

# Formalization of Bachmair and Ganzinger’s Ordered Resolution Prover

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

February 6, 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

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Map Function on Two Parallel Lists</b>	<b>1</b>
<b>3</b>	<b>Supremum and Liminf of Lazy Lists</b>	<b>3</b>
3.1	Library . . . . .	3
3.2	Supremum . . . . .	3
3.3	Supremum up-to . . . . .	4
3.4	Liminf . . . . .	4
3.5	Liminf up-to . . . . .	5
<b>4</b>	<b>Relational Chains over Lazy Lists</b>	<b>6</b>
4.1	Chains . . . . .	6
4.2	A Coinductive Puzzle . . . . .	7
4.3	Full Chains . . . . .	11
<b>5</b>	<b>Clausal Logic</b>	<b>11</b>
5.1	Literals . . . . .	12
5.2	Clauses . . . . .	14
<b>6</b>	<b>Herbrand Intepretation</b>	<b>15</b>
<b>7</b>	<b>Abstract Substitutions</b>	<b>18</b>
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
<b>8</b>	<b>Refutational Inference Systems</b>	<b>35</b>
8.1	Preliminaries	35
8.2	Refutational Completeness	36
8.3	Compactness	36
<b>9</b>	<b>Candidate Models for Ground Resolution</b>	<b>37</b>
<b>10</b>	<b>Ground Unordered Resolution Calculus</b>	<b>41</b>
10.1	Inference Rule	41
10.2	Inference System	42
<b>11</b>	<b>Ground Ordered Resolution Calculus with Selection</b>	<b>42</b>
11.1	Inference Rule	42
11.2	Inference System	44
<b>12</b>	<b>Theorem Proving Processes</b>	<b>44</b>
<b>13</b>	<b>The Standard Redundancy Criterion</b>	<b>46</b>
<b>14</b>	<b>First-Order Ordered Resolution Calculus with Selection</b>	<b>49</b>
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
<b>15</b>	<b>An Ordered Resolution Prover for First-Order Clauses</b>	<b>54</b>

## 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](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](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](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**

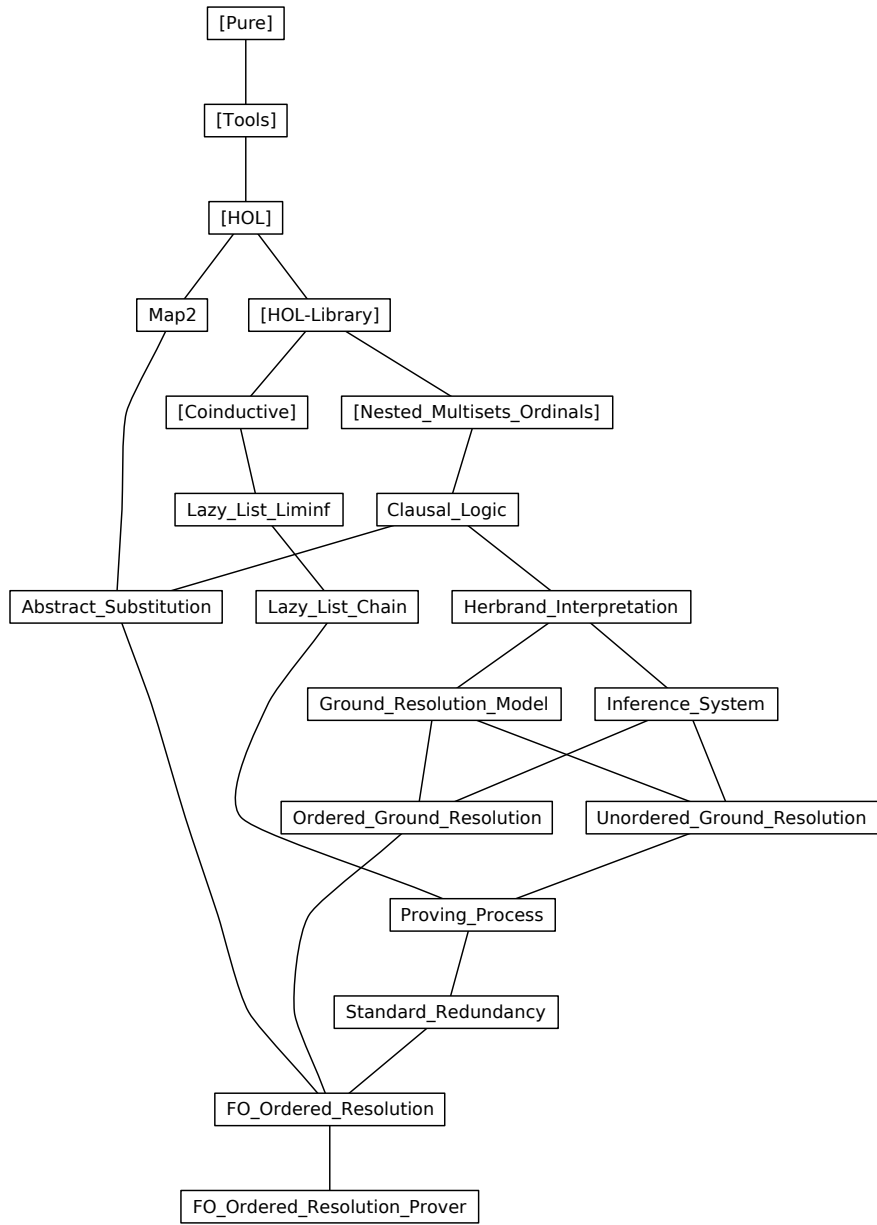


Figure 1: Theory dependency graph

$Sup\_l\!l\!i\!s\!t\ Xs = (\bigcup i \in \{i. \text{enat } i < \text{llength } Xs\}. \text{lnth } Xs\ i)$

**lemma**  $lnth\_subset\_Sup\_l\!l\!i\!s\!t$ :  $\text{enat } i < \text{llength } Xs \implies \text{lnth } Xs\ i \subseteq Sup\_l\!l\!i\!s\!t\ Xs$   
 ⟨proof⟩

**lemma**  $Sup\_l\!l\!i\!s\!t\_imp\_exists\_index$ :  $x \in Sup\_l\!l\!i\!s\!t\ Xs \implies \exists i. \text{enat } i < \text{llength } Xs \wedge x \in \text{lnth } Xs\ i$   
 ⟨proof⟩

**lemma**  $exists\_index\_imp\_Sup\_l\!l\!i\!s\!t$ :  $\text{enat } i < \text{llength } Xs \implies x \in \text{lnth } Xs\ i \implies x \in Sup\_l\!l\!i\!s\!t\ Xs$   
 ⟨proof⟩

**lemma**  $Sup\_l\!l\!i\!s\!t\_LNil[simp]$ :  $Sup\_l\!l\!i\!s\!t\ LNil = \{\}$   
 ⟨proof⟩

**lemma**  $Sup\_l\!l\!i\!s\!t\_LCons[simp]$ :  $Sup\_l\!l\!i\!s\!t\ (LCons\ X\ Xs) = X \cup Sup\_l\!l\!i\!s\!t\ Xs$   
 ⟨proof⟩

**lemma**  $lhd\_subset\_Sup\_l\!l\!i\!s\!t$ :  $\neg \text{lnull } Xs \implies \text{lhd } Xs \subseteq Sup\_l\!l\!i\!s\!t\ Xs$   
 ⟨proof⟩

### 3.3 Supremum up-to

**definition**  $Sup\_upto\_l\!l\!i\!s\!t$  :: 'a set  $l\!l\!i\!s\!t \Rightarrow \text{enat} \Rightarrow$  'a set **where**  
 $Sup\_upto\_l\!l\!i\!s\!t\ Xs\ j = (\bigcup i \in \{i. \text{enat } i < \text{llength } Xs \wedge \text{enat } i \leq j\}. \text{lnth } Xs\ i)$

**lemma**  $Sup\_upto\_l\!l\!i\!s\!t\_eq\_Sup\_l\!l\!i\!s\!t\_ltake$ :  $Sup\_upto\_l\!l\!i\!s\!t\ Xs\ j = Sup\_l\!l\!i\!s\!t\ (\text{ltake } (eSuc\ j)\ Xs)$   
 ⟨proof⟩

**lemma**  $Sup\_upto\_l\!l\!i\!s\!t\_enat\_0[simp]$ :  
 $Sup\_upto\_l\!l\!i\!s\!t\ Xs\ (\text{enat } 0) = (\text{if } \text{lnull } Xs \text{ then } \{\} \text{ else } \text{lhd } Xs)$   
 ⟨proof⟩

**lemma**  $Sup\_upto\_l\!l\!i\!s\!t\_Suc[simp]$ :  
 $Sup\_upto\_l\!l\!i\!s\!t\ Xs\ (\text{enat } (Suc\ j)) =$   
 $Sup\_upto\_l\!l\!i\!s\!t\ Xs\ (\text{enat } j) \cup (\text{if } \text{enat } (Suc\ j) < \text{llength } Xs \text{ then } \text{lnth } Xs\ (Suc\ j) \text{ else } \{\})$   
 ⟨proof⟩

**lemma**  $Sup\_upto\_l\!l\!i\!s\!t\_infinity[simp]$ :  $Sup\_upto\_l\!l\!i\!s\!t\ Xs\ \infty = Sup\_l\!l\!i\!s\!t\ Xs$   
 ⟨proof⟩

**lemma**  $Sup\_upto\_l\!l\!i\!s\!t\_0[simp]$ :  $Sup\_upto\_l\!l\!i\!s\!t\ Xs\ 0 = (\text{if } \text{lnull } Xs \text{ then } \{\} \text{ else } \text{lhd } Xs)$   
 ⟨proof⟩

**lemma**  $Sup\_upto\_l\!l\!i\!s\!t\_eSuc[simp]$ :  
 $Sup\_upto\_l\!l\!i\!s\!t\ Xs\ (eSuc\ j) =$   
 (case  $j$  of  
    $\text{enat } k \Rightarrow Sup\_upto\_l\!l\!i\!s\!t\ Xs\ (\text{enat } (Suc\ k))$   
    $|\ \infty \Rightarrow Sup\_l\!l\!i\!s\!t\ Xs)$   
 ⟨proof⟩

**lemma**  $Sup\_upto\_l\!l\!i\!s\!t\_mono[simp]$ :  $j \leq k \implies Sup\_upto\_l\!l\!i\!s\!t\ Xs\ j \subseteq Sup\_upto\_l\!l\!i\!s\!t\ Xs\ k$   
 ⟨proof⟩

**lemma**  $Sup\_upto\_l\!l\!i\!s\!t\_subset\_Sup\_l\!l\!i\!s\!t$ :  $Sup\_upto\_l\!l\!i\!s\!t\ Xs\ j \subseteq Sup\_l\!l\!i\!s\!t\ Xs$   
 ⟨proof⟩

**lemma**  $elem\_Sup\_l\!l\!i\!s\!t\_imp\_Sup\_upto\_l\!l\!i\!s\!t$ :  
 $x \in Sup\_l\!l\!i\!s\!t\ Xs \implies \exists j < \text{llength } Xs. x \in Sup\_upto\_l\!l\!i\!s\!t\ Xs\ (\text{enat } j)$   
 ⟨proof⟩

**lemma**  $lnth\_subset\_Sup\_upto\_l\!l\!i\!s\!t$ :  $j < \text{llength } Xs \implies \text{lnth } Xs\ j \subseteq Sup\_upto\_l\!l\!i\!s\!t\ Xs\ j$   
 ⟨proof⟩

**lemma**  $finite\_Sup\_l\!l\!i\!s\!t\_imp\_Sup\_upto\_l\!l\!i\!s\!t$ :

**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**

$Liminf\_upto\_l\!l\!i\!s\!t\ Xs\ k =$   
 $(\bigcup i \in \{i. \text{enat } i < \text{llength } Xs \wedge \text{enat } i \leq k\}.$   
 $\bigcap j \in \{j. i \leq j \wedge \text{enat } j < \text{llength } Xs \wedge \text{enat } j \leq k\}. \text{lnth } Xs\ j)$

**lemma**  $Liminf\_upto\_l\!l\!i\!s\!t\_eq\_Liminf\_l\!l\!i\!s\!t\_ltake:$   
 $Liminf\_upto\_l\!l\!i\!s\!t\ Xs\ j = Liminf\_l\!l\!i\!s\!t\ (\text{ltake } (eSuc\ j)\ Xs)$   
 $\langle \text{proof} \rangle$

**lemma**  $Liminf\_upto\_l\!l\!i\!s\!t\_enat[simp]:$   
 $Liminf\_upto\_l\!l\!i\!s\!t\ Xs\ (\text{enat } k) =$   
 $(\text{if } \text{enat } k < \text{llength } Xs \text{ then } \text{lnth } Xs\ k \text{ else if } \text{lnull } Xs \text{ then } \{\} \text{ else } \text{llast } Xs)$   
 $\langle \text{proof} \rangle$

**lemma**  $Liminf\_upto\_l\!l\!i\!s\!t\_infinity[simp]:$   $Liminf\_upto\_l\!l\!i\!s\!t\ Xs\ \infty = Liminf\_l\!l\!i\!s\!t\ Xs$   
 $\langle \text{proof} \rangle$

**lemma**  $Liminf\_upto\_l\!l\!i\!s\!t\_0[simp]:$   
 $Liminf\_upto\_l\!l\!i\!s\!t\ Xs\ 0 = (\text{if } \text{lnull } Xs \text{ then } \{\} \text{ else } \text{lhs } Xs)$   
 $\langle \text{proof} \rangle$

**lemma**  $Liminf\_upto\_l\!l\!i\!s\!t\_eSuc[simp]:$   
 $Liminf\_upto\_l\!l\!i\!s\!t\ Xs\ (eSuc\ j) =$   
 $(\text{case } j \text{ of}$   
 $\text{enat } k \Rightarrow Liminf\_upto\_l\!l\!i\!s\!t\ Xs\ (\text{enat } (Suc\ k))$   
 $| \infty \Rightarrow Liminf\_l\!l\!i\!s\!t\ Xs)$   
 $\langle \text{proof} \rangle$

**lemma**  $elem\_Liminf\_l\!l\!i\!s\!t\_imp\_Liminf\_upto\_l\!l\!i\!s\!t:$   
 $x \in Liminf\_l\!l\!i\!s\!t\ Xs \Longrightarrow$   
 $\exists i < \text{llength } Xs. \forall j. i \leq j \wedge j < \text{llength } Xs \longrightarrow x \in Liminf\_upto\_l\!l\!i\!s\!t\ Xs\ (\text{enat } j)$   
 $\langle \text{proof} \rangle$

**end**

## 4 Relational Chains over Lazy Lists

**theory**  $Lazy\_List\_Chain$   
**imports**  
 $HOL-Library.BNF\_Corec$   
 $Lazy\_List\_Liminf$

**begin**

A chain is a lazy list of elements such that all pairs of consecutive elements are related by a given relation. A full chain is either an infinite chain or a finite chain that cannot be extended. The inspiration for this theory is Section 4.1 (“Theorem Proving Processes”) of Bachmair and Ganzinger’s chapter.

### 4.1 Chains

**coinductive**  $chain :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ l\!l\!i\!s\!t \Rightarrow bool$  **for**  $R :: 'a \Rightarrow 'a \Rightarrow bool$  **where**  
 $chain\_singleton: chain\ R\ (LCons\ x\ LNil)$   
 $| chain\_cons: chain\ R\ xs \Longrightarrow R\ x\ (\text{lhs } xs) \Longrightarrow chain\ R\ (LCons\ x\ xs)$

**lemma**  
 $chain\_LNil[simp]: \neg chain\ R\ LNil$  **and**  
 $chain\_not\_lnull: chain\ R\ xs \Longrightarrow \neg \text{lnull } xs$   
 $\langle \text{proof} \rangle$

**lemma**  $chain\_lappend:$   
**assumes**  
 $r\_xs: chain\ R\ xs$  **and**  
 $r\_ys: chain\ R\ ys$  **and**  
 $mid: R\ (\text{llast } xs)\ (\text{lhs } ys)$   
**shows**  $chain\ R\ (\text{lappend } xs\ ys)$

*<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*:  $\text{chain } R (\text{LCons } x \ xs)$

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

## 4.2 A Coinductive Puzzle

**primrec** *prepend* **where**

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

**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]*:  $\text{prepend } xs \ \text{LNil} = \text{llist\_of } xs$   
 ⟨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*:  $\text{prepend } xs (\text{LCons } y \ ys) = \text{prepend } (xs @ [y]) \ ys$   
 ⟨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*:  $P \ \text{LNil}$   
**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  $\longrightarrow$   $(\forall y. \neg R$  (*llast 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**  
 $I \models_l L \iff (\text{if } \text{is\_pos } L \text{ then } (\lambda P. P) \text{ else } \text{Not}) (\text{atm\_of } L \in I)$

**lemma** *true\_lit\_simps*[*simp*]:  
 $I \models_l \text{Pos } A \iff A \in I$   
 $I \models_l \text{Neg } A \iff A \notin I$   
 ⟨proof⟩

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

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

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

**lemma** *true\_cls\_singleton*[*iff*]:  $I \models \{\#L\} \iff I \models_l L$   
 ⟨proof⟩

**lemma** *true\_cls\_add\_mset*[*iff*]:  $I \models \text{add\_mset } C D \iff I \models_l C \vee I \models D$   
 ⟨proof⟩

**lemma** *true\_cls\_union*[*iff*]:  $I \models C + D \iff I \models C \vee I \models D$   
 ⟨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*]:  $I \models \text{replicate\_mset } n L \iff n \neq 0 \wedge I \models_l L$   
 ⟨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**  
 $AAA \cdot ass\ \sigma = (\lambda AAA. AAA \cdot as\ \sigma) \text{ ' } AAA$

**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]*:  $A \cdot a \text{ id\_subst} = A$  **and**

*subst\_atm\_comp\_subst[simp]*:  $A \cdot a (\sigma \odot \tau) = (A \cdot a \sigma) \cdot a \tau$  **and**

*subst\_ext*:  $(\bigwedge A. A \cdot a \sigma = A \cdot a \tau) \Longrightarrow \sigma = \tau$  **and**

*make\_ground\_subst*:  $\text{is\_ground\_cls } (C \cdot \sigma) \Longrightarrow \exists \tau. \text{is\_ground\_subst } \tau \wedge C \cdot \tau = C \cdot \sigma$  **and**

*wf\_strictly\_generalizes\_atm*: *wfP strictly\_generalizes\_atm*

**begin**

**lemma** *subst\_ext\_iff*:  $\sigma = \tau \longleftrightarrow (\forall A. A \cdot a \sigma = A \cdot a \tau)$

*<proof>*

#### 7.3.1 Identity Substitution

**lemma** *id\_subst\_comp\_subst[simp]*:  $\text{id\_subst} \odot \sigma = \sigma$

*<proof>*

**lemma** *comp\_subst\_id\_subst[simp]*:  $\sigma \odot \text{id\_subst} = \sigma$

*<proof>*

**lemma** *id\_subst\_comp\_substs[simp]*:  $\text{replicate } (\text{length } \sigma s) \text{ id\_subst} \odot s \sigma s = \sigma s$

*<proof>*

**lemma** *comp\_substs\_id\_subst[simp]*:  $\sigma s \odot s \text{ replicate } (\text{length } \sigma s) \text{ id\_subst} = \sigma s$

*<proof>*

**lemma** *subst\_atms\_id\_subst[simp]*:  $AA \cdot as \text{ id\_subst} = AA$

*<proof>*

**lemma** *subst\_atmss\_id\_subst[simp]*:  $AAA \cdot ass \text{ id\_subst} = AAA$

*<proof>*

**lemma** *subst\_atm\_list\_id\_subst[simp]*:  $As \cdot al \text{ id\_subst} = As$

*<proof>*

**lemma** *subst\_atm\_mset\_id\_subst[simp]*:  $AA \cdot am \text{ id\_subst} = AA$

*<proof>*

**lemma** *subst\_atm\_mset\_list\_id\_subst[simp]*:  $AAs \cdot aml \text{ id\_subst} = AAs$

*<proof>*

**lemma** *subst\_atm\_mset\_lists\_id\_subst[simp]*:  $AAs \cdot\cdot aml \text{ replicate } (\text{length } AAs) \text{ id\_subst} = AAs$

*<proof>*

**lemma** *subst\_lit\_id\_subst[simp]*:  $L \cdot l \text{ id\_subst} = L$

*<proof>*

**lemma** *subst\_cls\_id\_subst[simp]*:  $C \cdot \text{id\_subst} = C$

*<proof>*

**lemma** *subst\_cls\_id\_subst[simp]*:  $CC \cdot cs \text{ id\_subst} = CC$

*<proof>*

**lemma** *subst\_cls\_list\_id\_subst[simp]*:  $Cs \cdot cl \text{ id\_subst} = Cs$

*<proof>*

**lemma** *subst\_cls\_lists\_id\_subst[simp]*:  $Cs \cdot cl \text{ replicate } (\text{length } Cs) \text{ id\_subst} = Cs$   
*<proof>*

**lemma** *subst\_cls\_mset\_id\_subst[simp]*:  $CC \cdot cm \text{ id\_subst} = CC$   
*<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]*:  $AA \cdot as (\tau \odot \sigma) = AA \cdot as \tau \cdot as \sigma$   
*<proof>*

**lemma** *subst\_atmss\_comp\_subst[simp]*:  $AAA \cdot ass (\tau \odot \sigma) = AAA \cdot ass \tau \cdot ass \sigma$   
*<proof>*

**lemma** *subst\_atm\_list\_comp\_subst[simp]*:  $As \cdot al (\tau \odot \sigma) = As \cdot al \tau \cdot al \sigma$   
*<proof>*

**lemma** *subst\_atm\_mset\_comp\_subst[simp]*:  $AA \cdot am (\tau \odot \sigma) = AA \cdot am \tau \cdot am \sigma$   
*<proof>*

**lemma** *subst\_atm\_mset\_list\_comp\_subst[simp]*:  $AAs \cdot aml (\tau \odot \sigma) = (AAs \cdot aml \tau) \cdot aml \sigma$   
*<proof>*

**lemma** *subst\_atm\_mset\_lists\_comp\_substs[simp]*:  $AAs \cdot aml (\tau s \odot s \sigma s) = AAs \cdot aml \tau s \cdot aml \sigma s$   
*<proof>*

**lemma** *subst\_lit\_comp\_subst[simp]*:  $L \cdot l (\tau \odot \sigma) = L \cdot l \tau \cdot l \sigma$   
*<proof>*

**lemma** *subst\_cls\_comp\_subst[simp]*:  $C \cdot (\tau \odot \sigma) = C \cdot \tau \cdot \sigma$   
*<proof>*

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

**lemma** *subst\_cls\_list\_comp\_subst[simp]*:  $Cs \cdot cl (\tau \odot \sigma) = Cs \cdot cl \tau \cdot cl \sigma$   
*<proof>*

**lemma** *subst\_cls\_lists\_comp\_substs[simp]*:  $Cs \cdot cl (\tau s \odot s \sigma s) = Cs \cdot cl \tau s \cdot cl \sigma s$   
*<proof>*

**lemma** *subst\_cls\_mset\_comp\_subst[simp]*:  $CC \cdot cm (\tau \odot \sigma) = CC \cdot cm \tau \cdot cm \sigma$   
*<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**  $sum\_list\_subsetq\_mset\_is\_ground\_cls\_list[simp]: sum\_list Cs \subseteq\# sum\_list Ds \Longrightarrow is\_ground\_cls\_list Ds \Longrightarrow is\_ground\_cls\_list Cs$   
 ⟨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]*:  $is\_ground\_class\ CC \implies CC \cdot cs\ \sigma = CC$   
 ⟨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*:  $grounding\_of\_class\ (CC \cdot cs\ \mu) \subseteq grounding\_of\_class\ CC$   
 ⟨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 *is\_unifier*  $\sigma$  *AA*  $\implies A \in \text{AA} \implies B \in \text{AA} \implies A \cdot a \sigma = B \cdot a \sigma$   
 ⟨proof⟩

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

**lemma** *is\_unifiers\_subst\_atm\_eqI*:  
 assumes *finite AA is\_unifiers*  $\sigma$  *AAA AA*  $\in$  *AAA A*  $\in$  *AA B*  $\in$  *AA*  
 shows  $A \cdot a \sigma = B \cdot a \sigma$   
 ⟨proof⟩

**theorem** *is\_unifiers\_comp*:  
 $\text{is\_unifiers } \sigma (\text{set\_mset } ' \text{set } (\text{map2 } \text{add\_mset } \text{As } \text{Bs}) \cdot \text{ass } \eta) \iff$   
 $\text{is\_unifiers } (\eta \odot \sigma) (\text{set\_mset } ' \text{set } (\text{map2 } \text{add\_mset } \text{As } \text{Bs}))$   
 ⟨proof⟩

### 7.3.21 Most General Unifier

**lemma** *is\_mgu\_is\_unifiers*:  $\text{is\_mgu } \sigma$  *AAA*  $\implies \text{is\_unifiers } \sigma$  *AAA*  
 ⟨proof⟩

**lemma** *is\_mgu\_is\_most\_general*:  $\text{is\_mgu } \sigma$  *AAA*  $\implies \text{is\_unifiers } \tau$  *AAA*  $\implies \exists \gamma. \tau = \sigma \odot \gamma$   
 ⟨proof⟩

**lemma** *is\_unifiers\_is\_unifier*:  $\text{is\_unifiers } \sigma$  *AAA*  $\implies \text{AA} \in \text{AAA} \implies \text{is\_unifier } \sigma$  *AA*  
 ⟨proof⟩

**lemma** *is\_imgu\_is\_mgu[intro]*:  $\text{is\_imgu } \sigma$  *AAA*  $\implies \text{is\_mgu } \sigma$  *AAA*  
 ⟨proof⟩

**lemma** *is\_imgu\_comp\_idempotent[simp]*:  $\text{is\_imgu } \sigma$  *AAA*  $\implies \sigma \odot \sigma = \sigma$   
 ⟨proof⟩

**lemma** *is\_imgu\_subst\_atm\_idempotent[simp]*:  $\text{is\_imgu } \sigma$  *AAA*  $\implies A \cdot a \sigma \cdot a \sigma = A \cdot a \sigma$   
 ⟨proof⟩

**lemma** *is\_imgu\_subst\_atms\_idempotent[simp]*:  $\text{is\_imgu } \sigma$  *AAA*  $\implies \text{AA} \cdot \text{as } \sigma \cdot \text{as } \sigma = \text{AA} \cdot \text{as } \sigma$   
 ⟨proof⟩

**lemma** *is\_imgu\_subst\_lit\_idemptotent[simp]*:  $\text{is\_imgu } \sigma$  *AAA*  $\implies L \cdot l \sigma \cdot l \sigma = L \cdot l \sigma$   
 ⟨proof⟩

**lemma** *is\_imgu\_subst\_cls\_idemptotent[simp]*:  $\text{is\_imgu } \sigma$  *AAA*  $\implies C \cdot \sigma \cdot \sigma = C \cdot \sigma$   
 ⟨proof⟩

**lemma** *is\_imgu\_subst\_cls\_idemptotent[simp]*:  $\text{is\_imgu } \sigma$  *AAA*  $\implies \text{CC} \cdot \text{cs } \sigma \cdot \text{cs } \sigma = \text{CC} \cdot \text{cs } \sigma$   
 ⟨proof⟩

### 7.3.22 Generalization and Subsumption

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

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

**lemma** *strict\_subset\_subst\_strictly\_subsumes*:  $C \cdot \eta \subset\# D \implies \text{strictly\_subsumes } C D$   
(proof)

**lemma** *generalizes\_lit\_refl[simp]*:  $\text{generalizes\_lit } L L$   
(proof)

**lemma** *generalizes\_lit\_trans*:  
 $\text{generalizes\_lit } L1 L2 \implies \text{generalizes\_lit } L2 L3 \implies \text{generalizes\_lit } L1 L3$   
(proof)

**lemma** *generalizes\_refl[simp]*:  $\text{generalizes } C C$   
(proof)

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

**lemma** *subsumes\_refl*:  $\text{subsumes } C C$   
(proof)

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

**lemma** *strictly\_generalizes\_irrefl*:  $\neg \text{strictly\_generalizes } C C$   
(proof)

**lemma** *strictly\_generalizes\_antisym*:  $\text{strictly\_generalizes } C D \implies \neg \text{strictly\_generalizes } D C$   
(proof)

**lemma** *strictly\_generalizes\_trans*:  
 $\text{strictly\_generalizes } C D \implies \text{strictly\_generalizes } D E \implies \text{strictly\_generalizes } C E$   
(proof)

**lemma** *strictly\_subsumes\_irrefl*:  $\neg \text{strictly\_subsumes } C C$   
(proof)

**lemma** *strictly\_subsumes\_antisym*:  $\text{strictly\_subsumes } C D \implies \neg \text{strictly\_subsumes } D C$   
(proof)

**lemma** *strictly\_subsumes\_trans*:  
 $\text{strictly\_subsumes } C D \implies \text{strictly\_subsumes } D E \implies \text{strictly\_subsumes } C E$   
(proof)

**lemma** *subset\_strictly\_subsumes*:  $C \subset\# D \implies \text{strictly\_subsumes } C D$   
(proof)

**lemma** *strictly\_generalizes\_neq*:  $\text{strictly\_generalizes } D' D \implies D' \neq D \cdot \sigma$   
(proof)

**lemma** *strictly\_subsumes\_neq*:  $\text{strictly\_subsumes } D' D \implies D' \neq D \cdot \sigma$   
(proof)

**lemma** *variants\_imp\_exists\_substitution*:  $\text{variants } D D' \implies \exists \sigma. D \cdot \sigma = D'$   
(proof)

**lemma** *strictly\_subsumes\_variants*:  
**assumes**  $\text{strictly\_subsumes } E D$  **and**  $\text{variants } D D'$   
**shows**  $\text{strictly\_subsumes } E D'$   
(proof)

**lemma** *neg\_strictly\_subsumes\_variants*:  
**assumes**  $\neg \text{strictly\_subsumes } E D$  **and**  $\text{variants } D D'$   
**shows**  $\neg \text{strictly\_subsumes } E D'$   
(proof)

end

**locale** *substitution\_renamings* = *substitution subst\_atm id\_subst comp\_subst*  
**for**  
  *subst\_atm* :: 'a  $\Rightarrow$  's  $\Rightarrow$  'a **and**  
  *id\_subst* :: 's **and**  
  *comp\_subst* :: 's  $\Rightarrow$  's  $\Rightarrow$  's +  
**fixes**  
  *renamings\_apart* :: 'a clause list  $\Rightarrow$  's list **and**  
  *atm\_of\_atms* :: 'a list  $\Rightarrow$  'a  
**assumes**  
  *renamings\_apart\_length*: *length (renamings\_apart Cs) = length Cs* **and**  
  *renamings\_apart\_renaming*:  $\varrho \in \text{set } (\text{renamings\_apart } Cs) \Longrightarrow \text{is\_renaming } \varrho$  **and**  
  *renamings\_apart\_var\_disjoint*: *var\_disjoint (Cs ..cl (renamings\_apart Cs))* **and**  
  *atm\_of\_atms\_subst*:  
     $\bigwedge As Bs. \text{atm\_of\_atms } As \cdot a \sigma = \text{atm\_of\_atms } Bs \longleftrightarrow \text{map } (\lambda A. A \cdot a \sigma) As = Bs$   
**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  $\Rightarrow$  '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+  
**fixes**  
  *mgu* :: 'a set set  $\Rightarrow$  's option  
**assumes**  
  *mgu\_sound*: *finite AAA*  $\Longrightarrow (\forall AA \in AAA. \text{finite } AA) \Longrightarrow \text{mgu } AAA = \text{Some } \sigma \Longrightarrow \text{is\_mgu } \sigma \text{ } AAA$  **and**  
  *mgu\_complete*:  
    *finite AAA*  $\Longrightarrow (\forall AA \in AAA. \text{finite } AA) \Longrightarrow \text{is\_unifiers } \sigma \text{ } AAA \Longrightarrow \exists \tau. \text{mgu } AAA = \text{Some } \tau$   
**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*  $\sigma = \text{mgu } (\text{set\_mset } ' \text{set } (\text{map2 } \text{add\_mset } As \ AAs))$  **and**  
  *i\_lt*: *i < n* **and**  
  *a\_in*: *A*  $\in \# AAs ! i$   
**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_premis_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_premis_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*:  $C \in CC \Rightarrow C \in \text{saturate } CC$   
| *step*:  $\text{Infer } CC' D E \in \Gamma \Rightarrow (\bigwedge C'. C' \in \# CC' \Rightarrow C' \in \text{saturate } CC) \Rightarrow D \in \text{saturate } CC \Rightarrow E \in \text{saturate } CC$

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

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

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

**end**

**context** *sound\_inference\_system*  
**begin**

**theorem** *saturate\_sound*:  $C \in \text{saturate } CC \Rightarrow I \models_s CC \Rightarrow I \models C$   
<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*  $\Rightarrow$  *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  $\lambda_{\cdot}$ .  $\{\}$  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  $\varepsilon_C$  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 =  $\{\text{Infer } \{\#C\# \} D E \mid C D E. \text{unord\_resolve } C D E\}$ 
```

```
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* = { $\gamma$ .  $\gamma \in \Gamma \wedge \text{concl\_of } \gamma \in N$ }

**sublocale** *effective\_redundancy\_criterion*  $\Gamma$  *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*  $\Gamma$  *Rf* *Ri*  
  **shows** *redundancy\_criterion*  $\Gamma'$  *Rf* ( $\lambda N$ . *Ri*  $N \cup (\Gamma' - \Gamma)$ )  
  ⟨*proof*⟩

**lemma** *redundancy\_criterion\_standard\_extension\_saturated\_upto\_iff*:  
  **assumes**  $\Gamma \subseteq \Gamma'$  **and** *redundancy\_criterion*  $\Gamma$  *Rf* *Ri*  
  **shows** *redundancy\_criterion.saturated\_upto*  $\Gamma$  *Rf* *Ri* *M*  $\longleftrightarrow$   
    *redundancy\_criterion.saturated\_upto*  $\Gamma'$  *Rf* ( $\lambda N$ . *Ri*  $N \cup (\Gamma' - \Gamma)$ ) *M*  
  ⟨*proof*⟩

**lemma** *redundancy\_criterion\_standard\_extension\_effective*:  
  **assumes**  $\Gamma \subseteq \Gamma'$  **and** *effective\_redundancy\_criterion*  $\Gamma$  *Rf* *Ri*  
  **shows** *effective\_redundancy\_criterion*  $\Gamma'$  *Rf* ( $\lambda N$ . *Ri*  $N \cup (\Gamma' - \Gamma)$ )  
  ⟨*proof*⟩

**lemma** *redundancy\_criterion\_standard\_extension\_fair\_iff*:  
  **assumes**  $\Gamma \subseteq \Gamma'$  **and** *effective\_redundancy\_criterion*  $\Gamma$  *Rf* *Ri*  
  **shows** *effective\_redundancy\_criterion.fair\_cls\_seq*  $\Gamma'$  *Rf* ( $\lambda N$ . *Ri*  $N \cup (\Gamma' - \Gamma)$ ) *Ns*  $\longleftrightarrow$   
    *effective\_redundancy\_criterion.fair\_cls\_seq*  $\Gamma$  *Rf* *Ri* *Ns*  
  ⟨*proof*⟩

**theorem** *redundancy\_criterion\_standard\_extension\_fair\_derive\_saturated\_upto*:  
**assumes**  
*subs*:  $\Gamma \subseteq \Gamma'$  **and**  
*red*: *redundancy\_criterion*  $\Gamma$  *Rf Ri* **and**  
*red'*: *sat\_preserving\_effective\_redundancy\_criterion*  $\Gamma'$  *Rf*  $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))$  **and**  
*deriv*: *chain* (*redundancy\_criterion.derive*  $\Gamma'$  *Rf*) *Ns* **and**  
*fair*: *effective\_redundancy\_criterion.fair\_cls\_seq*  $\Gamma'$  *Rf*  $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))$  *Ns*  
**shows** *redundancy\_criterion.saturated\_upto*  $\Gamma$  *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*  $\Gamma$  **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*:

$S DA = \text{negs } (\text{mset } As) \vee S DA = \{\#\} \wedge \text{length } As = 1 \wedge \text{maximal\_wrt } (As ! 0 \cdot a \sigma) (DA \cdot \sigma) \Longrightarrow$   
 $\text{eligible } \sigma As DA$

**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*:

$\text{length } CAs = n \Longrightarrow$

$\text{length } Cs = n \Longrightarrow$

$\text{length } AAs = n \Longrightarrow$

$\text{length } As = n \Longrightarrow$

$n \neq 0 \Longrightarrow$

$(\forall i < n. CAs ! i = Cs ! i + \text{poss } (AAs ! i)) \Longrightarrow$

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

$\text{Some } \sigma = \text{mgu } (\text{set\_mset } ' \text{set } (\text{map2 } \text{add\_mset } As AAs)) \Longrightarrow$

$\text{eligible } \sigma As (D + \text{negs } (\text{mset } As)) \Longrightarrow$

$(\forall i < n. \text{strictly\_maximal\_wrt } (As ! i \cdot a \sigma) (Cs ! i \cdot \sigma)) \Longrightarrow$

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

$\text{ord\_resolve } CAs (D + \text{negs } (\text{mset } As)) AAs As \sigma ((\sum \# (\text{mset } Cs) + D) \cdot \sigma)$

**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*:

$\text{length } CAs = n \Longrightarrow$

$\text{length } AAs = n \Longrightarrow$

$\text{length } As = n \Longrightarrow$

$(\forall i < n. \text{poss } (AAs ! i) \subseteq\# CAs ! i) \Longrightarrow$

$\text{negs } (\text{mset } As) \subseteq\# DA \Longrightarrow$

$g = \text{hd } (\text{renamings\_apart } (DA \# CAs)) \Longrightarrow$

$gs = \text{tl } (\text{renamings\_apart } (DA \# CAs)) \Longrightarrow$

$$\text{ord\_resolve } (CAs \cdot\cdot cl \ \varrho s) (DA \cdot \varrho) (AAs \cdot\cdot aml \ \varrho s) (As \cdot al \ \varrho) \sigma E \implies \\ \text{ord\_resolve\_rename } CAs DA AAs As \sigma E$$

**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*  $\eta$   
**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*  $\eta$   
**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**  $S\_M\_not\_grounding\_of\_cls$ :  $C \notin \text{grounding\_of\_cls } M \implies S\_M C = S C$   
⟨proof⟩

**lemma**  $S\_M\_selects\_subsetq$ :  $S\_M C \subseteq\# C$   
⟨proof⟩

**lemma**  $S\_M\_selects\_neg\_lits$ :  $L \in\# S\_M C \implies \text{is\_neg } L$   
⟨proof⟩

**end**

**end**

The following corresponds to Lemma 4.12:

**lemma**  $\text{ground\_resolvent\_subset}$ :  
**assumes**  
 $gr\_cas$ :  $\text{is\_ground\_cls\_list } CAs$  **and**  
 $gr\_da$ :  $\text{is\_ground\_cls } DA$  **and**  
 $res\_e$ :  $\text{ord\_resolve } S CAs DA AAs As \sigma E$   
**shows**  $E \subseteq\# \sum\# (\text{mset } CAs) + DA$   
⟨proof⟩

**lemma**  $\text{ord\_resolve\_obtain\_clauses}$ :  
**assumes**  
 $res\_e$ :  $\text{ord\_resolve } (S\_M S M) CAs DA AAs As \sigma E$  **and**  
 $select$ :  $\text{selection } S$  **and**  
 $grounding$ :  $\{DA\} \cup \text{set } CAs \subseteq \text{grounding\_of\_cls } M$  **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**  
 $sel\_stable$ :  $\bigwedge \varrho C. \text{is\_renaming } \varrho \implies S (C \cdot \varrho) = S C \cdot \varrho$  **and**  
 $res\_e$ :  $\text{ord\_resolve } (S\_M S M) CAs DA AAs As \sigma E$  **and**  
 $select$ :  $\text{selection } S$  **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  $\langle$ proof $\rangle$ 

```

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
   $\langle$ proof $\rangle$ 

```

```

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)
   $\langle$ proof $\rangle$ 

```

**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*  $\sigma$

**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*  $\sigma$  **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*:  $St \rightsquigarrow St' \implies I \models sr\_ext.derive$  (*grounding\_of\_state*  $St'$ )  $\iff I \models sr\_ext.derive$  (*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*:  $C \in \text{Liminf\_llist } Gs - \text{sr.Rf } (\text{Liminf\_llist } Gs)$  **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*:  $C \in \text{Liminf\_llist } Gs - \text{sr.Rf } (\text{Liminf\_llist } Gs)$  **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*:  $C \in \text{Liminf\_llist } Gs - \text{sr.Rf } (\text{Liminf\_llist } Gs)$  **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**  $\text{Liminf\_llist } Gs - \text{sr.Rf } (\text{Liminf\_llist } Gs)$   
 $\subseteq \text{grounding\_of\_cls } (\text{Q\_of\_state } (\text{Liminf\_state } Sts))$   
 <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*:  $\{\#\} \in \text{Liminf\_llist } (\text{lmap grounding\_of\_state } Sts)$   
**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**