

Formalization of Bachmair and Ganzinger’s Ordered Resolution Prover

Anders Schlichtkrull, Jasmin Christian Blanchette, Dmitriy Traytel, and Uwe Waldmann

June 10, 2026

Abstract

This Isabelle/HOL formalization covers Sections 2 to 4 of Bachmair and Ganzinger’s “Resolution Theorem Proving” chapter in the *Handbook of Automated Reasoning*. This includes soundness and completeness of unordered and ordered variants of ground resolution with and without literal selection, the standard redundancy criterion, a general framework for refutational theorem proving, and soundness and completeness of an abstract first-order prover.

Contents

1	Introduction	1
2	Map Function on Two Parallel Lists	1
3	Supremum and Liminf of Lazy Lists	3
3.1	Library	3
3.2	Supremum	3
3.3	Supremum up-to	4
3.4	Liminf	4
3.5	Liminf up-to	5
4	Relational Chains over Lazy Lists	6
4.1	Chains	6
4.2	A Coinductive Puzzle	7
4.3	Full Chains	11
5	Clausal Logic	11
5.1	Literals	12
5.2	Clauses	14
6	Herbrand Intepretation	15
7	Abstract Substitutions	18
7.1	Library	18
7.2	Substitution Operators	18
7.3	Substitution Lemmas	20
7.3.1	Identity Substitution	21
7.3.2	Associativity of Composition	21
7.3.3	Compatibility of Substitution and Composition	21
7.3.4	“Commutativity” of Membership and Substitution	22
7.3.5	Signs and Substitutions	22
7.3.6	Substitution on Literal(s)	22
7.3.7	Substitution on Empty	23
7.3.8	Substitution on a Union	24
7.3.9	Substitution on a Singleton	24
7.3.10	Substitution on (#)	25
7.3.11	Substitution on tl	25

7.3.12	Substitution on (!)	25
7.3.13	Substitution on Various Other Functions	26
7.3.14	Renamings	26
7.3.15	Monotonicity	27
7.3.16	Size after Substitution	28
7.3.17	Variable Disjointness	28
7.3.18	Ground Expressions and Substitutions	28
7.3.19	Subsumption	31
7.3.20	Unifiers	31
7.3.21	Most General Unifier	32
7.3.22	Generalization and Subsumption	32
7.3.23	Generalization and Subsumption	34
7.4	Most General Unifiers	34
7.5	Idempotent Most General Unifiers	34
8	Refutational Inference Systems	35
8.1	Preliminaries	35
8.2	Refutational Completeness	36
8.3	Compactness	36
9	Candidate Models for Ground Resolution	37
10	Ground Unordered Resolution Calculus	41
10.1	Inference Rule	41
10.2	Inference System	42
11	Ground Ordered Resolution Calculus with Selection	42
11.1	Inference Rule	42
11.2	Inference System	44
12	Theorem Proving Processes	44
13	The Standard Redundancy Criterion	46
14	First-Order Ordered Resolution Calculus with Selection	49
14.1	Library	49
14.2	Calculus	49
14.3	Soundness	50
14.4	Other Basic Properties	51
14.5	Inference System	52
14.6	Lifting	52
15	An Ordered Resolution Prover for First-Order Clauses	54

1 Introduction

Bachmair and Ganzinger’s “Resolution Theorem Proving” chapter in the *Handbook of Automated Reasoning* is the standard reference on the topic. It defines a general framework for propositional and first-order resolution-based theorem proving. Resolution forms the basis for superposition, the calculus implemented in many popular automatic theorem provers.

This Isabelle/HOL formalization covers Sections 2.1, 2.2, 2.4, 2.5, 3, 4.1, 4.2, and 4.3 of Bachmair and Ganzinger’s chapter. Section 2 focuses on preliminaries. Section 3 introduces unordered and ordered variants of ground resolution with and without literal selection and proves them refutationally complete. Section 4.1 presents a framework for theorem provers based on refutation and saturation. Section 4.2 generalizes the refutational completeness argument and introduces the standard redundancy criterion, which can be used in conjunction with ordered resolution. Finally, Section 4.3 lifts the result to a first-order prover, specified as a calculus. Figure 1 shows the corresponding Isabelle theory structure.

We refer to the following publications for details:

Anders Schlichtkrull, Jasmin Christian Blanchette, Dmitriy Traytel, Uwe Waldmann:
 Formalizing Bachmair and Ganzinger’s Ordered Resolution Prover.
 IJCAR 2018: 89-107
http://matryoshka.gforge.inria.fr/pubs/rp_paper.pdf

Anders Schlichtkrull, Jasmin Blanchette, Dmitriy Traytel, Uwe Waldmann:
 Formalizing Bachmair and Ganzinger’s Ordered Resolution Prover.
 Journal of Automated Reasoning
http://matryoshka.gforge.inria.fr/pubs/rp_article.pdf

2 Map Function on Two Parallel Lists

```
theory Map2
  imports Main
begin
```

This theory defines a map function that applies a (curried) binary function elementwise to two parallel lists. The definition is taken from https://www.isa-afp.org/browser_info/current/AFP/Jinja/Listn.html.

abbreviation $map2 :: ('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow 'a\ list \Rightarrow 'b\ list \Rightarrow 'c\ list$ **where**
 $map2\ f\ xs\ ys \equiv map\ (case_prod\ f)\ (zip\ xs\ ys)$

lemma $map2_empty_iff[simp]: map2\ f\ xs\ ys = [] \longleftrightarrow xs = [] \vee ys = []$
 <proof>

lemma $image_map2: length\ t = length\ s \Longrightarrow g\ `set\ (map2\ f\ t\ s) = set\ (map2\ (\lambda a\ b.\ g\ (f\ a\ b))\ t\ s)$
 <proof>

lemma $map2_tl: length\ t = length\ s \Longrightarrow map2\ f\ (tl\ t)\ (tl\ s) = tl\ (map2\ f\ t\ s)$
 <proof>

lemma $map_zip_assoc:$
 $map\ f\ (zip\ (zip\ xs\ ys)\ zs) = map\ (\lambda(x,\ y,\ z).\ f\ ((x,\ y),\ z))\ (zip\ xs\ (zip\ ys\ zs))$
 <proof>

lemma $set_map2_ex:$
assumes $length\ t = length\ s$
shows $set\ (map2\ f\ s\ t) = \{x.\ \exists i < length\ t.\ x = f\ (s\ !\ i)\ (t\ !\ i)\}$
 <proof>

end

3 Supremum and Liminf of Lazy Lists

```
theory Lazy_List_Liminf
  imports Coinductive.Coinductive_List
begin
```

Lazy lists, as defined in the *Archive of Formal Proofs*, provide finite and infinite lists in one type, defined coinductively. The present theory introduces the concept of the union of all elements of a lazy list of sets and the limit of such a lazy list. The definitions are stated more generally in terms of lattices. The basis for this theory is Section 4.1 (“Theorem Proving Processes”) of Bachmair and Ganzinger’s chapter.

3.1 Library

lemma $less_llength_ltake: i < llength\ (ltake\ k\ Xs) \longleftrightarrow i < k \wedge i < llength\ Xs$
 <proof>

3.2 Supremum

definition $Sup_llist :: 'a\ set\ llist \Rightarrow 'a\ set$ **where**

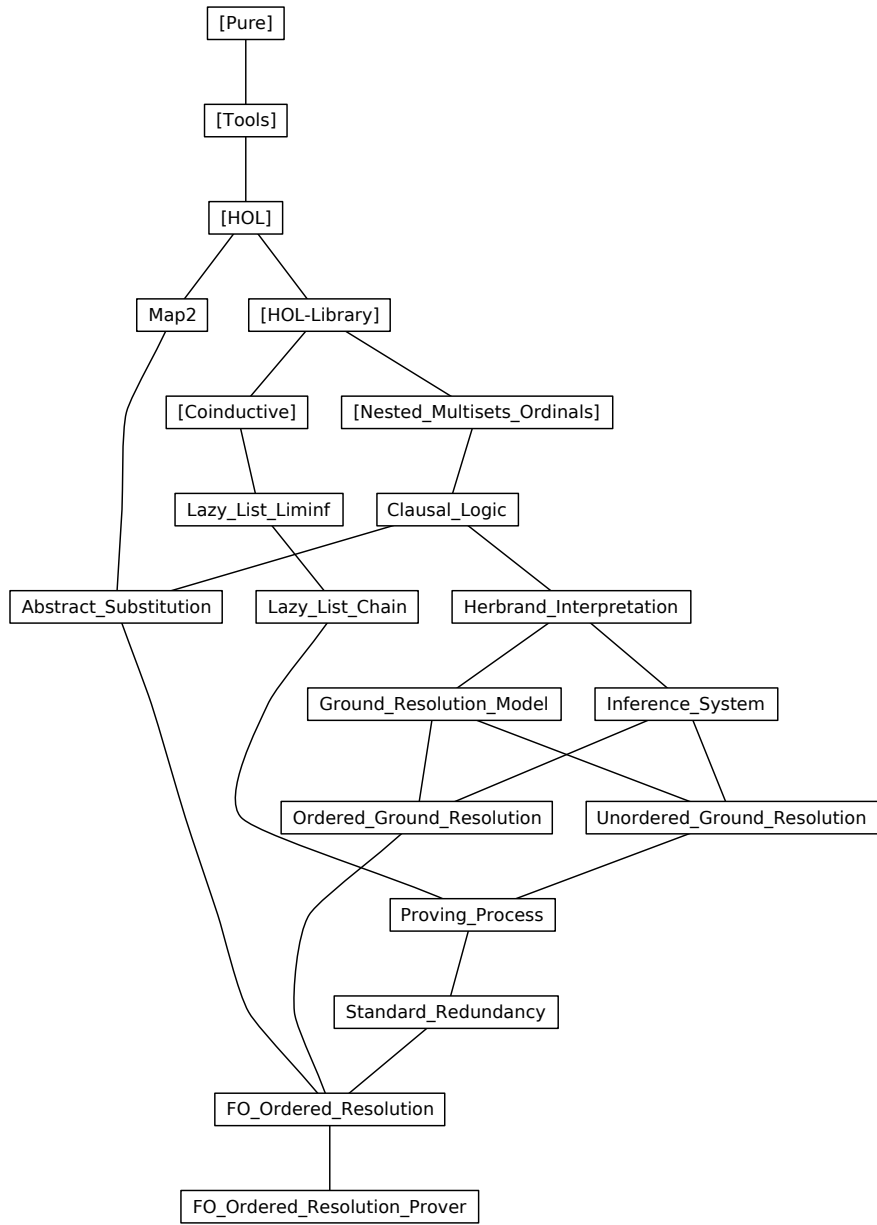


Figure 1: Theory dependency graph

$Sup_l\text{list } Xs = (\bigcup i \in \{i. \text{enat } i < \text{llength } Xs\}. \text{lnth } Xs \ i)$

lemma $\text{lnth_subset_Sup_l\text{list}}$: $\text{enat } i < \text{llength } Xs \implies \text{lnth } Xs \ i \subseteq \text{Sup_l\text{list } } Xs$
 ⟨proof⟩

lemma $\text{Sup_l\text{list_imp_exists_index}}$: $x \in \text{Sup_l\text{list } } Xs \implies \exists i. \text{enat } i < \text{llength } Xs \wedge x \in \text{lnth } Xs \ i$
 ⟨proof⟩

lemma $\text{exists_index_imp_Sup_l\text{list}}$: $\text{enat } i < \text{llength } Xs \implies x \in \text{lnth } Xs \ i \implies x \in \text{Sup_l\text{list } } Xs$
 ⟨proof⟩

lemma $\text{Sup_l\text{list_LNil[simp]}}$: $\text{Sup_l\text{list } } \text{LNil} = \{\}$
 ⟨proof⟩

lemma $\text{Sup_l\text{list_LCons[simp]}}$: $\text{Sup_l\text{list } } (\text{LCons } X \ Xs) = X \cup \text{Sup_l\text{list } } Xs$
 ⟨proof⟩

lemma $\text{lhs_subset_Sup_l\text{list}}$: $\neg \text{lnull } Xs \implies \text{lhs } Xs \subseteq \text{Sup_l\text{list } } Xs$
 ⟨proof⟩

3.3 Supremum up-to

definition $\text{Sup_upto_l\text{list}}$:: 'a set llist \Rightarrow enat \Rightarrow 'a set **where**
 $\text{Sup_upto_l\text{list } } Xs \ j = (\bigcup i \in \{i. \text{enat } i < \text{llength } Xs \wedge \text{enat } i \leq j\}. \text{lnth } Xs \ i)$

lemma $\text{Sup_upto_l\text{list_eq_Sup_l\text{list_l\text{take}}}$: $\text{Sup_upto_l\text{list } } Xs \ j = \text{Sup_l\text{list } } (\text{l\text{take } } (\text{eSuc } j) \ Xs)$
 ⟨proof⟩

lemma $\text{Sup_upto_l\text{list_enat_0[simp]}}$:
 $\text{Sup_upto_l\text{list } } Xs \ (\text{enat } 0) = (\text{if } \text{lnull } Xs \ \text{then } \{\} \ \text{else } \text{lhs } Xs)$
 ⟨proof⟩

lemma $\text{Sup_upto_l\text{list_Suc[simp]}}$:
 $\text{Sup_upto_l\text{list } } Xs \ (\text{enat } (\text{Suc } j)) =$
 $\text{Sup_upto_l\text{list } } Xs \ (\text{enat } j) \cup (\text{if } \text{enat } (\text{Suc } j) < \text{llength } Xs \ \text{then } \text{lnth } Xs \ (\text{Suc } j) \ \text{else } \{\})$
 ⟨proof⟩

lemma $\text{Sup_upto_l\text{list_infinity[simp]}}$: $\text{Sup_upto_l\text{list } } Xs \ \infty = \text{Sup_l\text{list } } Xs$
 ⟨proof⟩

lemma $\text{Sup_upto_l\text{list_0[simp]}}$: $\text{Sup_upto_l\text{list } } Xs \ 0 = (\text{if } \text{lnull } Xs \ \text{then } \{\} \ \text{else } \text{lhs } Xs)$
 ⟨proof⟩

lemma $\text{Sup_upto_l\text{list_eSuc[simp]}}$:
 $\text{Sup_upto_l\text{list } } Xs \ (\text{eSuc } j) =$
 (case j of
 $\text{enat } k \Rightarrow \text{Sup_upto_l\text{list } } Xs \ (\text{enat } (\text{Suc } k))$
 $|\ \infty \Rightarrow \text{Sup_l\text{list } } Xs)$
 ⟨proof⟩

lemma $\text{Sup_upto_l\text{list_mono[simp]}}$: $j \leq k \implies \text{Sup_upto_l\text{list } } Xs \ j \subseteq \text{Sup_upto_l\text{list } } Xs \ k$
 ⟨proof⟩

lemma $\text{Sup_upto_l\text{list_subset_Sup_l\text{list}}$: $\text{Sup_upto_l\text{list } } Xs \ j \subseteq \text{Sup_l\text{list } } Xs$
 ⟨proof⟩

lemma $\text{elem_Sup_l\text{list_imp_Sup_upto_l\text{list}}$:
 $x \in \text{Sup_l\text{list } } Xs \implies \exists j < \text{llength } Xs. x \in \text{Sup_upto_l\text{list } } Xs \ (\text{enat } j)$
 ⟨proof⟩

lemma $\text{lnth_subset_Sup_upto_l\text{list}}$: $j < \text{llength } Xs \implies \text{lnth } Xs \ j \subseteq \text{Sup_upto_l\text{list } } Xs \ j$
 ⟨proof⟩

lemma $\text{finite_Sup_l\text{list_imp_Sup_upto_l\text{list}}$:

assumes *finite X and* $X \subseteq \text{Sup_l}list\ Xs$
shows $\exists k. X \subseteq \text{Sup_upto_l}list\ Xs\ (\text{enat}\ k)$
 ⟨proof⟩

3.4 Liminf

definition *Liminf_l}list* :: 'a set l}list \Rightarrow 'a set **where**

Liminf_l}list $Xs =$
 $(\bigcup i \in \{i. \text{enat}\ i < \text{l}length\ Xs\}. \bigcap j \in \{j. i \leq j \wedge \text{enat}\ j < \text{l}length\ Xs\}. \text{l}nth\ Xs\ j)$

lemma *Liminf_l}list_LNil[simp]*: *Liminf_l}list* LNil = {}
 ⟨proof⟩

lemma *Liminf_l}list_LCons*:

Liminf_l}list (LCons X Xs) = (if lnull Xs then X else *Liminf_l}list* Xs) (**is** ?lhs = ?rhs)
 ⟨proof⟩

lemma *lfinite_Liminf_l}list*: *lfinite* Xs \Longrightarrow *Liminf_l}list* Xs = (if lnull Xs then {} else llast Xs)
 ⟨proof⟩

lemma *Liminf_l}list_ltl*: \neg lnull (ltl Xs) \Longrightarrow *Liminf_l}list* Xs = *Liminf_l}list* (ltl Xs)
 ⟨proof⟩

lemma *Liminf_l}list_subset_Sup_l}list*: *Liminf_l}list* Xs \subseteq *Sup_l}list* Xs
 ⟨proof⟩

lemma *image_Liminf_l}list_subset*: $f \text{ ' } \text{Liminf_l}list\ Ns \subseteq \text{Liminf_l}list\ (\text{lmap}\ ((\cdot)\ f)\ Ns)$
 ⟨proof⟩

lemma *Liminf_l}list_imp_exists_index*:

$x \in \text{Liminf_l}list\ Xs \Longrightarrow \exists i. \text{enat}\ i < \text{l}length\ Xs \wedge x \in \text{l}nth\ Xs\ i$
 ⟨proof⟩

lemma *not_Liminf_l}list_imp_exists_index*:

\neg lnull Xs $\Longrightarrow x \notin \text{Liminf_l}list\ Xs \Longrightarrow \text{enat}\ i < \text{l}length\ Xs \Longrightarrow$
 $(\exists j. i \leq j \wedge \text{enat}\ j < \text{l}length\ Xs \wedge x \notin \text{l}nth\ Xs\ j)$
 ⟨proof⟩

lemma *finite_subset_Liminf_l}list_imp_exists_index*:

assumes
nmil: \neg lnull Xs **and**
fin: *finite* X **and**
in_lim: $X \subseteq \text{Liminf_l}list\ Xs$
shows $\exists i. \text{enat}\ i < \text{l}length\ Xs \wedge X \subseteq (\bigcap j \in \{j. i \leq j \wedge \text{enat}\ j < \text{l}length\ Xs\}. \text{l}nth\ Xs\ j)$
 ⟨proof⟩

lemma *Liminf_l}list_lmap_image*:

assumes *f_inj*: *inj_on* f (*Sup_l}list* (lmap g xs))
shows *Liminf_l}list* (lmap ($\lambda x. f \text{ ' } g\ x$) xs) = *f* ' *Liminf_l}list* (lmap g xs) (**is** ?lhs = ?rhs)
 ⟨proof⟩

lemma *Liminf_l}list_lmap_union*:

assumes $\forall x \in \text{lset}\ xs. \forall Y \in \text{lset}\ xs. g\ x \cap h\ Y = \{\}$
shows *Liminf_l}list* (lmap ($\lambda x. g\ x \cup h\ x$) xs) =
Liminf_l}list (lmap g xs) \cup *Liminf_l}list* (lmap h xs) (**is** ?lhs = ?rhs)
 ⟨proof⟩

lemma *Liminf_set_filter_commute*:

Liminf_l}list (lmap ($\lambda X. \{x \in X. p\ x\}$) Xs) = $\{x \in \text{Liminf_l}list\ Xs. p\ x\}$
 ⟨proof⟩

3.5 Liminf up-to

definition *Liminf_upto_l}list* :: 'a set l}list \Rightarrow enat \Rightarrow 'a set **where**

<proof>

lemma *chain_length_pos*: $chain\ R\ xs \implies llength\ xs > 0$
<proof>

lemma *chain_ldropn*:
assumes $chain\ R\ xs$ **and** $enat\ n < llength\ xs$
shows $chain\ R\ (ldropn\ n\ xs)$
<proof>

lemma *inf_chain_ldropn_chain*: $chain\ R\ xs \implies \neg\ lfinite\ xs \implies chain\ R\ (ldropn\ n\ xs)$
<proof>

lemma *inf_chain_ltl_chain*: $chain\ R\ xs \implies \neg\ lfinite\ xs \implies chain\ R\ (ltl\ xs)$
<proof>

lemma *chain_lnth_rel*:
assumes
 $chain: chain\ R\ xs$ **and**
 $len: enat\ (Suc\ j) < llength\ xs$
shows $R\ (lnth\ xs\ j)\ (lnth\ xs\ (Suc\ j))$
<proof>

lemma *infinite_chain_lnth_rel*:
assumes $\neg\ lfinite\ c$ **and** $chain\ r\ c$
shows $r\ (lnth\ c\ i)\ (lnth\ c\ (Suc\ i))$
<proof>

lemma *lnth_rel_chain*:
assumes
 $\neg\ null\ xs$ **and**
 $\forall j. enat\ (j + 1) < llength\ xs \longrightarrow R\ (lnth\ xs\ j)\ (lnth\ xs\ (j + 1))$
shows $chain\ R\ xs$
<proof>

lemma *chain_lmap*:
assumes $\forall x\ y. R\ x\ y \longrightarrow R'\ (f\ x)\ (f\ y)$ **and** $chain\ R\ xs$
shows $chain\ R'\ (lmap\ f\ xs)$
<proof>

lemma *chain_mono*:
assumes $\forall x\ y. R\ x\ y \longrightarrow R'\ x\ y$ **and** $chain\ R\ xs$
shows $chain\ R'\ xs$
<proof>

lemma *chain_ldropnI*:
assumes
 $rel: \forall j. j \geq i \longrightarrow enat\ (Suc\ j) < llength\ xs \longrightarrow R\ (lnth\ xs\ j)\ (lnth\ xs\ (Suc\ j))$ **and**
 $si_lt: enat\ (Suc\ i) < llength\ xs$
shows $chain\ R\ (ldropn\ i\ xs)$
<proof>

lemma *chain_ldropn_lmapI*:
assumes
 $rel: \forall j. j \geq i \longrightarrow enat\ (Suc\ j) < llength\ xs \longrightarrow R\ (f\ (lnth\ xs\ j))\ (f\ (lnth\ xs\ (Suc\ j)))$ **and**
 $si_lt: enat\ (Suc\ i) < llength\ xs$
shows $chain\ R\ (ldropn\ i\ (lmap\ f\ xs))$
<proof>

lemma *lfinite_chain_imp_rtranclp_lhd_llast*: $lfinite\ xs \implies chain\ R\ xs \implies R^{**}\ (lhd\ xs)\ (llast\ xs)$
<proof>

lemma *tranclp_imp_exists_finite_chain_list*:

$R^{++} x y \implies \exists xs. \text{chain } R (\text{llist_of } (x \# xs @ [y]))$
 ⟨proof⟩

inductive-cases *chain_consE*: $\text{chain } R (LCons x xs)$

inductive-cases *chain_nontrivE*: $\text{chain } R (LCons x (LCons y xs))$

4.2 A Coinductive Puzzle

primrec *prepend* **where**

$\text{prepend } [] ys = ys$
 | $\text{prepend } (x \# xs) ys = LCons x (\text{prepend } xs ys)$

lemma *null_prepend[simp]*: $\text{null } (\text{prepend } xs ys) = (xs = [] \wedge \text{null } ys)$
 ⟨proof⟩

lemma *lhd_prepend[simp]*: $\text{lhd } (\text{prepend } xs ys) = (\text{if } xs \neq [] \text{ then } \text{hd } xs \text{ else } \text{lhd } ys)$
 ⟨proof⟩

lemma *prepend_LNil[simp]*: $\text{prepend } xs LNil = \text{llist_of } xs$
 ⟨proof⟩

lemma *lfinite_prepend[simp]*: $\text{lfinite } (\text{prepend } xs ys) \longleftrightarrow \text{lfinite } ys$
 ⟨proof⟩

lemma *llength_prepend[simp]*: $\text{llength } (\text{prepend } xs ys) = \text{length } xs + \text{llength } ys$
 ⟨proof⟩

lemma *llast_prepend[simp]*: $\neg \text{null } ys \implies \text{llast } (\text{prepend } xs ys) = \text{llast } ys$
 ⟨proof⟩

lemma *prepend_prepend*: $\text{prepend } xs (\text{prepend } ys zs) = \text{prepend } (xs @ ys) zs$
 ⟨proof⟩

lemma *chain_prepend*:
 $\text{chain } R (\text{llist_of } zs) \implies \text{last } zs = \text{lhd } xs \implies \text{chain } R xs \implies \text{chain } R (\text{prepend } zs (\text{tl } xs))$
 ⟨proof⟩

lemma *lmap_prepend[simp]*: $\text{lmap } f (\text{prepend } xs ys) = \text{prepend } (\text{map } f xs) (\text{lmap } f ys)$
 ⟨proof⟩

lemma *lset_prepend[simp]*: $\text{lset } (\text{prepend } xs ys) = \text{set } xs \cup \text{lset } ys$
 ⟨proof⟩

lemma *prepend_LCons*: $\text{prepend } xs (LCons y ys) = \text{prepend } (xs @ [y]) ys$
 ⟨proof⟩

lemma *lnth_prepend*:
 $\text{lnth } (\text{prepend } xs ys) i = (\text{if } i < \text{length } xs \text{ then } \text{nth } xs i \text{ else } \text{lnth } ys (i - \text{length } xs))$
 ⟨proof⟩

theorem *lfinite_less_induct[consumes 1, case_names less]*:
assumes *fin*: $\text{lfinite } xs$
and *step*: $\bigwedge xs. \text{lfinite } xs \implies (\bigwedge zs. \text{llength } zs < \text{llength } xs \implies P zs) \implies P xs$
shows $P xs$
 ⟨proof⟩

theorem *lfinite_prepend_induct[consumes 1, case_names LNil prepend]*:
assumes *lfinite* $\text{lfinite } xs$
and *LNil*: $P LNil$
and *prepend*: $\bigwedge xs. \text{lfinite } xs \implies (\bigwedge zs. (\exists ys. xs = \text{prepend } ys zs \wedge ys \neq []) \implies P zs) \implies P xs$
shows $P xs$
 ⟨proof⟩

coinductive *emb* :: $'a \text{ llist} \Rightarrow 'a \text{ llist} \Rightarrow \text{bool}$ **where**

$lfinite\ xs \implies emb\ LNil\ xs$
 $| emb\ xs\ ys \implies emb\ (LCons\ x\ xs)\ (prepend\ zs\ (LCons\ x\ ys))$

inductive-cases emb_LConsE : $emb\ (LCons\ z\ zs)\ ys$

inductive-cases emb_LNil1E : $emb\ LNil\ ys$

inductive-cases emb_LNil2E : $emb\ xs\ LNil$

lemma $emb_lfinite$:

assumes $emb\ xs\ ys$

shows $lfinite\ ys \longleftrightarrow lfinite\ xs$

$\langle proof \rangle$

inductive $prepend_cong1$ **for** X **where**

$prepend_cong1_base$: $X\ xs \implies prepend_cong1\ X\ xs$

$| prepend_cong1_prepend$: $prepend_cong1\ X\ ys \implies prepend_cong1\ X\ (prepend\ xs\ ys)$

lemma $prepend_cong1_alt$: $prepend_cong1\ X\ xs \longleftrightarrow (\exists\ ys\ zs.\ xs = prepend\ ys\ zs \wedge X\ zs)$

$\langle proof \rangle$

lemma $emb_prepend_coinduct_cong$ [$rotated, case_names\ emb$]:

assumes $(\bigwedge x1\ x2.\ X\ x1\ x2 \implies$

$(\exists\ xs.\ x1 = LNil \wedge x2 = xs \wedge lfinite\ xs)$

$\vee (\exists\ xs\ ys\ x\ zs.\ x1 = LCons\ x\ xs \wedge x2 = prepend\ zs\ (LCons\ x\ ys)$

$\wedge (prepend_cong1\ (X\ xs)\ ys \vee emb\ xs\ ys)))$ (**is** $\bigwedge x1\ x2.\ X\ x1\ x2 \implies ?bisim\ x1\ x2$)

shows $X\ x1\ x2 \implies emb\ x1\ x2$

$\langle proof \rangle$

lemma $emb_prepend$: $emb\ xs\ ys \implies emb\ xs\ (prepend\ zs\ ys)$

$\langle proof \rangle$

lemma $prepend_cong1_emb$: $prepend_cong1\ (emb\ xs)\ ys = emb\ xs\ ys$

$\langle proof \rangle$

lemma $prepend_cong_distrib$:

$prepend_cong1\ (P \sqcup Q)\ xs \longleftrightarrow prepend_cong1\ P\ xs \vee prepend_cong1\ Q\ xs$

$\langle proof \rangle$

lemma $emb_prepend_coinduct_aux$ [$case_names\ emb$]:

assumes $X\ x1\ x2\ (\bigwedge x1\ x2.\ X\ x1\ x2 \implies$

$(\exists\ xs.\ x1 = LNil \wedge x2 = xs \wedge lfinite\ xs)$

$\vee (\exists\ xs\ ys\ x\ zs.\ x1 = LCons\ x\ xs \wedge x2 = prepend\ zs\ (LCons\ x\ ys)$

$\wedge (prepend_cong1\ (X\ xs \sqcup emb\ xs)\ ys)))$

shows $emb\ x1\ x2$

$\langle proof \rangle$

lemma $emb_prepend_coinduct$ [$rotated, case_names\ emb$]:

assumes $(\bigwedge x1\ x2.\ X\ x1\ x2 \implies$

$(\exists\ xs.\ x1 = LNil \wedge x2 = xs \wedge lfinite\ xs)$

$\vee (\exists\ xs\ ys\ x\ zs\ zs'.\ x1 = LCons\ x\ xs \wedge x2 = prepend\ zs\ (LCons\ x\ (prepend\ zs'\ ys))$

$\wedge (X\ xs\ ys \vee emb\ xs\ ys)))$

shows $X\ x1\ x2 \implies emb\ x1\ x2$

$\langle proof \rangle$

context

begin

private coinductive $chain'$ **for** R **where**

$chain'\ R\ (LCons\ x\ LNil)$

$| chain\ R\ (l\ list_of\ (x\ \# \ zs\ @\ [lhd\ xs])) \implies$

$chain'\ R\ xs \implies chain'\ R\ (LCons\ x\ (prepend\ zs\ xs))$

private lemma $chain_imp_chain'$: $chain\ R\ xs \implies chain'\ R\ xs$

<proof> **lemma** *chain'_imp_chain*: $chain' R xs \implies chain R xs$

<proof> **lemma** *chain_chain'*: $chain = chain'$

<proof>

lemma *chain_prepend_coinduct*[*case_names chain*]:

$X x \implies (\bigwedge x. X x \implies$
 $(\exists z. x = LCons z LNil) \vee$
 $(\exists y xs zs. x = LCons y (prepend zs xs) \wedge$
 $(X xs \vee chain R xs) \wedge chain R (llist_of (y \# zs @ [lhd xs]))) \implies chain R x$
<proof>

end

context

fixes $R :: 'a \Rightarrow 'a \Rightarrow bool$

begin

private definition *pick where*

$pick\ x\ y = (SOME\ xs.\ chain\ R\ (llist_of\ (x\ \#\ xs\ @\ [y])))$

private lemma *pick[simp]*:

assumes $R^{++}\ x\ y$

shows $chain\ R\ (llist_of\ (x\ \#\ pick\ x\ y\ @\ [y]))$

<proof> **friend-of-corec** *prepend where*

$prepend\ xs\ ys = (case\ xs\ of\ [] \Rightarrow$

$(case\ ys\ of\ LNil \Rightarrow LNil \mid LCons\ x\ xs \Rightarrow LCons\ x\ xs) \mid x\ \#\ xs' \Rightarrow LCons\ x\ (prepend\ xs'\ ys))$

<proof> **corec** *wit where*

$wit\ xs = (case\ xs\ of\ LCons\ x\ (LCons\ y\ xs) \Rightarrow$

$LCons\ x\ (prepend\ (pick\ x\ y)\ (wit\ (LCons\ y\ xs))) \mid _ \Rightarrow xs)$

private lemma

wit_LNil[simp]: $wit\ LNil = LNil$ **and**

wit_singleton[simp]: $wit\ (LCons\ x\ LNil) = LCons\ x\ LNil$ **and**

wit_LCons2: $wit\ (LCons\ x\ (LCons\ y\ xs)) =$

$(LCons\ x\ (prepend\ (pick\ x\ y)\ (wit\ (LCons\ y\ xs))))$

<proof> **lemma** *lnull_wit[simp]*: $lnull\ (wit\ xs) \longleftrightarrow lnull\ xs$

<proof> **lemma** *lhd_wit[simp]*: $chain\ R^{++}\ xs \implies lhd\ (wit\ xs) = lhd\ xs$

<proof> **lemma** *LNil_eq_iff_lnull*: $LNil = xs \longleftrightarrow lnull\ xs$

<proof>

lemma *emb_wit[simp]*: $chain\ R^{++}\ xs \implies emb\ xs\ (wit\ xs)$

<proof> **lemma** *lfinite_wit[simp]*:

assumes $chain\ R^{++}\ xs$

shows $lfinite\ (wit\ xs) \longleftrightarrow lfinite\ xs$

<proof> **lemma** *llast_wit[simp]*:

assumes $chain\ R^{++}\ xs$

shows $llast\ (wit\ xs) = llast\ xs$

<proof>

lemma *chain_tranclp_imp_exists_chain*:

$chain\ R^{++}\ xs \implies$

$\exists ys.\ chain\ R\ ys \wedge emb\ xs\ ys \wedge lhd\ ys = lhd\ xs \wedge llast\ ys = llast\ xs$

<proof>

lemma *emb_lset_mono[rotated]*: $x \in lset\ xs \implies emb\ xs\ ys \implies x \in lset\ ys$

<proof>

lemma *emb_Ball_lset_antimono*:

assumes $emb\ Xs\ Ys$

shows $\forall Y \in lset\ Ys.\ x \in Y \implies \forall X \in lset\ Xs.\ x \in X$

<proof>

lemma *emb_lfinite_antimono[rotated]*: $lfinite\ ys \implies emb\ xs\ ys \implies lfinite\ xs$

<proof>

lemma *emb_Liminf_llist_mono_aux*:

assumes *emb Xs Ys and* \neg *lfinite Xs and* \neg *lfinite Ys and* $\forall j \geq i. x \in \text{lnth } Ys\ j$
shows $\forall j \geq i. x \in \text{lnth } Xs\ j$

<proof>

lemma *emb_Liminf_llist_infinite*:

assumes *emb Xs Ys and* \neg *lfinite Xs*
shows *Liminf_llist Ys* \subseteq *Liminf_llist Xs*

<proof>

lemma *emb_lmap*: *emb xs ys* \implies *emb (lmap f xs) (lmap f ys)*

<proof>

end

lemma *chain_inf_llist_if_infinite_chain_function*:

assumes $\forall i. r (f (Suc\ i)) (f\ i)$
shows \neg *lfinite (inf_llist f)* \wedge *chain* r^{-1-1} (*inf_llist f*)

<proof>

lemma *infinite_chain_function_iff_infinite_chain_llist*:

$(\exists f. \forall i. r (f (Suc\ i)) (f\ i)) \iff (\exists c. \neg$ *lfinite c* \wedge *chain* r^{-1-1} *c*)

<proof>

lemma *wfP_iff_no_infinite_down_chain_llist*: *wfP r* \iff $(\nexists c. \neg$ *lfinite c* \wedge *chain* r^{-1-1} *c*)

<proof>

4.3 Full Chains

coinductive *full_chain* :: $('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow 'a\ \text{llist} \Rightarrow \text{bool}$ **for** *R* :: $'a \Rightarrow 'a \Rightarrow \text{bool}$ **where**

full_chain_singleton: $(\forall y. \neg R\ x\ y) \implies$ *full_chain R (LCons x LNil)*

| *full_chain_cons*: *full_chain R xs* \implies *R x (lhd xs)* \implies *full_chain R (LCons x xs)*

lemma

full_chain_LNil[simp]: \neg *full_chain R LNil* **and**
full_chain_not_lnull: *full_chain R xs* \implies \neg *lnull xs*

<proof>

lemma *full_chain_ldropn*:

assumes *full*: *full_chain R xs* **and** *enat n < llength xs*
shows *full_chain R (ldropn n xs)*

<proof>

lemma *full_chain_iff_chain*:

full_chain R xs \iff *chain R xs* \wedge (*lfinite xs* \longrightarrow $(\forall y. \neg R (l\text{last } xs)\ y)$)

<proof>

lemma *full_chain_imp_chain*: *full_chain R xs* \implies *chain R xs*

<proof>

lemma *full_chain_length_pos*: *full_chain R xs* \implies *llength xs > 0*

<proof>

lemma *full_chain_lnth_rel*:

full_chain R xs \implies *enat (Suc j) < llength xs* \implies *R (lnth xs j) (lnth xs (Suc j))*

<proof>

lemma *full_chain_lnth_not_rel*:

assumes
full: *full_chain R xs* **and**
sj: *enat (Suc j) = llength xs*
shows $\neg R (lnth\ xs\ j)\ y$

<proof>

inductive-cases *full_chain_consE*: *full_chain* *R* (*LCons* *x xs*)

inductive-cases *full_chain_nontrivE*: *full_chain* *R* (*LCons* *x (LCons y xs)*)

lemma *full_chain_tranclp_imp_exists_full_chain*:

assumes *full*: *full_chain* R^{++} *xs*

shows $\exists ys. \text{full_chain } R \text{ } ys \wedge \text{emb } xs \text{ } ys \wedge \text{lhs } ys = \text{lhs } xs \wedge \text{llast } ys = \text{llast } xs$

<proof>

end

5 Clausal Logic

theory *Clausal_Logic*

imports *Nested_Multisets_Ordinals.Multiset_More*

begin

Resolution operates on clauses, which are disjunctions of literals. The material formalized here corresponds roughly to Sections 2.1 (“Formulas and Clauses”) of Bachmair and Ganzinger, excluding the formula and term syntax.

5.1 Literals

Literals consist of a polarity (positive or negative) and an atom, of type *'a*.

datatype *'a literal* =

is_pos: *Pos* (*atm_of*: *'a*)

| *Neg* (*atm_of*: *'a*)

abbreviation *is_neg* :: *'a literal* \Rightarrow *bool* **where**

is_neg *L* $\equiv \neg$ *is_pos* *L*

lemma *Pos_atm_of_iff[simp]*: *Pos* (*atm_of* *L*) = *L* \longleftrightarrow *is_pos* *L*

<proof>

lemma *Neg_atm_of_iff[simp]*: *Neg* (*atm_of* *L*) = *L* \longleftrightarrow *is_neg* *L*

<proof>

lemma *set_literal_atm_of*: *set_literal* *L* = {*atm_of* *L*}

<proof>

lemma *ex_lit_cases*: $(\exists L. P \text{ } L) \longleftrightarrow (\exists A. P (\text{Pos } A) \vee P (\text{Neg } A))$

<proof>

instantiation *literal* :: (*type*) *uminus*

begin

definition *uminus_literal* :: *'a literal* \Rightarrow *'a literal* **where**

uminus *L* = (if *is_pos* *L* then *Neg* else *Pos*) (*atm_of* *L*)

instance *<proof>*

end

lemma

uminus_Pos[simp]: \neg *Pos* *A* = *Neg* *A* **and**

uminus_Neg[simp]: \neg *Neg* *A* = *Pos* *A*

<proof>

lemma *atm_of_uminus[simp]*: *atm_of* (\neg *L*) = *atm_of* *L*

<proof>

lemma *uminus_of_uminus_id[simp]*: $- (- (x :: 'v \text{ literal})) = x$
<proof>

lemma *uminus_not_id[simp]*: $x \neq - (x :: 'v \text{ literal})$
<proof>

lemma *uminus_not_id'[simp]*: $- x \neq (x :: 'v \text{ literal})$
<proof>

lemma *uminus_eq_inj[iff]*: $- (a :: 'v \text{ literal}) = - b \longleftrightarrow a = b$
<proof>

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

lemma *is_pos_neg_not_is_pos*: $is_pos (- L) \longleftrightarrow \neg is_pos L$
<proof>

instantiation *literal* :: (*preorder*) *preorder*
begin

definition *less_literal* :: '*a literal* \Rightarrow '*a literal* \Rightarrow *bool* **where**
less_literal L M $\longleftrightarrow atm_of L < atm_of M \vee atm_of L \leq atm_of M \wedge is_neg L < is_neg M$

definition *less_eq_literal* :: '*a literal* \Rightarrow '*a literal* \Rightarrow *bool* **where**
less_eq_literal L M $\longleftrightarrow atm_of L < atm_of M \vee atm_of L \leq atm_of M \wedge is_neg L \leq is_neg M$

instance
<proof>

end

instantiation *literal* :: (*order*) *order*
begin

instance
<proof>

end

lemma *pos_less_neg[simp]*: $Pos A < Neg A$
<proof>

lemma *pos_less_pos_iff[simp]*: $Pos A < Pos B \longleftrightarrow A < B$
<proof>

lemma *pos_less_neg_iff[simp]*: $Pos A < Neg B \longleftrightarrow A \leq B$
<proof>

lemma *neg_less_pos_iff[simp]*: $Neg A < Pos B \longleftrightarrow A < B$
<proof>

lemma *neg_less_neg_iff[simp]*: $Neg A < Neg B \longleftrightarrow A < B$
<proof>

lemma *pos_le_neg[simp]*: $Pos A \leq Neg A$
<proof>

lemma *pos_le_pos_iff[simp]*: $Pos A \leq Pos B \longleftrightarrow A \leq B$
<proof>

lemma *pos_le_neg_iff[simp]*: $Pos A \leq Neg B \longleftrightarrow A \leq B$
<proof>

lemma *neg_le_pos_iff[simp]*: $Neg\ A \leq Pos\ B \longleftrightarrow A < B$
 ⟨proof⟩

lemma *neg_le_neg_iff[simp]*: $Neg\ A \leq Neg\ B \longleftrightarrow A \leq B$
 ⟨proof⟩

lemma *leq_imp_less_eq_atm_of*: $L \leq M \implies atm_of\ L \leq atm_of\ M$
 ⟨proof⟩

instantiation *literal* :: (linorder) linorder
begin

instance
 ⟨proof⟩

end

instantiation *literal* :: (wellorder) wellorder
begin

instance
 ⟨proof⟩

end

5.2 Clauses

Clauses are (finite) multisets of literals.

type-synonym *'a clause* = *'a literal multiset*

abbreviation *map_clause* :: (*'a* \Rightarrow *'b*) \Rightarrow *'a clause* \Rightarrow *'b clause* **where**
map_clause *f* $\equiv image_mset\ (map_literal\ f)$

abbreviation *rel_clause* :: (*'a* \Rightarrow *'b* \Rightarrow bool) \Rightarrow *'a clause* \Rightarrow *'b clause* \Rightarrow bool **where**
rel_clause *R* $\equiv rel_mset\ (rel_literal\ R)$

abbreviation *poss* :: *'a multiset* \Rightarrow *'a clause* **where** *poss* *AA* $\equiv \{\#Pos\ A.\ A \in\# AA\}$

abbreviation *negs* :: *'a multiset* \Rightarrow *'a clause* **where** *negs* *AA* $\equiv \{\#Neg\ A.\ A \in\# AA\}$

lemma *Max_in_lits*: $C \neq \{\#\} \implies Max_mset\ C \in\# C$
 ⟨proof⟩

lemma *Max_atm_of_set_mset_commute*: $C \neq \{\#\} \implies Max\ (atm_of\ 'set_mset\ C) = atm_of\ (Max_mset\ C)$
 ⟨proof⟩

lemma *Max_pos_neg_less_multiset*:
assumes *max*: $Max_mset\ C = Pos\ A$ **and** *neg*: $Neg\ A \in\# D$
shows $C < D$
 ⟨proof⟩

lemma *pos_Max_imp_neg_notin*: $Max_mset\ C = Pos\ A \implies Neg\ A \notin\# C$
 ⟨proof⟩

lemma *less_eq_Max_lit*: $C \neq \{\#\} \implies C \leq D \implies Max_mset\ C \leq Max_mset\ D$
 ⟨proof⟩

definition *atms_of* :: *'a clause* \Rightarrow *'a set* **where**
atms_of *C* = *atm_of 'set_mset C*

lemma *atms_of_empty[simp]*: $atms_of\ \{\#\} = \{\}$
 ⟨proof⟩

lemma *atms_of_singleton[simp]*: $atms_of \{\#L\# \} = \{atm_of L\}$
 ⟨proof⟩

lemma *atms_of_add_mset[simp]*: $atms_of (add_mset a A) = insert (atm_of a) (atms_of A)$
 ⟨proof⟩

lemma *atms_of_union_mset[simp]*: $atms_of (A \cup\# B) = atms_of A \cup atms_of B$
 ⟨proof⟩

lemma *finite_atms_of[iff]*: $finite (atms_of C)$
 ⟨proof⟩

lemma *atm_of_lit_in_atms_of*: $L \in\# C \implies atm_of L \in atms_of C$
 ⟨proof⟩

lemma *atms_of_plus[simp]*: $atms_of (C + D) = atms_of C \cup atms_of D$
 ⟨proof⟩

lemma *in_atms_of_minusD*: $x \in atms_of (A - B) \implies x \in atms_of A$
 ⟨proof⟩

lemma *pos_lit_in_atms_of*: $Pos A \in\# C \implies A \in atms_of C$
 ⟨proof⟩

lemma *neg_lit_in_atms_of*: $Neg A \in\# C \implies A \in atms_of C$
 ⟨proof⟩

lemma *atm_imp_pos_or_neg_lit*: $A \in atms_of C \implies Pos A \in\# C \vee Neg A \in\# C$
 ⟨proof⟩

lemma *atm_iff_pos_or_neg_lit*: $A \in atms_of L \iff Pos A \in\# L \vee Neg A \in\# L$
 ⟨proof⟩

lemma *atm_of_eq_atm_of*: $atm_of L = atm_of L' \iff (L = L' \vee L = -L')$
 ⟨proof⟩

lemma *atm_of_in_atm_of_set_iff_in_set_or_uminus_in_set*: $atm_of L \in atm_of ' I \iff (L \in I \vee -L \in I)$
 ⟨proof⟩

lemma *lits_subseteq_imp_atms_subseteq*: $set_mset C \subseteq set_mset D \implies atms_of C \subseteq atms_of D$
 ⟨proof⟩

lemma *atms_empty_iff_empty[iff]*: $atms_of C = \{\} \iff C = \{\#\}$
 ⟨proof⟩

lemma
atms_of_poss[simp]: $atms_of (poss AA) = set_mset AA$ **and**
atms_of_negs[simp]: $atms_of (negs AA) = set_mset AA$
 ⟨proof⟩

lemma *less_eq_Max_atms_of*: $C \neq \{\#\} \implies C \leq D \implies Max (atms_of C) \leq Max (atms_of D)$
 ⟨proof⟩

lemma *le_multiset_Max_in_imp_Max*:
 $Max (atms_of D) = A \implies C \leq D \implies A \in atms_of C \implies Max (atms_of C) = A$
 ⟨proof⟩

lemma *atm_of_Max_lit[simp]*: $C \neq \{\#\} \implies atm_of (Max_mset C) = Max (atms_of C)$
 ⟨proof⟩

lemma *Max_lit_eq_pos_or_neg_Max_atm*:
 $C \neq \{\#\} \implies Max_mset C = Pos (Max (atms_of C)) \vee Max_mset C = Neg (Max (atms_of C))$
 ⟨proof⟩

lemma *atms_less_imp_lit_less_pos*: $(\bigwedge B. B \in \text{atms_of } C \implies B < A) \implies L \in \# C \implies L < \text{Pos } A$
 ⟨proof⟩

lemma *atms_less_eq_imp_lit_less_eq_neg*: $(\bigwedge B. B \in \text{atms_of } C \implies B \leq A) \implies L \in \# C \implies L \leq \text{Neg } A$
 ⟨proof⟩

end

6 Herbrand Intepretation

theory *Herbrand_Interpretation*
imports *Clausal_Logic*
begin

The material formalized here corresponds roughly to Sections 2.2 (“Herbrand Interpretations”) of Bachmair and Ganzinger, excluding the formula and term syntax.

A Herbrand interpretation is a set of ground atoms that are to be considered true.

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

definition *true_lit* :: *'a interp* \Rightarrow *'a literal* \Rightarrow *bool* (**infix** $\langle \models_l \rangle$ 50) **where**
 $I \models_l L \longleftrightarrow (\text{if } \text{is_pos } L \text{ then } (\lambda P. P) \text{ else } \text{Not}) (\text{atm_of } L \in I)$

lemma *true_lit_simps*[*simp*]:
 $I \models_l \text{Pos } A \longleftrightarrow A \in I$
 $I \models_l \text{Neg } A \longleftrightarrow A \notin I$
 ⟨proof⟩

lemma *true_lit_iff*[*iff*]: $I \models_l L \longleftrightarrow (\exists A. L = \text{Pos } A \wedge A \in I \vee L = \text{Neg } A \wedge A \notin I)$
 ⟨proof⟩

definition *true_cls* :: *'a interp* \Rightarrow *'a clause* \Rightarrow *bool* (**infix** $\langle \models \rangle$ 50) **where**
 $I \models C \longleftrightarrow (\exists L \in \# C. I \models_l L)$

lemma *true_cls_empty*[*iff*]: $\neg I \models \{\#\}$
 ⟨proof⟩

lemma *true_cls_singleton*[*iff*]: $I \models \{\#L\} \longleftrightarrow I \models_l L$
 ⟨proof⟩

lemma *true_cls_add_mset*[*iff*]: $I \models \text{add_mset } C D \longleftrightarrow I \models_l C \vee I \models D$
 ⟨proof⟩

lemma *true_cls_union*[*iff*]: $I \models C + D \longleftrightarrow I \models C \vee I \models D$
 ⟨proof⟩

lemma *true_cls_mono*: $\text{set_mset } C \subseteq \text{set_mset } D \implies I \models C \implies I \models D$
 ⟨proof⟩

lemma
assumes $I \subseteq J$
shows
false_to_true_imp_ex_pos: $\neg I \models C \implies J \models C \implies \exists A \in J. \text{Pos } A \in \# C$ **and**
true_to_false_imp_ex_neg: $I \models C \implies \neg J \models C \implies \exists A \in J. \text{Neg } A \in \# C$
 ⟨proof⟩

lemma *true_cls_replicate_mset*[*iff*]: $I \models \text{replicate_mset } n L \longleftrightarrow n \neq 0 \wedge I \models_l L$
 ⟨proof⟩

lemma *pos_literal_in_imp_true_cls*[*intro*]: $\text{Pos } A \in \# C \implies A \in I \implies I \models C$
 ⟨proof⟩

lemma *neg_literal_notin_imp_true_cls*[intro]: $Neg A \in\# C \implies A \notin I \implies I \models C$
 ⟨proof⟩

lemma *pos_neg_in_imp_true*: $Pos A \in\# C \implies Neg A \in\# C \implies I \models C$
 ⟨proof⟩

definition *true_cls* :: 'a interp \Rightarrow 'a clause set \Rightarrow bool (**infix** $\langle\models\rangle$ 50) **where**
 $I \models_s CC \longleftrightarrow (\forall C \in CC. I \models C)$

lemma *true_cls_empty*[iff]: $I \models_s \{\}$
 ⟨proof⟩

lemma *true_cls_singleton*[iff]: $I \models_s \{C\} \longleftrightarrow I \models C$
 ⟨proof⟩

lemma *true_cls_insert*[iff]: $I \models_s insert C DD \longleftrightarrow I \models C \wedge I \models_s DD$
 ⟨proof⟩

lemma *true_cls_union*[iff]: $I \models_s CC \cup DD \longleftrightarrow I \models_s CC \wedge I \models_s DD$
 ⟨proof⟩

lemma *true_cls_Union*[iff]: $I \models_s \bigcup CCC \longleftrightarrow (\forall CC \in CCC. I \models_s CC)$
 ⟨proof⟩

lemma *true_cls_mono*: $DD \subseteq CC \implies I \models_s CC \implies I \models_s DD$
 ⟨proof⟩

lemma *true_cls_mono_strong*: $(\forall D \in DD. \exists C \in CC. C \subseteq\# D) \implies I \models_s CC \implies I \models_s DD$
 ⟨proof⟩

lemma *true_cls_subclause*: $C \subseteq\# D \implies I \models_s \{C\} \implies I \models_s \{D\}$
 ⟨proof⟩

abbreviation *satisfiable* :: 'a clause set \Rightarrow bool **where**
 $satisfiable CC \equiv \exists I. I \models_s CC$

lemma *satisfiable_antimono*: $CC \subseteq DD \implies satisfiable DD \implies satisfiable CC$
 ⟨proof⟩

lemma *unsatisfiable_mono*: $CC \subseteq DD \implies \neg satisfiable CC \implies \neg satisfiable DD$
 ⟨proof⟩

definition *true_cls_mset* :: 'a interp \Rightarrow 'a clause multiset \Rightarrow bool (**infix** $\langle\models\rangle$ 50) **where**
 $I \models_m CC \longleftrightarrow (\forall C \in\# CC. I \models C)$

lemma *true_cls_mset_empty*[iff]: $I \models_m \{\#\}$
 ⟨proof⟩

lemma *true_cls_mset_singleton*[iff]: $I \models_m \{\#C\#\} \longleftrightarrow I \models C$
 ⟨proof⟩

lemma *true_cls_mset_union*[iff]: $I \models_m CC + DD \longleftrightarrow I \models_m CC \wedge I \models_m DD$
 ⟨proof⟩

lemma *true_cls_mset_Union*[iff]: $I \models_m \sum\# CCC \longleftrightarrow (\forall CC \in\# CCC. I \models_m CC)$
 ⟨proof⟩

lemma *true_cls_mset_add_mset*[iff]: $I \models_m add_mset C CC \longleftrightarrow I \models C \wedge I \models_m CC$
 ⟨proof⟩

lemma *true_cls_mset_image_mset*[iff]: $I \models_m image_mset f A \longleftrightarrow (\forall x \in\# A. I \models f x)$
 ⟨proof⟩

lemma *true_cls_mset_mono*: $set_mset\ DD \subseteq set_mset\ CC \implies I \models_m CC \implies I \models_m DD$
 ⟨proof⟩

lemma *true_cls_mset_mono_strong*: $(\forall D \in\# DD. \exists C \in\# CC. C \subseteq\# D) \implies I \models_m CC \implies I \models_m DD$
 ⟨proof⟩

lemma *true_cls_set_mset[iff]*: $I \models_s set_mset\ CC \longleftrightarrow I \models_m CC$
 ⟨proof⟩

lemma *true_cls_mset_set[simp]*: $finite\ CC \implies I \models_m mset_set\ CC \longleftrightarrow I \models_s CC$
 ⟨proof⟩

lemma *true_cls_mset_true_cls*: $I \models_m CC \implies C \in\# CC \implies I \models C$
 ⟨proof⟩

end

7 Abstract Substitutions

theory *Abstract_Substitution*
imports *Clausal_Logic Map2*
begin

Atoms and substitutions are abstracted away behind some locales, to avoid having a direct dependency on the IsaFoR library.

Conventions: 's substitutions, 'a atoms.

7.1 Library

lemma *f_Suc_decr_eventually_const*:
fixes $f :: nat \Rightarrow nat$
assumes $leg: \forall i. f (Suc\ i) \leq f\ i$
shows $\exists l. \forall l' \geq l. f\ l' = f (Suc\ l')$
 ⟨proof⟩

7.2 Substitution Operators

locale *substitution_ops* =
fixes
 $subst_atm :: 'a \Rightarrow 's \Rightarrow 'a$ **and**
 $id_subst :: 's$ **and**
 $comp_subst :: 's \Rightarrow 's \Rightarrow 's$
begin

abbreviation *subst_atm_abbrev* :: $'a \Rightarrow 's \Rightarrow 'a$ (**infixl** $\langle \cdot a \rangle$ 67) **where**
 $subst_atm_abbrev \equiv subst_atm$

abbreviation *comp_subst_abbrev* :: $'s \Rightarrow 's \Rightarrow 's$ (**infixl** $\langle \odot \rangle$ 67) **where**
 $comp_subst_abbrev \equiv comp_subst$

definition *comp_substs* :: $'s\ list \Rightarrow 's\ list \Rightarrow 's\ list$ (**infixl** $\langle \odot s \rangle$ 67) **where**
 $\sigma s \odot s \tau s = map2\ comp_subst\ \sigma s\ \tau s$

definition *subst_atms* :: $'a\ set \Rightarrow 's \Rightarrow 'a\ set$ (**infixl** $\langle \cdot as \rangle$ 67) **where**
 $AA \cdot as\ \sigma = (\lambda A. A \cdot a\ \sigma) \text{ ' } AA$

definition *subst_atmss* :: $'a\ set\ set \Rightarrow 's \Rightarrow 'a\ set\ set$ (**infixl** $\langle \cdot ass \rangle$ 67) **where**
 $AAA \cdot ass\ \sigma = (\lambda AAA. AA \cdot as\ \sigma) \text{ ' } AAA$

definition *subst_atm_list* :: $'a\ list \Rightarrow 's \Rightarrow 'a\ list$ (**infixl** $\langle \cdot al \rangle$ 67) **where**
 $As \cdot al\ \sigma = map\ (\lambda A. A \cdot a\ \sigma)\ As$

definition $subst_atm_mset :: 'a\ multiset \Rightarrow 's \Rightarrow 'a\ multiset$ (**infixl** $\langle \cdot am \rangle$ 67) **where**
 $AA \cdot am \sigma = image_mset (\lambda A. A \cdot a \sigma) AA$

definition

$subst_atm_mset_list :: 'a\ multiset\ list \Rightarrow 's \Rightarrow 'a\ multiset\ list$ (**infixl** $\langle \cdot aml \rangle$ 67)

where

$AAA \cdot aml \sigma = map (\lambda AA. AA \cdot am \sigma) AAA$

definition

$subst_atm_mset_lists :: 'a\ multiset\ list \Rightarrow 's\ list \Rightarrow 'a\ multiset\ list$ (**infixl** $\langle \cdot \cdot aml \rangle$ 67)

where

$AA_s \cdot \cdot aml \sigma_s = map2 (\cdot am) AA_s \sigma_s$

definition $subst_lit :: 'a\ literal \Rightarrow 's \Rightarrow 'a\ literal$ (**infixl** $\langle \cdot l \rangle$ 67) **where**

$L \cdot l \sigma = map_literal (\lambda A. A \cdot a \sigma) L$

lemma $atm_of_subst_lit[simp]$: $atm_of (L \cdot l \sigma) = atm_of L \cdot a \sigma$

(proof)

definition $subst_cls :: 'a\ clause \Rightarrow 's \Rightarrow 'a\ clause$ (**infixl** $\langle \cdot \rangle$ 67) **where**

$AA \cdot \sigma = image_mset (\lambda A. A \cdot l \sigma) AA$

definition $subst_clss :: 'a\ clause\ set \Rightarrow 's \Rightarrow 'a\ clause\ set$ (**infixl** $\langle \cdot cs \rangle$ 67) **where**

$AA \cdot cs \sigma = (\lambda A. A \cdot \sigma) ' AA$

definition $subst_cls_list :: 'a\ clause\ list \Rightarrow 's \Rightarrow 'a\ clause\ list$ (**infixl** $\langle \cdot cl \rangle$ 67) **where**

$Cs \cdot cl \sigma = map (\lambda A. A \cdot \sigma) Cs$

definition $subst_cls_lists :: 'a\ clause\ list \Rightarrow 's\ list \Rightarrow 'a\ clause\ list$ (**infixl** $\langle \cdot \cdot cl \rangle$ 67) **where**

$Cs \cdot \cdot cl \sigma_s = map2 (\cdot) Cs \sigma_s$

definition $subst_cls_mset :: 'a\ clause\ multiset \Rightarrow 's \Rightarrow 'a\ clause\ multiset$ (**infixl** $\langle \cdot cm \rangle$ 67) **where**

$CC \cdot cm \sigma = image_mset (\lambda A. A \cdot \sigma) CC$

lemma $subst_cls_add_mset[simp]$: $add_mset L C \cdot \sigma = add_mset (L \cdot l \sigma) (C \cdot \sigma)$

(proof)

lemma $subst_cls_mset_add_mset[simp]$: $add_mset C CC \cdot cm \sigma = add_mset (C \cdot \sigma) (CC \cdot cm \sigma)$

(proof)

definition $generalizes_atm :: 'a \Rightarrow 'a \Rightarrow bool$ **where**

$generalizes_atm A B \longleftrightarrow (\exists \sigma. A \cdot a \sigma = B)$

definition $strictly_generalizes_atm :: 'a \Rightarrow 'a \Rightarrow bool$ **where**

$strictly_generalizes_atm A B \longleftrightarrow generalizes_atm A B \wedge \neg generalizes_atm B A$

definition $generalizes_lit :: 'a\ literal \Rightarrow 'a\ literal \Rightarrow bool$ **where**

$generalizes_lit L M \longleftrightarrow (\exists \sigma. L \cdot l \sigma = M)$

definition $strictly_generalizes_lit :: 'a\ literal \Rightarrow 'a\ literal \Rightarrow bool$ **where**

$strictly_generalizes_lit L M \longleftrightarrow generalizes_lit L M \wedge \neg generalizes_lit M L$

definition $generalizes :: 'a\ clause \Rightarrow 'a\ clause \Rightarrow bool$ **where**

$generalizes C D \longleftrightarrow (\exists \sigma. C \cdot \sigma = D)$

definition $strictly_generalizes :: 'a\ clause \Rightarrow 'a\ clause \Rightarrow bool$ **where**

$strictly_generalizes C D \longleftrightarrow generalizes C D \wedge \neg generalizes D C$

definition $subsumes :: 'a\ clause \Rightarrow 'a\ clause \Rightarrow bool$ **where**

$subsumes C D \longleftrightarrow (\exists \sigma. C \cdot \sigma \subseteq_{\#} D)$

definition $strictly_subsumes :: 'a\ clause \Rightarrow 'a\ clause \Rightarrow bool$ **where**

$strictly_subsumes C D \longleftrightarrow subsumes C D \wedge \neg subsumes D C$

definition *variants* :: 'a clause \Rightarrow 'a clause \Rightarrow bool **where**
variants C D \longleftrightarrow generalizes C D \wedge generalizes D C

definition *is_renaming* :: 's \Rightarrow bool **where**
is_renaming $\sigma \longleftrightarrow (\exists \tau. \sigma \odot \tau = id_subst)$

definition *is_renaming_list* :: 's list \Rightarrow bool **where**
is_renaming_list $\sigma s \longleftrightarrow (\forall \sigma \in set \sigma s. is_renaming \sigma)$

definition *inv_renaming* :: 's \Rightarrow 's **where**
inv_renaming $\sigma = (SOME \tau. \sigma \odot \tau = id_subst)$

definition *is_ground_atm* :: 'a \Rightarrow bool **where**
is_ground_atm A $\longleftrightarrow (\forall \sigma. A = A \cdot a \sigma)$

definition *is_ground_atms* :: 'a set \Rightarrow bool **where**
is_ground_atms AA = $(\forall A \in AA. is_ground_atm A)$

definition *is_ground_atm_list* :: 'a list \Rightarrow bool **where**
is_ground_atm_list As $\longleftrightarrow (\forall A \in set As. is_ground_atm A)$

definition *is_ground_atm_mset* :: 'a multiset \Rightarrow bool **where**
is_ground_atm_mset AA $\longleftrightarrow (\forall A. A \in \# AA \longrightarrow is_ground_atm A)$

definition *is_ground_lit* :: 'a literal \Rightarrow bool **where**
is_ground_lit L $\longleftrightarrow is_ground_atm (atm_of L)$

definition *is_ground_cls* :: 'a clause \Rightarrow bool **where**
is_ground_cls C $\longleftrightarrow (\forall L. L \in \# C \longrightarrow is_ground_lit L)$

definition *is_ground_cls* :: 'a clause set \Rightarrow bool **where**
is_ground_cls CC $\longleftrightarrow (\forall C \in CC. is_ground_cls C)$

definition *is_ground_cls_list* :: 'a clause list \Rightarrow bool **where**
is_ground_cls_list CC $\longleftrightarrow (\forall C \in set CC. is_ground_cls C)$

definition *is_ground_subst* :: 's \Rightarrow bool **where**
is_ground_subst $\sigma \longleftrightarrow (\forall A. is_ground_atm (A \cdot a \sigma))$

definition *is_ground_subst_list* :: 's list \Rightarrow bool **where**
is_ground_subst_list $\sigma s \longleftrightarrow (\forall \sigma \in set \sigma s. is_ground_subst \sigma)$

definition *grounding_of_cls* :: 'a clause \Rightarrow 'a clause set **where**
grounding_of_cls C = $\{C \cdot \sigma \mid \sigma. is_ground_subst \sigma\}$

definition *grounding_of_cls* :: 'a clause set \Rightarrow 'a clause set **where**
grounding_of_cls CC = $(\bigcup C \in CC. grounding_of_cls C)$

definition *is_unifier* :: 's \Rightarrow 'a set \Rightarrow bool **where**
is_unifier $\sigma AA \longleftrightarrow card (AA \cdot as \sigma) \leq 1$

definition *is_unifiers* :: 's \Rightarrow 'a set set \Rightarrow bool **where**
is_unifiers $\sigma AAA \longleftrightarrow (\forall AA \in AAA. is_unifier \sigma AA)$

definition *is_mgu* :: 's \Rightarrow 'a set set \Rightarrow bool **where**
is_mgu $\sigma AAA \longleftrightarrow is_unifiers \sigma AAA \wedge (\forall \tau. is_unifiers \tau AAA \longrightarrow (\exists \gamma. \tau = \sigma \odot \gamma))$

definition *is_imgu* :: 's \Rightarrow 'a set set \Rightarrow bool **where**
is_imgu $\sigma AAA \longleftrightarrow is_unifiers \sigma AAA \wedge (\forall \tau. is_unifiers \tau AAA \longrightarrow \tau = \sigma \odot \tau)$

definition *var_disjoint* :: 'a clause list \Rightarrow bool **where**
var_disjoint Cs \longleftrightarrow

$(\forall \sigma s. \text{length } \sigma s = \text{length } Cs \longrightarrow (\exists \tau. \forall i < \text{length } Cs. \forall S. S \subseteq\# Cs ! i \longrightarrow S \cdot \sigma s ! i = S \cdot \tau))$

end

7.3 Substitution Lemmas

locale substitution = substitution_ops subst_atm id_subst comp_subst

for

subst_atm :: 'a \Rightarrow 's \Rightarrow 'a and

id_subst :: 's and

comp_subst :: 's \Rightarrow 's \Rightarrow 's +

assumes

subst_atm_id_subst[simp]: A · a id_subst = A and

subst_atm_comp_subst[simp]: A · a (σ ⊙ τ) = (A · a σ) · a τ and

subst_ext: (∧ A. A · a σ = A · a τ) \implies σ = τ and

make_ground_subst: is_ground_cls (C · σ) \implies ∃ τ. is_ground_subst τ ∧ C · τ = C · σ and

wf_strictly_generalizes_atm: wfP strictly_generalizes_atm

begin

lemma subst_ext_iff: σ = τ \longleftrightarrow (∧ A. A · a σ = A · a τ)
 <proof>

7.3.1 Identity Substitution

lemma id_subst_comp_subst[simp]: id_subst ⊙ σ = σ
 <proof>

lemma comp_subst_id_subst[simp]: σ ⊙ id_subst = σ
 <proof>

lemma id_subst_comp_substs[simp]: replicate (length σ s) id_subst ⊙ s σ s = σ s
 <proof>

lemma comp_substs_id_subst[simp]: σ s ⊙ s replicate (length σ s) id_subst = σ s
 <proof>

lemma subst_atms_id_subst[simp]: AA · as id_subst = AA
 <proof>

lemma subst_atmss_id_subst[simp]: AAA · ass id_subst = AAA
 <proof>

lemma subst_atm_list_id_subst[simp]: As · al id_subst = As
 <proof>

lemma subst_atm_mset_id_subst[simp]: AA · am id_subst = AA
 <proof>

lemma subst_atm_mset_list_id_subst[simp]: AAs · aml id_subst = AAs
 <proof>

lemma subst_atm_mset_lists_id_subst[simp]: AAs · aml replicate (length AAs) id_subst = AAs
 <proof>

lemma subst_lit_id_subst[simp]: L · l id_subst = L
 <proof>

lemma subst_cls_id_subst[simp]: C · id_subst = C
 <proof>

lemma subst_cls_id_subst[simp]: CC · cs id_subst = CC
 <proof>

lemma subst_cls_list_id_subst[simp]: Cs · cl id_subst = Cs

<proof>

lemma *subst_cls_lists_id_subst[simp]*: $Cs \cdot cl$ replicate (length Cs) $id_subst = Cs$
<proof>

lemma *subst_cls_mset_id_subst[simp]*: $CC \cdot cm$ $id_subst = CC$
<proof>

7.3.2 Associativity of Composition

lemma *comp_subst_assoc[simp]*: $\sigma \odot (\tau \odot \gamma) = \sigma \odot \tau \odot \gamma$
<proof>

7.3.3 Compatibility of Substitution and Composition

lemma *subst_atms_comp_subst[simp]*: $AA \cdot as$ $(\tau \odot \sigma) = AA \cdot as$ $\tau \cdot as$ σ
<proof>

lemma *subst_atmss_comp_subst[simp]*: $AAA \cdot ass$ $(\tau \odot \sigma) = AAA \cdot ass$ $\tau \cdot ass$ σ
<proof>

lemma *subst_atm_list_comp_subst[simp]*: $As \cdot al$ $(\tau \odot \sigma) = As \cdot al$ $\tau \cdot al$ σ
<proof>

lemma *subst_atm_mset_comp_subst[simp]*: $AA \cdot am$ $(\tau \odot \sigma) = AA \cdot am$ $\tau \cdot am$ σ
<proof>

lemma *subst_atm_mset_list_comp_subst[simp]*: $AAs \cdot aml$ $(\tau \odot \sigma) = (AAs \cdot aml$ $\tau) \cdot aml$ σ
<proof>

lemma *subst_atm_mset_lists_comp_substs[simp]*: $AAs \cdot aml$ $(\tau s \odot s \sigma s) = AAs \cdot aml$ $\tau s \cdot aml$ σs
<proof>

lemma *subst_lit_comp_subst[simp]*: $L \cdot l$ $(\tau \odot \sigma) = L \cdot l$ $\tau \cdot l$ σ
<proof>

lemma *subst_cls_comp_subst[simp]*: $C \cdot$ $(\tau \odot \sigma) = C \cdot \tau \cdot \sigma$
<proof>

lemma *subst_clscomp_subst[simp]*: $CC \cdot cs$ $(\tau \odot \sigma) = CC \cdot cs$ $\tau \cdot cs$ σ
<proof>

lemma *subst_cls_list_comp_subst[simp]*: $Cs \cdot cl$ $(\tau \odot \sigma) = Cs \cdot cl$ $\tau \cdot cl$ σ
<proof>

lemma *subst_cls_lists_comp_substs[simp]*: $Cs \cdot cl$ $(\tau s \odot s \sigma s) = Cs \cdot cl$ $\tau s \cdot cl$ σs
<proof>

lemma *subst_cls_mset_comp_subst[simp]*: $CC \cdot cm$ $(\tau \odot \sigma) = CC \cdot cm$ $\tau \cdot cm$ σ
<proof>

7.3.4 “Commutativity” of Membership and Substitution

lemma *Melem_subst_atm_mset[simp]*: $A \in\#$ $AA \cdot am$ $\sigma \longleftrightarrow (\exists B. B \in\#$ $AA \wedge A = B \cdot a$ $\sigma)$
<proof>

lemma *Melem_subst_cls[simp]*: $L \in\#$ $C \cdot \sigma \longleftrightarrow (\exists M. M \in\#$ $C \wedge L = M \cdot l$ $\sigma)$
<proof>

lemma *Melem_subst_cls_mset[simp]*: $AA \in\#$ $CC \cdot cm$ $\sigma \longleftrightarrow (\exists BB. BB \in\#$ $CC \wedge AA = BB \cdot \sigma)$
<proof>

7.3.5 Signs and Substitutions

lemma *subst_lit_is_neg[simp]*: is_neg $(L \cdot l$ $\sigma) = is_neg$ L

<proof>

lemma *subst_lit_is_pos[simp]*: $is_pos (L \cdot l \sigma) = is_pos L$
<proof>

lemma *subst_minus[simp]*: $(- L) \cdot l \mu = - (L \cdot l \mu)$
<proof>

7.3.6 Substitution on Literal(s)

lemma *eql_neg_lit_eql_atm[simp]*: $(Neg A' \cdot l \eta) = Neg A \iff A' \cdot a \eta = A$
<proof>

lemma *eql_pos_lit_eql_atm[simp]*: $(Pos A' \cdot l \eta) = Pos A \iff A' \cdot a \eta = A$
<proof>

lemma *subst_cls_negs[simp]*: $(negs AA) \cdot \sigma = negs (AA \cdot am \sigma)$
<proof>

lemma *subst_cls_poss[simp]*: $(poss AA) \cdot \sigma = poss (AA \cdot am \sigma)$
<proof>

lemma *atms_of_subst_atms*: $atms_of C \cdot as \sigma = atms_of (C \cdot \sigma)$
<proof>

lemma *in_image_Neg_is_neg[simp]*: $L \cdot l \sigma \in Neg \text{ ' } AA \implies is_neg L$
<proof>

lemma *subst_lit_in_negs_subst_is_neg*: $L \cdot l \sigma \in \# (negs AA) \cdot \tau \implies is_neg L$
<proof>

lemma *subst_lit_in_negs_is_neg*: $L \cdot l \sigma \in \# negs AA \implies is_neg L$
<proof>

7.3.7 Substitution on Empty

lemma *subst_atms_empty[simp]*: $\{\} \cdot as \sigma = \{\}$
<proof>

lemma *subst_atmss_empty[simp]*: $\{\} \cdot ass \sigma = \{\}$
<proof>

lemma *comp_substs_empty_iff[simp]*: $\sigma s \odot s \eta s = [] \iff \sigma s = [] \vee \eta s = []$
<proof>

lemma *subst_atm_list_empty[simp]*: $[] \cdot al \sigma = []$
<proof>

lemma *subst_atm_mset_empty[simp]*: $\{\#\} \cdot am \sigma = \{\#\}$
<proof>

lemma *subst_atm_mset_list_empty[simp]*: $[] \cdot aml \sigma = []$
<proof>

lemma *subst_atm_mset_lists_empty[simp]*: $[] \cdot \cdot aml \sigma s = []$
<proof>

lemma *subst_cls_empty[simp]*: $\{\#\} \cdot \sigma = \{\#\}$
<proof>

lemma *subst_cls_empty[simp]*: $\{\} \cdot cs \sigma = \{\}$
<proof>

lemma *subst_cls_list_empty[simp]*: $[] \cdot cl \sigma = []$

<proof>

lemma *subst_cls_lists_empty[simp]*: $[\] \cdot cl \sigma = [\]$
<proof>

lemma *subst_scls_mset_empty[simp]*: $\{\#\} \cdot cm \sigma = \{\#\}$
<proof>

lemma *subst_atms_empty_iff[simp]*: $AA \cdot as \eta = \{\}$ \longleftrightarrow $AA = \{\}$
<proof>

lemma *subst_atmss_empty_iff[simp]*: $AAA \cdot ass \eta = \{\}$ \longleftrightarrow $AAA = \{\}$
<proof>

lemma *subst_atm_list_empty_iff[simp]*: $As \cdot al \eta = [\]$ \longleftrightarrow $As = [\]$
<proof>

lemma *subst_atm_mset_empty_iff[simp]*: $AA \cdot am \eta = \{\#\}$ \longleftrightarrow $AA = \{\#\}$
<proof>

lemma *subst_atm_mset_list_empty_iff[simp]*: $AAs \cdot aml \eta = [\]$ \longleftrightarrow $AAs = [\]$
<proof>

lemma *subst_atm_mset_lists_empty_iff[simp]*: $AAs \cdot aml \eta s = [\]$ \longleftrightarrow $(AAs = [\] \vee \eta s = [\])$
<proof>

lemma *subst_cls_empty_iff[simp]*: $C \cdot \eta = \{\#\}$ \longleftrightarrow $C = \{\#\}$
<proof>

lemma *subst_cls_empty_iff[simp]*: $CC \cdot cs \eta = \{\}$ \longleftrightarrow $CC = \{\}$
<proof>

lemma *subst_cls_list_empty_iff[simp]*: $Cs \cdot cl \eta = [\]$ \longleftrightarrow $Cs = [\]$
<proof>

lemma *subst_cls_lists_empty_iff[simp]*: $Cs \cdot cl \eta s = [\]$ \longleftrightarrow $Cs = [\] \vee \eta s = [\]$
<proof>

lemma *subst_cls_mset_empty_iff[simp]*: $CC \cdot cm \eta = \{\#\}$ \longleftrightarrow $CC = \{\#\}$
<proof>

7.3.8 Substitution on a Union

lemma *subst_atms_union[simp]*: $(AA \cup BB) \cdot as \sigma = AA \cdot as \sigma \cup BB \cdot as \sigma$
<proof>

lemma *subst_atmss_union[simp]*: $(AAA \cup BBB) \cdot ass \sigma = AAA \cdot ass \sigma \cup BBB \cdot ass \sigma$
<proof>

lemma *subst_atm_list_append[simp]*: $(As @ Bs) \cdot al \sigma = As \cdot al \sigma @ Bs \cdot al \sigma$
<proof>

lemma *subst_atm_mset_union[simp]*: $(AA + BB) \cdot am \sigma = AA \cdot am \sigma + BB \cdot am \sigma$
<proof>

lemma *subst_atm_mset_list_append[simp]*: $(AAs @ BBs) \cdot aml \sigma = AAs \cdot aml \sigma @ BBs \cdot aml \sigma$
<proof>

lemma *subst_cls_union[simp]*: $(C + D) \cdot \sigma = C \cdot \sigma + D \cdot \sigma$
<proof>

lemma *subst_cls_union[simp]*: $(CC \cup DD) \cdot cs \sigma = CC \cdot cs \sigma \cup DD \cdot cs \sigma$
<proof>

lemma *subst_cls_list_append[simp]*: $(Cs @ Ds) \cdot cl \sigma = Cs \cdot cl \sigma @ Ds \cdot cl \sigma$
 ⟨proof⟩

lemma *subst_cls_lists_append[simp]*:
 $length\ Cs = length\ \sigma s \implies length\ Cs' = length\ \sigma s' \implies$
 $(Cs @ Cs') \cdot cl (\sigma s @ \sigma s') = Cs \cdot cl \sigma s @ Cs' \cdot cl \sigma s'$
 ⟨proof⟩

lemma *subst_cls_mset_union[simp]*: $(CC + DD) \cdot cm \sigma = CC \cdot cm \sigma + DD \cdot cm \sigma$
 ⟨proof⟩

7.3.9 Substitution on a Singleton

lemma *subst_atms_single[simp]*: $\{A\} \cdot as \sigma = \{A \cdot a \sigma\}$
 ⟨proof⟩

lemma *subst_atmss_single[simp]*: $\{AA\} \cdot ass \sigma = \{AA \cdot as \sigma\}$
 ⟨proof⟩

lemma *subst_atm_list_single[simp]*: $[A] \cdot al \sigma = [A \cdot a \sigma]$
 ⟨proof⟩

lemma *subst_atm_mset_single[simp]*: $\{\#A\#\} \cdot am \sigma = \{\#A \cdot a \sigma\#\}$
 ⟨proof⟩

lemma *subst_atm_mset_list[simp]*: $[AA] \cdot aml \sigma = [AA \cdot am \sigma]$
 ⟨proof⟩

lemma *subst_cls_single[simp]*: $\{\#L\#\} \cdot \sigma = \{\#L \cdot l \sigma\#\}$
 ⟨proof⟩

lemma *subst_cls_single[simp]*: $\{C\} \cdot cs \sigma = \{C \cdot \sigma\}$
 ⟨proof⟩

lemma *subst_cls_list_single[simp]*: $[C] \cdot cl \sigma = [C \cdot \sigma]$
 ⟨proof⟩

lemma *subst_cls_lists_single[simp]*: $[C] \cdot cl [\sigma] = [C \cdot \sigma]$
 ⟨proof⟩

lemma *subst_cls_mset_single[simp]*: $\{\#C\#\} \cdot cm \sigma = \{\#C \cdot \sigma\#\}$
 ⟨proof⟩

7.3.10 Substitution on (#)

lemma *subst_atm_list_Cons[simp]*: $(A \# As) \cdot al \sigma = A \cdot a \sigma \# As \cdot al \sigma$
 ⟨proof⟩

lemma *subst_atm_mset_list_Cons[simp]*: $(A \# As) \cdot aml \sigma = A \cdot am \sigma \# As \cdot aml \sigma$
 ⟨proof⟩

lemma *subst_atm_mset_lists_Cons[simp]*: $(C \# Cs) \cdot aml (\sigma \# \sigma s) = C \cdot am \sigma \# Cs \cdot aml \sigma s$
 ⟨proof⟩

lemma *subst_cls_list_Cons[simp]*: $(C \# Cs) \cdot cl \sigma = C \cdot \sigma \# Cs \cdot cl \sigma$
 ⟨proof⟩

lemma *subst_cls_lists_Cons[simp]*: $(C \# Cs) \cdot cl (\sigma \# \sigma s) = C \cdot \sigma \# Cs \cdot cl \sigma s$
 ⟨proof⟩

7.3.11 Substitution on tl

lemma *subst_atm_list_tl[simp]*: $tl (As \cdot al \sigma) = tl As \cdot al \sigma$
 ⟨proof⟩

lemma *subst_atm_mset_list_tl[simp]*: $tl (AAs \cdot aml \sigma) = tl AAs \cdot aml \sigma$
 ⟨proof⟩

lemma *subst_cls_list_tl[simp]*: $tl (Cs \cdot cl \sigma) = tl Cs \cdot cl \sigma$
 ⟨proof⟩

lemma *subst_cls_lists_tl[simp]*: $length Cs = length \sigma s \implies tl (Cs \cdot cl \sigma s) = tl Cs \cdot cl tl \sigma s$
 ⟨proof⟩

7.3.12 Substitution on (!)

lemma *comp_substs_nth[simp]*:
 $length \tau s = length \sigma s \implies i < length \tau s \implies (\tau s \odot s \sigma s) ! i = (\tau s ! i) \odot (\sigma s ! i)$
 ⟨proof⟩

lemma *subst_atm_list_nth[simp]*: $i < length As \implies (As \cdot al \tau) ! i = As ! i \cdot a \tau$
 ⟨proof⟩

lemma *subst_atm_mset_list_nth[simp]*: $i < length AAs \implies (AAs \cdot aml \eta) ! i = (AAs ! i) \cdot am \eta$
 ⟨proof⟩

lemma *subst_atm_mset_lists_nth[simp]*:
 $length AAs = length \sigma s \implies i < length AAs \implies (AAs \cdot aml \sigma s) ! i = (AAs ! i) \cdot am (\sigma s ! i)$
 ⟨proof⟩

lemma *subst_cls_list_nth[simp]*: $i < length Cs \implies (Cs \cdot cl \tau) ! i = (Cs ! i) \cdot \tau$
 ⟨proof⟩

lemma *subst_cls_lists_nth[simp]*:
 $length Cs = length \sigma s \implies i < length Cs \implies (Cs \cdot cl \sigma s) ! i = (Cs ! i) \cdot (\sigma s ! i)$
 ⟨proof⟩

7.3.13 Substitution on Various Other Functions

lemma *subst_cls_image[simp]*: $image f X \cdot cs \sigma = \{f x \cdot \sigma \mid x. x \in X\}$
 ⟨proof⟩

lemma *subst_cls_mset_image_mset[simp]*: $image_mset f X \cdot cm \sigma = \{\# f x \cdot \sigma. x \in \# X \#\}$
 ⟨proof⟩

lemma *mset_subst_atm_list_subst_atm_mset[simp]*: $mset (As \cdot al \sigma) = mset (As) \cdot am \sigma$
 ⟨proof⟩

lemma *mset_subst_cls_list_subst_cls_mset*: $mset (Cs \cdot cl \sigma) = (mset Cs) \cdot cm \sigma$
 ⟨proof⟩

lemma *sum_list_subst_cls_list_subst_cls[simp]*: $sum_list (Cs \cdot cl \eta) = sum_list Cs \cdot \eta$
 ⟨proof⟩

lemma *set_mset_subst_cls_mset_subst_cls*: $set_mset (CC \cdot cm \mu) = (set_mset CC) \cdot cs \mu$
 ⟨proof⟩

lemma *Neg_Melem_subst_atm_subst_cls[simp]*: $Neg A \in \# C \implies Neg (A \cdot a \sigma) \in \# C \cdot \sigma$
 ⟨proof⟩

lemma *Pos_Melem_subst_atm_subst_cls[simp]*: $Pos A \in \# C \implies Pos (A \cdot a \sigma) \in \# C \cdot \sigma$
 ⟨proof⟩

lemma *in_atms_of_subst[simp]*: $B \in atms_of C \implies B \cdot a \sigma \in atms_of (C \cdot \sigma)$
 ⟨proof⟩

7.3.14 Renamings

lemma *is_renaming_id_subst[simp]: is_renaming id_subst*
 ⟨proof⟩

lemma *is_renamingD: is_renaming $\sigma \implies (\forall A1 A2. A1 \cdot a \sigma = A2 \cdot a \sigma \longleftrightarrow A1 = A2)$*
 ⟨proof⟩

lemma *inv_renaming_cancel_r[simp]: is_renaming $r \implies r \odot \text{inv_renaming } r = \text{id_subst}$*
 ⟨proof⟩

lemma *inv_renaming_cancel_r_list[simp]:*
is_renaming_list $rs \implies rs \odot s \text{ map inv_renaming } rs = \text{replicate (length } rs) \text{ id_subst}$
 ⟨proof⟩

lemma *Nil_comp_substs[simp]: [] $\odot s = []$*
 ⟨proof⟩

lemma *comp_substs_Nil[simp]: $s \odot s [] = []$*
 ⟨proof⟩

lemma *is_renaming_idempotent_id_subst: is_renaming $r \implies r \odot r = r \implies r = \text{id_subst}$*
 ⟨proof⟩

lemma *is_renaming_left_id_subst_right_id_subst:*
is_renaming $r \implies s \odot r = \text{id_subst} \implies r \odot s = \text{id_subst}$
 ⟨proof⟩

lemma *is_renaming_closure: is_renaming $r1 \implies is_renaming r2 \implies is_renaming (r1 \odot r2)$*
 ⟨proof⟩

lemma *is_renaming_inv_renaming_cancel_atm[simp]: is_renaming $\rho \implies A \cdot a \rho \cdot a \text{ inv_renaming } \rho = A$*
 ⟨proof⟩

lemma *is_renaming_inv_renaming_cancel_atms[simp]: is_renaming $\rho \implies AA \cdot as \rho \cdot as \text{ inv_renaming } \rho = AA$*
 ⟨proof⟩

lemma *is_renaming_inv_renaming_cancel_atmss[simp]: is_renaming $\rho \implies AAA \cdot ass \rho \cdot ass \text{ inv_renaming } \rho = AAA$*
 ⟨proof⟩

lemma *is_renaming_inv_renaming_cancel_atm_list[simp]: is_renaming $\rho \implies As \cdot al \rho \cdot al \text{ inv_renaming } \rho = As$*
 ⟨proof⟩

lemma *is_renaming_inv_renaming_cancel_atm_mset[simp]: is_renaming $\rho \implies AA \cdot am \rho \cdot am \text{ inv_renaming } \rho = AA$*
 ⟨proof⟩

lemma *is_renaming_inv_renaming_cancel_atm_mset_list[simp]: is_renaming $\rho \implies (AAs \cdot aml \rho) \cdot aml \text{ inv_renaming } \rho = AAs$*
 ⟨proof⟩

lemma *is_renaming_list_inv_renaming_cancel_atm_mset_lists[simp]:*
length $AAs = \text{length } \rho s \implies is_renaming_list \rho s \implies AAs \cdot aml \rho s \cdot aml \text{ map inv_renaming } \rho s = AAs$
 ⟨proof⟩

lemma *is_renaming_inv_renaming_cancel_lit[simp]: is_renaming $\rho \implies (L \cdot l \rho) \cdot l \text{ inv_renaming } \rho = L$*
 ⟨proof⟩

lemma *is_renaming_inv_renaming_cancel_cls[simp]: is_renaming $\rho \implies C \cdot \rho \cdot \text{inv_renaming } \rho = C$*
 ⟨proof⟩

lemma *is_renaming_inv_renaming_cancel_cls[simp]:*
is_renaming $\rho \implies CC \cdot cs \rho \cdot cs \text{ inv_renaming } \rho = CC$

<proof>

lemma *is_renaming_inv_renaming_cancel_cls_list[simp]*:
 $is_renaming\ \varrho \implies Cs \cdot cl\ \varrho \cdot cl\ inv_renaming\ \varrho = Cs$
<proof>

lemma *is_renaming_list_inv_renaming_cancel_cls_list[simp]*:
 $length\ Cs = length\ \varrho s \implies is_renaming_list\ \varrho s \implies Cs \cdot cl\ \varrho s \cdot cl\ map\ inv_renaming\ \varrho s = Cs$
<proof>

lemma *is_renaming_inv_renaming_cancel_cls_mset[simp]*:
 $is_renaming\ \varrho \implies CC \cdot cm\ \varrho \cdot cm\ inv_renaming\ \varrho = CC$
<proof>

7.3.15 Monotonicity

lemma *subst_cls_mono*: $set_mset\ C \subseteq set_mset\ D \implies set_mset\ (C \cdot \sigma) \subseteq set_mset\ (D \cdot \sigma)$
<proof>

lemma *subst_cls_mono_mset*: $C \subseteq\# D \implies C \cdot \sigma \subseteq\# D \cdot \sigma$
<proof>

lemma *subst_subset_mono*: $D \subset\# C \implies D \cdot \sigma \subset\# C \cdot \sigma$
<proof>

7.3.16 Size after Substitution

lemma *size_subst[simp]*: $size\ (D \cdot \sigma) = size\ D$
<proof>

lemma *subst_atm_list_length[simp]*: $length\ (As \cdot al\ \sigma) = length\ As$
<proof>

lemma *length_subst_atm_mset_list[simp]*: $length\ (AAs \cdot aml\ \eta) = length\ AAs$
<proof>

lemma *subst_atm_mset_lists_length[simp]*: $length\ (AAs \cdot aml\ \sigma s) = \min\ (length\ AAs)\ (length\ \sigma s)$
<proof>

lemma *subst_cls_list_length[simp]*: $length\ (Cs \cdot cl\ \sigma) = length\ Cs$
<proof>

lemma *comp_substs_length[simp]*: $length\ (\tau s \odot s\ \sigma s) = \min\ (length\ \tau s)\ (length\ \sigma s)$
<proof>

lemma *subst_cls_lists_length[simp]*: $length\ (Cs \cdot cl\ \sigma s) = \min\ (length\ Cs)\ (length\ \sigma s)$
<proof>

7.3.17 Variable Disjointness

lemma *var_disjoint_clauses*:
assumes *var_disjoint* Cs
shows $\forall \sigma s. length\ \sigma s = length\ Cs \implies (\exists \tau. Cs \cdot cl\ \sigma s = Cs \cdot cl\ \tau)$
<proof>

7.3.18 Ground Expressions and Substitutions

lemma *ex_ground_subst*: $\exists \sigma. is_ground_subst\ \sigma$
<proof>

lemma *is_ground_cls_list_Cons[simp]*:
 $is_ground_cls_list\ (C \# Cs) = (is_ground_cls\ C \wedge is_ground_cls_list\ Cs)$
<proof>

Ground union lemma $is_ground_atms_union[simp]: is_ground_atms (AA \cup BB) \longleftrightarrow is_ground_atms AA \wedge is_ground_atms BB$
 ⟨proof⟩

lemma $is_ground_atm_mset_union[simp]: is_ground_atm_mset (AA + BB) \longleftrightarrow is_ground_atm_mset AA \wedge is_ground_atm_mset BB$
 ⟨proof⟩

lemma $is_ground_cls_union[simp]: is_ground_cls (C + D) \longleftrightarrow is_ground_cls C \wedge is_ground_cls D$
 ⟨proof⟩

lemma $is_ground_class_union[simp]: is_ground_class (CC \cup DD) \longleftrightarrow is_ground_class CC \wedge is_ground_class DD$
 ⟨proof⟩

lemma $is_ground_cls_list_is_ground_cls_sum_list[simp]: is_ground_cls_list Cs \Longrightarrow is_ground_cls (sum_list Cs)$
 ⟨proof⟩

Grounding simplifications lemma $grounding_of_class_empty[simp]: grounding_of_class \{\} = \{\}$
 ⟨proof⟩

lemma $grounding_of_class_singleton[simp]: grounding_of_class \{C\} = grounding_of_cls C$
 ⟨proof⟩

lemma $grounding_of_class_insert: grounding_of_class (insert C N) = grounding_of_cls C \cup grounding_of_class N$
 ⟨proof⟩

lemma $grounding_of_class_union: grounding_of_class (A \cup B) = grounding_of_class A \cup grounding_of_class B$
 ⟨proof⟩

Grounding monotonicity lemma $is_ground_cls_mono: C \subseteq\# D \Longrightarrow is_ground_cls D \Longrightarrow is_ground_cls C$
 ⟨proof⟩

lemma $is_ground_class_mono: CC \subseteq DD \Longrightarrow is_ground_class DD \Longrightarrow is_ground_class CC$
 ⟨proof⟩

lemma $grounding_of_class_mono: CC \subseteq DD \Longrightarrow grounding_of_class CC \subseteq grounding_of_class DD$
 ⟨proof⟩

lemma $sum_list_subseq_mset_is_ground_cls_list[simp]: sum_list Cs \subseteq\# sum_list Ds \Longrightarrow is_ground_cls_list Ds \Longrightarrow is_ground_cls_list Cs$
 ⟨proof⟩

Substituting on ground expression preserves ground lemma $is_ground_comp_subst[simp]: is_ground_subst \sigma \Longrightarrow is_ground_subst (\tau \odot \sigma)$
 ⟨proof⟩

lemma $ground_subst_ground_atm[simp]: is_ground_subst \sigma \Longrightarrow is_ground_atm (A \cdot a \sigma)$
 ⟨proof⟩

lemma $ground_subst_ground_lit[simp]: is_ground_subst \sigma \Longrightarrow is_ground_lit (L \cdot l \sigma)$
 ⟨proof⟩

lemma $ground_subst_ground_cls[simp]: is_ground_subst \sigma \Longrightarrow is_ground_cls (C \cdot \sigma)$
 ⟨proof⟩

lemma $ground_subst_ground_class[simp]: is_ground_subst \sigma \Longrightarrow is_ground_class (CC \cdot cs \sigma)$
 ⟨proof⟩

lemma *ground_subst_ground_cls_list[simp]*: $is_ground_subst\ \sigma \implies is_ground_cls_list\ (Cs \cdot cl\ \sigma)$
 ⟨proof⟩

lemma *ground_subst_ground_cls_lists[simp]*:
 $\forall \sigma \in set\ \sigma s. is_ground_subst\ \sigma \implies is_ground_cls_list\ (Cs \cdot cl\ \sigma s)$
 ⟨proof⟩

lemma *subst_cls_eq_grounding_of_cls_subset_eq*:
assumes $D \cdot \sigma = C$
shows $grounding_of_cls\ C \subseteq grounding_of_cls\ D$
 ⟨proof⟩

Substituting on ground expression has no effect **lemma** *is_ground_subst_atm[simp]*: $is_ground_atm\ A \implies A \cdot a\ \sigma = A$
 ⟨proof⟩

lemma *is_ground_subst_atms[simp]*: $is_ground_atms\ AA \implies AA \cdot as\ \sigma = AA$
 ⟨proof⟩

lemma *is_ground_subst_atm_mset[simp]*: $is_ground_atm_mset\ AA \implies AA \cdot am\ \sigma = AA$
 ⟨proof⟩

lemma *is_ground_subst_atm_list[simp]*: $is_ground_atm_list\ As \implies As \cdot al\ \sigma = As$
 ⟨proof⟩

lemma *is_ground_subst_atm_list_member[simp]*:
 $is_ground_atm_list\ As \implies i < length\ As \implies As\ !\ i \cdot a\ \sigma = As\ !\ i$
 ⟨proof⟩

lemma *is_ground_subst_lit[simp]*: $is_ground_lit\ L \implies L \cdot l\ \sigma = L$
 ⟨proof⟩

lemma *is_ground_subst_cls[simp]*: $is_ground_cls\ C \implies C \cdot \sigma = C$
 ⟨proof⟩

lemma *is_ground_subst_cls[simp]*: $is_ground_cls\ CC \implies CC \cdot cs\ \sigma = CC$
 ⟨proof⟩

lemma *is_ground_subst_cls_lists[simp]*:
assumes $length\ P = length\ Cs$ **and** $is_ground_cls_list\ Cs$
shows $Cs \cdot cl\ P = Cs$
 ⟨proof⟩

lemma *is_ground_subst_lit_iff*: $is_ground_lit\ L \iff (\forall \sigma. L = L \cdot l\ \sigma)$
 ⟨proof⟩

lemma *is_ground_subst_cls_iff*: $is_ground_cls\ C \iff (\forall \sigma. C = C \cdot \sigma)$
 ⟨proof⟩

Grounding of substitutions **lemma** *grounding_of_subst_cls_subset*: $grounding_of_cls\ (C \cdot \mu) \subseteq grounding_of_cls\ C$
 ⟨proof⟩

lemma *grounding_of_subst_cls_subset*: $grounding_of_cls\ (CC \cdot cs\ \mu) \subseteq grounding_of_cls\ CC$
 ⟨proof⟩

lemma *grounding_of_subst_cls_renaming_ident[simp]*:
assumes $is_renaming\ \varrho$
shows $grounding_of_cls\ (C \cdot \varrho) = grounding_of_cls\ C$
 ⟨proof⟩

lemma *grounding_of_subst_cls_renaming_ident[simp]*:
assumes $is_renaming\ \varrho$

shows $\text{grounding_of_clss } (CC \cdot cs \varrho) = \text{grounding_of_clss } CC$
 ⟨proof⟩

Members of ground expressions are ground lemma $\text{is_ground_cls_as_atms}: \text{is_ground_cls } C \longleftrightarrow (\forall A \in \text{atms_of } C. \text{is_ground_atm } A)$
 ⟨proof⟩

lemma $\text{is_ground_cls_imp_is_ground_lit}: L \in \# C \implies \text{is_ground_cls } C \implies \text{is_ground_lit } L$
 ⟨proof⟩

lemma $\text{is_ground_cls_imp_is_ground_atm}: A \in \text{atms_of } C \implies \text{is_ground_cls } C \implies \text{is_ground_atm } A$
 ⟨proof⟩

lemma $\text{is_ground_cls_is_ground_atms_atms_of}[simp]: \text{is_ground_cls } C \implies \text{is_ground_atms } (\text{atms_of } C)$
 ⟨proof⟩

lemma $\text{grounding_ground}: C \in \text{grounding_of_clss } M \implies \text{is_ground_cls } C$
 ⟨proof⟩

lemma $\text{is_ground_cls_if_in_grounding_of_cls}: C' \in \text{grounding_of_cls } C \implies \text{is_ground_cls } C'$
 ⟨proof⟩

lemma $\text{in_subset_eq_grounding_of_clss_is_ground_cls}[simp]:$
 $C \in CC \implies CC \subseteq \text{grounding_of_clss } DD \implies \text{is_ground_cls } C$
 ⟨proof⟩

lemma $\text{is_ground_cls_empty}[simp]: \text{is_ground_cls } \{\#\}$
 ⟨proof⟩

lemma $\text{is_ground_cls_add_mset}[simp]:$
 $\text{is_ground_cls } (\text{add_mset } L C) \longleftrightarrow \text{is_ground_lit } L \wedge \text{is_ground_cls } C$
 ⟨proof⟩

lemma $\text{grounding_of_cls_ground}: \text{is_ground_cls } C \implies \text{grounding_of_cls } C = \{C\}$
 ⟨proof⟩

lemma $\text{grounding_of_cls_empty}[simp]: \text{grounding_of_cls } \{\#\} = \{\{\#\}\}$
 ⟨proof⟩

lemma $\text{union_grounding_of_cls_ground}: \text{is_ground_clss } (\bigcup (\text{grounding_of_cls } 'N))$
 ⟨proof⟩

lemma $\text{is_ground_clss_grounding_of_clss}[simp]: \text{is_ground_clss } (\text{grounding_of_clss } N)$
 ⟨proof⟩

Grounding idempotence lemma $\text{grounding_of_grounding_of_cls}: E \in \text{grounding_of_cls } D \implies D \in \text{grounding_of_cls } C \implies E = D$
 ⟨proof⟩

lemma $\text{image_grounding_of_cls_grounding_of_cls}: \text{grounding_of_cls } ' \text{grounding_of_cls } C = (\lambda x. \{x\}) ' \text{grounding_of_cls } C$
 ⟨proof⟩

lemma $\text{grounding_of_clss_grounding_of_clss}[simp]: \text{grounding_of_clss } (\text{grounding_of_clss } N) = \text{grounding_of_clss } N$
 ⟨proof⟩

7.3.19 Subsumption

lemma $\text{subsumes_empty_left}[simp]: \text{subsumes } \{\#\} C$
 ⟨proof⟩

lemma $\text{strictly_subsumes_empty_left}[simp]: \text{strictly_subsumes } \{\#\} C \longleftrightarrow C \neq \{\#\}$
 ⟨proof⟩

7.3.20 Unifiers

lemma *card_le_one_alt*: $\text{finite } X \implies \text{card } X \leq 1 \iff X = \{\} \vee (\exists x. X = \{x\})$
 ⟨proof⟩

lemma *is_unifier_subst_atm_eqI*:
 assumes *finite AA*
 shows $\text{is_unifier } \sigma \text{ } AA \implies A \in AA \implies B \in AA \implies A \cdot a \sigma = B \cdot a \sigma$
 ⟨proof⟩

lemma *is_unifier_alt*:
 assumes *finite AA*
 shows $\text{is_unifier } \sigma \text{ } AA \iff (\forall A \in AA. \forall B \in AA. A \cdot a \sigma = B \cdot a \sigma)$
 ⟨proof⟩

lemma *is_unifiers_subst_atm_eqI*:
 assumes *finite AA is_unifiers* $\sigma \text{ } AAA \text{ } AA \in AAA \text{ } A \in AA \text{ } B \in AA$
 shows $A \cdot a \sigma = B \cdot a \sigma$
 ⟨proof⟩

theorem *is_unifiers_comp*:
 $\text{is_unifiers } \sigma \text{ } (\text{set_mset } ' \text{ set } (\text{map2 } \text{add_mset } \text{As } \text{Bs}) \cdot \text{ass } \eta) \iff$
 $\text{is_unifiers } (\eta \odot \sigma) \text{ } (\text{set_mset } ' \text{ set } (\text{map2 } \text{add_mset } \text{As } \text{Bs}))$
 ⟨proof⟩

7.3.21 Most General Unifier

lemma *is_mgu_is_unifiers*: $\text{is_mgu } \sigma \text{ } AAA \implies \text{is_unifiers } \sigma \text{ } AAA$
 ⟨proof⟩

lemma *is_mgu_is_most_general*: $\text{is_mgu } \sigma \text{ } AAA \implies \text{is_unifiers } \tau \text{ } AAA \implies \exists \gamma. \tau = \sigma \odot \gamma$
 ⟨proof⟩

lemma *is_unifiers_is_unifier*: $\text{is_unifiers } \sigma \text{ } AAA \implies AA \in AAA \implies \text{is_unifier } \sigma \text{ } AA$
 ⟨proof⟩

lemma *is_imgu_is_mgu[intro]*: $\text{is_imgu } \sigma \text{ } AAA \implies \text{is_mgu } \sigma \text{ } AAA$
 ⟨proof⟩

lemma *is_imgu_comp_idempotent[simp]*: $\text{is_imgu } \sigma \text{ } AAA \implies \sigma \odot \sigma = \sigma$
 ⟨proof⟩

lemma *is_imgu_subst_atm_idempotent[simp]*: $\text{is_imgu } \sigma \text{ } AAA \implies A \cdot a \sigma \cdot a \sigma = A \cdot a \sigma$
 ⟨proof⟩

lemma *is_imgu_subst_atms_idempotent[simp]*: $\text{is_imgu } \sigma \text{ } AAA \implies AA \cdot \text{as } \sigma \cdot \text{as } \sigma = AA \cdot \text{as } \sigma$
 ⟨proof⟩

lemma *is_imgu_subst_lit_idemptotent[simp]*: $\text{is_imgu } \sigma \text{ } AAA \implies L \cdot l \sigma \cdot l \sigma = L \cdot l \sigma$
 ⟨proof⟩

lemma *is_imgu_subst_cls_idemptotent[simp]*: $\text{is_imgu } \sigma \text{ } AAA \implies C \cdot \sigma \cdot \sigma = C \cdot \sigma$
 ⟨proof⟩

lemma *is_imgu_subst_cls_idemptotent[simp]*: $\text{is_imgu } \sigma \text{ } AAA \implies CC \cdot \text{cs } \sigma \cdot \text{cs } \sigma = CC \cdot \text{cs } \sigma$
 ⟨proof⟩

7.3.22 Generalization and Subsumption

lemma *variants_sym*: $\text{variants } D \text{ } D' \iff \text{variants } D' \text{ } D$
 ⟨proof⟩

lemma *variants_iff_subsumes*: $\text{variants } C \text{ } D \iff \text{subsumes } C \text{ } D \wedge \text{subsumes } D \text{ } C$
 ⟨proof⟩

lemma *strict_subset_subst_strictly_subsumes*: $C \cdot \eta \subset\# D \implies \text{strictly_subsumes } C D$
(proof)

lemma *generalizes_lit_refl[simp]*: *generalizes_lit* $L L$
(proof)

lemma *generalizes_lit_trans*:
generalizes_lit $L1 L2 \implies \text{generalizes_lit } L2 L3 \implies \text{generalizes_lit } L1 L3$
(proof)

lemma *generalizes_refl[simp]*: *generalizes* $C C$
(proof)

lemma *generalizes_trans*: *generalizes* $C D \implies \text{generalizes } D E \implies \text{generalizes } C E$
(proof)

lemma *subsumes_refl*: *subsumes* $C C$
(proof)

lemma *subsumes_trans*: *subsumes* $C D \implies \text{subsumes } D E \implies \text{subsumes } C E$
(proof)

lemma *strictly_generalizes_irrefl*: $\neg \text{strictly_generalizes } C C$
(proof)

lemma *strictly_generalizes_antisym*: *strictly_generalizes* $C D \implies \neg \text{strictly_generalizes } D C$
(proof)

lemma *strictly_generalizes_trans*:
strictly_generalizes $C D \implies \text{strictly_generalizes } D E \implies \text{strictly_generalizes } C E$
(proof)

lemma *strictly_subsumes_irrefl*: $\neg \text{strictly_subsumes } C C$
(proof)

lemma *strictly_subsumes_antisym*: *strictly_subsumes* $C D \implies \neg \text{strictly_subsumes } D C$
(proof)

lemma *strictly_subsumes_trans*:
strictly_subsumes $C D \implies \text{strictly_subsumes } D E \implies \text{strictly_subsumes } C E$
(proof)

lemma *subset_strictly_subsumes*: $C \subset\# D \implies \text{strictly_subsumes } C D$
(proof)

lemma *strictly_generalizes_neq*: *strictly_generalizes* $D' D \implies D' \neq D \cdot \sigma$
(proof)

lemma *strictly_subsumes_neq*: *strictly_subsumes* $D' D \implies D' \neq D \cdot \sigma$
(proof)

lemma *variants_imp_exists_substitution*: *variants* $D D' \implies \exists \sigma. D \cdot \sigma = D'$
(proof)

lemma *strictly_subsumes_variants*:
assumes *strictly_subsumes* $E D$ **and** *variants* $D D'$
shows *strictly_subsumes* $E D'$
(proof)

lemma *neg_strictly_subsumes_variants*:
assumes $\neg \text{strictly_subsumes } E D$ **and** *variants* $D D'$
shows $\neg \text{strictly_subsumes } E D'$
(proof)

end

```
locale substitution_renamings = substitution subst_atm id_subst comp_subst
  for
    subst_atm :: 'a ⇒ 's ⇒ 'a and
    id_subst :: 's and
    comp_subst :: 's ⇒ 's ⇒ 's +
  fixes
    renamings_apart :: 'a clause list ⇒ 's list and
    atm_of_atms :: 'a list ⇒ 'a
  assumes
    renamings_apart_length: length (renamings_apart Cs) = length Cs and
    renamings_apart_renaming: ρ ∈ set (renamings_apart Cs) ⇒ is_renaming ρ and
    renamings_apart_var_disjoint: var_disjoint (Cs ·cl (renamings_apart Cs)) and
    atm_of_atms_subst:
      ⋀ As Bs. atm_of_atms As ·a σ = atm_of_atms Bs ⇔ map (λA. A ·a σ) As = Bs
begin
```

7.3.23 Generalization and Subsumption

lemma *wf_strictly_generalizes*: wfP strictly_generalizes
<proof>

lemma *strictly_subsumes_has_minimum*:
 assumes $CC \neq \{\}$
 shows $\exists C \in CC. \forall D \in CC. \neg \text{strictly_subsumes } D C$
<proof>

lemma *wf_strictly_subsumes*: wfP strictly_subsumes
<proof>

end

7.4 Most General Unifiers

```
locale mgu = substitution_renamings subst_atm id_subst comp_subst renamings_apart atm_of_atms
  for
    subst_atm :: 'a ⇒ 's ⇒ 'a and
    id_subst :: 's and
    comp_subst :: 's ⇒ 's ⇒ 's and
    renamings_apart :: 'a literal multiset list ⇒ 's list and
    atm_of_atms :: 'a list ⇒ 'a+
  fixes
    mgu :: 'a set set ⇒ 's option
  assumes
    mgu_sound: finite AAA ⇒ (∀ AA ∈ AAA. finite AA) ⇒ mgu AAA = Some σ ⇒ is_mgu σ AAA and
    mgu_complete:
      finite AAA ⇒ (∀ AA ∈ AAA. finite AA) ⇒ is_unifiers σ AAA ⇒ ∃ τ. mgu AAA = Some τ
begin
```

lemmas *is_unifiers_mgu* = mgu_sound[unfolded is_mgu_def, THEN conjunct1]

lemmas *is_mgu_most_general* = mgu_sound[unfolded is_mgu_def, THEN conjunct2]

lemma *mgu_unifier*:
 assumes
 aslen: length As = n and
 aaslen: length AAs = n and
 mgu: Some σ = mgu (set_mset ' set (map2 add_mset As AAs)) and
 i_lt: i < n and
 a_in: A ∈ # AAs ! i
 shows $A \cdot a \sigma = As ! i \cdot a \sigma$
<proof>

end

7.5 Idempotent Most General Unifiers

```

locale imgu = mgu subst_atm id_subst comp_subst renamings_apart atm_of_atms mgu
for
  subst_atm :: 'a ⇒ 's ⇒ 'a and
  id_subst :: 's and
  comp_subst :: 's ⇒ 's ⇒ 's and
  renamings_apart :: 'a literal multiset list ⇒ 's list and
  atm_of_atms :: 'a list ⇒ 'a and
  mgu :: 'a set set ⇒ 's option +
assumes
  mgu_is_imgu: finite AAA ⇒ (∀ AA ∈ AAA. finite AA) ⇒ mgu AAA = Some σ ⇒ is_imgu σ AAA

```

end

8 Refutational Inference Systems

```

theory Inference_System
  imports Herbrand_Interpretation
begin

```

This theory gathers results from Section 2.4 (“Refutational Theorem Proving”), 3 (“Standard Resolution”), and 4.2 (“Counterexample-Reducing Inference Systems”) of Bachmair and Ganzinger’s chapter.

8.1 Preliminaries

Inferences have one distinguished main premise, any number of side premises, and a conclusion.

```

datatype 'a inference =
  Infer (side_prem_of: 'a clause multiset) (main_prem_of: 'a clause) (concl_of: 'a clause)

```

```

abbreviation prems_of :: 'a inference ⇒ 'a clause multiset where
  prems_of γ ≡ side_prem_of γ + {#main_prem_of γ#}

```

```

abbreviation concls_of :: 'a inference set ⇒ 'a clause set where
  concls_of Γ ≡ concl_of ' Γ

```

```

definition infer_from :: 'a clause set ⇒ 'a inference ⇒ bool where
  infer_from CC γ ↔ set_mset (prems_of γ) ⊆ CC

```

```

locale inference_system =
  fixes Γ :: 'a inference set
begin

```

```

definition inferences_from :: 'a clause set ⇒ 'a inference set where
  inferences_from CC = {γ. γ ∈ Γ ∧ infer_from CC γ}

```

```

definition inferences_between :: 'a clause set ⇒ 'a clause ⇒ 'a inference set where
  inferences_between CC C = {γ. γ ∈ Γ ∧ infer_from (CC ∪ {C}) γ ∧ C ∈ # prems_of γ}

```

```

lemma inferences_from_mono: CC ⊆ DD ⇒ inferences_from CC ⊆ inferences_from DD
  ⟨proof⟩

```

```

definition saturated :: 'a clause set ⇒ bool where
  saturated N ↔ concls_of (inferences_from N) ⊆ N

```

```

lemma saturatedD:
  assumes
    satur: saturated N and
    inf: Infer CC D E ∈ Γ and

```

$cc_subs_n: set_mset\ CC \subseteq N$ **and**
 $d_in_n: D \in N$
shows $E \in N$
 <proof>

end

Satisfiability preservation is a weaker requirement than soundness.

locale $sat_preserving_inference_system = inference_system +$
assumes $\Gamma_sat_preserving: satisfiable\ N \implies satisfiable\ (N \cup\ concls_of\ (inferences_from\ N))$

locale $sound_inference_system = inference_system +$
assumes $\Gamma_sound: Infer\ CC\ D\ E \in \Gamma \implies I \models_m\ CC \implies I \models\ D \implies I \models\ E$
begin

lemma $\Gamma_sat_preserving:$
assumes $sat_n: satisfiable\ N$
shows $satisfiable\ (N \cup\ concls_of\ (inferences_from\ N))$
 <proof>

sublocale $sat_preserving_inference_system$
 <proof>

end

locale $reductive_inference_system = inference_system\ \Gamma$ **for** $\Gamma :: ('a :: wellorder)\ inference\ set +$
assumes $\Gamma_reductive: \gamma \in \Gamma \implies concls_of\ \gamma < main_prem_of\ \gamma$

8.2 Refutational Completeness

Refutational completeness can be established once and for all for counterexample-reducing inference systems. The material formalized here draws from both the general framework of Section 4.2 and the concrete instances of Section 3.

locale $counterex_reducing_inference_system =$
 $inference_system\ \Gamma$ **for** $\Gamma :: ('a :: wellorder)\ inference\ set +$
fixes $I_of :: 'a\ clause\ set \Rightarrow 'a\ interp$
assumes $\Gamma_counterex_reducing:$
 $\{\#\} \notin N \implies D \in N \implies \neg I_of\ N \models D \implies (\bigwedge C. C \in N \implies \neg I_of\ N \models C \implies D \leq C) \implies$
 $\exists CC\ E. set_mset\ CC \subseteq N \wedge I_of\ N \models_m\ CC \wedge Infer\ CC\ D\ E \in \Gamma \wedge \neg I_of\ N \models E \wedge E < D$
begin

lemma $ex_min_counterex:$
fixes $N :: ('a :: wellorder)\ clause\ set$
assumes $\neg I \models_s\ N$
shows $\exists C \in N. \neg I \models C \wedge (\forall D \in N. D < C \longrightarrow I \models D)$
 <proof>

theorem $saturated_model:$
assumes
 $satur: saturated\ N$ **and**
 $ec_ni_n: \{\#\} \notin N$
shows $I_of\ N \models_s\ N$
 <proof>

Cf. Corollary 3.10:

corollary $saturated_complete: saturated\ N \implies \neg satisfiable\ N \implies \{\#\} \in N$
 <proof>

end

8.3 Compactness

Bachmair and Ganzinger claim that compactness follows from refutational completeness but leave the proof to the readers' imagination. Our proof relies on an inductive definition of saturation in terms of a base set of clauses.

context *inference_system*
begin

inductive-set *saturate* :: 'a clause set \Rightarrow 'a clause set **for** *CC* :: 'a clause set **where**
base: $C \in CC \implies C \in \text{saturate } CC$
| *step*: $\text{Infer } CC' D E \in \Gamma \implies (\bigwedge C'. C' \in \# CC' \implies C' \in \text{saturate } CC) \implies D \in \text{saturate } CC \implies E \in \text{saturate } CC$

lemma *saturate_mono*: $C \in \text{saturate } CC \implies CC \subseteq DD \implies C \in \text{saturate } DD$
<proof>

lemma *saturated_saturate[simp, intro]*: *saturated* (*saturate* *N*)
<proof>

lemma *saturate_finite*: $C \in \text{saturate } CC \implies \exists DD. DD \subseteq CC \wedge \text{finite } DD \wedge C \in \text{saturate } DD$
<proof>

end

context *sound_inference_system*
begin

theorem *saturate_sound*: $C \in \text{saturate } CC \implies I \models_s CC \implies I \models C$
<proof>

end

context *sat_preserving_inference_system*
begin

This result surely holds, but we have yet to prove it. The challenge is: Every time a new clause is introduced, we also get a new interpretation (by the definition of *sat_preserving_inference_system*). But the interpretation we want here is then the one that exists "at the limit". Maybe we can use compactness to prove it.

theorem *saturate_sat_preserving*: *satisfiable* *CC* \implies *satisfiable* (*saturate* *CC*)
<proof>

end

locale *sound_counterex_reducing_inference_system* =
counterex_reducing_inference_system + *sound_inference_system*
begin

Compactness of clausal logic is stated as Theorem 3.12 for the case of unordered ground resolution. The proof below is a generalization to any sound counterexample-reducing inference system. The actual theorem will become available once the locale has been instantiated with a concrete inference system.

theorem *clausal_logic_compact*:
fixes *N* :: ('a :: wellorder) clause set
shows $\neg \text{satisfiable } N \iff (\exists DD \subseteq N. \text{finite } DD \wedge \neg \text{satisfiable } DD)$
<proof>

end

end

9 Candidate Models for Ground Resolution

```

theory Ground_Resolution_Model
  imports Herbrand_Interpretation
begin

```

The proofs of refutational completeness for the two resolution inference systems presented in Section 3 (“Standard Resolution”) of Bachmair and Ganzinger’s chapter share mostly the same candidate model construction. The literal selection capability needed for the second system is ignored by the first one, by taking λ_{\cdot} . $\{\}$ as instantiation for the S parameter.

```

locale selection =
  fixes  $S :: 'a \text{ clause} \Rightarrow 'a \text{ clause}$ 
  assumes
     $S\_selects\_subsetq: S \ C \subseteq\# \ C$  and
     $S\_selects\_neg\_lits: L \in\# \ S \ C \Longrightarrow is\_neg \ L$ 

```

```

locale ground_resolution_with_selection = selection  $S$ 
  for  $S :: ('a :: wellorder) \text{ clause} \Rightarrow 'a \text{ clause}$ 
begin

```

The following commands corresponds to Definition 3.14, which generalizes Definition 3.1. *production* C is denoted ε_C in the chapter; *interp* C is denoted I_C ; *Interp* C is denoted I^C ; and *Interp* N is denoted I_N . The mutually recursive definition from the chapter is massaged to simplify the termination argument. The *production_unfold* lemma below gives the intended characterization.

```

context
  fixes  $N :: 'a \text{ clause set}$ 
begin

```

```

function production ::  $'a \text{ clause} \Rightarrow 'a \text{ interp}$  where
  production  $C =$ 
     $\{A. C \in N \wedge C \neq \{\#\} \wedge Max\_mset \ C = Pos \ A \wedge \neg (\bigcup D \in \{D. D < C\}. \text{production } D) \models C \wedge S \ C = \{\#\}\}$ 
     $\langle proof \rangle$ 
termination  $\langle proof \rangle$ 

```

```

declare production.simps [simp del]

```

```

definition interp ::  $'a \text{ clause} \Rightarrow 'a \text{ interp}$  where
  interp  $C = (\bigcup D \in \{D. D < C\}. \text{production } D)$ 

```

```

lemma production_unfold:
   $\text{production } C = \{A. C \in N \wedge C \neq \{\#\} \wedge Max\_mset \ C = Pos \ A \wedge \neg \text{interp } C \models C \wedge S \ C = \{\#\}\}$ 
   $\langle proof \rangle$ 

```

```

abbreviation productive ::  $'a \text{ clause} \Rightarrow bool$  where
  productive  $C \equiv \text{production } C \neq \{\}$ 

```

```

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

```

```

lemma producesD:  $\text{produces } C \ A \Longrightarrow C \in N \wedge C \neq \{\#\} \wedge Pos \ A = Max\_mset \ C \wedge \neg \text{interp } C \models C \wedge S \ C = \{\#\}$ 
   $\langle proof \rangle$ 

```

```

definition Interp ::  $'a \text{ clause} \Rightarrow 'a \text{ interp}$  where
  Interp  $C = \text{interp } C \cup \text{production } C$ 

```

```

lemma interp_subsetq_Interp[simp]:  $\text{interp } C \subseteq \text{Interp } C$ 
   $\langle proof \rangle$ 

```

```

lemma Interp_as_UNION:  $\text{Interp } C = (\bigcup D \in \{D. D \leq C\}. \text{production } D)$ 
   $\langle proof \rangle$ 

```

```

lemma productive_not_empty:  $\text{productive } C \Longrightarrow C \neq \{\#\}$ 

```

<proof>

lemma *productive_imp_produces_Max_literal*: $productive\ C \implies produces\ C\ (atm_of\ (Max_mset\ C))$
<proof>

lemma *productive_imp_produces_Max_atom*: $productive\ C \implies produces\ C\ (Max\ (atms_of\ C))$
<proof>

lemma *produces_imp_Max_literal*: $produces\ C\ A \implies A = atm_of\ (Max_mset\ C)$
<proof>

lemma *produces_imp_Max_atom*: $produces\ C\ A \implies A = Max\ (atms_of\ C)$
<proof>

lemma *produces_imp_Pos_in_lits*: $produces\ C\ A \implies Pos\ A \in\# C$
<proof>

lemma *productive_in_N*: $productive\ C \implies C \in N$
<proof>

lemma *produces_imp_atms_leq*: $produces\ C\ A \implies B \in atms_of\ C \implies B \leq A$
<proof>

lemma *produces_imp_neg_notin_lits*: $produces\ C\ A \implies \neg Neg\ A \in\# C$
<proof>

lemma *less_eq_imp_interp_subseteq_interp*: $C \leq D \implies interp\ C \subseteq interp\ D$
<proof>

lemma *less_eq_imp_interp_subseteq_Interp*: $C \leq D \implies interp\ C \subseteq Interp\ D$
<proof>

lemma *less_imp_production_subseteq_interp*: $C < D \implies production\ C \subseteq interp\ D$
<proof>

lemma *less_eq_imp_production_subseteq_Interp*: $C \leq D \implies production\ C \subseteq Interp\ D$
<proof>

lemma *less_imp_Interp_subseteq_interp*: $C < D \implies Interp\ C \subseteq interp\ D$
<proof>

lemma *less_eq_imp_Interp_subseteq_Interp*: $C \leq D \implies Interp\ C \subseteq Interp\ D$
<proof>

lemma *not_Interp_to_interp_imp_less*: $A \notin Interp\ C \implies A \in interp\ D \implies C < D$
<proof>

lemma *not_interp_to_interp_imp_less*: $A \notin interp\ C \implies A \in interp\ D \implies C < D$
<proof>

lemma *not_Interp_to_Interp_imp_less*: $A \notin Interp\ C \implies A \in Interp\ D \implies C < D$
<proof>

lemma *not_interp_to_Interp_imp_le*: $A \notin interp\ C \implies A \in Interp\ D \implies C \leq D$
<proof>

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

lemma *interp_subseteq_INTERP*: $interp\ C \subseteq INTERP$
<proof>

lemma *production_subseteq_INTERP*: $production\ C \subseteq INTERP$

<proof>

lemma *Interp_subseteq_INTERP*: $Interp\ C \subseteq INTERP$
<proof>

lemma *produces_imp_in_interp*:
assumes a_in_c : $Neg\ A \in\# C$ **and** d : *produces* $D\ A$
shows $A \in\ interp\ C$
<proof>

lemma *neg_notin_Interp_not_produce*: $Neg\ A \in\# C \implies A \notin Interp\ D \implies C \leq D \implies \neg\ produces\ D''\ A$
<proof>

lemma *in_production_imp_produces*: $A \in\ production\ C \implies produces\ C\ A$
<proof>

lemma *not_produces_imp_notin_production*: $\neg\ produces\ C\ A \implies A \notin production\ C$
<proof>

lemma *not_produces_imp_notin_interp*: $(\bigwedge D. \neg\ produces\ D\ A) \implies A \notin interp\ C$
<proof>

The results below corresponds to Lemma 3.4.

lemma *Interp_imp_general*:
assumes
 c_le_d : $C \leq D$ **and**
 d_lt_d' : $D < D'$ **and**
 c_at_d : $Interp\ D \models C$ **and**
 $subs$: $interp\ D' \subseteq (\bigcup C \in CC. production\ C)$
shows $(\bigcup C \in CC. production\ C) \models C$
<proof>

lemma *Interp_imp_interp*: $C \leq D \implies D < D' \implies Interp\ D \models C \implies interp\ D' \models C$
<proof>

lemma *Interp_imp_Interp*: $C \leq D \implies D \leq D' \implies Interp\ D \models C \implies Interp\ D' \models C$
<proof>

lemma *Interp_imp_INTERP*: $C \leq D \implies Interp\ D \models C \implies INTERP \models C$
<proof>

lemma *interp_imp_general*:
assumes
 c_le_d : $C \leq D$ **and**
 d_le_d' : $D \leq D'$ **and**
 c_at_d : $interp\ D \models C$ **and**
 $subs$: $interp\ D' \subseteq (\bigcup C \in CC. production\ C)$
shows $(\bigcup C \in CC. production\ C) \models C$
<proof>

lemma *interp_imp_interp*: $C \leq D \implies D \leq D' \implies interp\ D \models C \implies interp\ D' \models C$
<proof>

lemma *interp_imp_Interp*: $C \leq D \implies D \leq D' \implies interp\ D \models C \implies Interp\ D' \models C$
<proof>

lemma *interp_imp_INTERP*: $C \leq D \implies interp\ D \models C \implies INTERP \models C$
<proof>

lemma *productive_imp_not_interp*: $productive\ C \implies \neg\ interp\ C \models C$
<proof>

This corresponds to Lemma 3.3:

lemma *productive_imp_Interp*:

assumes *productive C*

shows $\text{Interp } C \models C$

<proof>

lemma *productive_imp_INTERP*: $\text{productive } C \implies \text{INTERP} \models C$

<proof>

This corresponds to Lemma 3.5:

lemma *max_pos_imp_Interp*:

assumes $C \in N$ **and** $C \neq \{\#\}$ **and** $\text{Max_mset } C = \text{Pos } A$ **and** $S \ C = \{\#\}$

shows $\text{Interp } C \models C$

<proof>

The following results correspond to Lemma 3.6:

lemma *max_atm_imp_Interp*:

assumes

c_in_n: $C \in N$ **and**

pos_in: $\text{Pos } A \in \# \ C$ **and**

max_atm: $A = \text{Max } (\text{atms_of } C)$ **and**

s_c_e: $S \ C = \{\#\}$

shows $\text{Interp } C \models C$

<proof>

lemma *not_Interp_imp_general*:

assumes

d'_le_d: $D' \leq D$ **and**

in_n_or_max_gt: $D' \in N \wedge S \ D' = \{\#\} \vee \text{Max } (\text{atms_of } D') < \text{Max } (\text{atms_of } D)$ **and**

d'_at_d: $\neg \text{Interp } D \models D'$ **and**

d_lt_c: $D < C$ **and**

subs: $\text{interp } C \subseteq (\bigcup C \in CC. \text{production } C)$

shows $\neg (\bigcup C \in CC. \text{production } C) \models D'$

<proof>

lemma *not_Interp_imp_not_interp*:

$D' \leq D \implies D' \in N \wedge S \ D' = \{\#\} \vee \text{Max } (\text{atms_of } D') < \text{Max } (\text{atms_of } D) \implies \neg \text{Interp } D \models D' \implies$

$D < C \implies \neg \text{interp } C \models D'$

<proof>

lemma *not_Interp_imp_not_Interp*:

$D' \leq D \implies D' \in N \wedge S \ D' = \{\#\} \vee \text{Max } (\text{atms_of } D') < \text{Max } (\text{atms_of } D) \implies \neg \text{Interp } D \models D' \implies$

$D < C \implies \neg \text{Interp } C \models D'$

<proof>

lemma *not_Interp_imp_not_INTERP*:

$D' \leq D \implies D' \in N \wedge S \ D' = \{\#\} \vee \text{Max } (\text{atms_of } D') < \text{Max } (\text{atms_of } D) \implies \neg \text{Interp } D \models D' \implies$

$\neg \text{INTERP} \models D'$

<proof>

Lemma 3.7 is a problem child. It is stated below but not proved; instead, a counterexample is displayed.

This is not much of a problem, because it is not invoked in the rest of the chapter.

lemma

assumes $D \in N$ **and** $\bigwedge D'. D' < D \implies \text{Interp } D' \models C$

shows $\text{interp } D \models C$

<proof>

lemma

assumes $d: D = \{\#\}$ **and** $n: N = \{D, C\}$ **and** $c: C = \{\#\text{Pos } A\}$

shows $D \in N$ **and** $\bigwedge D'. D' < D \implies \text{Interp } D' \models C$ **and** $\neg \text{interp } D \models C$

<proof>

end

end

end

10 Ground Unordered Resolution Calculus

```
theory Unordered_Ground_Resolution  
  imports Inference_System Ground_Resolution_Model  
begin
```

Unordered ground resolution is one of the two inference systems studied in Section 3 (“Standard Resolution”) of Bachmair and Ganzinger’s chapter.

10.1 Inference Rule

Unordered ground resolution consists of a single rule, called *unord_resolve* below, which is sound and counterexample-reducing.

```
locale ground_resolution_without_selection  
begin
```

```
sublocale ground_resolution_with_selection where  $S = \lambda\_ . \{\#\}$   
  <proof>
```

```
inductive unord_resolve :: 'a clause  $\Rightarrow$  'a clause  $\Rightarrow$  'a clause  $\Rightarrow$  bool where  
  unord_resolve ( $C + \text{replicate\_mset } (\text{Suc } n) (\text{Pos } A)$ ) ( $\text{add\_mset } (\text{Neg } A) D$ ) ( $C + D$ )
```

```
lemma unord_resolve_sound: unord_resolve  $C D E \Longrightarrow I \models C \Longrightarrow I \models D \Longrightarrow I \models E$   
  <proof>
```

The following result corresponds to Theorem 3.8, except that the conclusion is strengthened slightly to make it fit better with the counterexample-reducing inference system framework.

```
theorem unord_resolve_counterex_reducing:
```

```
  assumes
```

```
    ec_ni_n:  $\{\#\} \notin N$  and
```

```
    c_in_n:  $C \in N$  and
```

```
    c_cex:  $\neg \text{INTERP } N \models C$  and
```

```
    c_min:  $\bigwedge D. D \in N \Longrightarrow \neg \text{INTERP } N \models D \Longrightarrow C \leq D$ 
```

```
  obtains  $D E$  where
```

```
     $D \in N$ 
```

```
     $\text{INTERP } N \models D$ 
```

```
    productive  $N D$ 
```

```
    unord_resolve  $D C E$ 
```

```
     $\neg \text{INTERP } N \models E$ 
```

```
     $E < C$ 
```

```
  <proof>
```

10.2 Inference System

Theorem 3.9 and Corollary 3.10 are subsumed in the counterexample-reducing inference system framework, which is instantiated below.

```
definition unord_Γ :: 'a inference set where  
  unord_Γ =  $\{\text{Infer } \{\#\#\} D E \mid C D E. \text{unord\_resolve } C D E\}$ 
```

```
sublocale unord_Γ_sound_counterex_reducing?:
```

```
  sound_counterex_reducing_inference_system unord_Γ INTERP  
  <proof>
```

```
lemmas clausal_logic_compact = unord_Γ_sound_counterex_reducing.clausal_logic_compact
```

end

Theorem 3.12, compactness of clausal logic, has finally been derived for a concrete inference system:

lemmas *clausal_logic_compact* = *ground_resolution_without_selection.clausal_logic_compact*

end

11 Ground Ordered Resolution Calculus with Selection

theory *Ordered_Ground_Resolution*

imports *Inference_System Ground_Resolution_Model*

begin

Ordered ground resolution with selection is the second inference system studied in Section 3 (“Standard Resolution”) of Bachmair and Ganzinger’s chapter.

11.1 Inference Rule

Ordered ground resolution consists of a single rule, called *ord_resolve* below. Like *unord_resolve*, the rule is sound and counterexample-reducing. In addition, it is reductive.

context *ground_resolution_with_selection*

begin

The following inductive definition corresponds to Figure 2.

definition *maximal_wrt* :: 'a \Rightarrow 'a literal multiset \Rightarrow bool **where**
maximal_wrt A DA \longleftrightarrow DA = {#} \vee A = Max (atms_of DA)

definition *strictly_maximal_wrt* :: 'a \Rightarrow 'a literal multiset \Rightarrow bool **where**
strictly_maximal_wrt A CA \longleftrightarrow ($\forall B \in$ atms_of CA. B < A)

inductive *eligible* :: 'a list \Rightarrow 'a clause \Rightarrow bool **where**
eligible: (S DA = negs (mset As)) \vee (S DA = {#} \wedge length As = 1 \wedge *maximal_wrt* (As ! 0) DA) \implies
eligible As DA

lemma (S DA = negs (mset As) \vee S DA = {#} \wedge length As = 1 \wedge *maximal_wrt* (As ! 0) DA) \longleftrightarrow
eligible As DA
 <proof>

inductive

ord_resolve :: 'a clause list \Rightarrow 'a clause \Rightarrow 'a multiset list \Rightarrow 'a list \Rightarrow 'a clause \Rightarrow bool

where

ord_resolve:

length CAs = n \implies

length Cs = n \implies

length AAs = n \implies

length As = n \implies

n \neq 0 \implies

($\forall i < n$. CAs ! i = Cs ! i + poss (AAs ! i)) \implies

($\forall i < n$. AAs ! i \neq {#}) \implies

($\forall i < n$. $\forall A \in \#$ AAs ! i. A = As ! i) \implies

eligible As (D + negs (mset As)) \implies

($\forall i < n$. *strictly_maximal_wrt* (As ! i) (Cs ! i)) \implies

($\forall i < n$. S (CAs ! i) = {#}) \implies

ord_resolve CAs (D + negs (mset As)) AAs As ($\sum \#$ (mset Cs) + D)

lemma *ord_resolve_sound*:

assumes

res_e: *ord_resolve* CAs DA AAs As E **and**

cc_true: I \models_m mset CAs **and**

d_true: I \models DA

shows I \models E

<proof>

lemma *filter_neg_atm_of_S*: $\{\#Neg (atm_of L). L \in \# S C\# \} = S C$
 ⟨proof⟩

This corresponds to Lemma 3.13:

lemma *ord_resolve_reductive*:
assumes *ord_resolve CAs DA AAs As E*
shows $E < DA$
 ⟨proof⟩

This corresponds to Theorem 3.15:

theorem *ord_resolve_counterex_reducing*:
assumes
ec_ni_n: $\{\#\} \notin N$ **and**
d_in_n: $DA \in N$ **and**
d_cex: $\neg INTERP N \models DA$ **and**
d_min: $\bigwedge C. C \in N \implies \neg INTERP N \models C \implies DA \leq C$
obtains *CAs AAs As E* **where**
set CAs $\subseteq N$
 $INTERP N \models_m mset CAs$
 $\bigwedge CA. CA \in set CAs \implies productive N CA$
ord_resolve CAs DA AAs As E
 $\neg INTERP N \models E$
 $E < DA$
 ⟨proof⟩

lemma *ord_resolve_atms_of_concl_subset*:
assumes *ord_resolve CAs DA AAs As E*
shows $atms_of E \subseteq (\bigcup C \in set CAs. atms_of C) \cup atms_of DA$
 ⟨proof⟩

11.2 Inference System

Theorem 3.16 is subsumed in the counterexample-reducing inference system framework, which is instantiated below. Unlike its unordered cousin, ordered resolution is additionally a reductive inference system.

definition *ord_Γ* :: 'a inference set **where**
ord_Γ = $\{Infer (mset CAs) DA E \mid CAs DA AAs As E. ord_resolve CAs DA AAs As E\}$

sublocale *ord_Γ_sound_counterex_reducing?*:
sound_counterex_reducing_inference_system ground_resolution_with_selection.ord_Γ S
ground_resolution_with_selection.INTERP S +
reductive_inference_system ground_resolution_with_selection.ord_Γ S
 ⟨proof⟩

lemmas *clausal_logic_compact* = *ord_Γ_sound_counterex_reducing.clausal_logic_compact*

end

A second proof of Theorem 3.12, compactness of clausal logic:

lemmas *clausal_logic_compact* = *ground_resolution_with_selection.clausal_logic_compact*

end

12 Theorem Proving Processes

theory *Proving_Process*
imports *Unordered_Ground_Resolution Lazy_List_Chain*
begin

This material corresponds to Section 4.1 (“Theorem Proving Processes”) of Bachmair and Ganzinger’s chapter.

The locale assumptions below capture conditions R1 to R3 of Definition 4.1. Rf denotes $\mathcal{R}_{\mathcal{F}}$; Ri denotes $\mathcal{R}_{\mathcal{I}}$.

locale *redundancy_criterion* = *inference_system* +
fixes

$Rf :: 'a \text{ clause set} \Rightarrow 'a \text{ clause set}$ **and**
 $Ri :: 'a \text{ clause set} \Rightarrow 'a \text{ inference set}$

assumes

$Ri_subset_I: Ri\ N \subseteq \Gamma$ **and**
 $Rf_mono: N \subseteq N' \Longrightarrow Rf\ N \subseteq Rf\ N'$ **and**
 $Ri_mono: N \subseteq N' \Longrightarrow Ri\ N \subseteq Ri\ N'$ **and**
 $Rf_indep: N' \subseteq Rf\ N \Longrightarrow Rf\ N \subseteq Rf\ (N - N')$ **and**
 $Ri_indep: N' \subseteq Rf\ N \Longrightarrow Ri\ N \subseteq Ri\ (N - N')$ **and**
 $Rf_sat: \text{satisfiable}\ (N - Rf\ N) \Longrightarrow \text{satisfiable}\ N$

begin

definition *saturated_upto* :: $'a \text{ clause set} \Rightarrow \text{bool}$ **where**

$\text{saturated_upto}\ N \longleftrightarrow \text{inferences_from}\ (N - Rf\ N) \subseteq Ri\ N$

inductive *derive* :: $'a \text{ clause set} \Rightarrow 'a \text{ clause set} \Rightarrow \text{bool}$ (**infix** $\langle \triangleright \rangle$ 50) **where**

$\text{deduction_deletion}: N - M \subseteq \text{concls_of}\ (\text{inferences_from}\ M) \Longrightarrow M - N \subseteq Rf\ N \Longrightarrow M \triangleright N$

lemma *derive_subset*: $M \triangleright N \Longrightarrow N \subseteq M \cup \text{concls_of}\ (\text{inferences_from}\ M)$

<proof>

end

locale *sat_preserving_redundancy_criterion* =

sat_preserving_inference_system $\Gamma :: ('a :: \text{wellorder}) \text{ inference set} + \text{redundancy_criterion}$
begin

lemma *deriv_sat_preserving*:

assumes

$\text{deriv}: \text{chain}\ (\triangleright)\ Ns$ **and**
 $\text{sat_n0}: \text{satisfiable}\ (\text{lhd}\ Ns)$

shows $\text{satisfiable}\ (\text{Sup_llist}\ Ns)$

<proof>

This corresponds to Lemma 4.2:

lemma

assumes $\text{deriv}: \text{chain}\ (\triangleright)\ Ns$

shows

$Rf_Sup_subset_Rf_Liminf: Rf\ (\text{Sup_llist}\ Ns) \subseteq Rf\ (\text{Liminf_llist}\ Ns)$ **and**
 $Ri_Sup_subset_Ri_Liminf: Ri\ (\text{Sup_llist}\ Ns) \subseteq Ri\ (\text{Liminf_llist}\ Ns)$ **and**
 $\text{sat_limit_iff}: \text{satisfiable}\ (\text{Liminf_llist}\ Ns) \longleftrightarrow \text{satisfiable}\ (\text{lhd}\ Ns)$

<proof>

lemma

assumes $\text{chain}\ (\triangleright)\ Ns$

shows

$Rf_limit_Sup: Rf\ (\text{Liminf_llist}\ Ns) = Rf\ (\text{Sup_llist}\ Ns)$ **and**
 $Ri_limit_Sup: Ri\ (\text{Liminf_llist}\ Ns) = Ri\ (\text{Sup_llist}\ Ns)$

<proof>

end

The assumption below corresponds to condition R4 of Definition 4.1.

locale *effective_redundancy_criterion* = *redundancy_criterion* +

assumes $Ri_effective: \gamma \in \Gamma \Longrightarrow \text{concl_of}\ \gamma \in N \cup Rf\ N \Longrightarrow \gamma \in Ri\ N$

begin

definition *fair_cls_seq* :: $'a \text{ clause set llist} \Rightarrow \text{bool}$ **where**

$\text{fair_cls_seq}\ Ns \longleftrightarrow (\text{let}\ N' = \text{Liminf_llist}\ Ns - Rf\ (\text{Liminf_llist}\ Ns) \text{ in}$
 $\text{concls_of}\ (\text{inferences_from}\ N' - Ri\ N') \subseteq \text{Sup_llist}\ Ns \cup Rf\ (\text{Sup_llist}\ Ns))$

end

locale *sat_preserving_effective_redundancy_criterion* =
 sat_preserving_inference_system $\Gamma :: ('a :: wellorder)$ *inference set* +
 effective_redundancy_criterion

begin

sublocale *sat_preserving_redundancy_criterion*
 <proof>

The result below corresponds to Theorem 4.3.

theorem *fair_derive_saturated_upto*:
 assumes
 deriv: *chain* (\triangleright) *Ns* **and**
 fair: *fair_cls_seq* *Ns*
 shows *saturated_upto* (*Liminf_llist* *Ns*)
 <proof>

end

This corresponds to the trivial redundancy criterion defined on page 36 of Section 4.1.

locale *trivial_redundancy_criterion* = *inference_system*
begin

definition *Rf* :: 'a *clause set* \Rightarrow 'a *clause set* **where**
 Rf _ = {}

definition *Ri* :: 'a *clause set* \Rightarrow 'a *inference set* **where**
 Ri *N* = { γ . $\gamma \in \Gamma \wedge \text{concl_of } \gamma \in N$ }

sublocale *effective_redundancy_criterion* Γ *Rf Ri*
 <proof>

lemma *saturated_upto_iff*: *saturated_upto* *N* \longleftrightarrow *concls_of* (*inferences_from* *N*) \subseteq *N*
 <proof>

end

The following lemmas corresponds to the standard extension of a redundancy criterion defined on page 38 of Section 4.1.

lemma *redundancy_criterion_standard_extension*:
 assumes $\Gamma \subseteq \Gamma'$ **and** *redundancy_criterion* Γ *Rf Ri*
 shows *redundancy_criterion* Γ' *Rf* (λN . *Ri* $N \cup (\Gamma' - \Gamma)$)
 <proof>

lemma *redundancy_criterion_standard_extension_saturated_upto_iff*:
 assumes $\Gamma \subseteq \Gamma'$ **and** *redundancy_criterion* Γ *Rf Ri*
 shows *redundancy_criterion.saturated_upto* Γ *Rf Ri* *M* \longleftrightarrow
 redundancy_criterion.saturated_upto Γ' *Rf* (λN . *Ri* $N \cup (\Gamma' - \Gamma)$) *M*
 <proof>

lemma *redundancy_criterion_standard_extension_effective*:
 assumes $\Gamma \subseteq \Gamma'$ **and** *effective_redundancy_criterion* Γ *Rf Ri*
 shows *effective_redundancy_criterion* Γ' *Rf* (λN . *Ri* $N \cup (\Gamma' - \Gamma)$)
 <proof>

lemma *redundancy_criterion_standard_extension_fair_iff*:
 assumes $\Gamma \subseteq \Gamma'$ **and** *effective_redundancy_criterion* Γ *Rf Ri*
 shows *effective_redundancy_criterion.fair_cls_seq* Γ' *Rf* (λN . *Ri* $N \cup (\Gamma' - \Gamma)$) *Ns* \longleftrightarrow
 effective_redundancy_criterion.fair_cls_seq Γ *Rf Ri* *Ns*
 <proof>

theorem *redundancy_criterion_standard_extension_fair_derive_saturated_upto*:
assumes
subs: $\Gamma \subseteq \Gamma'$ **and**
red: *redundancy_criterion* Γ *Rf Ri* **and**
red': *sat_preserving_effective_redundancy_criterion* Γ' *Rf* $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))$ **and**
deriv: *chain* (*redundancy_criterion.derive* Γ' *Rf*) *Ns* **and**
fair: *effective_redundancy_criterion.fair_cls_seq* Γ' *Rf* $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))$ *Ns*
shows *redundancy_criterion.saturated_upto* Γ *Rf Ri* (*Liminf_llist* *Ns*)
 \langle *proof* \rangle

end

13 The Standard Redundancy Criterion

theory *Standard_Redundancy*
imports *Proving_Process*
begin

This material is based on Section 4.2.2 (“The Standard Redundancy Criterion”) of Bachmair and Ganzinger’s chapter.

locale *standard_redundancy_criterion* =
inference_system Γ **for** $\Gamma :: ('a :: wellorder)$ *inference set*
begin

definition *redundant_infer* :: *'a clause set* \Rightarrow *'a inference* \Rightarrow *bool* **where**
redundant_infer $N\ \gamma \longleftrightarrow$
 $(\exists DD. \text{set_mset } DD \subseteq N \wedge (\forall I. I \models DD + \text{side_prems_of } \gamma \longrightarrow I \models \text{concl_of } \gamma)$
 $\wedge (\forall D. D \in \# DD \longrightarrow D < \text{main_prem_of } \gamma))$

definition *Rf* :: *'a clause set* \Rightarrow *'a clause set* **where**
 $Rf\ N = \{C. \exists DD. \text{set_mset } DD \subseteq N \wedge (\forall I. I \models DD \longrightarrow I \models C) \wedge (\forall D. D \in \# DD \longrightarrow D < C)\}$

definition *Ri* :: *'a clause set* \Rightarrow *'a inference set* **where**
 $Ri\ N = \{\gamma \in \Gamma. \text{redundant_infer } N\ \gamma\}$

lemma *tautology_Rf*:
assumes *Pos* $A \in \# C$
assumes *Neg* $A \in \# C$
shows $C \in Rf\ N$
 \langle *proof* \rangle

lemma *tautology_redundant_infer*:
assumes
pos: *Pos* $A \in \# \text{concl_of } \iota$ **and**
neg: *Neg* $A \in \# \text{concl_of } \iota$
shows *redundant_infer* $N\ \iota$
 \langle *proof* \rangle

lemma *contradiction_Rf*: $\{\#\} \in N \Longrightarrow Rf\ N = UNIV - \{\#\}$
 \langle *proof* \rangle

The following results correspond to Lemma 4.5. The lemma *wlog_non_Rf* generalizes the core of the argument.

lemma *Rf_mono*: $N \subseteq N' \Longrightarrow Rf\ N \subseteq Rf\ N'$
 \langle *proof* \rangle

lemma *wlog_non_Rf*:
assumes *ex*: $\exists DD. \text{set_mset } DD \subseteq N \wedge (\forall I. I \models DD + CC \longrightarrow I \models E) \wedge (\forall D'. D' \in \# DD \longrightarrow D' < D)$
shows $\exists DD. \text{set_mset } DD \subseteq N - Rf\ N \wedge (\forall I. I \models DD + CC \longrightarrow I \models E) \wedge (\forall D'. D' \in \# DD \longrightarrow D' < D)$
 \langle *proof* \rangle

lemma *Rf_imp_ex_non_Rf*:

assumes $C \in Rf\ N$
shows $\exists CC. set_mset\ CC \subseteq N - Rf\ N \wedge (\forall I. I \models_m CC \longrightarrow I \models C) \wedge (\forall C'. C' \in\# CC \longrightarrow C' < C)$
 $\langle proof \rangle$

lemma $Rf_subs_Rf_diff_Rf: Rf\ N \subseteq Rf\ (N - Rf\ N)$
 $\langle proof \rangle$

lemma $Rf_eq_Rf_diff_Rf: Rf\ N = Rf\ (N - Rf\ N)$
 $\langle proof \rangle$

The following results correspond to Lemma 4.6.

lemma $Ri_mono: N \subseteq N' \Longrightarrow Ri\ N \subseteq Ri\ N'$
 $\langle proof \rangle$

lemma $Ri_subs_Ri_diff_Rf: Ri\ N \subseteq Ri\ (N - Rf\ N)$
 $\langle proof \rangle$

lemma $Ri_eq_Ri_diff_Rf: Ri\ N = Ri\ (N - Rf\ N)$
 $\langle proof \rangle$

lemma $Ri_subset_Gamma: Ri\ N \subseteq \Gamma$
 $\langle proof \rangle$

lemma $Rf_indep: N' \subseteq Rf\ N \Longrightarrow Rf\ N \subseteq Rf\ (N - N')$
 $\langle proof \rangle$

lemma $Ri_indep: N' \subseteq Rf\ N \Longrightarrow Ri\ N \subseteq Ri\ (N - N')$
 $\langle proof \rangle$

lemma $Rf_model:$
assumes $I \models_s N - Rf\ N$
shows $I \models_s N$
 $\langle proof \rangle$

lemma $Rf_sat: satisfiable\ (N - Rf\ N) \Longrightarrow satisfiable\ N$
 $\langle proof \rangle$

The following corresponds to Theorem 4.7:

sublocale $redundancy_criterion\ \Gamma\ Rf\ Ri$
 $\langle proof \rangle$

end

locale $standard_redundancy_criterion_reductive =$
 $standard_redundancy_criterion + reductive_inference_system$
begin

The following corresponds to Theorem 4.8:

lemma $Ri_effective:$
assumes
 $in_gamma: \gamma \in \Gamma$ **and**
 $concl_of_in_n_un_rf_n: concl_of\ \gamma \in N \cup Rf\ N$
shows $\gamma \in Ri\ N$
 $\langle proof \rangle$

sublocale $effective_redundancy_criterion\ \Gamma\ Rf\ Ri$
 $\langle proof \rangle$

lemma $contradiction_Rf: \{\#\} \in N \Longrightarrow Ri\ N = \Gamma$
 $\langle proof \rangle$

end

```

locale standard_redundancy_criterion_counterex_reducing =
  standard_redundancy_criterion + counterex_reducing_inference_system
begin

```

The following result corresponds to Theorem 4.9.

```

lemma saturated_upto_complete_if:

```

```

  assumes
    satur: saturated_upto N and
    unsat:  $\neg$  satisfiable N
  shows  $\{\#\} \in N$ 
  <proof>

```

```

theorem saturated_upto_complete:

```

```

  assumes saturated_upto N
  shows  $\neg$  satisfiable N  $\longleftrightarrow$   $\{\#\} \in N$ 
  <proof>

```

```

end

```

```

end

```

14 First-Order Ordered Resolution Calculus with Selection

```

theory FO_Ordered_Resolution

```

```

  imports Abstract_Substitution Ordered_Ground_Resolution Standard_Redundancy
begin

```

This material is based on Section 4.3 (“A Simple Resolution Prover for First-Order Clauses”) of Bachmair and Ganzinger’s chapter. Specifically, it formalizes the ordered resolution calculus for first-order standard clauses presented in Figure 4 and its related lemmas and theorems, including soundness and Lemma 4.12 (the lifting lemma).

The following corresponds to pages 41–42 of Section 4.3, until Figure 5 and its explanation.

```

locale FO_resolution = mgu subst_atm id_subst comp_subst renamings_apart atm_of_atms mgu
for

```

```

  subst_atm :: 'a :: wellorder  $\Rightarrow$  's  $\Rightarrow$  'a and
  id_subst :: 's and
  comp_subst :: 's  $\Rightarrow$  's  $\Rightarrow$  's and
  renamings_apart :: 'a literal multiset list  $\Rightarrow$  's list and
  atm_of_atms :: 'a list  $\Rightarrow$  'a and
  mgu :: 'a set set  $\Rightarrow$  's option +

```

```

fixes

```

```

  less_atm :: 'a  $\Rightarrow$  'a  $\Rightarrow$  bool

```

```

assumes

```

```

  less_atm_stable: less_atm A B  $\Longrightarrow$  less_atm (A  $\cdot$  a  $\sigma$ ) (B  $\cdot$  a  $\sigma$ ) and
  less_atm_ground: is_ground_atm A  $\Longrightarrow$  is_ground_atm B  $\Longrightarrow$  less_atm A B  $\Longrightarrow$  A < B

```

```

begin

```

14.1 Library

```

lemma Bex_cartesian_product:  $(\exists xy \in A \times B. P\ xy) \equiv (\exists x \in A. \exists y \in B. P\ (x, y))$ 
  <proof>

```

```

lemma eql_map_neg_lit_eql_atm:

```

```

  assumes map  $(\lambda L. L \cdot l\ \eta)$  (map Neg As') = map Neg As
  shows As'  $\cdot$  al  $\eta$  = As
  <proof>

```

```

lemma instance_list:

```

```

  assumes negs (mset As) = SDA'  $\cdot$   $\eta$ 
  shows  $\exists$  As'. negs (mset As') = SDA'  $\wedge$  As'  $\cdot$  al  $\eta$  = As
  <proof>

```

lemma *map2_add_mset_map*:

assumes $\text{length } AAs' = n$ **and** $\text{length } As' = n$

shows $\text{map2 } \text{add_mset } (As' \cdot \text{al } \eta) (AAs' \cdot \text{aml } \eta) = \text{map2 } \text{add_mset } As' AAs' \cdot \text{aml } \eta$
(*proof*)

context

fixes $S :: 'a \text{ clause} \Rightarrow 'a \text{ clause}$

begin

14.2 Calculus

The following corresponds to Figure 4.

definition *maximal_wrt* :: $'a \Rightarrow 'a \text{ literal multiset} \Rightarrow \text{bool}$ **where**

$\text{maximal_wrt } A C \longleftrightarrow (\forall B \in \text{atms_of } C. \neg \text{less_atm } A B)$

definition *strictly_maximal_wrt* :: $'a \Rightarrow 'a \text{ literal multiset} \Rightarrow \text{bool}$ **where**

$\text{strictly_maximal_wrt } A C \equiv \forall B \in \text{atms_of } C. A \neq B \wedge \neg \text{less_atm } A B$

lemma *strictly_maximal_wrt_maximal_wrt*: $\text{strictly_maximal_wrt } A C \Longrightarrow \text{maximal_wrt } A C$

(*proof*)

lemma *maximal_wrt_subst*: $\text{maximal_wrt } (A \cdot a \sigma) (C \cdot \sigma) \Longrightarrow \text{maximal_wrt } A C$

(*proof*)

lemma *strictly_maximal_wrt_subst*:

$\text{strictly_maximal_wrt } (A \cdot a \sigma) (C \cdot \sigma) \Longrightarrow \text{strictly_maximal_wrt } A C$

(*proof*)

inductive *eligible* :: $'s \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ clause} \Rightarrow \text{bool}$ **where**

eligible:

$S DA = \text{negs } (\text{mset } As) \vee S DA = \{\#\} \wedge \text{length } As = 1 \wedge \text{maximal_wrt } (As ! 0 \cdot a \sigma) (DA \cdot \sigma) \Longrightarrow$
eligible $\sigma As DA$

inductive

ord_resolve

:: $'a \text{ clause list} \Rightarrow 'a \text{ clause} \Rightarrow 'a \text{ multiset list} \Rightarrow 'a \text{ list} \Rightarrow 's \Rightarrow 'a \text{ clause} \Rightarrow \text{bool}$

where

ord_resolve:

$\text{length } CAs = n \Longrightarrow$

$\text{length } Cs = n \Longrightarrow$

$\text{length } AAs = n \Longrightarrow$

$\text{length } As = n \Longrightarrow$

$n \neq 0 \Longrightarrow$

$(\forall i < n. CAs ! i = Cs ! i + \text{poss } (AAs ! i)) \Longrightarrow$

$(\forall i < n. AAs ! i \neq \{\#\}) \Longrightarrow$

$\text{Some } \sigma = \text{mgu } (\text{set_mset } ' \text{set } (\text{map2 } \text{add_mset } As AAs)) \Longrightarrow$

eligible $\sigma As (D + \text{negs } (\text{mset } As)) \Longrightarrow$

$(\forall i < n. \text{strictly_maximal_wrt } (As ! i \cdot a \sigma) (Cs ! i \cdot \sigma)) \Longrightarrow$

$(\forall i < n. S (CAs ! i) = \{\#\}) \Longrightarrow$

ord_resolve $CAs (D + \text{negs } (\text{mset } As)) AAs As \sigma ((\sum \# (\text{mset } Cs) + D) \cdot \sigma)$

inductive

ord_resolve_rename

:: $'a \text{ clause list} \Rightarrow 'a \text{ clause} \Rightarrow 'a \text{ multiset list} \Rightarrow 'a \text{ list} \Rightarrow 's \Rightarrow 'a \text{ clause} \Rightarrow \text{bool}$

where

ord_resolve_rename:

$\text{length } CAs = n \Longrightarrow$

$\text{length } AAs = n \Longrightarrow$

$\text{length } As = n \Longrightarrow$

$(\forall i < n. \text{poss } (AAs ! i) \subseteq \# CAs ! i) \Longrightarrow$

$\text{negs } (\text{mset } As) \subseteq \# DA \Longrightarrow$

$\varrho = \text{hd } (\text{renamings_apart } (DA \# CAs)) \Longrightarrow$

$\varrho s = \text{tl } (\text{renamings_apart } (DA \# CAs)) \Longrightarrow$

$$\text{ord_resolve } (CAs \cdot\!\cdot\! cl \ \varrho s) (DA \cdot \varrho) (AAs \cdot\!\cdot\! aml \ \varrho s) (As \cdot al \ \varrho) \sigma E \implies \\ \text{ord_resolve_rename } CAs DA AAs As \sigma E$$

lemma *ord_resolve_empty_main_prem*: $\neg \text{ord_resolve } Cs \{\#\} AAs As \sigma E$
 ⟨proof⟩

lemma *ord_resolve_rename_empty_main_prem*: $\neg \text{ord_resolve_rename } Cs \{\#\} AAs As \sigma E$
 ⟨proof⟩

14.3 Soundness

Soundness is not discussed in the chapter, but it is an important property.

lemma *ord_resolve_ground_inst_sound*:
assumes
res_e: $\text{ord_resolve } CAs DA AAs As \sigma E$ **and**
cc_inst_true: $I \models_m \text{mset } CAs \cdot cm \ \sigma \cdot cm \ \eta$ **and**
d_inst_true: $I \models DA \cdot \sigma \cdot \eta$ **and**
ground_subst_η: $\text{is_ground_subst } \eta$
shows $I \models E \cdot \eta$
 ⟨proof⟩

The previous lemma is not only used to prove soundness, but also the following lemma which is used to prove Lemma 4.10.

lemma *ord_resolve_rename_ground_inst_sound*:
assumes
ord_resolve_rename $CAs DA AAs As \sigma E$ **and**
 $\varrho s = tl (\text{renamings_apart } (DA \# CAs))$ **and**
 $\varrho = hd (\text{renamings_apart } (DA \# CAs))$ **and**
 $I \models_m (\text{mset } (CAs \cdot\!\cdot\! cl \ \varrho s)) \cdot cm \ \sigma \cdot cm \ \eta$ **and**
 $I \models DA \cdot \varrho \cdot \sigma \cdot \eta$ **and**
is_ground_subst η
shows $I \models E \cdot \eta$
 ⟨proof⟩

Here follows the soundness theorem for the resolution rule.

theorem *ord_resolve_sound*:
assumes
res_e: $\text{ord_resolve } CAs DA AAs As \sigma E$ **and**
cc_d_true: $\bigwedge \sigma. \text{is_ground_subst } \sigma \implies I \models_m (\text{mset } CAs + \{\#DA\# \}) \cdot cm \ \sigma$ **and**
ground_subst_η: $\text{is_ground_subst } \eta$
shows $I \models E \cdot \eta$
 ⟨proof⟩

lemma *subst_sound*:
assumes
 $\bigwedge \sigma. \text{is_ground_subst } \sigma \implies I \models C \cdot \sigma$ **and**
is_ground_subst η
shows $I \models C \cdot \varrho \cdot \eta$
 ⟨proof⟩

lemma *subst_sound_scl*:
assumes
len: $\text{length } P = \text{length } CAs$ **and**
true_cas: $\bigwedge \sigma. \text{is_ground_subst } \sigma \implies I \models_m \text{mset } CAs \cdot cm \ \sigma$ **and**
ground_subst_η: $\text{is_ground_subst } \eta$
shows $I \models_m \text{mset } (CAs \cdot\!\cdot\! cl \ P) \cdot cm \ \eta$
 ⟨proof⟩

Here follows the soundness theorem for the resolution rule with renaming.

lemma *ord_resolve_rename_sound*:
assumes
res_e: $\text{ord_resolve_rename } CAs DA AAs As \sigma E$ **and**

$cc_d_true: \bigwedge \sigma. is_ground_subst \sigma \implies I \models m ((mset\ CAs) + \{\#DA\#}) \cdot cm \sigma$ **and**
 $ground_subst_eta: is_ground_subst \eta$
shows $I \models E \cdot \eta$
 <proof>

14.4 Other Basic Properties

lemma *ord_resolve_unique:*

assumes
 $ord_resolve\ CAs\ DA\ AAs\ As\ \sigma\ E$ **and**
 $ord_resolve\ CAs\ DA\ AAs\ As\ \sigma'\ E'$
shows $\sigma = \sigma' \wedge E = E'$
 <proof>

lemma *ord_resolve_rename_unique:*

assumes
 $ord_resolve_rename\ CAs\ DA\ AAs\ As\ \sigma\ E$ **and**
 $ord_resolve_rename\ CAs\ DA\ AAs\ As\ \sigma'\ E'$
shows $\sigma = \sigma' \wedge E = E'$
 <proof>

lemma *ord_resolve_max_side_prem:* $ord_resolve\ CAs\ DA\ AAs\ As\ \sigma\ E \implies length\ CAs \leq size\ DA$
 <proof>

lemma *ord_resolve_rename_max_side_prem:*

$ord_resolve_rename\ CAs\ DA\ AAs\ As\ \sigma\ E \implies length\ CAs \leq size\ DA$
 <proof>

14.5 Inference System

definition *ord_FO_Γ* :: 'a inference set **where**

$ord_FO_Γ = \{Infer\ (mset\ CAs)\ DA\ E \mid CAs\ DA\ AAs\ As\ \sigma\ E.\ ord_resolve_rename\ CAs\ DA\ AAs\ As\ \sigma\ E\}$

interpretation *ord_FO_resolution:* inference_system *ord_FO_Γ* <proof>

lemma *finite_ord_FO_resolution_inferences_between:*

assumes *fin_cc:* finite *CC*
shows finite (*ord_FO_resolution.inferences_between* *CC* *C*)
 <proof>

lemma *ord_FO_resolution_inferences_between_empty_empty:*

$ord_FO_resolution.inferences_between\ \{\}\ \{\#\} = \{\}$
 <proof>

14.6 Lifting

The following corresponds to the passage between Lemmas 4.11 and 4.12.

context

fixes *M* :: 'a clause set
assumes *select:* selection *S*
begin

interpretation *selection*

<proof>

definition *S_M* :: 'a literal multiset \Rightarrow 'a literal multiset **where**

$S_M\ C =$
 (if $C \in grounding_of_class\ M$ then
 (SOME C' . $\exists D\ \sigma. D \in M \wedge C = D \cdot \sigma \wedge C' = S\ D \cdot \sigma \wedge is_ground_subst\ \sigma$)
 else
 $S\ C$)

lemma *S_M_grounding_of_class:*

assumes $C \in \text{grounding_of_cls } M$
obtains $D \sigma$ **where**
 $D \in M \wedge C = D \cdot \sigma \wedge S_M C = S D \cdot \sigma \wedge \text{is_ground_subst } \sigma$
⟨proof⟩

lemma $S_M \text{ not_grounding_of_cls}$: $C \notin \text{grounding_of_cls } M \implies S_M C = S C$
⟨proof⟩

lemma $S_M \text{ selects_subsetq}$: $S_M C \subseteq\# C$
⟨proof⟩

lemma $S_M \text{ selects_neg_lits}$: $L \in\# S_M C \implies \text{is_neg } L$
⟨proof⟩

end

end

The following corresponds to Lemma 4.12:

lemma $\text{ground_resolvent_subset}$:
assumes
 $\text{gr_cas}: \text{is_ground_cls_list } CAs$ **and**
 $\text{gr_da}: \text{is_ground_cls } DA$ **and**
 $\text{res_e}: \text{ord_resolve } S CAs DA AAs As \sigma E$
shows $E \subseteq\# \sum\# (\text{mset } CAs) + DA$
⟨proof⟩

lemma $\text{ord_resolve_obtain_clauses}$:
assumes
 $\text{res_e}: \text{ord_resolve } (S_M S M) CAs DA AAs As \sigma E$ **and**
 $\text{select}: \text{selection } S$ **and**
 $\text{grounding}: \{DA\} \cup \text{set } CAs \subseteq \text{grounding_of_cls } M$ **and**
 $n: \text{length } CAs = n$ **and**
 $d: DA = D + \text{negs } (\text{mset } As)$ **and**
 $c: (\forall i < n. CAs ! i = Cs ! i + \text{poss } (AAs ! i)) \text{length } Cs = n \text{length } AAs = n$
obtains $DA0 \eta0 CAs0 \eta s0 As0 AAs0 D0 Cs0$ **where**
 $\text{length } CAs0 = n$
 $\text{length } \eta s0 = n$
 $DA0 \in M$
 $DA0 \cdot \eta0 = DA$
 $S DA0 \cdot \eta0 = S_M S M DA$
 $\forall CA0 \in \text{set } CAs0. CA0 \in M$
 $CAs0 \cdot\text{cl } \eta s0 = CAs$
 $\text{map } S CAs0 \cdot\text{cl } \eta s0 = \text{map } (S_M S M) CAs$
 $\text{is_ground_subst } \eta0$
 $\text{is_ground_subst_list } \eta s0$
 $As0 \cdot\text{al } \eta0 = As$
 $AAs0 \cdot\text{aml } \eta s0 = AAs$
 $\text{length } As0 = n$
 $D0 \cdot \eta0 = D$
 $DA0 = D0 + (\text{negs } (\text{mset } As0))$
 $S_M S M (D + \text{negs } (\text{mset } As)) \neq \{\#\} \implies \text{negs } (\text{mset } As0) = S DA0$
 $\text{length } Cs0 = n$
 $Cs0 \cdot\text{cl } \eta s0 = Cs$
 $\forall i < n. CAs0 ! i = Cs0 ! i + \text{poss } (AAs0 ! i)$
 $\text{length } AAs0 = n$
⟨proof⟩

lemma $\text{ord_resolve_rename_lifting}$:
assumes
 $\text{sel_stable}: \bigwedge \rho C. \text{is_renaming } \rho \implies S (C \cdot \rho) = S C \cdot \rho$ **and**
 $\text{res_e}: \text{ord_resolve } (S_M S M) CAs DA AAs As \sigma E$ **and**
 $\text{select}: \text{selection } S$ **and**

```

grounding: {DA} ∪ set CAs ⊆ grounding_of_cls M
obtains ηs η η2 CAs0 DA0 AAs0 As0 E0 τ where
  is_ground_subst η
  is_ground_subst_list ηs
  is_ground_subst η2
  ord_resolve_rename S CAs0 DA0 AAs0 As0 τ E0
  CAs0 ·cl ηs = CAs DA0 · η = DA E0 · η2 = E
  {DA0} ∪ set CAs0 ⊆ M
  length CAs0 = length CAs
  length ηs = length CAs
⟨proof⟩

```

```

lemma ground_ord_resolve_ground:
assumes
  select: selection S and
  CAs_p: ground_resolution_with_selection.ord_resolve S CAs DA AAs As E and
  ground_cas: is_ground_cls_list CAs and
  ground_da: is_ground_cls DA
shows is_ground_cls E
⟨proof⟩

```

```

lemma ground_ord_resolve_imp_ord_resolve:
assumes
  ground_da: ⟨is_ground_cls DA⟩ and
  ground_cas: ⟨is_ground_cls_list CAs⟩ and
  gr: ground_resolution_with_selection S_G and
  gr_res: ⟨ground_resolution_with_selection.ord_resolve S_G CAs DA AAs As E⟩
shows ⟨∃ σ. ord_resolve S_G CAs DA AAs As σ E⟩
⟨proof⟩

```

end

end

15 An Ordered Resolution Prover for First-Order Clauses

```

theory FO_Ordered_Resolution_Prover
imports FO_Ordered_Resolution
begin

```

This material is based on Section 4.3 (“A Simple Resolution Prover for First-Order Clauses”) of Bachmair and Ganzinger’s chapter. Specifically, it formalizes the RP prover defined in Figure 5 and its related lemmas and theorems, including Lemmas 4.10 and 4.11 and Theorem 4.13 (completeness).

```

definition is_least :: (nat ⇒ bool) ⇒ nat ⇒ bool where
  is_least P n ⟷ P n ∧ (∀ n' < n. ¬ P n')

```

```

lemma least_exists: P n ⟹ ∃ n. is_least P n
⟨proof⟩

```

The following corresponds to page 42 and 43 of Section 4.3, from the explanation of RP to Lemma 4.10.

```

type-synonym 'a state = 'a clause set × 'a clause set × 'a clause set

```

```

locale FO_resolution_prover =
  FO_resolution subst_atm id_subst comp_subst renamings_apart atm_of_atms mgu less_atm +
  selection S
for
  S :: ('a :: wellorder) clause ⇒ 'a clause and
  subst_atm :: 'a ⇒ 's ⇒ 'a and
  id_subst :: 's and
  comp_subst :: 's ⇒ 's ⇒ 's and
  renamings_apart :: 'a clause list ⇒ 's list and
  atm_of_atms :: 'a list ⇒ 'a and

```

```

  mgu :: 'a set set ⇒ 's option and
  less_atm :: 'a ⇒ 'a ⇒ bool +
assumes
  sel_stable:  $\bigwedge \varrho C. \text{is\_renaming } \varrho \implies S (C \cdot \varrho) = S C \cdot \varrho$ 
begin

```

```

fun N_of_state :: 'a state ⇒ 'a clause set where
  N_of_state (N, P, Q) = N

```

```

fun P_of_state :: 'a state ⇒ 'a clause set where
  P_of_state (N, P, Q) = P

```

O denotes relation composition in Isabelle, so the formalization uses Q instead.

```

fun Q_of_state :: 'a state ⇒ 'a clause set where
  Q_of_state (N, P, Q) = Q

```

```

abbreviation cls_of_state :: 'a state ⇒ 'a clause set where
  cls_of_state St  $\equiv$  N_of_state St  $\cup$  P_of_state St  $\cup$  Q_of_state St

```

```

abbreviation grounding_of_state :: 'a state ⇒ 'a clause set where
  grounding_of_state St  $\equiv$  grounding_of_cls (cls_of_state St)

```

```

interpretation ord_FO_resolution: inference_system ord_FO  $\Gamma$  S <proof>

```

The following inductive predicate formalizes the resolution prover in Figure 5.

```

inductive RP :: 'a state ⇒ 'a state ⇒ bool (infix  $\rightsquigarrow$  50) where
  tautology_deletion: Neg A  $\in$  # C  $\implies$  Pos A  $\in$  # C  $\implies$  (N  $\cup$  {C}, P, Q)  $\rightsquigarrow$  (N, P, Q)
| forward_subsumption: D  $\in$  P  $\cup$  Q  $\implies$  subsumes D C  $\implies$  (N  $\cup$  {C}, P, Q)  $\rightsquigarrow$  (N, P, Q)
| backward_subsumption_P: D  $\in$  N  $\implies$  strictly_subsumes D C  $\implies$  (N, P  $\cup$  {C}, Q)  $\rightsquigarrow$  (N, P, Q)
| backward_subsumption_Q: D  $\in$  N  $\implies$  strictly_subsumes D C  $\implies$  (N, P, Q  $\cup$  {C})  $\rightsquigarrow$  (N, P, Q)
| forward_reduction: D + {#L#}  $\in$  P  $\cup$  Q  $\implies$  - L = L'  $\cdot$  l  $\sigma \implies$  D  $\cdot$   $\sigma \subseteq$  # C  $\implies$ 
  (N  $\cup$  {C + {#L#}}, P, Q)  $\rightsquigarrow$  (N  $\cup$  {C}, P, Q)
| backward_reduction_P: D + {#L#}  $\in$  N  $\implies$  - L = L'  $\cdot$  l  $\sigma \implies$  D  $\cdot$   $\sigma \subseteq$  # C  $\implies$ 
  (N, P  $\cup$  {C + {#L#}}, Q)  $\rightsquigarrow$  (N, P  $\cup$  {C}, Q)
| backward_reduction_Q: D + {#L#}  $\in$  N  $\implies$  - L = L'  $\cdot$  l  $\sigma \implies$  D  $\cdot$   $\sigma \subseteq$  # C  $\implies$ 
  (N, P, Q  $\cup$  {C + {#L#}})  $\rightsquigarrow$  (N, P  $\cup$  {C}, Q)
| clause_processing: (N  $\cup$  {C}, P, Q)  $\rightsquigarrow$  (N, P  $\cup$  {C}, Q)
| inference_computation: N = concls_of (ord_FO_resolution.inferences_between Q C)  $\implies$ 
  ({}, P  $\cup$  {C}, Q)  $\rightsquigarrow$  (N, P, Q  $\cup$  {C})

```

```

lemma final_RP:  $\neg$  ({} , {} , Q)  $\rightsquigarrow$  St
  <proof>

```

```

definition Sup_state :: 'a state llist ⇒ 'a state where
  Sup_state Sts =
  (Sup_llist (lmap N_of_state Sts), Sup_llist (lmap P_of_state Sts),
   Sup_llist (lmap Q_of_state Sts))

```

```

definition Liminf_state :: 'a state llist ⇒ 'a state where
  Liminf_state Sts =
  (Liminf_llist (lmap N_of_state Sts), Liminf_llist (lmap P_of_state Sts),
   Liminf_llist (lmap Q_of_state Sts))

```

context

```

fixes Sts Sts' :: 'a state llist
assumes Sts: lfinite Sts lfinite Sts'  $\neg$  lnull Sts  $\neg$  lnull Sts' llast Sts' = llast Sts
begin

```

lemma

```

  N_of_Liminf_state_fin: N_of_state (Liminf_state Sts') = N_of_state (Liminf_state Sts) and
  P_of_Liminf_state_fin: P_of_state (Liminf_state Sts') = P_of_state (Liminf_state Sts) and
  Q_of_Liminf_state_fin: Q_of_state (Liminf_state Sts') = Q_of_state (Liminf_state Sts)
  <proof>

```

lemma *Liminf_state_fin*: $\text{Liminf_state } Sts' = \text{Liminf_state } Sts$
 ⟨proof⟩

end

context

fixes *Sts Sts'* :: 'a state llist
assumes *Sts*: $\neg \text{lfinite } Sts \text{ emb } Sts \ Sts'$

begin

lemma

N_of_Liminf_state_inf: $N_of_state (\text{Liminf_state } Sts') \subseteq N_of_state (\text{Liminf_state } Sts)$ **and**
P_of_Liminf_state_inf: $P_of_state (\text{Liminf_state } Sts') \subseteq P_of_state (\text{Liminf_state } Sts)$ **and**
Q_of_Liminf_state_inf: $Q_of_state (\text{Liminf_state } Sts') \subseteq Q_of_state (\text{Liminf_state } Sts)$
 ⟨proof⟩

lemma *cls_of_Liminf_state_inf*:

cls_of_state ($\text{Liminf_state } Sts'$) \subseteq *cls_of_state* ($\text{Liminf_state } Sts$)
 ⟨proof⟩

end

definition *fair_state_seq* :: 'a state llist \Rightarrow bool **where**

fair_state_seq *Sts* $\longleftrightarrow N_of_state (\text{Liminf_state } Sts) = \{\} \wedge P_of_state (\text{Liminf_state } Sts) = \{\}$

The following formalizes Lemma 4.10.

context

fixes *Sts* :: 'a state llist

begin

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

S_Q = *S_M* *S* (*Q_of_state* ($\text{Liminf_state } Sts$))

interpretation *sq*: selection *S_Q*

⟨proof⟩

interpretation *gr*: ground_resolution_with_selection *S_Q*

⟨proof⟩

interpretation *sr*: standard_redundancy_criterion_reductive *gr.ord_Γ*

⟨proof⟩

interpretation *sr*: standard_redundancy_criterion_counterex_reducing *gr.ord_Γ*

ground_resolution_with_selection.INTERP *S_Q*

⟨proof⟩

The extension of ordered resolution mentioned in 4.10. We let it consist of all sound rules.

definition *ground_sound_Γ* :: 'a inference set **where**

ground_sound_Γ = {*Infer* *CC D E* | *CC D E*. ($\forall I. I \models_m CC \longrightarrow I \models D \longrightarrow I \models E$)}

We prove that we indeed defined an extension.

lemma *gd_ord_Γ_ngd_ord_Γ*: *gr.ord_Γ* \subseteq *ground_sound_Γ*

⟨proof⟩

lemma *sound_ground_sound_Γ*: *sound_inference_system* *ground_sound_Γ*

⟨proof⟩

lemma *sat_preserving_ground_sound_Γ*: *sat_preserving_inference_system* *ground_sound_Γ*

⟨proof⟩

definition *sr_ext_Ri* :: 'a clause set \Rightarrow 'a inference set **where**

sr_ext_Ri *N* = *sr.Ri* *N* \cup (*ground_sound_Γ* - *gr.ord_Γ*)

interpretation *sr_ext*:

sat_preserving_redundancy_criterion ground_sound_Γ sr.Rf sr_ext_Ri
(proof)

lemma *strict_subset_subsumption_redundant_clause*:

assumes

sub: $D \cdot \sigma \subseteq\# C$ **and**

ground_σ: *is_ground_subst* σ

shows $C \in sr.Rf$ (*grounding_of_cls* D)

(proof)

lemma *strict_subset_subsumption_redundant_cls*:

assumes

$D \cdot \sigma \subseteq\# C$ **and**

is_ground_subst σ **and**

$D \in CC$

shows $C \in sr.Rf$ (*grounding_of_cls* CC)

(proof)

lemma *strict_subset_subsumption_grounding_redundant_cls*:

assumes

Dσ_subset_C: $D \cdot \sigma \subseteq\# C$ **and**

D_in_St: $D \in CC$

shows *grounding_of_cls* $C \subseteq sr.Rf$ (*grounding_of_cls* CC)

(proof)

lemma *derive_if_remove_subsumed*:

assumes

$D \in cls_of_state$ St **and**

subsumes D C

shows *sr_ext.derive* (*grounding_of_state* $St \cup$ *grounding_of_cls* C) (*grounding_of_state* St)

(proof)

lemma *reduction_in_concls_of*:

assumes

$C\mu \in$ *grounding_of_cls* C **and**

$D + \{\#L'\# \} \in CC$ **and**

$- L = L' \cdot l \sigma$ **and**

$D \cdot \sigma \subseteq\# C$

shows $C\mu \in$ *concls_of* (*sr_ext.inferences_from* (*grounding_of_cls* ($CC \cup \{C + \{\#L'\#\}$))))

(proof)

lemma *reduction_derivable*:

assumes

$D + \{\#L'\# \} \in CC$ **and**

$- L = L' \cdot l \sigma$ **and**

$D \cdot \sigma \subseteq\# C$

shows *sr_ext.derive* (*grounding_of_cls* ($CC \cup \{C + \{\#L'\#\}$))) (*grounding_of_cls* ($CC \cup \{C\}$))

(proof)

The following corresponds the part of Lemma 4.10 that states we have a theorem proving process:

lemma *RP_ground_derive*:

$St \rightsquigarrow St' \implies sr_ext.derive$ (*grounding_of_state* St) (*grounding_of_state* St')

(proof)

A useful consequence:

theorem *RP_model*: $St \rightsquigarrow St' \implies I \models_s$ *grounding_of_state* $St' \longleftrightarrow I \models_s$ *grounding_of_state* St

(proof)

Another formulation of the part of Lemma 4.10 that states we have a theorem proving process:

lemma *ground_derive_chain*: *chain* (\rightsquigarrow) $Sts \implies$ *chain* *sr_ext.derive* (*lmap* *grounding_of_state* Sts)

(proof)

The following is used to prove Lemma 4.11:

lemma *Sup_llist_grounding_of_state_ground*:

assumes $C \in \text{Sup_llist } (\text{lmap } \text{grounding_of_state } \text{Sts})$

shows $\text{is_ground_cls } C$

<proof>

lemma *Liminf_grounding_of_state_ground*:

$C \in \text{Liminf_llist } (\text{lmap } \text{grounding_of_state } \text{Sts}) \implies \text{is_ground_cls } C$

<proof>

lemma *in_Sup_llist_in_Sup_state*:

assumes $C \in \text{Sup_llist } (\text{lmap } \text{grounding_of_state } \text{Sts})$

shows $\exists D \sigma. D \in \text{cls_of_state } (\text{Sup_state } \text{Sts}) \wedge D \cdot \sigma = C \wedge \text{is_ground_subst } \sigma$

<proof>

lemma

$N_of_state_Liminf: N_of_state (Liminf_state \text{Sts}) = Liminf_llist (\text{lmap } N_of_state \text{Sts})$ **and**

$P_of_state_Liminf: P_of_state (Liminf_state \text{Sts}) = Liminf_llist (\text{lmap } P_of_state \text{Sts})$

<proof>

lemma *eventually_removed_from_N*:

assumes

$d_in: D \in N_of_state (\text{lnth } \text{Sts } i)$ **and**

$fair: fair_state_seq \text{Sts}$ **and**

$i_Sts: \text{enat } i < \text{llength } \text{Sts}$

shows $\exists l. D \in N_of_state (\text{lnth } \text{Sts } l) \wedge D \notin N_of_state (\text{lnth } \text{Sts } (\text{Suc } l)) \wedge i \leq l$

$\wedge \text{enat } (\text{Suc } l) < \text{llength } \text{Sts}$

<proof>

lemma *eventually_removed_from_P*:

assumes

$d_in: D \in P_of_state (\text{lnth } \text{Sts } i)$ **and**

$fair: fair_state_seq \text{Sts}$ **and**

$i_Sts: \text{enat } i < \text{llength } \text{Sts}$

shows $\exists l. D \in P_of_state (\text{lnth } \text{Sts } l) \wedge D \notin P_of_state (\text{lnth } \text{Sts } (\text{Suc } l)) \wedge i \leq l$

$\wedge \text{enat } (\text{Suc } l) < \text{llength } \text{Sts}$

<proof>

lemma *instance_if_subsumed_and_in_limit*:

assumes

$deriv: \text{chain } (\rightsquigarrow) \text{Sts}$ **and**

$ns: Gs = \text{lmap } \text{grounding_of_state } \text{Sts}$ **and**

$c: C \in \text{Liminf_llist } Gs - sr.Rf (\text{Liminf_llist } Gs)$ **and**

$d: D \in \text{cls_of_state } (\text{lnth } \text{Sts } i) \text{ enat } i < \text{llength } \text{Sts} \text{ subsumes } D C$

shows $\exists \sigma. D \cdot \sigma = C \wedge \text{is_ground_subst } \sigma$

<proof>

lemma *from_Q_to_Q_inf*:

assumes

$deriv: \text{chain } (\rightsquigarrow) \text{Sts}$ **and**

$fair: fair_state_seq \text{Sts}$ **and**

$ns: Gs = \text{lmap } \text{grounding_of_state } \text{Sts}$ **and**

$c: C \in \text{Liminf_llist } Gs - sr.Rf (\text{Liminf_llist } Gs)$ **and**

$d: D \in Q_of_state (\text{lnth } \text{Sts } i) \text{ enat } i < \text{llength } \text{Sts} \text{ subsumes } D C$ **and**

$d_least: \forall E \in \{E. E \in (\text{cls_of_state } (\text{Sup_state } \text{Sts})) \wedge \text{subsumes } E C\}.$

$\neg \text{strictly_subsumes } E D$

shows $D \in Q_of_state (Liminf_state \text{Sts})$

<proof>

lemma *from_P_to_Q*:

assumes

$deriv: \text{chain } (\rightsquigarrow) \text{Sts}$ **and**

$fair: fair_state_seq \text{Sts}$ **and**

ns: $Gs = \text{lmap grounding_of_state } Sts$ **and**
c: $C \in \text{Liminf_l1ist } Gs - \text{sr.Rf } (\text{Liminf_l1ist } Gs)$ **and**
d: $D \in \text{P_of_state } (\text{lth } Sts \ i) \text{ enat } i < \text{llength } Sts \text{ subsumes } D \ C$ **and**
d_least: $\forall E \in \{E. E \in (\text{cls_of_state } (\text{Sup_state } Sts)) \wedge \text{subsumes } E \ C\}.$
 $\neg \text{strictly_subsumes } E \ D$
shows $\exists l. D \in \text{Q_of_state } (\text{lth } Sts \ l) \wedge \text{enat } l < \text{llength } Sts$
 <proof>

lemma *from_N_to_P_or_Q*:

assumes
deriv: $\text{chain } (\rightsquigarrow) \ Sts$ **and**
fair: $\text{fair_state_seq } Sts$ **and**
ns: $Gs = \text{lmap grounding_of_state } Sts$ **and**
c: $C \in \text{Liminf_l1ist } Gs - \text{sr.Rf } (\text{Liminf_l1ist } Gs)$ **and**
d: $D \in \text{N_of_state } (\text{lth } Sts \ i) \text{ enat } i < \text{llength } Sts \text{ subsumes } D \ C$ **and**
d_least: $\forall E \in \{E. E \in (\text{cls_of_state } (\text{Sup_state } Sts)) \wedge \text{subsumes } E \ C\}.$ $\neg \text{strictly_subsumes } E \ D$
shows $\exists l \ D' \ \sigma'. D' \in \text{P_of_state } (\text{lth } Sts \ l) \cup \text{Q_of_state } (\text{lth } Sts \ l) \wedge$
 $\text{enat } l < \text{llength } Sts \wedge$
 $(\forall E \in \{E. E \in (\text{cls_of_state } (\text{Sup_state } Sts)) \wedge \text{subsumes } E \ C\}.$ $\neg \text{strictly_subsumes } E \ D') \wedge$
 $D' \cdot \sigma' = C \wedge \text{is_ground_subst } \sigma' \wedge \text{subsumes } D' \ C$
 <proof>

lemma *eventually_in_Qinf*:

assumes
deriv: $\text{chain } (\rightsquigarrow) \ Sts$ **and**
D_p: $D \in \text{cls_of_state } (\text{Sup_state } Sts)$
 $\text{subsumes } D \ C \ \forall E \in \{E. E \in (\text{cls_of_state } (\text{Sup_state } Sts)) \wedge \text{subsumes } E \ C\}.$
 $\neg \text{strictly_subsumes } E \ D$ **and**
fair: $\text{fair_state_seq } Sts$ **and**
ns: $Gs = \text{lmap grounding_of_state } Sts$ **and**
c: $C \in \text{Liminf_l1ist } Gs - \text{sr.Rf } (\text{Liminf_l1ist } Gs)$ **and**
ground_C: $\text{is_ground_cls } C$
shows $\exists D' \ \sigma'. D' \in \text{Q_of_state } (\text{Liminf_state } Sts) \wedge D' \cdot \sigma' = C \wedge \text{is_ground_subst } \sigma'$
 <proof>

The following corresponds to Lemma 4.11:

lemma *fair_imp_Liminf_minus_Rf_subset_ground_Liminf_state*:

assumes
deriv: $\text{chain } (\rightsquigarrow) \ Sts$ **and**
fair: $\text{fair_state_seq } Sts$ **and**
ns: $Gs = \text{lmap grounding_of_state } Sts$
shows $\text{Liminf_l1ist } Gs - \text{sr.Rf } (\text{Liminf_l1ist } Gs)$
 $\subseteq \text{grounding_of_cls } (\text{Q_of_state } (\text{Liminf_state } Sts))$
 <proof>

The following corresponds to (one direction of) Theorem 4.13:

lemma *subsetq_Liminf_state_eventually_always*:

fixes CC
assumes
finite CC **and**
 $CC \neq \{\}$ **and**
 $CC \subseteq \text{Q_of_state } (\text{Liminf_state } Sts)$
shows $\exists j. \text{enat } j < \text{llength } Sts \wedge (\forall j' \geq \text{enat } j. j' < \text{llength } Sts \longrightarrow CC \subseteq \text{Q_of_state } (\text{lth } Sts \ j'))$
 <proof>

lemma *empty_clause_in_Q_of_Liminf_state*:

assumes
deriv: $\text{chain } (\rightsquigarrow) \ Sts$ **and**
fair: $\text{fair_state_seq } Sts$ **and**
empty_in: $\{\#\} \in \text{Liminf_l1ist } (\text{lmap grounding_of_state } Sts)$
shows $\{\#\} \in \text{Q_of_state } (\text{Liminf_state } Sts)$
 <proof>

lemma *grounding_of_state_Liminf_state_subseteq*:
 $\text{grounding_of_state } (\text{Liminf_state } Sts) \subseteq \text{Liminf_llist } (\text{lmap } \text{grounding_of_state } Sts)$
 ⟨proof⟩

theorem *RP_sound*:
assumes
deriv: $\text{chain } (\rightsquigarrow) Sts$ **and**
 $\{\#\} \in \text{cls_of_state } (\text{Liminf_state } Sts)$
shows $\neg \text{satisfiable } (\text{grounding_of_state } (\text{lhd } Sts))$
 ⟨proof⟩

theorem *RP_saturated_if_fair*:
assumes
deriv: $\text{chain } (\rightsquigarrow) Sts$ **and**
fair: $\text{fair_state_seq } Sts$ **and**
empty_Q0: $Q_of_state (\text{lhd } Sts) = \{\}$
shows $\text{sr.saturated_upto } (\text{Liminf_llist } (\text{lmap } \text{grounding_of_state } Sts))$
 ⟨proof⟩

corollary *RP_complete_if_fair*:
assumes
deriv: $\text{chain } (\rightsquigarrow) Sts$ **and**
fair: $\text{fair_state_seq } Sts$ **and**
empty_Q0: $Q_of_state (\text{lhd } Sts) = \{\}$ **and**
unsat: $\neg \text{satisfiable } (\text{grounding_of_state } (\text{lhd } Sts))$
shows $\{\#\} \in Q_of_state (\text{Liminf_state } Sts)$
 ⟨proof⟩

end

end

end