t I N'$
using *assms* *rtrancp-simplify-preserve-partial-tree*[*of* $N N' t I$] *trancp-into-rtrancp*
unfolding *full-def* **by** *fast*

lemma *resolution-preserve-partial-tree*:
assumes *resolution* $S S'$
and *partial-interps* $t I (fst S)$
shows *partial-interps* $t I (fst S')$
using *assms* **apply** *induction*
using *full1-simplify-preserve-partial-tree* *fst-conv* **apply** *metis*
using *full-simplify-preserve-partial-tree* *inference-preserve-partial-tree* **by** *fastforce*

lemma *rtrancp-resolution-preserve-partial-tree*:
assumes *resolution*** $S S'$
and *partial-interps* $t I (fst S)$
shows *partial-interps* $t I (fst S')$
using *assms* **apply** *induction*
using *resolution-preserve-partial-tree* **by** *fast+*
thm *nat-less-induct* *nat.induct*

lemma *nat-ge-induct*[*case-names* $0 Suc$]:
assumes $P 0$
and $(\bigwedge n. (\bigwedge m. m < Suc\ n \implies P\ m) \implies P\ (Suc\ n))$
shows $P\ n$
using *assms* **apply** (*induct rule: nat-less-induct*)
by (*rename-tac* n , *case-tac* n) *auto*

```

lemma wf-always-more-step-False:
  assumes wf R
  shows  $(\forall x. \exists z. (z, x) \in R) \implies \text{False}$ 
  using assms unfolding wf-def by (meson Domain.DomainI assms wfE-min)

lemma finite-finite-mset-element-of-mset[simp]:
  assumes finite N
  shows finite  $\{f \varphi L \mid \varphi L. \varphi \in N \wedge L \in \# \varphi \wedge P \varphi L\}$ 
  using assms
proof (induction N rule: finite-induct)
  case empty
  show ?case by auto
next
  case (insert x N) note finite = this(1) and IH = this(3)
  have  $\{f \varphi L \mid \varphi L. (\varphi = x \vee \varphi \in N) \wedge L \in \# \varphi \wedge P \varphi L\} \subseteq \{f x L \mid L. L \in \# x \wedge P x L\}$ 
     $\cup \{f \varphi L \mid \varphi L. \varphi \in N \wedge L \in \# \varphi \wedge P \varphi L\}$  by auto
  moreover have finite  $\{f x L \mid L. L \in \# x\}$  by auto
  ultimately show ?case using IH finite-subset by fastforce
qed

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

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

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

interpretation sum-count-ge-2:
  folding  $(\lambda \varphi. \text{op} + (\text{msetsum } \{\# \text{count } \varphi L \mid L \in \# \varphi. 2 \leq \text{count } \varphi L \#\})) 0$ 
rewrites
  folding.F  $(\lambda \varphi. \text{op} + (\text{msetsum } \{\# \text{count } \varphi L \mid L \in \# \varphi. 2 \leq \text{count } \varphi L \#\})) 0 = \text{sum-count-ge-2}$ 
proof –
  show folding  $(\lambda \varphi. \text{op} + (\text{msetsum } (\text{image-mset } (\text{count } \varphi) \{\# L \in \# \varphi. 2 \leq \text{count } \varphi L \#\})))$ 
    by standard auto
  then interpret sum-count-ge-2:
    folding  $(\lambda \varphi. \text{op} + (\text{msetsum } \{\# \text{count } \varphi L \mid L \in \# \varphi. 2 \leq \text{count } \varphi L \#\})) 0 .$ 
  show folding.F  $(\lambda \varphi. \text{op} + (\text{msetsum } (\text{image-mset } (\text{count } \varphi) \{\# L \in \# \varphi. 2 \leq \text{count } \varphi L \#\}))) 0$ 
     $= \text{sum-count-ge-2}$  by (auto simp add: sum-count-ge-2-def)
qed

lemma finite-incl-le-setsum:
  finite (B::'a multiset set)  $\implies A \subseteq B \implies \Xi A \leq \Xi B$ 
proof (induction arbitrary:A rule: finite-induct)
  case empty

```

```

then show ?case by simp
next
case (insert a F) note finite = this(1) and aF = this(2) and IH = this(3) and AF = this(4)
show ?case
proof (cases a ∈ A)
  assume a ∉ A
  then have A ⊆ F using AF by auto
  then show ?case using IH[of A] by (simp add: aF local.finite)
next
assume aA: a ∈ A
then have A - {a} ⊆ F using AF by auto
then have  $\Xi (A - \{a\}) \leq \Xi F$  using IH by blast
then show ?case
proof -
  obtain nn :: nat ⇒ nat ⇒ nat where
     $\forall x0\ x1. (\exists v2. x0 = x1 + v2) = (x0 = x1 + nn\ x0\ x1)$ 
  by moura
  then have  $\Xi F = \Xi (A - \{a\}) + nn (\Xi F) (\Xi (A - \{a\}))$ 
  by (meson  $\langle \Xi (A - \{a\}) \leq \Xi F \rangle$  le-iff-add)
  then show ?thesis
  by (metis (no-types) le-iff-add aA aF add.assoc finite.insertI finite-subset
    insert.premis local.finite sum-count-ge-2.insert sum-count-ge-2.remove)
qed
qed
qed

```

lemma *simplify-finite-measure-decrease:*

simplify N N' ⇒ finite N ⇒ card N' + $\Xi N' < \text{card } N + \Xi N$

proof (*induction rule: simplify.induct*)

case (*tautology-deletion A P*) **note** *an = this(1)* **and** *fin = this(2)*

let $?N' = N - \{A + \{\#Pos\ P\# \} + \{\#Neg\ P\# \}\}$

have $\text{card } ?N' < \text{card } N$

by (*meson card-Diff1-less tautology-deletion.hyps tautology-deletion.premis*)

moreover have $?N' \subseteq N$ **by** *auto*

then have $\text{sum-count-ge-2 } ?N' \leq \text{sum-count-ge-2 } N$ **using** *finite-incl-le-setsum[OF fin]* **by** *blast*

ultimately show ?case **by** *linarith*

next

case (*condensation A L*) **note** *AN = this(1)* **and** *fin = this(2)*

let $?C' = A + \{\#L\#\}$

let $?C = A + \{\#L\#\} + \{\#L\#\}$

let $?N' = N - \{?C\} \cup \{?C'\}$

have $\text{card } ?N' \leq \text{card } N$

using *AN* **by** (*metis (no-types, lifting) Diff-subset Un-empty-right Un-insert-right card.remove*
card-insert-if card-mono fin finite-Diff order-refl)

moreover have $\Xi \{?C'\} < \Xi \{?C\}$

proof -

have *mset-decomp:*

$\{\# La \in \# A. (L = La \longrightarrow La \in \# A) \wedge (L \neq La \longrightarrow 2 \leq \text{count } A\ La)\#\}$

$= \{\# La \in \# A. L \neq La \wedge 2 \leq \text{count } A\ La\#\} +$

$\{\# La \in \# A. L = La \wedge \text{Suc } 0 \leq \text{count } A\ L\#\}$

by (*auto simp: multiset-eq-iff ac-simps*)

have *mset-decomp2:* $\{\# La \in \# A. L \neq La \longrightarrow 2 \leq \text{count } A\ La\#\} =$

$\{\# La \in \# A. L \neq La \wedge 2 \leq \text{count } A\ La\#\} + \text{replicate-mset } (\text{count } A\ L)\ L$

by (*auto simp: multiset-eq-iff*)

show ?thesis


```

    by (auto simp: mset-decomp mset-decomp2 filter-mset-eq ac-simps)
  qed
have  $\Xi \ ?N' < \Xi \ N$ 
proof cases
  assume  $a1: \ ?C' \in N$ 
  then show  $\ ?thesis$ 
  proof -
    have  $f2: \bigwedge m \ M. \ insert \ (m::'a \ literal \ multiset) \ (M - \{m\}) = M \cup \{m\} \vee m \notin M$ 
      using Un-empty-right insert-Diff by blast
    have  $f3: \bigwedge m \ M \ Ma. \ insert \ (m::'a \ literal \ multiset) \ M - insert \ m \ Ma = M - insert \ m \ Ma$ 
      by simp
    then have  $f4: \bigwedge M \ m. \ M - \{m::'a \ literal \ multiset\} = M \cup \{m\} \vee m \in M$ 
      using Diff-insert-absorb Un-empty-right by fastforce
    have  $f5: insert \ (A + \{\#L\# \} + \{\#L\# \}) \ N = N$ 
      using  $f3 \ f2$  Un-empty-right condensation.hyps insert-iff by fastforce
    have  $\bigwedge m \ M. \ insert \ (m::'a \ literal \ multiset) \ M = M \cup \{m\} \vee m \notin M$ 
      using  $f3 \ f2$  Un-empty-right add.right-neutral insert-iff by fastforce
    then have  $\Xi \ (N - \{A + \{\#L\# \} + \{\#L\# \}\}) < \Xi \ N$ 
      using  $f5 \ f4$  by (metis Un-empty-right  $\langle \Xi \ \{A + \{\#L\# \}\} < \Xi \ \{A + \{\#L\# \} + \{\#L\# \}\rangle$ 
        add.right-neutral add-diff-cancel-left' add-gr-0 diff-less fin finite.emptyI not-le
        sum-count-ge-2.empty sum-count-ge-2.insert-remove trans-le-add2)
    then show  $\ ?thesis$ 
      using  $f3 \ f2 \ a1$  by (metis (no-types) Un-empty-right Un-insert-right condensation.hyps
        insert-iff multi-self-add-other-not-self)
  qed
next
  assume  $\ ?C' \notin N$ 
  have mset-decomp:
     $\{\# \ La \in \# \ A. \ (L = La \longrightarrow Suc \ 0 \leq count \ A \ La) \wedge (L \neq La \longrightarrow 2 \leq count \ A \ La)\#\}$ 
     $= \{\# \ La \in \# \ A. \ L \neq La \wedge 2 \leq count \ A \ La\#\} +$ 
     $\{\# \ La \in \# \ A. \ L = La \wedge Suc \ 0 \leq count \ A \ L\#\}$ 
    by (auto simp: multiset-eq-iff ac-simps)
  have mset-decomp2:  $\{\# \ La \in \# \ A. \ L \neq La \longrightarrow 2 \leq count \ A \ La\# \} =$ 
     $\{\# \ La \in \# \ A. \ L \neq La \wedge 2 \leq count \ A \ La\#\} + replicate\text{-}mset \ (count \ A \ L) \ L$ 
    by (auto simp: multiset-eq-iff)

  show  $\ ?thesis$ 
    using  $\langle \Xi \ \{A + \{\#L\# \}\} < \Xi \ \{A + \{\#L\# \} + \{\#L\# \}\rangle$  condensation.hyps fin
    sum-count-ge-2.remove[of - A + \{\#L\# \} + \{\#L\# \}]  $\langle \ ?C' \notin N \rangle$ 
    by (auto simp: mset-decomp mset-decomp2 filter-mset-eq)
  qed
ultimately show  $\ ?case$  by linarith
next
case (subsumption  $A \ B$ ) note  $AN = this(1)$  and  $AB = this(2)$  and  $BN = this(3)$  and  $fin = this(4)$ 
have  $card \ (N - \{B\}) < card \ N$  using  $BN$  by (meson card-Diff1-less subsumption.prems)
moreover have  $\Xi \ (N - \{B\}) \leq \Xi \ N$ 
  by (simp add: Diff-subset finite-incl-le-setsum subsumption.prems)
ultimately show  $\ ?case$  by linarith
qed

lemma simplify-terminates:
  wf  $\{(N', N). \ finite \ N \wedge simplify \ N \ N'\}$ 
  using assms apply (rule wfP-if-measure[of finite simplify  $\lambda N. card \ N + \Xi \ N$ ])
  using simplify-finite-measure-decrease by blast

```

lemma *wf-terminates*:

assumes *wf r*

shows $\exists N'. (N', N) \in r^* \wedge (\forall N''. (N'', N') \notin r)$

proof –

let $?P = \lambda N. (\exists N'. (N', N) \in r^* \wedge (\forall N''. (N'', N') \notin r))$

have $(\forall x. (\forall y. (y, x) \in r \longrightarrow ?P y) \longrightarrow ?P x)$

proof *clarify*

fix *x*

assume $H: \forall y. (y, x) \in r \longrightarrow ?P y$

{ assume $\exists y. (y, x) \in r$

then obtain *y* **where** $y: (y, x) \in r$ **by** *blast*

then have $?P y$ **using** *H* **by** *blast*

then have $?P x$ **using** *y* **by** (*meson rtrancl.rtrancl-into-rtrancl*)

}

moreover {

assume $\neg(\exists y. (y, x) \in r)$

then have $?P x$ **by** *auto*

}

ultimately show $?P x$ **by** *blast*

qed

moreover have $(\forall x. (\forall y. (y, x) \in r \longrightarrow ?P y) \longrightarrow ?P x) \longrightarrow \text{All } ?P$

using *assms unfolding wf-def* **by** (*rule allE*)

ultimately have $\text{All } ?P$ **by** *blast*

then show $?P N$ **by** *blast*

qed

lemma *rtranclp-simplify-terminates*:

assumes *fin: finite N*

shows $\exists N'. \text{simplify}^{**} N N' \wedge \text{simplified } N'$

proof –

have $H: \{(N', N). \text{finite } N \wedge \text{simplify } N N'\} = \{(N', N). \text{simplify } N N' \wedge \text{finite } N\}$ **by** *auto*

then have $\text{wf: wf } \{(N', N). \text{simplify } N N' \wedge \text{finite } N\}$

using *simplify-terminates* **by** (*simp add: H*)

obtain N' **where** $N': (N', N) \in \{(b, a). \text{simplify } a b \wedge \text{finite } a\}^*$ **and**

more: $(\forall N''. (N'', N') \notin \{(b, a). \text{simplify } a b \wedge \text{finite } a\})$

using *Prop-Resolution.wf-terminates[OF wf, of N]* **by** *blast*

have $1: \text{simplify}^{**} N N'$

using N' **by** (*induction rule: rtrancl.induct*) *auto*

then have *finite N'* **using** *fin rtranclp-simplify-preserves-finite* **by** *blast*

then have $2: \forall N''. \neg \text{simplify } N' N''$ **using** *more* **by** *auto*

show *?thesis* **using** $1\ 2$ **by** *blast*

qed

lemma *finite-simplified-full1-simp*:

assumes *finite N*

shows $\text{simplified } N \vee (\exists N'. \text{full1 simplify } N N')$

using *rtranclp-simplify-terminates[OF assms]* **unfolding** *full1-def*

by (*metis Nitpick.rtranclp-unfold*)

lemma *finite-simplified-full-simp*:

assumes *finite N*

shows $\exists N'. \text{full simplify } N N'$

using *rtrancp-simplify-terminates*[*OF assms*] **unfolding** *full-def* **by** *metis*

lemma *can-decrease-tree-size-resolution:*

fixes $\psi :: 'v \text{ state}$ **and** $\text{tree} :: 'v \text{ sem-tree}$

assumes *finite* (*fst* ψ) **and** *already-used-inv* ψ

and *partial-interps* tree I (*fst* ψ)

and *simplified* (*fst* ψ)

shows $\exists (\text{tree}' :: 'v \text{ sem-tree}) \psi'. \text{resolution}^{**} \psi \psi' \wedge \text{partial-interps } \text{tree}' I (\text{fst } \psi')$

$\wedge (\text{sem-tree-size } \text{tree}' < \text{sem-tree-size } \text{tree} \vee \text{sem-tree-size } \text{tree} = 0)$

using *assms*

proof (*induct arbitrary: I rule: sem-tree-size*)

case (*bigger* xs I) **note** $IH = \text{this}(1)$ **and** $\text{finite} = \text{this}(2)$ **and** $a-u-i = \text{this}(3)$ **and** $\text{part} = \text{this}(4)$

and $\text{simp} = \text{this}(5)$

{ **assume** $\text{sem-tree-size } xs = 0$
then have $?case$ **using** part **by** *blast*
}

moreover {

assume $sn0: \text{sem-tree-size } xs > 0$

obtain ag ad v **where** $xs: xs = \text{Node } v$ ag ad **using** $sn0$ **by** (*cases* xs , *auto*)

{

assume $\text{sem-tree-size } ag = 0 \wedge \text{sem-tree-size } ad = 0$

then have $ag: ag = \text{Leaf}$ **and** $ad: ad = \text{Leaf}$ **by** (*cases* ag , *auto*, *cases* ad , *auto*)

then obtain χ χ' **where**

$\chi: \neg I \cup \{\text{Pos } v\} \models \chi$ **and**

$\text{tot}\chi: \text{total-over-m } (I \cup \{\text{Pos } v\}) \{\chi\}$ **and**

$\chi\psi: \chi \in \text{fst } \psi$ **and**

$\chi': \neg I \cup \{\text{Neg } v\} \models \chi'$ **and**

$\text{tot}\chi': \text{total-over-m } (I \cup \{\text{Neg } v\}) \{\chi'\}$ **and** $\chi'\psi: \chi' \in \text{fst } \psi$

using part **unfolding** xs **by** *auto*

have $\text{Pos}v: \text{Pos } v \notin \# \chi$ **using** χ **unfolding** *true-cls-def* *true-lit-def* **by** *auto*

have $\text{Neg}v: \text{Neg } v \notin \# \chi'$ **using** χ' **unfolding** *true-cls-def* *true-lit-def* **by** *auto*

{

assume $\text{Neg}\chi: \neg \text{Neg } v \in \# \chi$

then have $\neg I \models \chi$ **using** χ $\text{Pos}v$ **unfolding** *true-cls-def* *true-lit-def* **by** *auto*

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

using $\text{Pos}v$ $\text{Neg}\chi$ *atm-imp-pos-or-neg-lit* $\text{tot}\chi$ **unfolding** *total-over-m-def* *total-over-set-def*
by *fastforce*

ultimately have *partial-interps* $\text{Leaf } I$ (*fst* ψ)

and $\text{sem-tree-size } \text{Leaf} < \text{sem-tree-size } xs$

and $\text{resolution}^{**} \psi \psi$

unfolding xs **by** (*auto* *simp* *add: $\chi\psi$*)

}

moreover {

assume $\text{Pos}\chi: \neg \text{Pos } v \in \# \chi'$

then have $I\chi: \neg I \models \chi'$ **using** χ' $\text{Pos}v$ **unfolding** *true-cls-def* *true-lit-def* **by** *auto*

moreover have $\text{total-over-m } I \{\chi'\}$

using $\text{Neg}v$ $\text{Pos}\chi$ *atm-imp-pos-or-neg-lit* $\text{tot}\chi'$

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

ultimately have *partial-interps* $\text{Leaf } I$ (*fst* ψ)

and $\text{sem-tree-size } \text{Leaf} < \text{sem-tree-size } xs$

and $\text{resolution}^{**} \psi \psi$ **using** $\chi'\psi$ $I\chi$ **unfolding** xs **by** *auto*

}

```

moreover {
  assume  $neg: Neg\ v \in \# \chi$  and  $pos: Pos\ v \in \# \chi'$ 
  have  $count\ \chi\ (Neg\ v) = 1$ 
    using  $simplified-count[OF\ simp\ \chi\psi]$   $neg$ 
    by ( $simp\ add: dual-order.antisym$ )
  have  $count\ \chi'\ (Pos\ v) = 1$ 
    using  $simplified-count[OF\ simp\ \chi'\psi]$   $pos$ 
    by ( $simp\ add: dual-order.antisym$ )

  obtain  $C$  where  $\chi C: \chi = C + \{\#Neg\ v\# \}$  and  $negC: Neg\ v \notin \# C$  and  $posC: Pos\ v \notin \# C$ 
    by ( $metis\ (no-types,\ lifting)\ One-nat-def\ Posv\ Suc-eq-plus1-left\ \langle count\ \chi\ (Neg\ v) = 1 \rangle$ 
       $add-diff-cancel-left'\ count-diff\ count-greater-eq-one-iff\ count-single\ insert-DiffM$ 
       $insert-DiffM2\ less-numeral-extra(3)\ multi-member-skip\ not-le\ not-less-eq-eq$ )

  obtain  $C'$  where
     $\chi C': \chi' = C' + \{\#Pos\ v\# \}$  and
     $posC': Pos\ v \notin \# C'$  and
     $negC': Neg\ v \notin \# C'$ 
    by ( $metis\ (no-types,\ lifting)\ One-nat-def\ Negv\ Suc-eq-plus1-left\ \langle count\ \chi'\ (Pos\ v) = 1 \rangle$ 
       $add-diff-cancel-left'\ count-diff\ count-greater-eq-one-iff\ count-single\ insert-DiffM$ 
       $insert-DiffM2\ less-numeral-extra(3)\ multi-member-skip\ not-le\ not-less-eq-eq$ )

  have  $totC: total-over-m\ I\ \{C\}$ 
    using  $tot\chi\ tot-over-m-remove[of\ I\ Pos\ v\ C]\ negC\ posC$  unfolding  $\chi C$ 
    by ( $metis\ total-over-m-sum\ uminus-Neg\ uminus-of-uminus-id$ )
  have  $totC': total-over-m\ I\ \{C'\}$ 
    using  $tot\chi'\ total-over-m-sum\ tot-over-m-remove[of\ I\ Neg\ v\ C']\ negC'\ posC'$ 
    unfolding  $\chi C'$  by ( $metis\ total-over-m-sum\ uminus-Neg$ )
  have  $\neg I \models C + C'$ 
    using  $\chi\ \chi'\ \chi C\ \chi C'$  by  $auto$ 
  then have  $part-I-\psi''': partial-interps\ Leaf\ I\ (fst\ \psi \cup \{C + C'\})$ 
    using  $totC\ totC'\ \neg I \models C + C'$  by ( $metis\ Un-insert-right\ insertI1$ 
       $partial-interps.simps(1)\ total-over-m-sum$ )
  {
    assume  $(\{\#Pos\ v\# \} + C', \{\#Neg\ v\# \} + C) \notin snd\ \psi$ 
    then have  $inf''': inference\ \psi\ (fst\ \psi \cup \{C + C'\}, snd\ \psi \cup \{(\chi', \chi)\})$ 
      by ( $metis\ \chi'\psi\ \chi C\ \chi C'\ \chi\psi\ add.commute\ inference-step\ prod.collapse\ resolution$ )
    obtain  $N'$  where  $full: full\ simplify\ (fst\ \psi \cup \{C + C'\})\ N'$ 
      by ( $metis\ finite-simplified-full-simp\ fst-conv\ inf''\ inference-preserves-finite$ 
         $local.finite$ )
    have  $resolution\ \psi\ (N', snd\ \psi \cup \{(\chi', \chi)\})$ 
      using  $resolution.intros(2)[OF\ -\ simp\ full,\ of\ snd\ \psi\ snd\ \psi \cup \{(\chi', \chi)\}]\ inf''$ 
      by ( $metis\ surjective-pairing$ )
    moreover have  $partial-interps\ Leaf\ I\ N'$ 
      using  $full-simplify-preserve-partial-tree[OF\ full\ part-I-\psi''']$  .
    moreover have  $sem-tree-size\ Leaf < sem-tree-size\ xs$  unfolding  $xs$  by  $auto$ 
    ultimately have  $?case$ 
      by ( $metis\ (no-types)\ prod.sel(1)\ rtranclp.rtrancl-into-rtrancl\ rtranclp.rtrancl-refl$ )
  }
}
moreover {
  assume  $a: (\{\#Pos\ v\# \} + C', \{\#Neg\ v\# \} + C) \in snd\ \psi$ 
  then have  $(\exists \chi \in fst\ \psi. (\forall I. total-over-m\ I\ \{C+C'\} \longrightarrow total-over-m\ I\ \{\chi\})$ 
     $\wedge (\forall I. total-over-m\ I\ \{\chi\} \longrightarrow I \models \chi \longrightarrow I \models C' + C)) \vee tautology\ (C' + C)$ 
  proof -
    obtain  $p$  where  $p: Pos\ p \in \# (\{\#Pos\ v\# \} + C') \wedge Neg\ p \in \# (\{\#Neg\ v\# \} + C)$ 

```

$\wedge ((\exists \chi \in \text{fst } \psi. (\forall I. \text{total-over-m } I \{(\{\#Pos \ v\# \} + C') - \{\#Pos \ p\# \} + ((\{\#Neg \ v\# \} + C) - \{\#Neg \ p\# \})) \longrightarrow \text{total-over-m } I \{\chi\}) \wedge (\forall I. \text{total-over-m } I \{\chi\} \longrightarrow I \models \chi \longrightarrow I \models (\{\#Pos \ v\# \} + C') - \{\#Pos \ p\# \} + ((\{\#Neg \ v\# \} + C) - \{\#Neg \ p\# \}))) \vee \text{tautology } ((\{\#Pos \ v\# \} + C') - \{\#Pos \ p\# \} + ((\{\#Neg \ v\# \} + C) - \{\#Neg \ p\# \})))$

using *a* **by** (*blast intro: allE[OF a-u-i[unfolded subsumes-def Ball-def],*
of $(\{\#Pos \ v\# \} + C', \{\#Neg \ v\# \} + C))$

{ assume $p \neq v$
then have $Pos \ p \in \# \ C' \wedge Neg \ p \in \# \ C$ **using** *p* **by** *force*
then have *?thesis* **by** *auto*
}

moreover {
assume $p = v$
then have *?thesis* **using** *p* **by** (*metis add.commute add-diff-cancel-left')*
}

ultimately show *?thesis* **by** *auto*
qed

moreover {
assume $\exists \chi \in \text{fst } \psi. (\forall I. \text{total-over-m } I \{C+C'\} \longrightarrow \text{total-over-m } I \{\chi\})$
 $\wedge (\forall I. \text{total-over-m } I \{\chi\} \longrightarrow I \models \chi \longrightarrow I \models C' + C)$
then obtain ϑ **where**
 $\vartheta: \vartheta \in \text{fst } \psi$ **and**
 $\text{tot-}\vartheta\text{-}CC': \forall I. \text{total-over-m } I \{C+C'\} \longrightarrow \text{total-over-m } I \{\vartheta\}$ **and**
 $\vartheta\text{-inv}: \forall I. \text{total-over-m } I \{\vartheta\} \longrightarrow I \models \vartheta \longrightarrow I \models C' + C$ **by** *blast*
have *partial-interps Leaf I (fst ψ)*
using *tot-}\vartheta\text{-}CC' \vartheta \vartheta\text{-inv totC totC' } \hookrightarrow I \models C + C'* *total-over-m-sum* **by** *fastforce*
moreover have *sem-tree-size Leaf < sem-tree-size xs* **unfolding** *xs* **by** *auto*
ultimately have *?case* **by** *blast*
}

moreover {
assume *tautCC'*: *tautology* $(C' + C)$
have *total-over-m I {C'+C}* **using** *totC totC' total-over-m-sum* **by** *auto*
then have $\neg \text{tautology } (C' + C)$
using $\hookrightarrow I \models C + C'$ **unfolding** *add.commute[of C C'] total-over-m-def*
unfolding *tautology-def* **by** *auto*
then have *False* **using** *tautCC'* **unfolding** *tautology-def* **by** *auto*
}

ultimately have *?case* **by** *auto*
}

ultimately have *?case* **by** *auto*
}

ultimately have *?case* **using** *part* **by** (*metis (no-types) sem-tree-size.simps(1)*)
}

moreover {
assume *size-ag: sem-tree-size ag > 0*
have *sem-tree-size ag < sem-tree-size xs* **unfolding** *xs* **by** *auto*
moreover have *partial-interps ag (I ∪ {Pos v}) (fst ψ)*
and *partad: partial-interps ad (I ∪ {Neg v}) (fst ψ)*
using *part partial-interps.simps(2)* **unfolding** *xs* **by** *metis+*
moreover
have *sem-tree-size ag < sem-tree-size xs* \implies *finite (fst ψ)* \implies *already-used-inv ψ*
 \implies *partial-interps ag (I ∪ {Pos v}) (fst ψ)* \implies *simplified (fst ψ)*
 $\implies \exists \text{tree}' \ \psi'. \text{resolution}^* \ \psi \ \psi' \wedge \text{partial-interps tree}' (I \cup \{\text{Pos } v\}) (\text{fst } \psi')$
 $\wedge (\text{sem-tree-size tree}' < \text{sem-tree-size ag} \vee \text{sem-tree-size ag} = 0)$
using *IH[of ag I ∪ {Pos v}]* **by** *auto*
ultimately obtain $\psi' :: 'v \text{ state}$ **and** $\text{tree}' :: 'v \text{ sem-tree}$ **where**

```

  inf: resolution**  $\psi \psi'$ 
  and part: partial-interps tree' ( $I \cup \{Pos\ v\}$ ) (fst  $\psi'$ )
  and size: sem-tree-size tree' < sem-tree-size ag  $\vee$  sem-tree-size ag = 0
  using finite part rtranclp.rtrancl-refl a-u-i simp by blast

  have partial-interps ad ( $I \cup \{Neg\ v\}$ ) (fst  $\psi'$ )
    using rtranclp-resolution-preserve-partial-tree inf partad by fast
  then have partial-interps (Node v tree' ad) I (fst  $\psi'$ ) using part by auto
  then have ?case using inf size size-ag part unfolding xs by fastforce
}
moreover {
  assume size-ad: sem-tree-size ad > 0
  have sem-tree-size ad < sem-tree-size xs unfolding xs by auto
  moreover
    have
      partag: partial-interps ag ( $I \cup \{Pos\ v\}$ ) (fst  $\psi$ ) and
      partial-interps ad ( $I \cup \{Neg\ v\}$ ) (fst  $\psi$ )
      using part partial-interps.simps(2) unfolding xs by metis+
  moreover have sem-tree-size ad < sem-tree-size xs  $\longrightarrow$  finite (fst  $\psi$ )  $\longrightarrow$  already-used-inv  $\psi$ 
     $\longrightarrow$  ( partial-interps ad ( $I \cup \{Neg\ v\}$ ) (fst  $\psi$ )  $\longrightarrow$  simplified (fst  $\psi$ )
     $\longrightarrow$  ( $\exists$  tree'  $\psi'$ . resolution**  $\psi \psi' \wedge$  partial-interps tree' ( $I \cup \{Neg\ v\}$ ) (fst  $\psi'$ )
     $\wedge$  (sem-tree-size tree' < sem-tree-size ad  $\vee$  sem-tree-size ad = 0)))
    using IH by blast
  ultimately obtain  $\psi' :: 'v$  state and tree' :: 'v sem-tree where
    inf: resolution**  $\psi \psi'$ 
    and part: partial-interps tree' ( $I \cup \{Neg\ v\}$ ) (fst  $\psi'$ )
    and size: sem-tree-size tree' < sem-tree-size ad  $\vee$  sem-tree-size ad = 0
    using finite part rtranclp.rtrancl-refl a-u-i simp by blast

  have partial-interps ag ( $I \cup \{Pos\ v\}$ ) (fst  $\psi'$ )
    using rtranclp-resolution-preserve-partial-tree inf partag by fast
  then have partial-interps (Node v ag tree') I (fst  $\psi'$ ) using part by auto
  then have ?case using inf size size-ad unfolding xs by fastforce
}
ultimately have ?case by auto
}
ultimately show ?case by auto
qed

```

lemma resolution-completeness-inv:

```

fixes  $\psi :: 'v :: linorder$  state
assumes
  unsat:  $\neg$ satisfiable (fst  $\psi$ ) and
  finite: finite (fst  $\psi$ ) and
  a-u-v: already-used-inv  $\psi$ 
shows  $\exists \psi'. (resolution^{**} \psi \psi' \wedge \{\#\} \in fst \psi')$ 
proof -
  obtain tree where partial-interps tree {} (fst  $\psi$ )
  using partial-interps-build-sem-tree-atms assms by metis
  then show ?thesis
  using unsat finite a-u-v
  proof (induct tree arbitrary:  $\psi$  rule: sem-tree-size)
    case (bigger tree  $\psi$ ) note H = this
    {
      fix  $\chi$ 

```

```

assume tree: tree = Leaf
obtain  $\chi$  where  $\chi$ :  $\neg \{\}$   $\models \chi$  and tot $\chi$ : total-over-m  $\{\}$   $\{\chi\}$  and  $\chi\psi$ :  $\chi \in \text{fst } \psi$ 
  using H unfolding tree by auto
moreover have  $\{\#\} = \chi$ 
  using H atms-empty-iff-empty tot $\chi$ 
  unfolding true-cls-def total-over-m-def total-over-set-def by fastforce
moreover have resolution**  $\psi$   $\psi$  by auto
ultimately have ?case by metis
}
moreover {
  fix v tree1 tree2
  assume tree: tree = Node v tree1 tree2
  obtain  $\psi_0$  where  $\psi_0$ : resolution**  $\psi$   $\psi_0$  and simp: simplified (fst  $\psi_0$ )
  proof –
    { assume simplified (fst  $\psi$ )
      moreover have resolution**  $\psi$   $\psi$  by auto
      ultimately have thesis using that by blast
    }
    moreover {
      assume  $\neg$ simplified (fst  $\psi$ )
      then have  $\exists \psi'. \text{full1 simplify } (\text{fst } \psi) \psi'$ 
        by (metis Nitpick.rtranclp-unfold bigger.premis(3) full1-def
          rtranclp-simplify-terminates)
      then obtain N where full1 simplify (fst  $\psi$ ) N by metis
      then have resolution  $\psi$  (N, snd  $\psi$ )
        using resolution.intros(1)[of fst  $\psi$  N snd  $\psi$ ] by auto
      moreover have simplified N
        using  $\langle \text{full1 simplify } (\text{fst } \psi) N \rangle$  unfolding full1-def by blast
      ultimately have ?thesis using that by force
    }
    ultimately show ?thesis by auto
  }
qed

have p: partial-interps tree  $\{\}$  (fst  $\psi_0$ )
and uns: unsatisfiable (fst  $\psi_0$ )
and f: finite (fst  $\psi_0$ )
and a-u-v: already-used-inv  $\psi_0$ 
  using  $\psi_0$  bigger.premis(1) rtranclp-resolution-preserve-partial-tree apply blast
  using  $\psi_0$  bigger.premis(2) rtranclp-resolution-preserves-unsat apply blast
  using  $\psi_0$  bigger.premis(3) rtranclp-resolution-finite apply blast
  using rtranclp-resolution-already-used-inv[OF  $\psi_0$  bigger.premis(4)] by blast
obtain tree'  $\psi'$  where
  inf: resolution**  $\psi_0$   $\psi'$  and
  part': partial-interps tree'  $\{\}$  (fst  $\psi'$ ) and
  decrease: sem-tree-size tree' < sem-tree-size tree  $\vee$  sem-tree-size tree = 0
  using can-decrease-tree-size-resolution[OF f a-u-v p simp] unfolding tautology-def
  by meson
have s: sem-tree-size tree' < sem-tree-size tree using decrease unfolding tree by auto
have fin: finite (fst  $\psi'$ )
  using f inf rtranclp-resolution-finite by blast
have unsat: unsatisfiable (fst  $\psi'$ )
  using rtranclp-resolution-preserves-unsat inf uns by metis
have a-u-i': already-used-inv  $\psi'$ 
  using a-u-v inf rtranclp-resolution-already-used-inv[of  $\psi_0$   $\psi'$ ] by auto

```

```

      have ?case
      using inf rtrancpl-trans[of resolution]  $H(1)[OF\ s\ part'\ unsat\ fin\ a-u-i']\ \psi_0$  by blast
    }
    ultimately show ?case by (cases tree, auto)
  qed
qed

```

```

lemma resolution-preserves-already-used-inv:
  assumes resolution  $S\ S'$ 
  and already-used-inv  $S$ 
  shows already-used-inv  $S'$ 
  using assms
  apply (induct rule: resolution.induct)
  apply (rule full1-simplify-already-used-inv; simp)
  apply (rule full-simplify-already-used-inv, simp)
  apply (rule inference-preserves-already-used-inv, simp)
  apply blast
done

```

```

lemma rtrancpl-resolution-preserves-already-used-inv:
  assumes resolution**  $S\ S'$ 
  and already-used-inv  $S$ 
  shows already-used-inv  $S'$ 
  using assms
  apply (induct rule: rtrancpl-induct)
  apply simp
  using resolution-preserves-already-used-inv by fast

```

```

lemma resolution-completeness:
  fixes  $\psi :: 'v :: linorder\ state$ 
  assumes unsat:  $\neg\text{satisfiable}\ (fst\ \psi)$ 
  and finite:  $finite\ (fst\ \psi)$ 
  and snd  $\psi = \{\}$ 
  shows  $\exists\ \psi'.\ (resolution**\ \psi\ \psi' \wedge \{\#\} \in fst\ \psi')$ 
proof -
  have already-used-inv  $\psi$  unfolding assms by auto
  then show ?thesis using assms resolution-completeness-inv by blast
qed

```

```

lemma rtrancpl-preserves-sat:
  assumes simplify**  $S\ S'$ 
  and satisfiable  $S$ 
  shows satisfiable  $S'$ 
  using assms apply induction
  apply simp
  by (meson satisfiable-carac satisfiable-def simplify-preserves-un-sat-eq)

```

```

lemma resolution-preserves-sat:
  assumes resolution  $S\ S'$ 
  and satisfiable  $(fst\ S)$ 
  shows satisfiable  $(fst\ S')$ 
  using assms apply (induction rule: resolution.induct)
  using rtrancpl-preserves-sat trancpl-into-rtrancpl unfolding full1-def apply fastforce
  by (metis fst-conv full-def inference-preserves-un-sat rtrancpl-preserves-sat
    satisfiable-carac' satisfiable-def)

```



```

lemma rtrancpl-resolution-preserves-sat:
  assumes resolution**  $S\ S'$ 
  and satisfiable (fst  $S$ )
  shows satisfiable (fst  $S'$ )
  using assms apply (induction rule: rtrancpl-induct)
  apply simp
  using resolution-preserves-sat by blast

lemma resolution-soundness:
  fixes  $\psi :: 'v :: \text{linorder state}$ 
  assumes resolution**  $\psi\ \psi'$  and  $\{\#\} \in \text{fst } \psi'$ 
  shows unsatisfiable (fst  $\psi$ )
  using assms by (meson rtrancpl-resolution-preserves-sat satisfiable-def true-cls-empty
    true-clss-def)

lemma resolution-soundness-and-completeness:
  fixes  $\psi :: 'v :: \text{linorder state}$ 
  assumes finite: finite (fst  $\psi$ )
  and snd: snd  $\psi = \{\}$ 
  shows  $(\exists \psi'. (\text{resolution** } \psi\ \psi' \wedge \{\#\} \in \text{fst } \psi')) \longleftrightarrow \text{unsatisfiable } (\text{fst } \psi)$ 
  using assms resolution-completeness resolution-soundness by metis

lemma simplified-falsity:
  assumes simp: simplified  $\psi$ 
  and  $\{\#\} \in \psi$ 
  shows  $\psi = \{\{\#\}\}$ 
proof (rule ccontr)
  assume  $H: \neg ?thesis$ 
  then obtain  $\chi$  where  $\chi \in \psi$  and  $\chi \neq \{\#\}$  using assms(2) by blast
  then have  $\{\#\} \subsetneq \chi$  by (simp add: mset-less-empty-nonempty)
  then have simplify  $\psi$   $(\psi - \{\chi\})$ 
    using simplify.subsumption[OF assms(2)  $\langle \{\#\} \subsetneq \chi \rangle \langle \chi \in \psi \rangle$ ] by blast
  then show False using simp by blast
qed

lemma simplify-falsity-in-preserved:
  assumes simplify  $\chi s\ \chi s'$ 
  and  $\{\#\} \in \chi s$ 
  shows  $\{\#\} \in \chi s'$ 
  using assms
  by induction auto

lemma rtrancpl-simplify-falsity-in-preserved:
  assumes simplify**  $\chi s\ \chi s'$ 
  and  $\{\#\} \in \chi s$ 
  shows  $\{\#\} \in \chi s'$ 
  using assms
  by induction (auto intro: simplify-falsity-in-preserved)

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

```

```

proof
  assume ?B
  then show ?A by auto
next
  assume ?A
  then obtain  $\chi s$   $a-u-v$  where  $\chi s$ : resolution**  $\psi$  ( $\chi s$ ,  $a-u-v$ ) and  $F$ :  $\{\#\} \in \chi s$  by auto
  { assume simplified  $\chi s$ 
    then have ?B using simplified-falsity[ $OF - F$ ]  $\chi s$  by blast
  }
  moreover {
    assume  $\neg$  simplified  $\chi s$ 
    then obtain  $\chi s'$  where full1 simplify  $\chi s$   $\chi s'$ 
      by (metis  $\chi s$  assms finite-simplified-full1-simp fst-conv rtranclp-resolution-finite)
    then have  $\{\#\} \in \chi s'$ 
      unfolding full1-def by (meson  $F$  rtranclp-simplify-falsity-in-preserved
        trancplp-into-rtranclp)
    then have ?B
      by (metis  $\chi s$  (full1 simplify  $\chi s$   $\chi s'$ ) fst-conv full1-simp resolution-always-simplified
        rtranclp.rtrancl-into-rtrancl simplified-falsity)
  }
  ultimately show ?B by blast
qed

lemma resolution-soundness-and-completeness':
  fixes  $\psi :: 'v :: \text{linorder state}$ 
  assumes
    finite: finite (fst  $\psi$ ) and
    snd: snd  $\psi = \{\}$ 
  shows  $(\exists a-u-v. (\text{resolution** } \psi (\{\#\}, a-u-v))) \longleftrightarrow \text{unsatisfiable } (\text{fst } \psi)$ 
    using assms resolution-completeness resolution-soundness resolution-falsity-get-falsity-alone
    by metis

end
theory Prop-Superposition
imports Partial-Clausal-Logic ../lib/Herbrand-Interpretation
begin

```

2 Superposition

no-notation *Herbrand-Interpretation.true-cls* (**infix** \models 50)

notation *Herbrand-Interpretation.true-cls* (**infix** \models_h 50)

no-notation *Herbrand-Interpretation.true-clss* (**infix** \models_s 50)

notation *Herbrand-Interpretation.true-clss* (**infix** \models_{hs} 50)

lemma *herbrand-interp-iff-partial-interp-cls*:

$S \models_h C \longleftrightarrow \{Pos\ P|P. P \in S\} \cup \{Neg\ P|P. P \notin S\} \models C$

unfolding *Herbrand-Interpretation.true-cls-def Partial-Clausal-Logic.true-cls-def*
by *auto*

lemma *herbrand-consistent-interp*:

consistent-interp ($\{Pos\ P|P. P \in S\} \cup \{Neg\ P|P. P \notin S\}$)

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

lemma *herbrand-total-over-set*:

total-over-set ($\{\text{Pos } P \mid P. P \in S\} \cup \{\text{Neg } P \mid P. P \notin S\}$) *T*
unfolding *total-over-set-def* **by** *auto*

lemma *herbrand-total-over-m*:

total-over-m ($\{\text{Pos } P \mid P. P \in S\} \cup \{\text{Neg } P \mid P. P \notin S\}$) *T*
unfolding *total-over-m-def* **by** (*auto simp add: herbrand-total-over-set*)

lemma *herbrand-interp-iff-partial-interp-clss*:

$S \models_{hs} C \longleftrightarrow \{\text{Pos } P \mid P. P \in S\} \cup \{\text{Neg } P \mid P. P \notin S\} \models_s C$
unfolding *true-clss-def Ball-def herbrand-interp-iff-partial-interp-clss*
Partial-Clausal-Logic.true-clss-def **by** *auto*

definition *clss-lt* :: '*a*::wellorder clauses \Rightarrow '*a* clause \Rightarrow '*a* clauses **where**
clss-lt *N C* = $\{D \in N. D \# \subset \# C\}$

notation (*latex output*)

clss-lt ($-\hat{<}^{bsup} - \hat{<}^{esup}$)

locale *selection* =

fixes *S* :: '*a* clause \Rightarrow '*a* clause
assumes
S-selects-subseteq: $\bigwedge C. S C \leq \# C$ **and**
S-selects-neg-lits: $\bigwedge C L. L \in \# S C \implies \text{is-neg } L$

locale *ground-resolution-with-selection* =

selection *S* **for** *S* :: ('*a* :: wellorder) clause \Rightarrow '*a* clause

begin

context

fixes *N* :: '*a* clause set

begin

We do not create an equivalent of δ , but we directly defined N_C by inlining the definition.

function

production :: '*a* clause \Rightarrow '*a* interp

where

production *C* =
 $\{A. C \in N \wedge C \neq \{\#\} \wedge \text{Max } (\text{set-mset } C) = \text{Pos } A \wedge \text{count } C (\text{Pos } A) \leq 1$
 $\wedge \neg (\bigcup D \in \{D. D \# \subset \# C\}. \text{production } D) \models_h C \wedge S C = \{\#\}\}$

by *auto*

termination by (*relation* $\{(D, C). D \# \subset \# C\}$) (*auto simp: wf-less-multiset*)

declare *production.simps*[*simp del*]

definition *interp* :: '*a* clause \Rightarrow '*a* interp **where**

interp *C* = $(\bigcup D \in \{D. D \# \subset \# C\}. \text{production } D)$

lemma *production-unfold*:

production *C* = $\{A. C \in N \wedge C \neq \{\#\} \wedge \text{Max } (\text{set-mset } C) = \text{Pos } A \wedge \text{count } C (\text{Pos } A) \leq 1 \wedge \neg$
interp *C* $\models_h C \wedge S C = \{\#\}\}$

unfolding *interp-def* **by** (*rule production.simps*)

abbreviation *productive* *A* \equiv (*production* *A* $\neq \{\}$)

abbreviation *produces* :: '*a* clause \Rightarrow '*a* \Rightarrow bool **where**

$\text{produces } C \ A \equiv \text{production } C = \{A\}$

lemma *producesD*:

$\text{produces } C \ A \implies C \in N \wedge C \neq \{\#\} \wedge \text{Pos } A = \text{Max } (\text{set-mset } C) \wedge \text{count } C \ (\text{Pos } A) \leq 1 \wedge$
 $\neg \text{interp } C \models_h C \wedge S \ C = \{\#\}$

unfolding *production-unfold* **by** *auto*

lemma $\text{produces } C \ A \implies \text{Pos } A \in \# \ C$

by (*simp add: Max-in-lits producesD*)

lemma *interp'-def-in-set*:

$\text{interp } C = (\bigcup D \in \{D \in N. D \# \subseteq \# \ C\}. \text{production } D)$

unfolding *interp-def* **apply** *auto*

unfolding *production-unfold* **apply** *auto*

done

lemma *production-iff-produces*:

$\text{produces } D \ A \longleftrightarrow A \in \text{production } D$

unfolding *production-unfold* **by** *auto*

definition *Interp* :: 'a clause \Rightarrow 'a interp **where**

Interp *C* = *interp* *C* \cup *production* *C*

lemma

assumes *produces* *C* *P*

shows *Interp* *C* $\models_h C$

unfolding *Interp-def* *assms* **using** *producesD*[*OF* *assms*]

by (*metis* *Max-in-lits* *Un-insert-right* *insertI1* *pos-literal-in-imp-true-cls*)

definition *INTERP* :: 'a interp **where**

INTERP = $(\bigcup D \in N. \text{production } D)$

lemma *interp-subseteq-Interp*[*simp*]: *interp* *C* \subseteq *Interp* *C*

unfolding *Interp-def* **by** *simp*

lemma *Interp-as-UNION*: *Interp* *C* = $(\bigcup D \in \{D. D \# \subseteq \# \ C\}. \text{production } D)$

unfolding *Interp-def* *interp-def* *le-multiset-def* **by** *fast*

lemma *productive-not-empty*: *productive* *C* $\implies C \neq \{\#\}$

unfolding *production-unfold* **by** *auto*

lemma *productive-imp-produces-Max-literal*: *productive* *C* $\implies \text{produces } C \ (\text{atm-of } (\text{Max } (\text{set-mset } C)))$

unfolding *production-unfold* **by** (*auto simp del: atm-of-Max-lit*)

lemma *productive-imp-produces-Max-atom*: *productive* *C* $\implies \text{produces } C \ (\text{Max } (\text{atms-of } C))$

unfolding *atms-of-def* *Max-atm-of-set-mset-commute*[*OF* *productive-not-empty*]

by (*rule* *productive-imp-produces-Max-literal*)

lemma *produces-imp-Max-literal*: $\text{produces } C \ A \implies A = \text{atm-of } (\text{Max } (\text{set-mset } C))$

by (*metis* *Max-singleton* *insert-not-empty* *productive-imp-produces-Max-literal*)

lemma *produces-imp-Max-atom*: $\text{produces } C \ A \implies A = \text{Max } (\text{atms-of } C)$

by (*metis* *Max-singleton* *insert-not-empty* *productive-imp-produces-Max-atom*)

lemma *produces-imp-Pos-in-lits*: $\text{produces } C \ A \implies \text{Pos } A \in \# \ C$
by (*auto intro: Max-in-lits dest!: producesD*)

lemma *productive-in-N*: $\text{productive } C \implies C \in N$
unfolding *production-unfold* **by** *auto*

lemma *produces-imp-atms-leq*: $\text{produces } C \ A \implies B \in \text{atms-of } C \implies B \leq A$
by (*metis Max-ge finite-atms-of insert-not-empty productive-imp-produces-Max-atom singleton-inject*)

lemma *produces-imp-neg-notin-lits*: $\text{produces } C \ A \implies \neg \text{Neg } A \in \# \ C$
by (*rule pos-Max-imp-neg-notin*) (*auto dest: producesD*)

lemma *less-eq-imp-interp-subseteq-interp*: $C \ \# \subseteq \# \ D \implies \text{interp } C \subseteq \text{interp } D$
unfolding *interp-def* **by** *auto* (*metis multiset-order.order.strict-trans2*)

lemma *less-eq-imp-interp-subseteq-Interp*: $C \ \# \subseteq \# \ D \implies \text{interp } C \subseteq \text{Interp } D$
unfolding *Interp-def* **using** *less-eq-imp-interp-subseteq-interp* **by** *blast*

lemma *less-imp-production-subseteq-interp*: $C \ \# \subset \# \ D \implies \text{production } C \subseteq \text{interp } D$
unfolding *interp-def* **by** *fast*

lemma *less-eq-imp-production-subseteq-Interp*: $C \ \# \subseteq \# \ D \implies \text{production } C \subseteq \text{Interp } D$
unfolding *Interp-def* **using** *less-imp-production-subseteq-interp*
by (*metis multiset-order.le-imp-less-or-eq le-supI1 sup-ge2*)

lemma *less-imp-Interp-subseteq-interp*: $C \ \# \subset \# \ D \implies \text{Interp } C \subseteq \text{interp } D$
unfolding *Interp-def*
by (*auto simp: less-eq-imp-interp-subseteq-interp less-imp-production-subseteq-interp*)

lemma *less-eq-imp-Interp-subseteq-Interp*: $C \ \# \subseteq \# \ D \implies \text{Interp } C \subseteq \text{Interp } D$
using *less-imp-Interp-subseteq-interp*
unfolding *Interp-def* **by** (*metis multiset-order.le-imp-less-or-eq le-supI2 subset-refl sup-commute*)

lemma *false-Interp-to-true-interp-imp-less-multiset*: $A \notin \text{Interp } C \implies A \in \text{interp } D \implies C \ \# \subset \# \ D$
using *less-eq-imp-interp-subseteq-Interp multiset-linorder.not-less* **by** *blast*

lemma *false-interp-to-true-interp-imp-less-multiset*: $A \notin \text{interp } C \implies A \in \text{interp } D \implies C \ \# \subset \# \ D$
using *less-eq-imp-interp-subseteq-interp multiset-linorder.not-less* **by** *blast*

lemma *false-Interp-to-true-Interp-imp-less-multiset*: $A \notin \text{Interp } C \implies A \in \text{Interp } D \implies C \ \# \subset \# \ D$
using *less-eq-imp-Interp-subseteq-Interp multiset-linorder.not-less* **by** *blast*

lemma *false-interp-to-true-Interp-imp-le-multiset*: $A \notin \text{interp } C \implies A \in \text{Interp } D \implies C \ \# \subseteq \# \ D$
using *less-imp-Interp-subseteq-interp multiset-linorder.not-less* **by** *blast*

lemma *interp-subseteq-INTERP*: $\text{interp } C \subseteq \text{INTERP}$
unfolding *interp-def INTERP-def* **by** (*auto simp: production-unfold*)

lemma *production-subseteq-INTERP*: $\text{production } C \subseteq \text{INTERP}$
unfolding *INTERP-def* **using** *production-unfold* **by** *blast*

lemma *Interp-subseteq-INTERP*: $\text{Interp } C \subseteq \text{INTERP}$
unfolding *Interp-def* **by** (*auto intro!: interp-subseteq-INTERP production-subseteq-INTERP*)

This lemma corresponds to theorem 2.7.6 page 67 of Weidenbach's book.

lemma *produces-imp-in-interp*:

assumes *a-in-c*: $Neg A \in \# C$ **and** *d*: *produces* $D A$
shows $A \in interp C$

proof –

from *d* **have** $Max (set-mset D) = Pos A$

using *production-unfold* **by** *blast*

hence $D \# \subset \# \{ \# Neg A \# \}$

by (*auto intro*: *Max-pos-neg-less-multiset*)

moreover have $\{ \# Neg A \# \} \# \subseteq \# C$

by (*rule less-eq-imp-le-multiset*) (*rule mset-le-single*[*OF a-in-c*])

ultimately show *?thesis*

using *d* **by** (*blast dest*: *less-eq-imp-interp-subseteq-interp less-imp-production-subseteq-interp*)

qed

lemma *neg-notin-Interp-not-produce*: $Neg A \in \# C \implies A \notin Interp D \implies C \# \subseteq \# D \implies \neg produces D'' A$

by (*auto dest*: *produces-imp-in-interp less-eq-imp-interp-subseteq-Interp*)

lemma *in-production-imp-produces*: $A \in production C \implies produces C A$

by (*metis insert-absorb productive-imp-produces-Max-atom singleton-insert-inj-eq'*)

lemma *not-produces-imp-notin-production*: $\neg produces C A \implies A \notin production C$

by (*metis in-production-imp-produces*)

lemma *not-produces-imp-notin-interp*: $(\bigwedge D. \neg produces D A) \implies A \notin interp C$

unfolding *interp-def* **by** (*fast intro!*: *in-production-imp-produces*)

The results below corresponds to Lemma 3.4.

Nitpicking: If $D = D'$ and D is productive, $I^D \subseteq I_{D'}$ does not hold.

lemma *true-Interp-imp-general*:

assumes

c-le-d: $C \# \subseteq \# D$ **and**

d-lt-d': $D \# \subset \# D'$ **and**

c-at-d: $Interp D \models_h C$ **and**

subs: $interp D' \subseteq (\bigcup C \in CC. production C)$

shows $(\bigcup C \in CC. production C) \models_h C$

proof (*cases* $\exists A. Pos A \in \# C \wedge A \in Interp D$)

case *True*

then obtain *A* **where** *a-in-c*: $Pos A \in \# C$ **and** *a-at-d*: $A \in Interp D$

by *blast*

from *a-at-d* **have** $A \in interp D'$

using *d-lt-d'* *less-imp-Interp-subseteq-interp* **by** *blast*

thus *?thesis*

using *subs a-in-c* **by** (*blast dest*: *contra-subsetD*)

next

case *False*

then obtain *A* **where** *a-in-c*: $Neg A \in \# C$ **and** $A \notin Interp D$

using *c-at-d* *unfolding true-cls-def* **by** *blast*

hence $\bigwedge D''. \neg produces D'' A$

using *c-le-d* *neg-notin-Interp-not-produce* **by** *simp*

thus *?thesis*

using *a-in-c* *subs not-produces-imp-notin-production* **by** *auto*

qed

lemma *true-Interp-imp-interp*: $C \# \subseteq \# D \implies D \# \subset \# D' \implies Interp D \models_h C \implies interp D' \models_h C$

using *interp-def true-Interp-imp-general* **by** *simp*

lemma *true-Interp-imp-Interp*: $C \# \subseteq \# D \implies D \# \subset \# D' \implies \text{Interp } D \models_h C \implies \text{Interp } D' \models_h C$
using *Interp-as-UNION interp-subseteq-Interp true-Interp-imp-general* **by** *simp*

lemma *true-Interp-imp-INTERP*: $C \# \subseteq \# D \implies \text{Interp } D \models_h C \implies \text{INTERP} \models_h C$
using *INTERP-def interp-subseteq-INTERP*
true-Interp-imp-general[OF - less-multiset-right-total]
by *simp*

lemma *true-interp-imp-general*:
assumes
c-le-d: $C \# \subseteq \# D$ **and**
d-lt-d': $D \# \subset \# D'$ **and**
c-at-d: $\text{interp } D \models_h C$ **and**
subs: $\text{interp } D' \subseteq (\bigcup C \in CC. \text{production } C)$
shows $(\bigcup C \in CC. \text{production } C) \models_h C$
proof (*cases* $\exists A. \text{Pos } A \in \# C \wedge A \in \text{interp } D$)
case *True*
then obtain *A* **where** *a-in-c*: $\text{Pos } A \in \# C$ **and** *a-at-d*: $A \in \text{interp } D$
by *blast*
from *a-at-d* **have** $A \in \text{interp } D'$
using *d-lt-d' less-eq-imp-interp-subseteq-interp[OF multiset-order.less-imp-le]* **by** *blast*
thus *?thesis*
using *subs a-in-c* **by** (*blast dest: contra-subsetD*)
next
case *False*
then obtain *A* **where** *a-in-c*: $\text{Neg } A \in \# C$ **and** $A \notin \text{interp } D$
using *c-at-d unfolding true-cls-def* **by** *blast*
hence $\bigwedge D''. \neg \text{produces } D'' A$
using *c-le-d* **by** (*auto dest: produces-imp-in-interp less-eq-imp-interp-subseteq-interp*)
thus *?thesis*
using *a-in-c subs not-produces-imp-notin-production* **by** *auto*
qed

This lemma corresponds to theorem 2.7.6 page 67 of Weidenbach's book. Here the strict maximality is important

lemma *true-interp-imp-interp*: $C \# \subseteq \# D \implies D \# \subset \# D' \implies \text{interp } D \models_h C \implies \text{interp } D' \models_h C$
using *interp-def true-interp-imp-general* **by** *simp*

lemma *true-interp-imp-Interp*: $C \# \subseteq \# D \implies D \# \subset \# D' \implies \text{interp } D \models_h C \implies \text{Interp } D' \models_h C$
using *Interp-as-UNION interp-subseteq-Interp[of D'] true-interp-imp-general* **by** *simp*

lemma *true-interp-imp-INTERP*: $C \# \subseteq \# D \implies \text{interp } D \models_h C \implies \text{INTERP} \models_h C$
using *INTERP-def interp-subseteq-INTERP*
true-interp-imp-general[OF - less-multiset-right-total]
by *simp*

lemma *productive-imp-false-interp*: $\text{productive } C \implies \neg \text{interp } C \models_h C$
unfolding *production-unfold* **by** *auto*

This lemma corresponds to theorem 2.7.6 page 67 of Weidenbach's book. Here the strict maximality is important

lemma *cls-gt-double-pos-no-production*:
assumes *D*: $\{\# \text{Pos } P, \text{Pos } P \# \} \# \subset \# C$

shows $\neg \text{produces } C \ P$
proof –
 let $?D = \{\#Pos \ P, \ Pos \ P\#\}$
 note $D' = D[\text{unfolded less-multiset}_{HO}]$
consider
 (P) *count* $C \ (Pos \ P) \geq 2$
 | (Q) Q **where** $Q > Pos \ P$ **and** $Q \in \# \ C$
 using $HOL.spec[OF \ HOL.conjunct2[OF \ D], \ of \ Pos \ P]$ **by** (auto split: if-split-asm)
thus ?thesis
proof cases
 case Q
 have $Q \in \text{set-mset } C$
 using $Q(2)$ **by** (auto split: if-split-asm)
 then have $Max \ (\text{set-mset } C) > Pos \ P$
 using $Q(1) \ Max\text{-gr-iff}$ **by** blast
 thus ?thesis
 unfolding production-unfold **by** auto
 next
 case P
 thus ?thesis
 unfolding production-unfold **by** auto
 qed
 qed

This lemma corresponds to theorem 2.7.6 page 67 of Weidenbach's book.

lemma
 assumes $D: C + \{\#Neg \ P\# \} \# \subset \# \ D$
 shows *production* $D \neq \{P\}$
proof –
 note $D' = D[\text{unfolded less-multiset}_{HO}]$
consider
 (P) $Neg \ P \in \# \ D$
 | (Q) Q **where** $Q > Neg \ P$ **and** *count* $D \ Q > \text{count } (C + \{\#Neg \ P\# \}) \ Q$
 using $HOL.spec[OF \ HOL.conjunct2[OF \ D], \ of \ Neg \ P]$ *count-greater-zero-iff* **by** fastforce
thus ?thesis
proof cases
 case Q
 have $Q \in \text{set-mset } D$
 using $Q(2)$ *gr-implies-not0* **by** fastforce
 then have $Max \ (\text{set-mset } D) > Neg \ P$
 using $Q(1) \ Max\text{-gr-iff}$ **by** blast
 hence $Max \ (\text{set-mset } D) > Pos \ P$
 using less-trans[of $Pos \ P \ Neg \ P \ Max \ (\text{set-mset } D)$] **by** auto
 thus ?thesis
 unfolding production-unfold **by** auto
 next
 case P
 hence $Max \ (\text{set-mset } D) > Pos \ P$
 by (meson *Max-ge finite-set-mset le-less-trans linorder-not-le pos-less-neg*)
 thus ?thesis
 unfolding production-unfold **by** auto
 qed
 qed

lemma *in-interp-is-produced:*

assumes $P \in \text{INTERP}$
shows $\exists D. D + \{\#Pos\ P\# \} \in N \wedge \text{produces } (D + \{\#Pos\ P\# \})\ P$
using *assms unfolding INTERP-def UN-iff production-iff-produces Ball-def*
by (*metis ground-resolution-with-selection.produces-imp-Pos-in-lits insert-DiffM2*
ground-resolution-with-selection-axioms not-produces-imp-notin-production)

end
end

abbreviation $MMax\ M \equiv Max\ (set-mset\ M)$

2.1 We can now define the rules of the calculus

inductive *superposition-rules* :: 'a clause \Rightarrow 'a clause \Rightarrow 'a clause \Rightarrow bool **where**
factoring: *superposition-rules* $(C + \{\#Pos\ P\# \} + \{\#Pos\ P\# \})\ B\ (C + \{\#Pos\ P\# \})\ |$
superposition-l: *superposition-rules* $(C_1 + \{\#Pos\ P\# \})\ (C_2 + \{\#Neg\ P\# \})\ (C_1 + C_2)$

inductive *superposition* :: 'a clauses \Rightarrow 'a clauses \Rightarrow bool **where**
superposition: $A \in N \Longrightarrow B \in N \Longrightarrow \text{superposition-rules } A\ B\ C$
 $\Longrightarrow \text{superposition } N\ (N \cup \{C\})$

definition *abstract-red* :: 'a::wellorder clause \Rightarrow 'a clauses \Rightarrow bool **where**
abstract-red $C\ N = (clss-lt\ N\ C \models_p C)$

lemma *less-multiset[iff]*: $M < N \longleftrightarrow M \# \subset \# N$
unfolding *less-multiset-def* **by** *auto*

lemma *less-eq-multiset[iff]*: $M \leq N \longleftrightarrow M \# \subseteq \# N$
unfolding *less-eq-multiset-def* **by** *auto*

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

assumes
 $AB: A \models_{hs} B$ **and**
 $BC: B \models_p C$
shows $A \models_h C$

proof –

let $?I = \{Pos\ P\ | P. P \in A\} \cup \{Neg\ P\ | P. P \notin A\}$
have $B: ?I \models_s B$ **using** AB
by (*auto simp add: herbrand-interp-iff-partial-interp-clss*)

have $IH: \bigwedge I. \text{total-over-set } I\ (\text{atms-of } C) \Longrightarrow \text{total-over-m } I\ B \Longrightarrow \text{consistent-interp } I$
 $\Longrightarrow I \models_s B \Longrightarrow I \models C$ **using** BC
by (*auto simp add: true-clss-clss-def*)

show *?thesis*

unfolding *herbrand-interp-iff-partial-interp-clss*
by (*auto intro: IH[of ?I] simp add: herbrand-total-over-set herbrand-total-over-m*
herbrand-consistent-interp B)

qed

lemma *abstract-red-subset-mset-abstract-red*:

assumes
 $abstr: \text{abstract-red } C\ N$ **and**
 $c-lt-d: C \subseteq \# D$
shows $\text{abstract-red } D\ N$

proof –

```

have  $\{D \in N. D \# \subset \# C\} \subseteq \{D' \in N. D' \# \subset \# D\}$ 
  using c-lt-d less-eq-imp-le-multiset by fastforce
thus ?thesis
  using abstr unfolding abstract-red-def clss-lt-def
  by (metis (no-types, lifting) c-lt-d subset-mset.diff-add true-clss-clss-mono-r'
      true-clss-clss-subset)
qed

```

lemma *true-clss-clss-extended*:

```

assumes
   $A \models_p B$  and
  tot: total-over-m  $I$  ( $A$ ) and
  cons: consistent-interp  $I$  and
   $I \models_s A$ 
shows  $I \models B$ 
proof -
  let ? $I$  =  $I \cup \{Pos\ P \mid P. P \in \text{atms-of } B \wedge P \notin \text{atms-of-s } I\}$ 
  have consistent-interp ? $I$ 
    using cons unfolding consistent-interp-def atms-of-s-def atms-of-def
    apply (auto 1 5 simp add: image-iff)
    by (metis atm-of-uminus literal.sel(1))
  moreover have total-over-m ? $I$  ( $A \cup \{B\}$ )
  proof -
    obtain  $aa :: 'a \text{ set} \Rightarrow 'a \text{ literal set} \Rightarrow 'a$  where
       $f2: \forall x0\ x1. (\exists v2. v2 \in x0 \wedge Pos\ v2 \notin x1 \wedge Neg\ v2 \notin x1)$ 
       $\longleftrightarrow (aa\ x0\ x1 \in x0 \wedge Pos\ (aa\ x0\ x1) \notin x1 \wedge Neg\ (aa\ x0\ x1) \notin x1)$ 
    by moura
    have  $\forall a. a \notin \text{atms-of-ms } A \vee Pos\ a \in I \vee Neg\ a \in I$ 
      using tot by (simp add: total-over-m-def total-over-set-def)
    hence  $aa\ (\text{atms-of-ms } A \cup \text{atms-of-ms } \{B\})\ (I \cup \{Pos\ a \mid a. a \in \text{atms-of } B \wedge a \notin \text{atms-of-s } I\})$ 
       $\notin \text{atms-of-ms } A \cup \text{atms-of-ms } \{B\} \vee Pos\ (aa\ (\text{atms-of-ms } A \cup \text{atms-of-ms } \{B\})$ 
       $(I \cup \{Pos\ a \mid a. a \in \text{atms-of } B \wedge a \notin \text{atms-of-s } I\})) \in I$ 
       $\cup \{Pos\ a \mid a. a \in \text{atms-of } B \wedge a \notin \text{atms-of-s } I\}$ 
       $\vee Neg\ (aa\ (\text{atms-of-ms } A \cup \text{atms-of-ms } \{B\})$ 
       $(I \cup \{Pos\ a \mid a. a \in \text{atms-of } B \wedge a \notin \text{atms-of-s } I\})) \in I$ 
       $\cup \{Pos\ a \mid a. a \in \text{atms-of } B \wedge a \notin \text{atms-of-s } I\}$ 
    by auto
    hence total-over-set ( $I \cup \{Pos\ a \mid a. a \in \text{atms-of } B \wedge a \notin \text{atms-of-s } I\}$ )
      ( $\text{atms-of-ms } A \cup \text{atms-of-ms } \{B\}$ )
      using  $f2$  by (meson total-over-set-def)
    thus ?thesis
      by (simp add: total-over-m-def)
  qed
  moreover have ? $I \models_s A$ 
    using  $I \models_s A$  by auto
  ultimately have ? $I \models B$ 
    using  $\langle A \models_p B \rangle$  unfolding true-clss-clss-def by auto
  thus ?thesis

```

oops

lemma

```

assumes
   $CP: \neg \text{clss-lt } N\ (\{\#C\# \} + \{\#E\# \}) \models_p \{\#C\# \} + \{\#Neg\ P\# \}$  and
   $\text{clss-lt } N\ (\{\#C\# \} + \{\#E\# \}) \models_p \{\#E\# \} + \{\#Pos\ P\# \} \vee \text{clss-lt } N\ (\{\#C\# \} + \{\#E\# \}) \models_p$ 
   $\{\#C\# \} + \{\#Neg\ P\# \}$ 

```

shows *clss-lt* N ($\{\#C\# \} + \{\#E\# \} \models_p \{\#E\# \} + \{\#Pos\ P\# \}$)

oops

locale *ground-ordered-resolution-with-redundancy* =
ground-resolution-with-selection +
fixes *redundant* :: 'a::wellorder clause \Rightarrow 'a clauses \Rightarrow bool
assumes
redundant-iff-abstract: *redundant* $A\ N \longleftrightarrow$ *abstract-red* $A\ N$
begin
definition *saturated* :: 'a clauses \Rightarrow bool **where**
saturated $N \longleftrightarrow (\forall A\ B\ C. A \in N \longrightarrow B \in N \longrightarrow \neg \text{redundant } A\ N \longrightarrow \neg \text{redundant } B\ N$
 $\longrightarrow \text{superposition-rules } A\ B\ C \longrightarrow \text{redundant } C\ N \vee C \in N)$
lemma
assumes
saturated: *saturated* N **and**
finite: *finite* N **and**
empty: $\{\#\} \notin N$
shows *INTERP* $N \models_{hs} N$
proof (*rule ccontr*)
let $?N_{\mathcal{I}} = \text{INTERP } N$
assume $\neg ?thesis$
hence *not-empty*: $\{E \in N. \neg ?N_{\mathcal{I}} \models_h E\} \neq \{\}$
unfolding *true-clss-def Ball-def* **by** *auto*
def $D \equiv \text{Min } \{E \in N. \neg ?N_{\mathcal{I}} \models_h E\}$
have [*simp*]: $D \in N$
unfolding *D-def*
by (*metis* (*mono-tags*, *lifting*) *Min-in not-empty finite mem-Collect-eq rev-finite-subset subsetI*)
have *not-d-interp*: $\neg ?N_{\mathcal{I}} \models_h D$
unfolding *D-def*
by (*metis* (*mono-tags*, *lifting*) *Min-in finite mem-Collect-eq not-empty rev-finite-subset subsetI*)
have *cls-not-D*: $\bigwedge E. E \in N \Longrightarrow E \neq D \Longrightarrow \neg ?N_{\mathcal{I}} \models_h E \Longrightarrow D \leq E$
using *finite D-def* **by** (*auto simp del: less-eq-multiset*)
obtain $C\ L$ **where** $D: D = C + \{\#L\# \}$ **and** *LSD*: $L \in \# S\ D \vee (S\ D = \{\#\} \wedge \text{Max } (\text{set-mset } D) = L)$
proof (*cases* $S\ D = \{\#\}$)
case *False*
then obtain L **where** $L \in \# S\ D$
using *Max-in-lits* **by** *blast*
moreover
hence $L \in \# D$
using *S-selects-subseteq*[*of D*] **by** *auto*
hence $D = (D - \{\#L\# \}) + \{\#L\# \}$
by *auto*
ultimately show *?thesis* **using** *that* **by** *blast*
next
let $?L = \text{MMax } D$
case *True*
moreover
have $?L \in \# D$
by (*metis* (*no-types*, *lifting*) *Max-in-lits* $\langle D \in N \rangle$ *empty*)
hence $D = (D - \{\#?L\# \}) + \{\#?L\# \}$
by *auto*
ultimately show *?thesis* **using** *that* **by** *blast*

```

qed
have red:  $\neg$ redundant  $D$   $N$ 
proof (rule ccontr)
  assume red[simplified]:  $\sim\sim$ redundant  $D$   $N$ 
  have  $\forall E < D. E \in N \longrightarrow ?N_{\mathcal{I}} \models_h E$ 
    using cls-not- $D$  not-le by fastforce
  hence  $?N_{\mathcal{I}} \models_{hs} clss\text{-}lt$   $N$   $D$ 
    unfolding clss-lt-def true-clss-def Ball-def by blast
  thus False
    using red not-d-interp unfolding abstract-red-def redundant-iff-abstract
    using herbrand-true-clss-true-clss-cls-herbrand-true-clss by fast
qed

consider
  ( $L$ )  $P$  where  $L = Pos$   $P$  and  $S$   $D = \{\#\}$  and  $Max$  (set-mset  $D$ ) =  $Pos$   $P$ 
| ( $Lneg$ )  $P$  where  $L = Neg$   $P$ 
  using LSD  $S$ -selects-neg-lits[of  $L$   $D$ ] by (cases  $L$ ) auto
thus False
proof cases
  case  $L$  note  $P = this(1)$  and  $S = this(2)$  and  $max = this(3)$ 
  have count  $D$   $L > 1$ 
  proof (rule ccontr)
    assume  $\sim$  ?thesis
    hence count: count  $D$   $L = 1$ 
    unfolding  $D$  by (auto simp: not-in-iff)
    have  $\neg ?N_{\mathcal{I}} \models_h D$ 
    using not-d-interp true-interp-imp-INTERP ground-resolution-with-selection-axioms
    by blast
    hence produces  $N$   $D$   $P$ 
    using not-empty empty finite  $\langle D \in N \rangle$  count  $L$ 
    true-interp-imp-INTERP unfolding production-iff-produces unfolding production-unfold
    by (auto simp add: max not-empty)
    hence INTERP  $N \models_h D$ 
    unfolding  $D$ 
    by (metis pos-literal-in-imp-true-cls produces-imp-Pos-in-lits
    production-subseteq-INTERP singletonI subsetCE)
    thus False
    using not-d-interp by blast
  qed
then have  $Pos$   $P \in \#$   $C$ 
  by (simp add:  $P$   $D$ )
then obtain  $C'$  where  $C':D = C' + \{\#Pos$   $P\#\} + \{\#Pos$   $P\#\}$ 
  unfolding  $D$  by (metis (full-types)  $P$  insert-DiffM2)
have sup: superposition-rules  $D$   $D$  ( $D - \{\#L\#\}$ )
  unfolding  $C'$   $L$  by (auto simp add: superposition-rules.simps)
have  $C' + \{\#Pos$   $P\#\} \# \subset \#$   $C' + \{\#Pos$   $P\#\} + \{\#Pos$   $P\#\}$ 
  by auto
moreover have  $\neg ?N_{\mathcal{I}} \models_h (D - \{\#L\#\})$ 
  using not-d-interp unfolding  $C'$   $L$  by auto
ultimately have  $C' + \{\#Pos$   $P\#\} \notin N$ 
  by (metis (no-types, lifting)  $C'$   $P$  add-diff-cancel-right' cls-not- $D$  less-multiset
  multi-self-add-other-not-self not-le)
have  $D - \{\#L\#\} \# \subset \#$   $D$ 
  unfolding  $C'$   $L$  by auto
have  $c'\text{-}p\text{-}p$ :  $C' + \{\#Pos$   $P\#\} + \{\#Pos$   $P\#\} - \{\#Pos$   $P\#\} = C' + \{\#Pos$   $P\#\}$ 

```

```

  by auto
have redundant  $(C' + \{\#Pos P\}) N$ 
  using saturated red sup  $\langle D \in N \rangle \langle C' + \{\#Pos P\} \notin N \rangle$  unfolding saturated-def  $C' L c'-p-p$ 
  by blast
moreover have  $C' + \{\#Pos P\} \subseteq\# C' + \{\#Pos P\} + \{\#Pos P\}$ 
  by auto
ultimately show False
  using red unfolding  $C'$  redundant-iff-abstract by (blast dest:
    abstract-red-subset-mset-abstract-red)
next
case Lneg note  $L = this(1)$ 
have  $P \in ?N_{\mathcal{I}}$ 
  using not-d-interp unfolding  $D$  true-cls-def  $L$  by (auto split: if-split-asm)
then obtain  $E$  where
  DPN:  $E + \{\#Pos P\} \in N$  and
  prod: production  $N (E + \{\#Pos P\}) = \{P\}$ 
  using in-interp-is-produced by blast
have sup-EC: superposition-rules  $(E + \{\#Pos P\}) (C + \{\#Neg P\}) (E + C)$ 
  using superposition-l by fast
hence superposition  $N (N \cup \{E+C\})$ 
  using DPN  $\langle D \in N \rangle$  unfolding  $D L$  by (auto simp add: superposition.simps)
have
  PMax:  $Pos P = MMax (E + \{\#Pos P\})$  and
  count  $(E + \{\#Pos P\}) (Pos P) \leq 1$  and
   $S (E + \{\#Pos P\}) = \{\#\}$  and
   $\neg interp N (E + \{\#Pos P\}) \models_h E + \{\#Pos P\}$ 
  using prod unfolding production-unfold by auto
have  $Neg P \notin\# E$ 
  using prod produces-imp-neg-notin-lits by force
hence  $\bigwedge y. y \in\# (E + \{\#Pos P\})$ 
 $\implies count (E + \{\#Pos P\}) (Neg P) < count (C + \{\#Neg P\}) (Neg P)$ 
  using count-greater-zero-iff by fastforce
moreover have  $\bigwedge y. y \in\# (E + \{\#Pos P\}) \implies y < Neg P$ 
  using PMax by (metis DPN Max-less-iff empty finite-set-mset pos-less-neg
    set-mset-eq-empty-iff)
moreover have  $E + \{\#Pos P\} \neq C + \{\#Neg P\}$ 
  using prod produces-imp-neg-notin-lits by force
ultimately have  $E + \{\#Pos P\} \# \subset\# C + \{\#Neg P\}$ 
  unfolding less-multisetHO by (metis count-greater-zero-iff less-iff-Suc-add zero-less-Suc)
have ce-lt-d:  $C + E \# \subset\# D$ 
  unfolding  $D L$  by (simp add:  $\langle \bigwedge y. y \in\# E + \{\#Pos P\} \implies y < Neg P \rangle$  ex-gt-imp-less-multiset)
have  $?N_{\mathcal{I}} \models_h E + \{\#Pos P\}$ 
  using  $\langle P \in ?N_{\mathcal{I}} \rangle$  by blast
have  $?N_{\mathcal{I}} \models_h C+E \vee C+E \notin N$ 
  using ce-lt-d cls-not-D unfolding  $D$ -def by fastforce
have  $Pos P \notin\# C+E$ 
  using  $D \langle P \in ground-resolution-with-selection.INTERP S N \rangle$ 
 $\langle count (E + \{\#Pos P\}) (Pos P) \leq 1 \rangle$  multi-member-skip not-d-interp
  by (auto simp: not-in-iff)
hence  $\bigwedge y. y \in\# C+E$ 
 $\implies count (C+E) (Pos P) < count (E + \{\#Pos P\}) (Pos P)$ 
  using set-mset-def by fastforce

have  $\neg redundant (C + E) N$ 
  proof (rule ccontr)

```

```

assume  $red'[simplified]: \neg ?thesis$ 
have  $abs: clss-lt\ N\ (C + E) \models_p C + E$ 
  using redundant-iff-abstract  $red'$  unfolding abstract-red-def by auto
have  $clss-lt\ N\ (C + E) \models_p E + \{\#Pos\ P\# \} \vee clss-lt\ N\ (C + E) \models_p C + \{\#Neg\ P\# \}$ 
proof clarify
  assume  $CP: \neg clss-lt\ N\ (C + E) \models_p C + \{\#Neg\ P\# \}$ 
  { fix  $I$ 
    assume
       $total-over-m\ I\ (clss-lt\ N\ (C + E) \cup \{E + \{\#Pos\ P\# \}\})$  and
       $consistent-interp\ I$  and
       $I \models_s clss-lt\ N\ (C + E)$ 
      hence  $I \models C + E$ 
      using abs sorry
      moreover have  $\neg I \models C + \{\#Neg\ P\# \}$ 
      using  $CP$  unfolding true-clss-clss-def
      sorry
      ultimately have  $I \models E + \{\#Pos\ P\# \}$  by auto
    }
  then show  $clss-lt\ N\ (C + E) \models_p E + \{\#Pos\ P\# \}$ 
    unfolding true-clss-clss-def by auto
  qed
moreover have  $clss-lt\ N\ (C + E) \subseteq clss-lt\ N\ (C + \{\#Neg\ P\# \})$ 
  using ce-lt-d mult-less-trans unfolding clss-lt-def D L by force
ultimately have  $redundant\ (C + \{\#Neg\ P\# \})\ N \vee clss-lt\ N\ (C + E) \models_p E + \{\#Pos\ P\# \}$ 
  unfolding redundant-iff-abstract abstract-red-def using true-clss-clss-subset by blast
show False sorry
qed
moreover have  $\neg redundant\ (E + \{\#Pos\ P\# \})\ N$ 
sorry
ultimately have  $CEN: C + E \in N$ 
  using  $\langle D \in N \rangle \langle E + \{\#Pos\ P\# \} \in N \rangle$  saturated sup-EC red unfolding saturated-def D L
  by (metis union-commute)
have  $CED: C + E \neq D$ 
  using  $D$  ce-lt-d by auto
have  $interp: \neg INTERP\ N \models_h C + E$ 
sorry
show False
  using cls-not-D[OF CEN CED interp] ce-lt-d unfolding INTERP-def less-eq-multiset-def by
auto
qed
qed

```

end

lemma *tautology-is-redundant:*

```

assumes tautology C
shows abstract-red C N
using assms unfolding abstract-red-def true-clss-clss-def tautology-def by auto

```

lemma *subsumed-is-redundant:*

```

assumes  $AB: A \subset\# B$ 
and  $AN: A \in N$ 
shows abstract-red B N

```

proof —

```

have  $A \in clss-lt\ N\ B$  using  $AN\ AB$  unfolding clss-lt-def

```

```

    by (auto dest: less-eq-imp-le-multiset simp add: multiset-order.dual-order.order-iff-strict)
  thus ?thesis
    using AB unfolding abstract-red-def true-clss-clss-def Partial-Clausal-Logic.true-clss-def
    by blast
qed

```

```

inductive redundant :: 'a clause  $\Rightarrow$  'a clauses  $\Rightarrow$  bool where
  subsumption:  $A \in N \Longrightarrow A \subset\# B \Longrightarrow \text{redundant } B N$ 

```

```

lemma redundant-is-redundancy-criterion:

```

```

  fixes A :: 'a :: wellorder clause and N :: 'a :: wellorder clauses
  assumes redundant A N
  shows abstract-red A N
  using assms

```

```

proof (induction rule: redundant.induct)

```

```

  case (subsumption A B N)

```

```

  thus ?case

```

```

    using subsumed-is-redundant[of A N B] unfolding abstract-red-def clss-lt-def by auto
  qed

```

```

lemma redundant-mono:

```

```

  redundant A N  $\Longrightarrow A \subseteq\# B \Longrightarrow \text{redundant } B N$ 

```

```

apply (induction rule: redundant.induct)

```

```

by (meson subset-mset.less-le-trans subsumption)

```

```

locale truc =

```

```

  selection S for S :: nat clause  $\Rightarrow$  nat clause

```

```

begin

```

```

end

```

```

end

```