Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

April 20, 2016

Contents

theory *Prop-Resolution*
imports *Partial-Clausal-Logic List-More Wellfounded-More*

begin

Chapter 1

Resolution-based techniques

This chapter contains the formalisation of resolution and superposition.

1.1 Resolution

1.1.1 Simplification Rules

inductive *simplify* :: *'v clauses ⇒ 'v clauses ⇒ bool* **for** *N* :: *'v clause set* **where**
tautology-deletion:
A + {#Pos P#} + {#Neg P#} ∈ N ⟹ simplify N (N − {A + {#Pos P#} + {#Neg P#}})|
condensation:
A + {#L#} + {#L#} ∈ N ⟹ simplify N (N − {A + {#L#} + {#L#}} ∪ {A + {#L#}}) |
subsumption:
A ∈ N ⟹ A ⊂# B ⟹ B ∈ N ⟹ 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 ⊨s N' ⟶ I ⊨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 ⊨s N ⟶ I ⊨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 ⊨s N ⟶ I ⊨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 ⊨s N ⟷ I ⊨s N'*

⟨*proof*⟩

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

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

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

lemma *rtranclp-simplify-atms-of-ms*:
assumes *rtranclp simplify* ψ ψ'
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*⟩

1.1.2 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 $\psi \in S'$
⟨*proof*⟩

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

Subsumption

definition *subsumes* :: *'a literal multiset* ⇒ *'a literal multiset* ⇒ *bool* **where**
subsumes χ $\chi' \longleftrightarrow$
$(\forall I.$ *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

⟨*proof*⟩

lemma *subsumes-subsumption*:
assumes *subsumes D χ*
and *C* ⊂# *D* **and** ¬*tautology χ*
shows *subsumes C χ* ⟨*proof*⟩

lemma *subsumes-tautology*:
assumes *subsumes* (*C* + {#*Pos P*#} + {#*Neg P*#}) *χ*
shows *tautology χ*
⟨*proof*⟩

1.1.3 Inference Rule

type-synonym *'v state* = *'v clauses* × (*'v clause* × *'v clause*) *set*
inductive *inference-clause* :: *'v state* ⇒ *'v clause* × (*'v clause* × *'v clause*) *set* ⇒ *bool*
(**infix** ⇒Res *100*) **where**
resolution:
{#*Pos p*#} + *C* ∈ *N* ⟹ {#*Neg p*#} + *D* ∈ *N* ⟹ ({#*Pos p*#} + *C*, {#*Neg p*#} + *D*) ∉ *already-used*
⟹ *inference-clause* (*N*, *already-used*) (*C* + *D*, *already-used* ∪ {({#*Pos p*#} + *C*, {#*Neg p*#} + *D*)}) |
factoring: {#*L*#} + {#*L*#} + *C* ∈ *N* ⟹ *inference-clause* (*N*, *already-used*) (*C* + {#*L*#}, *already-used*)

inductive *inference* :: *'v state* ⇒ *'v state* ⇒ *bool* **where**
inference-step: *inference-clause S* (*clause*, *already-used*)
⟹ *inference S* (*fst S* ∪ {*clause*}, *already-used*)

abbreviation *already-used-inv*
:: *'a literal multiset set* × (*'a literal multiset* × *'a literal multiset*) *set* ⇒ *bool* **where**
already-used-inv state ≡
(∀(*A*, *B*) ∈ *snd state*. ∃ *p*. *Pos p* ∈# *A* ∧ *Neg p* ∈# *B* ∧
((∃ *χ* ∈ *fst state*. *subsumes χ* ((*A* − {#*Pos p*#}) + (*B* − {#*Neg p*#})))
∨ *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* ∪ {*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-used*$)$
shows *already-used-inv* $(N',$ *already-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-of* $(\{\#L\#\} + \{\#L\#\} + C) =$ *atms-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* $\psi\ \psi'$ **and** *finite* (*snd* ψ)
shows *finite* (*snd* ψ')
⟨*proof*⟩

lemma *rtranclp-inference-preserves-finite*:
assumes *rtranclp inference* $\psi\ \psi'$ **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* ⇒ *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*]:
(⋀*xs*:: *'v sem-tree*. (⋀*ys*:: *'v sem-tree*. *sem-tree-size ys* < *sem-tree-size xs* ⟹ *P ys*) ⟹ *P xs*)
⟹ *P xs*
⟨*proof*⟩

fun *partial-interps* :: *'v sem-tree* ⇒ *'v interp* ⇒ *'v clauses* ⇒ *bool* **where**
partial-interps Leaf I ψ = (∃ χ. ¬ *I* ⊨ χ ∧ χ ∈ ψ ∧ *total-over-m I* {χ}) |
partial-interps (*Node v ag ad*) *I* ψ ⟷
(*partial-interps ag* (*I* ∪ {*Pos v*}) ψ ∧ *partial-interps ad* (*I*∪ {*Neg v*}) ψ)

lemma *simplify-preserve-partial-leaf*:
simplify N N' ⟹ *partial-interps Leaf I N* ⟹ *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 ψ*
⟨*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 *ψ''* :: *'v clauses* × (*'v clause* × *'v clause* × *'v*) *set*
assumes *count χ L* = *n*
and *L* ∈# *χ* **and** *χ* ∈ *fst ψ*
shows ∃ *ψ' χ'*. *inference*** *ψ ψ'* ∧ *χ'* ∈ *fst ψ'* ∧ (∀ *L*. *L* ∈# *χ* ⟷ *L* ∈# *χ'*)
∧ *count χ' L* = *1*
∧ (∀ *φ*. *φ* ∈ *fst ψ* ⟶ *φ* ∈ *fst ψ'*)
∧ (*I* ⊨ *χ* ⟷ *I* ⊨ *χ'*)
∧ (∀ *I'*. *total-over-m I'* {*χ*} ⟶ *total-over-m I'* {*χ'*})
⟨*proof*⟩

lemma *can-decrease-tree-size*:
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*)
shows ∃ (*tree'*:: *'v sem-tree*) *ψ'*. *inference*** *ψ ψ'* ∧ *partial-interps tree' I* (*fst ψ'*)
∧ (*sem-tree-size tree'* < *sem-tree-size tree* ∨ *sem-tree-size tree* = *0*)
⟨*proof*⟩

lemma *inference-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 $\exists \psi'$. (*inference*** ψ $\psi' \wedge \{\#\} \in$ *fst* ψ')
⟨*proof*⟩

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

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

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

1.1.4 Lemma about the simplified state

abbreviation *simplified* $\psi \equiv$ (*no-step simplify* ψ)

lemma *simplified-count*:
  assumes *simp*: *simplified* ψ **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 $\{\psi\} \models p\ \varphi$
⟨*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 ¬*full1 simplify* $\psi\ \psi'$
⟨*proof*⟩

1.1.5 Resolution and Invariants

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

Invariants

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

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

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

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

lemma *resolution-always-simplified*:
assumes *resolution* $\psi\ \psi'$

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') ⊆ *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') ⊆ *atms-of-ms* (*fst* ψ)
⟨*proof*⟩

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

abbreviation *already-used-all-simple*
:: ('*a literal multiset* × '*a literal multiset*) *set* ⇒ '*a set* ⇒ *bool* **where**
already-used-all-simple already-used vars ≡
(∀ (*A*, *B*) ∈ *already-used*. *simplified* {*A*} ∧ *simplified* {*B*} ∧ *atms-of A* ⊆ *vars* ∧ *atms-of B* ⊆ *vars*)

lemma *already-used-all-simple-vars-incl*:
assumes *vars* ⊆ *vars'*
shows *already-used-all-simple a vars* ⟹ *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*) ⊆ *vars*
shows *already-used-all-simple* (*snd* (*fst S* ∪ {*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*) ⊆ *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*) ⊆ *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*) ⊆ *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* ⊂ *snd S′*
⟨*proof*⟩

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

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

lemma *tranclp-resolution-simplified-already-used-subset*:
assumes *tranclp resolution S S′*
and *simplified* (*fst S*)
shows *snd S* ⊂ *snd S′*
⟨*proof*⟩

abbreviation *already-used-top vars* ≡ *simple-clss vars* × *simple-clss vars*

lemma *already-used-all-simple-in-already-used-top*:
assumes *already-used-all-simple s vars* **and** *finite vars*
shows *s* ⊆ *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* ⊆ *var′* **and** *finite var′*
shows *already-used-top var* ⊆ *already-used-top var′*
⟨*proof*⟩

lemma *already-used-all-simple-finite*:
fixes *s* :: (*′a literal multiset* × *′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 ψ* ≡ *card* (*already-used-top vars* − *ψ*)

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*) ⊆ *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*) ⊆ *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 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*⟩

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*) ⊆ *vars* ∧ *simplified* (*fst x*)
∧ *finite* (*snd x*) ∧ *finite* (*fst x*) ∧ *already-used-all-simple* (*snd x*) *vars*) ∧ *resolution x y*}
⟨*proof*⟩

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

lemma *wf-resolution*:
assumes *f-vars*: *finite vars*
shows *wf* ({(*y*:: *'v*:: *linorder state*, *x*). (*atms-of-ms* (*fst x*) ⊆ *vars* ∧ *simplified* (*fst x*)
∧ *finite* (*snd x*) ∧ *finite* (*fst x*) ∧ *already-used-all-simple* (*snd x*) *vars*) ∧ *resolution x y*}
∪ {(*y*, *x*). (*atms-of-ms* (*fst x*) ⊆ *vars* ∧ ¬ *simplified* (*fst x*) ∧ *finite* (*snd x*) ∧ *finite* (*fst x*)

$\wedge$ *already-used-all-simple* (*snd x*) *vars*) $\wedge$ *resolution x y*}) (**is** *wf* (*?R* $\cup$ *?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* ψ' $\longrightarrow$ *satisfiable* ψ
⟨*proof*⟩

lemma *full1-simplify-preserves-unsat*:
assumes *full1 simplify* ψ ψ'
shows *satisfiable* ψ' $\longrightarrow$ *satisfiable* ψ
⟨*proof*⟩

lemma *full-simplify-preserves-unsat*:
assumes *full simplify* ψ ψ'
shows *satisfiable* ψ' $\longrightarrow$ *satisfiable* ψ
⟨*proof*⟩

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

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

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

lemma *full1-simplify-preserve-partial-tree*:
assumes *full1 simplify N N'*
and *partial-interps t I N*
shows *partial-interps t I N'*
⟨*proof*⟩

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

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

lemma *rtranclp-resolution-preserve-partial-tree*:
assumes *resolution*** *S S'*
and *partial-interps t I* (*fst S*)
shows *partial-interps t I* (*fst S'*)
⟨*proof*⟩
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* φ *L* |φ *L*. $\varphi \in N \wedge L \in\# \varphi \wedge P\ \varphi\ L$}
⟨*proof*⟩

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

interpretation *sum-count-ge-2*:
folding ($\lambda\varphi.$ *op* +(*msetsum* {#*count* φ *L* |*L* $\in\#$ φ. *2* $\leq$ *count* φ *L*#})) *0*
rewrites

folding.F ($\lambda\varphi$. *op* +(*msetsum* {#*count* φ *L* |*L* ∈# φ. *2* ≤ *count* φ *L*#})) *0* = *sum-count-ge-2*
⟨*proof*⟩

lemma *finite-incl-le-setsum*:
finite (*B*::'*a multiset set*) ⟹ *A* ⊆ *B* ⟹ Ξ *A* ≤ Ξ *B*
⟨*proof*⟩

lemma *simplify-finite-measure-decrease*:
simplify N N' ⟹ *finite N* ⟹ *card N'* + Ξ *N'* < *card N* + Ξ *N*
⟨*proof*⟩

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

lemma *wf-terminates*:
assumes *wf r*
shows ∃ *N'*.(*N'*, *N*)∈ *r** ∧ (∀ *N''*. (*N''*, *N'*)∉ *r*)
⟨*proof*⟩

lemma *rtranclp-simplify-terminates*:
assumes *fin*: *finite N*
shows ∃ *N'*. *simplify*** *N N'* ∧ *simplified N'*
⟨*proof*⟩

lemma *finite-simplified-full1-simp*:
assumes *finite N*
shows *simplified N* ∨ (∃ *N'*. *full1 simplify N N'*)
⟨*proof*⟩

lemma *finite-simplified-full-simp*:
assumes *finite N*
shows ∃ *N'*. *full simplify N N'*
⟨*proof*⟩

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 ∃(*tree'*:: '*v sem-tree*) ψ'. *resolution*** ψ ψ' ∧ *partial-interps tree' I* (*fst* ψ')
∧ (*sem-tree-size tree'* < *sem-tree-size tree* ∨ *sem-tree-size tree* = *0*)
⟨*proof*⟩

lemma *resolution-completeness-inv*:
fixes ψ :: '*v* ::*linorder state*
assumes
unsat: ¬*satisfiable* (*fst* ψ) **and**
finite: *finite* (*fst* ψ) **and**
a-u-v: *already-used-inv* ψ
shows ∃ ψ'. (*resolution*** ψ ψ' ∧ {#} ∈ *fst* ψ')
⟨*proof*⟩

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

assumes *resolution S S′*
and *already-used-inv S*
shows *already-used-inv S′*
⟨*proof*⟩

lemma *rtranclp-resolution-preserves-already-used-inv*:
assumes *resolution*** *S S′*
and *already-used-inv S*
shows *already-used-inv S′*
⟨*proof*⟩

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

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 $\psi' \wedge \{\#\} \in$ *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* $\chi s\ \chi s'$
and $\{\#\} \in \chi s$
shows $\{\#\} \in \chi s'$
⟨*proof*⟩

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

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

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

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

1.2 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* ⇒ *'a clause* ⇒ *'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* ⇒ *'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* ≡ (*production A* ≠ {})

abbreviation *produces* :: *'a clause* ⇒ *'a* ⇒ *bool* **where**
produces C A ≡ *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 = (⋃ *D* ∈ {*D* ∈ *N*. *D* #⊂# *C*}. *production D*)
⟨*proof*⟩

lemma *production-iff-produces*:
produces D A ⟷ *A* ∈ *production D*
⟨*proof*⟩

definition *Interp* :: *'a clause* ⇒ *'a interp* **where**
Interp C = *interp C* ∪ *production C*

lemma
assumes *produces C P*
shows *Interp C* ⊨*h C*
⟨*proof*⟩

definition *INTERP* :: *'a interp* **where**
INTERP = (⋃ *D* ∈*N*. *production D*)

lemma *interp-subseteq-Interp*[*simp*]: *interp C* ⊆ *Interp C*
⟨*proof*⟩

lemma *Interp-as-UNION*: *Interp C* = (⋃ *D* ∈ {*D*. *D* #⊆# *C*}. *production D*)
⟨*proof*⟩

lemma *productive-not-empty*: *productive C* ⟹ *C* ≠ {#}
⟨*proof*⟩

lemma *productive-imp-produces-Max-literal*: *productive C* ⟹ *produces C* (*atm-of* (*Max* (*set-mset C*)))
⟨*proof*⟩

lemma *productive-imp-produces-Max-atom*: *productive C* ⟹ *produces C* (*Max* (*atms-of C*))
⟨*proof*⟩

lemma *produces-imp-Max-literal*: *produces C A* ⟹ *A* = *atm-of* (*Max* (*set-mset C*))
⟨*proof*⟩

lemma *produces-imp-Max-atom*: *produces C A* ⟹ *A* = *Max* (*atms-of C*)
⟨*proof*⟩

lemma *produces-imp-Pos-in-lits*: *produces C A* ⟹ *Pos A* ∈# *C*
⟨*proof*⟩

lemma *productive-in-N*: *productive C* ⟹ *C* ∈ *N*
⟨*proof*⟩

lemma *produces-imp-atms-leq*: *produces C A* ⟹ *B* ∈ *atms-of C* ⟹ *B* ≤ *A*
⟨*proof*⟩

lemma *produces-imp-neg-notin-lits*: *produces C A* ⟹ *Neg A* ∉# *C*
⟨*proof*⟩

lemma *less-eq-imp-interp-subseteq-interp*: *C* #⊆# *D* ⟹ *interp C* ⊆ *interp D*

⟨*proof*⟩

lemma *less-eq-imp-interp-subseteq-Interp*: $C \mathrel{\#\subseteq\#} D \Longrightarrow \mathit{interp}\ C \subseteq \mathit{Interp}\ D$
⟨*proof*⟩

lemma *less-imp-production-subseteq-interp*: $C \mathrel{\#\subset\#} D \Longrightarrow \mathit{production}\ C \subseteq \mathit{interp}\ D$
⟨*proof*⟩

lemma *less-eq-imp-production-subseteq-Interp*: $C \mathrel{\#\subseteq\#} D \Longrightarrow \mathit{production}\ C \subseteq \mathit{Interp}\ D$
⟨*proof*⟩

lemma *less-imp-Interp-subseteq-interp*: $C \mathrel{\#\subset\#} D \Longrightarrow \mathit{Interp}\ C \subseteq \mathit{interp}\ D$
⟨*proof*⟩

lemma *less-eq-imp-Interp-subseteq-Interp*: $C \mathrel{\#\subseteq\#} D \Longrightarrow \mathit{Interp}\ C \subseteq \mathit{Interp}\ D$
⟨*proof*⟩

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

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

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

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

lemma *neg-notin-Interp-not-produce*: $\mathit{Neg}\ A \mathrel{\in\#} C \Longrightarrow A \notin \mathit{Interp}\ D \Longrightarrow C \mathrel{\#\subseteq\#} D \Longrightarrow \neg\ \mathit{produces}\ D''\ A$
⟨*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 #⊆# D **and**
d-lt-d': D #⊂# 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*⟩

lemma *true-Interp-imp-interp*: C #⊆# $D \Longrightarrow D$ #⊂# $D' \Longrightarrow$ *Interp* $D \models h$ $C \Longrightarrow$ *interp* $D' \models h$ C
⟨*proof*⟩

lemma *true-Interp-imp-Interp*: C #⊆# $D \Longrightarrow D$ #⊂# $D' \Longrightarrow$ *Interp* $D \models h$ $C \Longrightarrow$ *Interp* $D' \models h$ C
⟨*proof*⟩

lemma *true-Interp-imp-INTERP*: C #⊆# $D \Longrightarrow$ *Interp* $D \models h$ $C \Longrightarrow$ *INTERP* $\models h$ C
⟨*proof*⟩

lemma *true-interp-imp-general*:
assumes
c-le-d: C #⊆# D **and**
d-lt-d': D #⊂# 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*⟩

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 #⊆# $D \Longrightarrow D$ #⊂# $D' \Longrightarrow$ *interp* $D \models h$ $C \Longrightarrow$ *interp* $D' \models h$ C
⟨*proof*⟩

lemma *true-interp-imp-Interp*: C #⊆# $D \Longrightarrow D$ #⊂# $D' \Longrightarrow$ *interp* $D \models h$ $C \Longrightarrow$ *Interp* $D' \models h$ C
⟨*proof*⟩

lemma *true-interp-imp-INTERP*: C #⊆# $D \Longrightarrow$ *interp* $D \models h$ $C \Longrightarrow$ *INTERP* $\models h$ C
⟨*proof*⟩

lemma *productive-imp-false-interp*: *productive* $C \Longrightarrow \neg$ *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: {#*Pos* P, *Pos* P#} #⊂# C
shows ¬*produces* C P
⟨*proof*⟩

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

lemma
assumes D: C+{#*Neg* P#} #⊂# D
shows *production* $D \neq \{P\}$

⟨*proof*⟩

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

end
end

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

1.2.1 We can now define the rules of the calculus

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

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

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

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

lemma *true-clss-cls-extended*:
  assumes
    $A \models p\ B$ **and**
    tot: $total\text{-}over\text{-}m\ I\ A$ **and**
    cons: $consistent\text{-}interp\ I$ **and**
    I-A: $I \models s\ A$
  shows $I \models B$
⟨*proof*⟩
lemma

assumes
CP: ¬ *clss-lt N* ({#*C*#} + {#*E*#}) ⊨*p* {#*C*#} + {#*Neg P*#} **and**
clss-lt N ({#*C*#} + {#*E*#}) ⊨*p* {#*E*#} + {#*Pos P*#} ∨ *clss-lt N* ({#*C*#} + {#*E*#}) ⊨*p* {#*C*#} + {#*Neg P*#}
shows *clss-lt N* ({#*C*#} + {#*E*#}) ⊨*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* ⟷ *abstract-red A N*
begin
definition *saturated* :: *'a clauses* ⇒ *bool* **where**
saturated N ⟷ (∀ *A B C*. *A* ∈ *N* ⟶ *B* ∈ *N* ⟶ ¬*redundant A N* ⟶ ¬*redundant B N*
⟶ *superposition-rules A B C* ⟶ *redundant C N* ∨ *C* ∈ *N*)

lemma
assumes
saturated: *saturated N* **and**
finite: *finite N* **and**
empty: {#} ∉ *N*
shows *INTERP N* ⊨*hs N*
⟨*proof*⟩

end

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

lemma *subsumed-is-redundant*:
assumes *AB*: *A* ⊂# *B*
and *AN*: *A* ∈ *N*
shows *abstract-red B N*
⟨*proof*⟩

inductive *redundant* :: *'a clause* ⇒ *'a clauses* ⇒ *bool* **where**
subsumption: *A* ∈ *N* ⟹ *A* ⊂# *B* ⟹ *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 ⟹ *A* ⊆# *B* ⟹ *redundant B N*
⟨*proof*⟩

locale *truc* =
selection S **for** *S* :: *nat clause* ⇒ *nat clause*
begin

end

end