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 . . . 2
1.2.1 Subsumption . . . 3
1.3 Inference Rule . . . 3
1.4 Lemma about the simplified state . . . 8
1.5 Resolution and Invariants . . . 9
1.5.1 Invariants . . . 9
1.5.2 well-foundness if the relation . . . 13

2 Superposition 18
2.1 We can now define the rules of the calculus . . . 23

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 $I \models s\ N' \longrightarrow I \models s\ N$
⟨*proof*⟩

lemma *simplify-preserves-un-sat*:
fixes $N\ N'$ :: $'v$ *clauses*
assumes *simplify* $N\ N'$

and *total-over-m* I N
shows $I \models s\ N \longrightarrow I \models s\ N'$
⟨*proof*⟩

lemma *simplify-preserves-un-sat''*:
fixes $N\ N'$ :: *'v clauses*
assumes *simplify* $N\ N'$
and *total-over-m* $I\ N'$
shows $I \models s\ N \longrightarrow I \models s\ N'$
⟨*proof*⟩

lemma *simplify-preserves-un-sat-eq*:
fixes $N\ N'$ :: *'v clauses*
assumes *simplify* $N\ N'$
and *total-over-m* I N
shows $I \models s\ N \longleftrightarrow I \models s\ N'$
⟨*proof*⟩

lemma *simplify-preserves-finite*:
assumes *simplify* $\psi\ \psi'$
shows *finite* $\psi \longleftrightarrow$ *finite* ψ'
⟨*proof*⟩

lemma *rtranclp-simplify-preserves-finite*:
assumes *rtranclp simplify* $\psi\ \psi'$
shows *finite* $\psi \longleftrightarrow$ *finite* ψ'
⟨*proof*⟩

lemma *simplify-atms-of-ms*:
assumes *simplify* $\psi\ \psi'$
shows *atms-of-ms* $\psi' \subseteq$ *atms-of-ms* ψ
⟨*proof*⟩

lemma *rtranclp-simplify-atms-of-ms*:
assumes *rtranclp simplify* $\psi\ \psi'$
shows *atms-of-ms* $\psi' \subseteq$ *atms-of-ms* ψ
⟨*proof*⟩

lemma *factoring-imp-simplify*:
assumes $\{\#L\#\} + \{\#L\#\} + C \in N$
shows $\exists N'.$ *simplify* $N\ N'$
⟨*proof*⟩

1.2 Unconstrained Resolution

type-synonym *'v uncon-state* = *'v clauses*
inductive *uncon-res* :: *'v uncon-state* ⇒ *'v uncon-state* ⇒ *bool* **where**
resolution:
$\{\#Pos\ p\#\} + C \in N \Longrightarrow \{\#Neg\ p\#\} + D \in N \Longrightarrow (\{\#Pos\ p\#\} + C, \{\#Neg\ p\#\} + D) \notin$ *already-used*
$\Longrightarrow$ *uncon-res* $(N)\ (N \cup \{C + D\})$ |
factoring: $\{\#L\#\} + \{\#L\#\} + C \in N \Longrightarrow$ *uncon-res* $N\ (N \cup \{C + \{\#L\#\}\})$

lemma *uncon-res-increasing*:
assumes *uncon-res* $S\ S'$ **and** $\psi \in S$
shows $\psi \in S'$

$\langle proof \rangle$

lemma *rtranclp-uncon-inference-increasing*:
assumes *rtranclp uncon-res S S'* **and** $\psi \in S$
shows $\psi \in S'$
$\langle proof \rangle$

1.2.1 Subsumption

definition *subsumes* :: *'a literal multiset* $\Rightarrow$ *'a literal multiset* $\Rightarrow$ *bool* **where**
subsumes χ χ' $\longleftrightarrow$
$(\forall I.$ *total-over-m* I $\{\chi'\}$ $\longrightarrow$ *total-over-m* I $\{\chi\})$
$\wedge$ $(\forall I.$ *total-over-m* I $\{\chi\}$ $\longrightarrow$ $I \models \chi \longrightarrow I \models \chi')$

lemma *subsumes-refl*[*simp*]:
subsumes χ χ
$\langle proof \rangle$

lemma *subsumes-subsumption*:
assumes *subsumes* D χ
and $C \subset\# D$ **and** $\neg$*tautology* χ
shows *subsumes* C χ $\langle proof \rangle$

lemma *subsumes-tautology*:
assumes *subsumes* $(C + \{\#Pos\ P\#\} + \{\#Neg\ P\#\})$ χ
shows *tautology* χ
$\langle proof \rangle$

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** $\Rightarrow_{\mathrm{Res}}$ *100*) **where**
resolution:
$\{\#Pos\ p\#\} + C \in N \Longrightarrow \{\#Neg\ p\#\} + D \in N \Longrightarrow (\{\#Pos\ p\#\} + C, \{\#Neg\ p\#\} + D) \notin$ *already-used*
$\Longrightarrow$ *inference-clause* (N, *already-used*) ($C + D$, *already-used* $\cup$ $\{(\{\#Pos\ p\#\} + C, \{\#Neg\ p\#\} + D)\}$) |
factoring: $\{\#L\#\} + \{\#L\#\} + C \in N \Longrightarrow$ *inference-clause* (N, *already-used*) ($C + \{\#L\#\}$, *already-used*)

inductive *inference* :: *'v state* $\Rightarrow$ *'v state* $\Rightarrow$ *bool* **where**
inference-step: *inference-clause* S (*clause*, *already-used*)
$\Longrightarrow$ *inference* S (*fst* S $\cup$ $\{$*clause*$\}$, *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 $\equiv$
$(\forall (A, B) \in$ *snd state*. $\exists p.$ *Pos* $p \in\# A \wedge$ *Neg* $p \in\# B \wedge$
$((\exists \chi \in$ *fst state*. *subsumes* χ $((A - \{\#Pos\ p\#\}) + (B - \{\#Neg\ p\#\})))$
$\vee$ *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* $(fst\ S \cup \{fst\ S'\},\ snd\ S')$
⟨*proof*⟩

lemma *inference-preserves-already-used-inv*:
assumes *inference* $S\ S'$
and *already-used-inv* S
shows *already-used-inv* S'
⟨*proof*⟩

lemma *rtranclp-inference-preserves-already-used-inv*:
assumes *rtranclp inference* $S\ S'$
and *already-used-inv* S
shows *already-used-inv* S'
⟨*proof*⟩

lemma *subsumes-condensation*:
assumes *subsumes* $(C + \{\#L\#\} + \{\#L\#\})\ D$
shows *subsumes* $(C + \{\#L\#\})\ D$
⟨*proof*⟩

lemma *simplify-preserves-already-used-inv*:
assumes *simplify* $N\ N'$
and *already-used-inv* $(N,\ already\text{-}used)$
shows *already-used-inv* $(N',\ already\text{-}used)$
⟨*proof*⟩

lemma
factoring-satisfiable: $I \models \{\#L\#\} + \{\#L\#\} + C \longleftrightarrow I \models \{\#L\#\} + C$ **and**
resolution-satisfiable:
consistent-interp $I \Longrightarrow I \models \{\#Pos\ p\#\} + C \Longrightarrow I \models \{\#Neg\ p\#\} + D \Longrightarrow I \models C + D$ **and**
factoring-same-vars: $atms\text{-}of\ (\{\#L\#\} + \{\#L\#\} + C) = atms\text{-}of\ (\{\#L\#\} + C)$
⟨*proof*⟩

lemma *inference-increasing*:
assumes *inference* $S\ S'$ **and** $\psi \in fst\ S$
shows $\psi \in fst\ S'$
⟨*proof*⟩

lemma *rtranclp-inference-increasing*:
assumes *rtranclp inference* $S\ S'$ **and** $\psi \in fst\ S$
shows $\psi \in fst\ S'$
⟨*proof*⟩

lemma *inference-clause-already-used-increasing*:
assumes *inference-clause* $S\ S'$
shows $snd\ S \subseteq snd\ S'$
⟨*proof*⟩

lemma *inference-already-used-increasing*:
assumes *inference* $S\ S'$
shows $snd\ S \subseteq snd\ S'$
⟨*proof*⟩

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 $I \models s\ fst\ T \longleftrightarrow I \models s\ fst\ T \cup \{fst\ T'\}$
⟨*proof*⟩

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 $I \models s\ fst\ T \longleftrightarrow I \models s\ fst\ T'$
⟨*proof*⟩

lemma *inference-clause-preserves-atms-of-ms*:
assumes *inference-clause* $S\ S'$
shows *atms-of-ms* (*fst* (*fst* $S \cup \{fst\ S'\}$, *snd* S')) $\subseteq$ *atms-of-ms* (*fst* S)
⟨*proof*⟩

lemma *inference-preserves-atms-of-ms*:
fixes $N\ N' :: 'v\ clauses$
assumes *inference* $T\ T'$
shows *atms-of-ms* (*fst* T') $\subseteq$ *atms-of-ms* (*fst* T)
⟨*proof*⟩

lemma *inference-preserves-total*:
fixes $N\ N' :: 'v\ clauses$
assumes *inference* (N, *already-used*) (N', *already-used'*)
shows *total-over-m* $I\ N \Longrightarrow$ *total-over-m* $I\ N'$
⟨*proof*⟩

lemma *rtranclp-inference-preserves-total*:
assumes *rtranclp inference* $T\ T'$
shows *total-over-m* I (*fst* T) $\Longrightarrow$ *total-over-m* I (*fst* T')
⟨*proof*⟩

lemma *rtranclp-inference-preserves-un-sat*:
assumes *rtranclp inference* $N\ N'$
and *total-over-m* I (*fst* N)
and *consistent*: *consistent-interp* I
shows $I \models s\ fst\ N \longleftrightarrow I \models s\ fst\ N'$
⟨*proof*⟩

lemma *inference-preserves-finite*:
assumes *inference* $\psi\ \psi'$ **and** *finite* (*fst* ψ)
shows *finite* (*fst* ψ')
⟨*proof*⟩

lemma *inference-clause-preserves-finite-snd*:
assumes *inference-clause* $\psi\ \psi'$ **and** *finite* (*snd* ψ)

shows *finite* (*snd* ψ')
⟨*proof*⟩

lemma *inference-preserves-finite-snd*:
assumes *inference* ψ ψ' **and** *finite* (*snd* ψ)
shows *finite* (*snd* ψ')
⟨*proof*⟩

lemma *rtranclp-inference-preserves-finite*:
assumes *rtranclp inference* ψ ψ' **and** *finite* (*fst* ψ)
shows *finite* (*fst* ψ')
⟨*proof*⟩

lemma *consistent-interp-insert*:
assumes *consistent-interp I*
and *atm-of P* $\notin$ *atm-of* ' *I*
shows *consistent-interp* (*insert P I*)
⟨*proof*⟩

lemma *simplify-clause-preserves-sat*:
assumes *simp*: *simplify* ψ ψ'
and *satisfiable* ψ'
shows *satisfiable* ψ
⟨*proof*⟩

lemma *simplify-preserves-unsat*:
assumes *inference* ψ ψ'
shows *satisfiable* (*fst* ψ') $\longrightarrow$ *satisfiable* (*fst* ψ)
⟨*proof*⟩

lemma *inference-preserves-unsat*:
assumes *inference*** *S S'*
shows *satisfiable* (*fst S'*) $\longrightarrow$ *satisfiable* (*fst S*)
⟨*proof*⟩

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 sem-tree*. ($\bigwedge$*ys*:: *'v sem-tree*. *sem-tree-size ys* < *sem-tree-size xs* $\Longrightarrow$ *P ys*) $\Longrightarrow$ *P xs*)
$\Longrightarrow$ *P xs*
⟨*proof*⟩

fun *partial-interps* :: *'v sem-tree* $\Rightarrow$ *'v interp* $\Rightarrow$ *'v clauses* $\Rightarrow$ *bool* **where**
partial-interps Leaf I ψ = ($\exists\chi$. $\neg$ *I* $\models$ χ $\wedge$ $\chi \in \psi$ $\wedge$ *total-over-m I* $\{\chi\}$) |
partial-interps (*Node v ag ad*) *I* ψ $\longleftrightarrow$
(*partial-interps ag* (*I* $\cup$ {*Pos v*}) ψ $\wedge$ *partial-interps ad* (*I* $\cup$ {*Neg v*}) ψ)

lemma *simplify-preserve-partial-leaf*:
  simplify $N\ N' \Longrightarrow$ *partial-interps Leaf* $I\ N \Longrightarrow$ *partial-interps Leaf* $I\ N'$
  ⟨*proof*⟩

lemma *simplify-preserve-partial-tree*:
  assumes *simplify* $N\ N'$
  and *partial-interps* $t\ I\ N$
  shows *partial-interps* $t\ I\ N'$
  ⟨*proof*⟩

lemma *inference-preserve-partial-tree*:
  assumes *inference* $S\ S'$
  and *partial-interps* $t\ I$ (*fst* S)
  shows *partial-interps* $t\ I$ (*fst* S')
  ⟨*proof*⟩

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')
  ⟨*proof*⟩

function *build-sem-tree* :: *'v* :: *linorder set* ⇒ *'v clauses* ⇒ *'v sem-tree* **where**
build-sem-tree atms ψ =
  (*if atms* = {} ∨ ¬ *finite atms*
  then Leaf
  else Node (*Min atms*) (*build-sem-tree* (*Set.remove* (*Min atms*) *atms*) ψ)
    (*build-sem-tree* (*Set.remove* (*Min atms*) *atms*) ψ))
⟨*proof*⟩
termination
  ⟨*proof*⟩
declare *build-sem-tree.induct*[*case-names tree*]

lemma *unsatisfiable-empty*[*simp*]:
  ¬*unsatisfiable* {}
  ⟨*proof*⟩

lemma *partial-interps-build-sem-tree-atms-general*:
  fixes ψ :: *'v* :: *linorder clauses* **and** *p* :: *'v literal list*
  assumes *unsat*: *unsatisfiable* ψ **and** *finite* ψ **and** *consistent-interp* I
  and *finite atms*
  and *atms-of-ms* ψ = *atms* ∪ *atms-of-s* I **and** *atms* ∩ *atms-of-s* I = {}
  shows *partial-interps* (*build-sem-tree atms* ψ) $I\ \psi$
  ⟨*proof*⟩

lemma *partial-interps-build-sem-tree-atms*:
  fixes ψ :: *'v* :: *linorder clauses* **and** *p* :: *'v literal list*
  assumes *unsat*: *unsatisfiable* ψ **and** *finite*: *finite* ψ
  shows *partial-interps* (*build-sem-tree* (*atms-of-ms* ψ) ψ) {} ψ
⟨*proof*⟩

lemma *can-decrease-count*:
fixes $\psi'' :: 'v\ clauses \times ('v\ clause \times 'v\ clause \times 'v)\ set$
assumes $count\ \chi\ L = n$
and $L \in\#\ \chi$ **and** $\chi \in fst\ \psi$
shows $\exists \psi'\ \chi'.\ inference^{**}\ \psi\ \psi' \wedge \chi' \in fst\ \psi' \wedge (\forall L.\ L \in\#\ \chi \longleftrightarrow L \in\#\ \chi')$
$\wedge\ count\ \chi'\ L = 1$
$\wedge\ (\forall \varphi.\ \varphi \in fst\ \psi \longrightarrow \varphi \in fst\ \psi')$
$\wedge\ (I \models \chi \longleftrightarrow I \models \chi')$
$\wedge\ (\forall I'.\ total\text{-}over\text{-}m\ I'\ \{\chi\} \longrightarrow total\text{-}over\text{-}m\ I'\ \{\chi'\})$
⟨*proof*⟩

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

lemma *inference-completeness-inv*:
fixes $\psi :: 'v :: linorder\ state$
assumes
unsat: $\neg satisfiable\ (fst\ \psi)$ **and**
finite: $finite\ (fst\ \psi)$ **and**
a-u-v: $already\text{-}used\text{-}inv\ \psi$
shows $\exists \psi'.\ (inference^{**}\ \psi\ \psi' \wedge \{\#\} \in fst\ \psi')$
⟨*proof*⟩

lemma *inference-completeness*:
fixes $\psi :: 'v :: linorder\ state$
assumes *unsat*: $\neg satisfiable\ (fst\ \psi)$
and *finite*: $finite\ (fst\ \psi)$
and $snd\ \psi = \{\}$
shows $\exists \psi'.\ (rtranclp\ inference\ \psi\ \psi' \wedge \{\#\} \in fst\ \psi')$
⟨*proof*⟩

lemma *inference-soundness*:
fixes $\psi :: 'v :: linorder\ state$
assumes $rtranclp\ inference\ \psi\ \psi'$ **and** $\{\#\} \in fst\ \psi'$
shows $unsatisfiable\ (fst\ \psi)$
⟨*proof*⟩

lemma *inference-soundness-and-completeness*:
fixes $\psi :: 'v :: linorder\ state$
assumes *finite*: $finite\ (fst\ \psi)$
and $snd\ \psi = \{\}$
shows $(\exists \psi'.\ (inference^{**}\ \psi\ \psi' \wedge \{\#\} \in fst\ \psi')) \longleftrightarrow unsatisfiable\ (fst\ \psi)$
⟨*proof*⟩

1.4 Lemma about the simplified state

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

lemma *simplified-count*:
assumes *simp*: $simplified\ \psi$ **and** χ: $\chi \in \psi$

shows *count* χ $L \leq 1$
⟨*proof*⟩

lemma *simplified-no-both*:
assumes *simp*: *simplified* ψ **and** χ: $\chi \in \psi$
shows $\neg$ $(L \in\# \chi \wedge -L \in\# \chi)$
⟨*proof*⟩

lemma *simplified-not-tautology*:
assumes *simplified* $\{\psi\}$
shows $^{\sim}$*tautology* ψ
⟨*proof*⟩

lemma *simplified-remove*:
assumes *simplified* $\{\psi\}$
shows *simplified* $\{\psi - \{\#l\#\}\}$
⟨*proof*⟩

lemma *in-simplified-simplified*:
assumes *simp*: *simplified* ψ **and** *incl*: $\psi' \subseteq \psi$
shows *simplified* ψ'
⟨*proof*⟩

lemma *simplified-in*:
assumes *simplified* ψ
and $N \in \psi$
shows *simplified* $\{N\}$
⟨*proof*⟩

lemma *subsumes-imp-formula*:
assumes $\psi \leq\# \varphi$
shows $\{\psi\} \models p$ φ
⟨*proof*⟩

lemma *simplified-imp-distinct-mset-tauto*:
assumes *simp*: *simplified* ψ'
shows *distinct-mset-set* ψ' **and** $\forall \chi \in \psi'$. $\neg$*tautology* χ
⟨*proof*⟩

lemma *simplified-no-more-full1-simplified*:
assumes *simplified* ψ
shows $\neg$*full1 simplify* ψ ψ'
⟨*proof*⟩

1.5 Resolution and Invariants

inductive *resolution* :: *'v state* $\Rightarrow$ *'v state* $\Rightarrow$ *bool* **where**
full1-simp: *full1 simplify* N N' $\Longrightarrow$ *resolution* $(N$, *already-used*$)$ $(N'$, *already-used*$)$ |
inferring: *inference* $(N$, *already-used*$)$ $(N'$, *already-used'*$)$ $\Longrightarrow$ *simplified* N
$\Longrightarrow$ *full simplify* N' N'' $\Longrightarrow$ *resolution* $(N$, *already-used*$)$ $(N''$, *already-used'*$)$

1.5.1 Invariants

lemma *resolution-finite*:
assumes *resolution* ψ ψ' **and** *finite* $(fst\ \psi)$

shows *finite* (*fst* ψ')
⟨*proof*⟩

lemma *rtranclp-resolution-finite*:
assumes *resolution*** ψ ψ' **and** *finite* (*fst* ψ)
shows *finite* (*fst* ψ')
⟨*proof*⟩

lemma *resolution-finite-snd*:
assumes *resolution* ψ ψ' **and** *finite* (*snd* ψ)
shows *finite* (*snd* ψ')
⟨*proof*⟩

lemma *rtranclp-resolution-finite-snd*:
assumes *resolution*** ψ ψ' **and** *finite* (*snd* ψ)
shows *finite* (*snd* ψ')
⟨*proof*⟩

lemma *resolution-always-simplified*:
assumes *resolution* ψ ψ'
shows *simplified* (*fst* ψ')
⟨*proof*⟩

lemma *tranclp-resolution-always-simplified*:
assumes *tranclp resolution* ψ ψ'
shows *simplified* (*fst* ψ')
⟨*proof*⟩

lemma *resolution-atms-of*:
assumes *resolution* ψ ψ' **and** *finite* (*fst* ψ)
shows *atms-of-ms* (*fst* ψ') $\subseteq$ *atms-of-ms* (*fst* ψ)
⟨*proof*⟩

lemma *rtranclp-resolution-atms-of*:
assumes *resolution*** ψ ψ' **and** *finite* (*fst* ψ)
shows *atms-of-ms* (*fst* ψ') $\subseteq$ *atms-of-ms* (*fst* ψ)
⟨*proof*⟩

lemma *resolution-include*:
assumes *res*: *resolution* ψ ψ' **and** *finite*: *finite* (*fst* ψ)
shows *fst* ψ' $\subseteq$ *simple-clss* (*atms-of-ms* (*fst* ψ))
⟨*proof*⟩

lemma *rtranclp-resolution-include*:
assumes *res*: *tranclp resolution* ψ ψ' **and** *finite*: *finite* (*fst* ψ)
shows *fst* ψ' $\subseteq$ *simple-clss* (*atms-of-ms* (*fst* ψ))
⟨*proof*⟩

abbreviation *already-used-all-simple*
:: (*'a literal multiset* $\times$ *'a literal multiset*) *set* $\Rightarrow$ *'a set* $\Rightarrow$ *bool* **where**
already-used-all-simple already-used vars $\equiv$
($\forall$ (A, B) $\in$ *already-used*. *simplified* $\{A\}$ $\wedge$ *simplified* $\{B\}$ $\wedge$ *atms-of* A $\subseteq$ *vars* $\wedge$ *atms-of* B $\subseteq$ *vars*)

lemma *already-used-all-simple-vars-incl*:
assumes *vars* $\subseteq$ *vars'*

shows *already-used-all-simple a vars* $\Longrightarrow$ *already-used-all-simple a vars'*
⟨*proof*⟩

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* $\cup$ {*fst S'*}, *snd S'*)) *vars*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *rtranclp-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*
⟨*proof*⟩

lemma *inference-clause-simplified-already-used-subset*:
assumes *inference-clause S S'*
and *simplified* (*fst S*)
shows *snd S* $\subset$ *snd S'*
⟨*proof*⟩

lemma *inference-simplified-already-used-subset*:
assumes *inference S S'*
and *simplified* (*fst S*)
shows *snd S* $\subset$ *snd S'*
⟨*proof*⟩

lemma *resolution-simplified-already-used-subset*:
assumes *resolution S S'*
and *simplified* (*fst S*)
shows *snd S* $\subset$ *snd S'*
⟨*proof*⟩

lemma *tranclp-resolution-simplified-already-used-subset*:
assumes *tranclp resolution S S'*
and *simplified* (*fst S*)

shows *snd* $S \subset$ *snd* S'
⟨*proof*⟩

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 $s \subseteq$ *already-used-top vars*
⟨*proof*⟩

lemma *already-used-top-finite*:
assumes *finite vars*
shows *finite* (*already-used-top vars*)
⟨*proof*⟩

lemma *already-used-top-increasing*:
assumes *var* $\subseteq$ *var'* **and** *finite var'*
shows *already-used-top var* $\subseteq$ *already-used-top var'*
⟨*proof*⟩

lemma *already-used-all-simple-finite*:
fixes *s* :: (*'a literal multiset* $\times$ *'a literal multiset*) *set* **and** *vars* :: *'a set*
assumes *already-used-all-simple s vars* **and** *finite vars*
shows *finite s*
⟨*proof*⟩

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

lemma *resolution-card-simple-decreasing*:
assumes *res*: *resolution* ψ ψ'
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) $\subseteq$ *vars*
shows *card-simple vars* (*snd* ψ') $<$ *card-simple vars* (*snd* ψ)
⟨*proof*⟩

lemma *tranclp-resolution-card-simple-decreasing*:
assumes *tranclp resolution* ψ ψ' **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)
⟨*proof*⟩

lemma *tranclp-resolution-card-simple-decreasing-2*:
assumes *tranclp resolution* ψ ψ'
and *finite-fst*: *finite* (*fst* ψ)
and *empty-snd*: *snd* $\psi = \{\}$

and *simplified* (*fst* ψ)
shows *card-simple* (*atms-of-ms* (*fst* ψ)) (*snd* ψ') $<$ *card-simple* (*atms-of-ms* (*fst* ψ)) (*snd* ψ)
⟨*proof*⟩

1.5.2 well-foundness if the relation

lemma *wf-simplified-resolution*:
assumes *f-vars*: *finite vars*
shows *wf* $\{(y::\ 'v::\ linorder\ state,\ x).\ (atms\text{-}of\text{-}ms\ (fst\ x) \subseteq vars \wedge simplified\ (fst\ x)$
$\wedge\ finite\ (snd\ x) \wedge finite\ (fst\ x) \wedge already\text{-}used\text{-}all\text{-}simple\ (snd\ x)\ vars) \wedge resolution\ x\ y\}$
⟨*proof*⟩

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

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

lemma *rtrancp-simplify-already-used-inv*:
assumes $simplify^{**}\ S\ S'$
and *already-used-inv* (*S*, *N*)
shows *already-used-inv* (S', *N*)
⟨*proof*⟩

lemma *full1-simplify-already-used-inv*:
assumes *full1 simplify* $S\ S'$
and *already-used-inv* (*S*, *N*)
shows *already-used-inv* (S', *N*)
⟨*proof*⟩

lemma *full-simplify-already-used-inv*:
assumes *full simplify* $S\ S'$
and *already-used-inv* (*S*, *N*)
shows *already-used-inv* (S', *N*)
⟨*proof*⟩
lemma *resolution-already-used-inv*:
assumes *resolution* $S\ S'$
and *already-used-inv S*
shows *already-used-inv* S'
⟨*proof*⟩

lemma *rtranclp-resolution-already-used-inv*:
assumes $resolution^{**}\ S\ S'$
and *already-used-inv S*
shows *already-used-inv* S'
⟨*proof*⟩

lemma *rtanclp-simplify-preserves-unsat*:

assumes *simplify*** ψ ψ'
shows *satisfiable* $\psi' \longrightarrow$ *satisfiable* ψ
$\langle proof \rangle$

lemma *full1-simplify-preserves-unsat*:
assumes *full1 simplify* ψ ψ'
shows *satisfiable* $\psi' \longrightarrow$ *satisfiable* ψ
$\langle proof \rangle$

lemma *full-simplify-preserves-unsat*:
assumes *full simplify* ψ ψ'
shows *satisfiable* $\psi' \longrightarrow$ *satisfiable* ψ
$\langle proof \rangle$

lemma *resolution-preserves-unsat*:
assumes *resolution* ψ ψ'
shows *satisfiable* (*fst* ψ') $\longrightarrow$ *satisfiable* (*fst* ψ)
$\langle proof \rangle$

lemma *rtranclp-resolution-preserves-unsat*:
assumes *resolution*** ψ ψ'
shows *satisfiable* (*fst* ψ') $\longrightarrow$ *satisfiable* (*fst* ψ)
$\langle proof \rangle$

lemma *rtranclp-simplify-preserve-partial-tree*:
assumes *simplify*** N N'
and *partial-interps* t I N
shows *partial-interps* t I N'
$\langle proof \rangle$

lemma *full1-simplify-preserve-partial-tree*:
assumes *full1 simplify* N N'
and *partial-interps* t I N
shows *partial-interps* t I N'
$\langle proof \rangle$

lemma *full-simplify-preserve-partial-tree*:
assumes *full simplify* N N'
and *partial-interps* t I N
shows *partial-interps* t I N'
$\langle proof \rangle$

lemma *resolution-preserve-partial-tree*:
assumes *resolution* S S'
and *partial-interps* t I (*fst* S)
shows *partial-interps* t I (*fst* S')
$\langle proof \rangle$

lemma *rtranclp-resolution-preserve-partial-tree*:
assumes *resolution*** S S'
and *partial-interps* t I (*fst* S)
shows *partial-interps* t I (*fst* S')
$\langle proof \rangle$
thm *nat-less-induct nat.induct*

lemma *nat-ge-induct*[*case-names 0 Suc*]:
assumes $P\ 0$
and $(\bigwedge n.\ (\bigwedge m.\ m<Suc\ n \Longrightarrow P\ m) \Longrightarrow P\ (Suc\ n))$
shows $P\ n$
⟨*proof*⟩

lemma *wf-always-more-step-False*:
assumes $wf\ R$
shows $(\forall x.\ \exists z.\ (z,\ x)\in R) \Longrightarrow False$
⟨*proof*⟩

lemma *finite-finite-mset-element-of-mset*[*simp*]:
assumes $finite\ N$
shows $finite\ \{f\ \varphi\ L\ |\varphi\ L.\ \varphi \in N \wedge L \in\#\ \varphi \wedge P\ \varphi\ L\}$
⟨*proof*⟩

value *card*
value *filter-mset*
value $\{\#count\ \varphi\ L\ |L \in\#\ \varphi.\ 2 \le count\ \varphi\ L\#\}$
value $(\lambda\varphi.\ msetsum\ \{\#count\ \varphi\ L\ |L \in\#\ \varphi.\ 2 \le count\ \varphi\ L\#\})$

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

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

interpretation *sum-count-ge-2*:
$folding\ (\lambda\varphi.\ op\ +(msetsum\ \{\#count\ \varphi\ L\ |L \in\#\ \varphi.\ 2 \le count\ \varphi\ L\#\}))\ 0$
rewrites
$folding.F\ (\lambda\varphi.\ op\ +(msetsum\ \{\#count\ \varphi\ L\ |L \in\#\ \varphi.\ 2 \le count\ \varphi\ L\#\}))\ 0 = sum\text{-}count\text{-}ge\text{-}2$
⟨*proof*⟩

lemma *finite-incl-le-setsum*:
$finite\ (B::'a\ multiset\ set) \Longrightarrow A \subseteq B \Longrightarrow \Xi\ A \le \Xi\ B$
⟨*proof*⟩

lemma *simplify-finite-measure-decrease*:
$simplify\ N\ N' \Longrightarrow finite\ N \Longrightarrow card\ N' + \Xi\ N' < card\ N + \Xi\ N$
⟨*proof*⟩

lemma *simplify-terminates*:
$wf\ \{(N',\ N).\ finite\ N \wedge simplify\ N\ N'\}$
⟨*proof*⟩

lemma *wf-terminates*:
assumes $wf\ r$

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

lemma *rtranclp-simplify-terminates*:
assumes *fin*: *finite N*
shows $\exists N'.$ *simplify*** $N\ N' \wedge$ *simplified* N'
$\langle proof \rangle$

lemma *finite-simplified-full1-simp*:
assumes *finite N*
shows *simplified N* $\vee$ ($\exists N'.$ *full1 simplify N* N')
$\langle proof \rangle$

lemma *finite-simplified-full-simp*:
assumes *finite N*
shows $\exists N'.$ *full simplify N* N'
$\langle proof \rangle$

lemma *can-decrease-tree-size-resolution*:
fixes ψ :: *'v state* **and** *tree* :: *'v sem-tree*
assumes *finite* (*fst* ψ) **and** *already-used-inv* ψ
and *partial-interps tree I* (*fst* ψ)
and *simplified* (*fst* ψ)
shows $\exists$(*tree'*:: *'v sem-tree*) ψ'. *resolution*** $\psi\ \psi' \wedge$ *partial-interps tree' I* (*fst* ψ')
$\wedge$ (*sem-tree-size tree'* $<$ *sem-tree-size tree* $\vee$ *sem-tree-size tree* $=$ *0*)
$\langle proof \rangle$

lemma *resolution-completeness-inv*:
fixes ψ :: *'v ::linorder state*
assumes
unsat: $\neg$*satisfiable* (*fst* ψ) **and**
finite: *finite* (*fst* ψ) **and**
a-u-v: *already-used-inv* ψ
shows $\exists \psi'.$ (*resolution*** $\psi\ \psi' \wedge \{\#\} \in$ *fst* ψ')
$\langle proof \rangle$

lemma *resolution-preserves-already-used-inv*:
assumes *resolution S S'*
and *already-used-inv S*
shows *already-used-inv S'*
$\langle proof \rangle$

lemma *rtranclp-resolution-preserves-already-used-inv*:
assumes *resolution*** *S S'*
and *already-used-inv S*
shows *already-used-inv S'*
$\langle proof \rangle$

lemma *resolution-completeness*:
fixes ψ :: *'v ::linorder state*
assumes *unsat*: $\neg$*satisfiable* (*fst* ψ)
and *finite*: *finite* (*fst* ψ)
and *snd* $\psi = \{\}$
shows $\exists \psi'.$ (*resolution*** $\psi\ \psi' \wedge \{\#\} \in$ *fst* ψ')
$\langle proof \rangle$

lemma *rtranclp-preserves-sat*:
  assumes *simplify*** S S'
  and *satisfiable* S
  shows *satisfiable* S'
  ⟨*proof*⟩

lemma *resolution-preserves-sat*:
  assumes *resolution* S S'
  and *satisfiable* (*fst* S)
  shows *satisfiable* (*fst* S')
  ⟨*proof*⟩

lemma *rtranclp-resolution-preserves-sat*:
  assumes *resolution*** S S'
  and *satisfiable* (*fst* S)
  shows *satisfiable* (*fst* S')
  ⟨*proof*⟩

lemma *resolution-soundness*:
  fixes ψ :: $'v$ ::*linorder state*
  assumes *resolution*** ψ ψ' **and** $\{\#\} \in$ *fst* ψ'
  shows *unsatisfiable* (*fst* ψ)
  ⟨*proof*⟩

lemma *resolution-soundness-and-completeness*:
fixes ψ :: $'v$ ::*linorder state*
assumes *finite*: *finite* (*fst* ψ)
and *snd*: *snd* $\psi = \{\}$
shows ($\exists\,\psi'$. (*resolution*** ψ ψ' $\wedge$ $\{\#\} \in$ *fst* ψ')) $\longleftrightarrow$ *unsatisfiable* (*fst* ψ)
  ⟨*proof*⟩

lemma *simplified-falsity*:
  assumes *simp*: *simplified* ψ
  and $\{\#\} \in \psi$
  shows $\psi = \{\{\#\}\}$
⟨*proof*⟩

lemma *simplify-falsity-in-preserved*:
  assumes *simplify* χs $\chi s'$
  and $\{\#\} \in \chi s$
  shows $\{\#\} \in \chi s'$
  ⟨*proof*⟩

lemma *rtranclp-simplify-falsity-in-preserved*:
  assumes *simplify*** χs $\chi s'$
  and $\{\#\} \in \chi s$
  shows $\{\#\} \in \chi s'$
  ⟨*proof*⟩

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

⟨*proof*⟩

lemma *resolution-soundness-and-completeness'*:
fixes ψ :: *'v* ::*linorder state*
assumes
finite: *finite* (*fst* ψ)**and**
snd: *snd* ψ = {}
shows ($\exists$ *a-u-v*. (*resolution*** ψ ({{#}}, *a-u-v*))) $\longleftrightarrow$ *unsatisfiable* (*fst* ψ)
⟨*proof*⟩

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*:
$S \models h\ C \longleftrightarrow \{Pos\ P|P.\ P{\in}S\} \cup \{Neg\ P|P.\ P{\notin}S\} \models C$
⟨*proof*⟩

lemma *herbrand-consistent-interp*:
consistent-interp $(\{Pos\ P|P.\ P{\in}S\} \cup \{Neg\ P|P.\ P{\notin}S\})$
⟨*proof*⟩

lemma *herbrand-total-over-set*:
total-over-set $(\{Pos\ P|P.\ P{\in}S\} \cup \{Neg\ P|P.\ P{\notin}S\})\ T$
⟨*proof*⟩

lemma *herbrand-total-over-m*:
total-over-m $(\{Pos\ P|P.\ P{\in}S\} \cup \{Neg\ P|P.\ P{\notin}S\})\ T$
⟨*proof*⟩

lemma *herbrand-interp-iff-partial-interp-clss*:
$S \models hs\ C \longleftrightarrow \{Pos\ P|P.\ P{\in}S\} \cup \{Neg\ P|P.\ P{\notin}S\} \models s\ C$
⟨*proof*⟩

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 (-<^*bsup*>-<^*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: $\bigwedge C\ L.\ L \in\#\ S\ C \Longrightarrow$ *is-neg* L

locale *ground-resolution-with-selection* =
  selection S **for** *S* :: (*'a* :: *wellorder*) *clause* ⇒ *'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* ⇒ *'a interp*
where
  production C =
   $\{A.\ C \in N \wedge C \neq \{\#\} \wedge Max\ (set\text{-}mset\ C) = Pos\ A \wedge count\ C\ (Pos\ A) \leq 1$
     $\wedge \neg (\bigcup D \in \{D.\ D \#{\subset}\#\ C\}.\ production\ D) \models h\ C \wedge S\ C = \{\#\}\}$
  ⟨*proof*⟩
termination ⟨*proof*⟩

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

definition *interp* :: *'a clause* ⇒ *'a interp* **where**
  $interp\ C = (\bigcup D \in \{D.\ D \#{\subset}\#\ C\}.\ production\ D)$

lemma *production-unfold*:
  $production\ C = \{A.\ C \in N \wedge C \neq \{\#\} \wedge Max\ (set\text{-}mset\ C) = Pos\ A \wedge count\ C\ (Pos\ A) \leq 1 \wedge \neg$
$interp\ C \models h\ C \wedge S\ C = \{\#\}\}$
  ⟨*proof*⟩

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

abbreviation *produces* :: *'a clause* ⇒ *'a* ⇒ *bool* **where**
  $produces\ C\ A \equiv production\ C = \{A\}$

lemma *producesD*:
  $produces\ C\ A \Longrightarrow C \in N \wedge C \neq \{\#\} \wedge Pos\ A = Max\ (set\text{-}mset\ C) \wedge count\ C\ (Pos\ A) \leq 1 \wedge$
   $\neg\ interp\ C \models h\ C \wedge S\ C = \{\#\}$
  ⟨*proof*⟩

lemma $produces\ C\ A \Longrightarrow Pos\ A \in\#\ C$
  ⟨*proof*⟩

lemma *interp'-def-in-set*:
  $interp\ C = (\bigcup D \in \{D \in N.\ D \#{\subset}\#\ C\}.\ production\ D)$
  ⟨*proof*⟩

lemma *production-iff-produces*:
  $produces\ D\ A \longleftrightarrow A \in production\ D$
  ⟨*proof*⟩

definition *Interp* :: *'a clause* ⇒ *'a interp* **where**
  $Interp\ C = interp\ C \cup production\ C$

lemma
  assumes *produces C P*
  shows $Interp\ C \models h\ C$

$\langle proof \rangle$

definition *INTERP* :: *'a interp* **where**
$INTERP = (\bigcup D \in N.\ production\ D)$

lemma *interp-subseteq-Interp*[*simp*]: $interp\ C \subseteq Interp\ C$
$\langle proof \rangle$

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

lemma *productive-not-empty*: $productive\ C \Longrightarrow C \neq \{\#\}$
$\langle proof \rangle$

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

lemma *productive-imp-produces-Max-atom*: $productive\ C \Longrightarrow produces\ C\ (Max\ (atms\text{-}of\ C))$
$\langle proof \rangle$

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

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

lemma *produces-imp-Pos-in-lits*: $produces\ C\ A \Longrightarrow Pos\ A \in\#\ C$
$\langle proof \rangle$

lemma *productive-in-N*: $productive\ C \Longrightarrow C \in N$
$\langle proof \rangle$

lemma *produces-imp-atms-leq*: $produces\ C\ A \Longrightarrow B \in atms\text{-}of\ C \Longrightarrow B \leq A$
$\langle proof \rangle$

lemma *produces-imp-neg-notin-lits*: $produces\ C\ A \Longrightarrow \neg\ Neg\ A \in\#\ C$
$\langle proof \rangle$

lemma *less-eq-imp-interp-subseteq-interp*: $C \ \#{\subseteq}\#\ D \Longrightarrow interp\ C \subseteq interp\ D$
$\langle proof \rangle$

lemma *less-eq-imp-interp-subseteq-Interp*: $C \ \#{\subseteq}\#\ D \Longrightarrow interp\ C \subseteq Interp\ D$
$\langle proof \rangle$

lemma *less-imp-production-subseteq-interp*: $C \ \#{\subset}\#\ D \Longrightarrow production\ C \subseteq interp\ D$
$\langle proof \rangle$

lemma *less-eq-imp-production-subseteq-Interp*: $C \ \#{\subseteq}\#\ D \Longrightarrow production\ C \subseteq Interp\ D$
$\langle proof \rangle$

lemma *less-imp-Interp-subseteq-interp*: $C \ \#{\subset}\#\ D \Longrightarrow Interp\ C \subseteq interp\ D$
$\langle proof \rangle$

lemma *less-eq-imp-Interp-subseteq-Interp*: $C \ \#{\subseteq}\#\ D \Longrightarrow Interp\ C \subseteq Interp\ D$

⟨*proof*⟩

lemma *false-Interp-to-true-interp-imp-less-multiset*: $A \notin \mathit{Interp}\ C \Longrightarrow A \in \mathit{interp}\ D \Longrightarrow C\ \#{\subset}\#\ D$
⟨*proof*⟩

lemma *false-interp-to-true-interp-imp-less-multiset*: $A \notin \mathit{interp}\ C \Longrightarrow A \in \mathit{interp}\ D \Longrightarrow C\ \#{\subset}\#\ D$
⟨*proof*⟩

lemma *false-Interp-to-true-Interp-imp-less-multiset*: $A \notin \mathit{Interp}\ C \Longrightarrow A \in \mathit{Interp}\ D \Longrightarrow C\ \#{\subset}\#\ D$
⟨*proof*⟩

lemma *false-interp-to-true-Interp-imp-le-multiset*: $A \notin \mathit{interp}\ C \Longrightarrow A \in \mathit{Interp}\ D \Longrightarrow C\ \#{\subseteq}\#\ D$
⟨*proof*⟩

lemma *interp-subseteq-INTERP*: $\mathit{interp}\ C \subseteq \mathit{INTERP}$
⟨*proof*⟩

lemma *production-subseteq-INTERP*: $\mathit{production}\ C \subseteq \mathit{INTERP}$
⟨*proof*⟩

lemma *Interp-subseteq-INTERP*: $\mathit{Interp}\ C \subseteq \mathit{INTERP}$
⟨*proof*⟩

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

lemma *produces-imp-in-interp*:
assumes *a-in-c*: $\mathit{Neg}\ A \in\#\ C$ **and** *d*: $\mathit{produces}\ D\ A$
shows $A \in \mathit{interp}\ C$
⟨*proof*⟩

lemma *neg-notin-Interp-not-produce*: $\mathit{Neg}\ A \in\#\ C \Longrightarrow A \notin \mathit{Interp}\ D \Longrightarrow C\ \#{\subseteq}\#\ D \Longrightarrow \neg\ \mathit{produces}\ D''\ A$
⟨*proof*⟩

lemma *in-production-imp-produces*: $A \in \mathit{production}\ C \Longrightarrow \mathit{produces}\ C\ A$
⟨*proof*⟩

lemma *not-produces-imp-notin-production*: $\neg\ \mathit{produces}\ C\ A \Longrightarrow A \notin \mathit{production}\ C$
⟨*proof*⟩

lemma *not-produces-imp-notin-interp*: $(\bigwedge D.\ \neg\ \mathit{produces}\ D\ A) \Longrightarrow A \notin \mathit{interp}\ C$
⟨*proof*⟩

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: $\mathit{Interp}\ D \models h\ C$ **and**
subs: $\mathit{interp}\ D' \subseteq (\bigcup C \in CC.\ \mathit{production}\ C)$
shows $(\bigcup C \in CC.\ \mathit{production}\ C) \models h\ C$
⟨*proof*⟩

lemma *true-Interp-imp-interp*: $C\ \#{\subseteq}\#\ D \Longrightarrow D\ \#{\subset}\#\ D' \Longrightarrow \mathit{Interp}\ D \models h\ C \Longrightarrow \mathit{interp}\ D' \models h\ C$
⟨*proof*⟩

lemma *true-Interp-imp-Interp*: $C \ \#{\subseteq}\# \ D \Longrightarrow D \ \#{\subset}\# \ D' \Longrightarrow \mathit{Interp}\ D \models h\ C \Longrightarrow \mathit{Interp}\ D' \models h\ C$
⟨*proof*⟩

lemma *true-Interp-imp-INTERP*: $C \ \#{\subseteq}\# \ D \Longrightarrow \mathit{Interp}\ D \models h\ C \Longrightarrow \mathit{INTERP} \models h\ C$
⟨*proof*⟩

lemma *true-interp-imp-general*:
assumes
c-le-d: $C \ \#{\subseteq}\# \ D$ **and**
d-lt-d': $D \ \#{\subset}\# \ D'$ **and**
c-at-d: $\mathit{interp}\ D \models h\ C$ **and**
subs: $\mathit{interp}\ D' \subseteq (\bigcup C \in CC.\ \mathit{production}\ C)$
shows $(\bigcup C \in CC.\ \mathit{production}\ C) \models h\ C$
⟨*proof*⟩

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*: $C \ \#{\subseteq}\# \ D \Longrightarrow D \ \#{\subset}\# \ D' \Longrightarrow \mathit{interp}\ D \models h\ C \Longrightarrow \mathit{interp}\ D' \models h\ C$
⟨*proof*⟩

lemma *true-interp-imp-Interp*: $C \ \#{\subseteq}\# \ D \Longrightarrow D \ \#{\subset}\# \ D' \Longrightarrow \mathit{interp}\ D \models h\ C \Longrightarrow \mathit{Interp}\ D' \models h\ C$
⟨*proof*⟩

lemma *true-interp-imp-INTERP*: $C \ \#{\subseteq}\# \ D \Longrightarrow \mathit{interp}\ D \models h\ C \Longrightarrow \mathit{INTERP} \models h\ C$
⟨*proof*⟩

lemma *productive-imp-false-interp*: $\mathit{productive}\ C \Longrightarrow \neg\ \mathit{interp}\ C \models h\ C$
⟨*proof*⟩

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: $\{\#\mathit{Pos}\ P,\ \mathit{Pos}\ P\#\} \ \#{\subset}\# \ C$
shows $\neg \mathit{produces}\ C\ P$
⟨*proof*⟩

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

lemma
assumes D: $C{+}\{\#\mathit{Neg}\ P\#\} \ \#{\subset}\# \ D$
shows $\mathit{production}\ D \neq \{P\}$
⟨*proof*⟩

lemma *in-interp-is-produced*:
assumes $P \in \mathit{INTERP}$
shows $\exists\, D.\ D\ {+}\{\#\mathit{Pos}\ P\#\} \in N \wedge \mathit{produces}\ (D\ {+}\{\#\mathit{Pos}\ P\#\})\ P$
⟨*proof*⟩

end
end

abbreviation $\mathit{MMax}\ M \equiv \mathit{Max}\ (\mathit{set\text{-}mset}\ M)$

2.1 We can now define the rules of the calculus

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

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

lemma *less-multiset*[*iff*]: $M < N \longleftrightarrow M \#{\subset}\# N$
⟨*proof*⟩

lemma *less-eq-multiset*[*iff*]: $M \leq N \longleftrightarrow M \#{\subseteq}\# N$
⟨*proof*⟩

lemma *herbrand-true-clss-true-clss-cls-herbrand-true-clss*:
assumes
AB: $A \models hs\ B$ **and**
BC: $B \models p\ C$
shows $A \models h\ C$
⟨*proof*⟩

lemma *abstract-red-subset-mset-abstract-red*:
assumes
abstr: *abstract-red* $C\ N$ **and**
c-lt-d: $C \subseteq\# D$
shows *abstract-red* $D\ N$
⟨*proof*⟩

lemma *true-clss-cls-extended*:
assumes
$A \models p\ B$ **and**
tot: *total-over-m* $I\ (A)$ **and**
cons: *consistent-interp* I **and**
I-A: $I \models s\ A$
shows $I \models B$
⟨*proof*⟩
lemma
assumes
CP: $\neg$ *clss-lt* $N\ (\{\#C\#\} + \{\#E\#\}) \models p\ \{\#C\#\} + \{\#Neg\ P\#\}$ **and**
clss-lt $N\ (\{\#C\#\} + \{\#E\#\}) \models p\ \{\#E\#\} + \{\#Pos\ P\#\} \vee$ *clss-lt* $N\ (\{\#C\#\} + \{\#E\#\}) \models p$ $\{\#C\#\} + \{\#Neg\ P\#\}$
shows *clss-lt* $N\ (\{\#C\#\} + \{\#E\#\}) \models p\ \{\#E\#\} + \{\#Pos\ P\#\}$

⟨*proof*⟩

locale *ground-ordered-resolution-with-redundancy* =
ground-resolution-with-selection +
fixes *redundant* :: *'a::wellorder clause* ⇒ *'a clauses* ⇒ *bool*
assumes
redundant-iff-abstract: *redundant* $A\ N \longleftrightarrow$ *abstract-red* $A\ N$

begin
definition *saturated* :: $'a$ *clauses* $\Rightarrow$ *bool* **where**
saturated $N \longleftrightarrow (\forall\, A\ B\ C.\ A \in N \longrightarrow B \in N \longrightarrow \neg$*redundant* $A\ N \longrightarrow \neg$*redundant* $B\ N$
  $\longrightarrow$ *superposition-rules* $A\ B\ C \longrightarrow$ *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*⟩

end

lemma *tautology-is-redundant*:
  assumes *tautology* C
  shows *abstract-red* $C\ N$
  ⟨*proof*⟩

lemma *subsumed-is-redundant*:
  assumes *AB*: $A \subset\# B$
  and *AN*: $A \in N$
  shows *abstract-red* $B\ N$
⟨*proof*⟩

inductive *redundant* :: $'a$ *clause* $\Rightarrow$ $'a$ *clauses* $\Rightarrow$ *bool* **where**
subsumption: $A \in N \Longrightarrow A \subset\# B \Longrightarrow$ *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* $A\ N$
  ⟨*proof*⟩

lemma *redundant-mono*:
  redundant $A\ N \Longrightarrow A \subseteq\# B \Longrightarrow$ *redundant* $B\ N$
  ⟨*proof*⟩

locale *truc* =
    selection S **for** S :: *nat clause* $\Rightarrow$ *nat clause*
begin

end

end