

Formalization of Bachmair and Ganzinger’s Ordered Resolution Prover

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

June 20, 2018

Abstract

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

Contents

1	Introduction	2
2	Map Function on Two Parallel Lists	2
3	Liminf of Lazy Lists	4
4	Relational Chains over Lazy Lists	6
4.1	Chains	7
4.2	Full Chains	16
5	Clausal Logic	17
5.1	Literals	17
5.2	Clauses	20
6	Herbrand Intepretation	22
7	Abstract Substitutions	23
7.1	Library	24
7.2	Substitution Operators	24
7.3	Substitution Lemmas	27
7.3.1	Identity Substitution	27
7.3.2	Associativity of Composition	28
7.3.3	Compatibility of Substitution and Composition	28
7.3.4	“Commutativity” of Membership and Substitution	29
7.3.5	Signs and Substitutions	29
7.3.6	Substitution on Literal(s)	29
7.3.7	Substitution on Empty	29
7.3.8	Substitution on a Union	31
7.3.9	Substitution on a Singleton	31
7.3.10	Substitution on (#)	31
7.3.11	Substitution on tl	32
7.3.12	Substitution on (!)	32
7.3.13	Substitution on Various Other Functions	32
7.3.14	Renamings	33
7.3.15	Monotonicity	34
7.3.16	Size after Substitution	34
7.3.17	Variable Disjointness	34

7.3.18	Ground Expressions and Substitutions	35
7.3.19	Subsumption	37
7.3.20	Unifiers	37
7.3.21	Most General Unifier	37
7.3.22	Generalization and Subsumption	37
7.4	Most General Unifiers	40
8	Refutational Inference Systems	41
8.1	Preliminaries	41
8.2	Refutational Completeness	42
8.3	Compactness	43
9	Candidate Models for Ground Resolution	45
10	Ground Unordered Resolution Calculus	51
10.1	Inference Rule	51
10.2	Inference System	53
11	Ground Ordered Resolution Calculus with Selection	53
11.1	Inference Rule	53
11.2	Inference System	60
12	Theorem Proving Processes	60
13	The Standard Redundancy Criterion	65
14	First-Order Ordered Resolution Calculus with Selection	70
14.1	Library	71
14.2	Calculus	71
14.3	Soundness	72
14.4	Other Basic Properties	75
14.5	Inference System	75
14.6	Lifting	78
15	An Ordered Resolution Prover for First-Order Clauses	90

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. Finally, Section 4.2 generalizes the refutational completeness argument and introduces the standard redundancy criterion, which can be used in conjunction with ordered resolution. Section 4.3 lifts the result to a first-order prover, specified as a calculus. Figure 1 shows the corresponding Isabelle theory structure.

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.

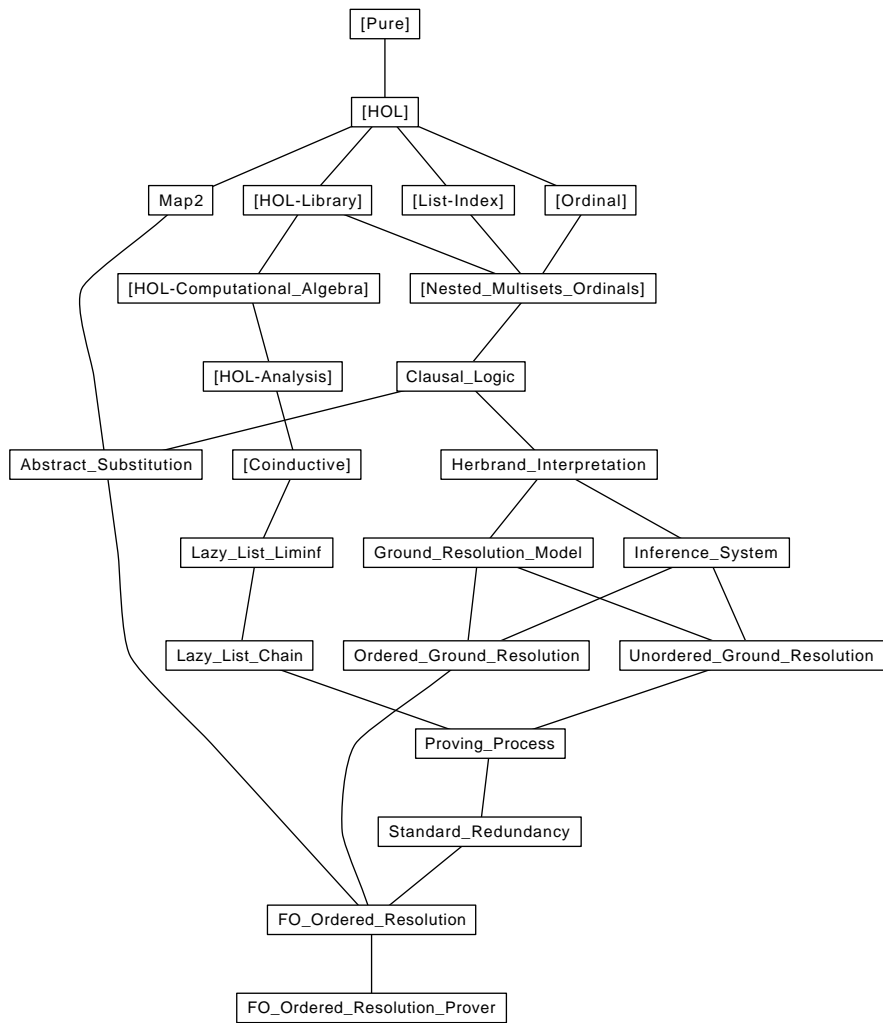


Figure 1: Theory dependency graph

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

lemma $map2_empty_iff[simp]$: $map2 f xs ys = [] \longleftrightarrow xs = [] \vee ys = []$
by (*metis Nil.is_map_conv list.exhaust list.simps(3) zip.simps(1) zip.Cons.Cons zip.Nil*)

lemma $image_map2$: $length t = length s \Longrightarrow g \text{ ` set } (map2 f t s) = set (map2 (\lambda a b. g (f a b)) t s)$
by auto

lemma $map2_tl$: $length t = length s \Longrightarrow map2 f (tl t) (tl s) = tl (map2 f t s)$
by (*metis (no_types, lifting) hd.Cons_tl list.sel(3) map2_empty_iff map_tl tl.Nil zip.Cons.Cons*)

lemma map_zip_assoc :
 $map f (zip (zip xs ys) zs) = map (\lambda(x, y, z). f ((x, y), z)) (zip xs (zip ys zs))$
by (*induct zs arbitrary: xs ys*) (*auto simp add: zip.simps(2) split: list.splits*)

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 (*rule; rule*)

fix x

assume $x \in set (map2 f s t)$

then obtain i **where** $i_p: i < length (map2 f s t) \wedge x = map2 f s t ! i$

by (*metis in_set_conv_nth*)

from i_p **have** $i < length t$

by auto

moreover from $this i_p$ **have** $x = f (s ! i) (t ! i)$

using *assms* **by auto**

ultimately show $x \in \{x. \exists i < length t. x = f (s ! i) (t ! i)\}$

using *assms* **by auto**

next

fix x

assume $x \in \{x. \exists i < length t. x = f (s ! i) (t ! i)\}$

then obtain i **where** $i_p: i < length t \wedge x = f (s ! i) (t ! i)$

by auto

then have $i < length (map2 f s t)$

using *assms* **by auto**

moreover from i_p **have** $x = map2 f s t ! i$

using *assms* **by auto**

ultimately show $x \in set (map2 f s t)$

by (*metis in_set_conv_nth*)

qed

end

3 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.

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

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

lemma $lnth_subset_Sup_llist$: $enat i < llength xs \Longrightarrow lnth xs i \subseteq Sup_llist xs$
unfolding Sup_llist_def **by auto**

lemma $Sup_llist_LNil[simp]$: $Sup_llist LNil = \{\}$
unfolding Sup_llist_def **by auto**

lemma *Sup_llist_LCons*[simp]: $Sup_llist (LCons X Xs) = X \cup Sup_llist Xs$
unfolding *Sup_llist_def*
proof (*intro subset_antisym subsetI*)
fix x
assume $x \in (\bigcup i \in \{i. enat i < llength (LCons X Xs)\}. lnth (LCons X Xs) i)$
then obtain i **where** $len: enat i < llength (LCons X Xs)$ **and** $nth: x \in lnth (LCons X Xs) i$
by *blast*
from nth **have** $x \in X \vee i > 0 \wedge x \in lnth Xs (i - 1)$
by (*metis lnth_LCons' neq0_conv*)
then have $x \in X \vee (\exists i. enat i < llength Xs \wedge x \in lnth Xs i)$
by (*metis len Suc_pred' eSuc.enat iless_Suc_eq less_irrefl llength_LCons not_less order_trans*)
then show $x \in X \cup (\bigcup i \in \{i. enat i < llength Xs\}. lnth Xs i)$
by *blast*
qed ((*auto*)[], *metis i0_lb lnth_0 zero_enat_def, metis Suc_ile_eq lnth_Suc_LCons*)

lemma *lhd_subset_Sup_llist*: $\neg lnull Xs \implies lhd Xs \subseteq Sup_llist Xs$
by (*cases Xs*) *simp_all*

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

lemma *Sup_upto_llist_mono*: $j \leq k \implies Sup_upto_llist Xs j \subseteq Sup_upto_llist Xs k$
unfolding *Sup_upto_llist_def* **by** *auto*

lemma *Sup_upto_llist_subset_Sup_llist*: $j \leq k \implies Sup_upto_llist Xs j \subseteq Sup_llist Xs$
unfolding *Sup_llist_def Sup_upto_llist_def* **by** *auto*

lemma *elem_Sup_llist_imp_Sup_upto_llist*: $x \in Sup_llist Xs \implies \exists j. x \in Sup_upto_llist Xs j$
unfolding *Sup_llist_def Sup_upto_llist_def* **by** *blast*

lemma *finite_Sup_llist_imp_Sup_upto_llist*:
assumes *finite X* **and** $X \subseteq Sup_llist Xs$
shows $\exists k. X \subseteq Sup_upto_llist Xs k$
using *assms*

proof *induct*
case (*insert x X*)
then have $x: x \in Sup_llist Xs$ **and** $X: X \subseteq Sup_llist Xs$
by *simp+*
from x **obtain** k **where** $k: x \in Sup_upto_llist Xs k$
using *elem_Sup_llist_imp_Sup_upto_llist* **by** *fast*
from X **obtain** k' **where** $k': X \subseteq Sup_upto_llist Xs k'$
using *insert.hyps*(\exists) **by** *fast*
have $insert x X \subseteq Sup_upto_llist Xs (max k k')$
using $k k'$
by (*metis insert_absorb insert_subset Sup_upto_llist_mono max.cobounded2 max commute order.trans*)
then show *?case*
by *fast*
qed *simp*

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

lemma *Liminf_llist_subset_Sup_llist*: $Liminf_llist Xs \subseteq Sup_llist Xs$
unfolding *Liminf_llist_def Sup_llist_def* **by** *fast*

lemma *Liminf_llist_LNil*[simp]: $Liminf_llist LNil = \{\}$
unfolding *Liminf_llist_def* **by** *simp*

lemma *Liminf_llist_LCons*:
 $Liminf_llist (LCons X Xs) = (if lnull Xs then X else Liminf_llist Xs)$ (**is** *?lhs = ?rhs*)

```

proof (cases lnull Xs)
  case nnull: False
  show ?thesis
  proof
    {
      fix x
      assume  $\exists i. \text{enat } i \leq \text{llength } Xs$ 
       $\wedge (\forall j. i \leq j \wedge \text{enat } j \leq \text{llength } Xs \longrightarrow x \in \text{lnth } (LCons X Xs) j)$ 
      then have  $\exists i. \text{enat } (Suc\ i) \leq \text{llength } Xs$ 
       $\wedge (\forall j. Suc\ i \leq j \wedge \text{enat } j \leq \text{llength } Xs \longrightarrow x \in \text{lnth } (LCons X Xs) j)$ 
      by (cases llength Xs,
        metis not_lnull_conv[THEN iffD1, OF nnull] Suc.le_D eSuc.enat eSuc.ile_mono
          llength_LCons not_less_eq_eq zero.enat_def zero.le,
          metis Suc.leD enat_ord_code(3))
      then have  $\exists i. \text{enat } i < \text{llength } Xs \wedge (\forall j. i \leq j \wedge \text{enat } j < \text{llength } Xs \longrightarrow x \in \text{lnth } Xs\ j)$ 
      by (metis Suc.ile_eq Suc.n_not_le.n lift_Suc_mono.le lnth_Suc_LCons nat.le_linear)
    }
  then show ?lhs  $\subseteq$  ?rhs
  by (simp add: Liminf_llist_def nnull) (rule subsetI, simp)

  {
    fix x
    assume  $\exists i. \text{enat } i < \text{llength } Xs \wedge (\forall j. i \leq j \wedge \text{enat } j < \text{llength } Xs \longrightarrow x \in \text{lnth } Xs\ j)$ 
    then obtain i where
      i:  $\text{enat } i < \text{llength } Xs$  and
      j:  $\forall j. i \leq j \wedge \text{enat } j < \text{llength } Xs \longrightarrow x \in \text{lnth } Xs\ j$ 
    by blast

    have  $\text{enat } (Suc\ i) \leq \text{llength } Xs$ 
    using i by (simp add: Suc.ile_eq)
    moreover have  $\forall j. Suc\ i \leq j \wedge \text{enat } j \leq \text{llength } Xs \longrightarrow x \in \text{lnth } (LCons X Xs) j$ 
    using Suc.ile_eq Suc.le_D j by force
    ultimately have  $\exists i. \text{enat } i \leq \text{llength } Xs \wedge (\forall j. i \leq j \wedge \text{enat } j \leq \text{llength } Xs \longrightarrow x \in \text{lnth } (LCons X Xs) j)$ 
    by blast
  }
  then show ?rhs  $\subseteq$  ?lhs
  by (simp add: Liminf_llist_def nnull) (rule subsetI, simp)
qed
qed (simp add: Liminf_llist_def enat_0_iff(1))

lemma lfinite_Liminf_llist: lfinite Xs  $\implies$  Liminf_llist Xs = (if lnull Xs then {} else llast Xs)
proof (induction rule: lfinite_induct)
  case (LCons xs)
  then obtain y ys where
    xs:  $xs = LCons\ y\ ys$ 
  by (meson not_lnull_conv)
  show ?case
  unfolding xs by (simp add: Liminf_llist_LCons LCons.IH[unfolded xs, simplified] llast_LCons)
qed (simp add: Liminf_llist_def)

lemma Liminf_llist_ltl:  $\neg$  lnull (ltl Xs)  $\implies$  Liminf_llist Xs = Liminf_llist (ltl Xs)
  by (metis Liminf_llist_LCons lhd_LCons_ltl lnull_ltlI)

```

4 Relational Chains over Lazy Lists

```

theory Lazy_List_Chain
  imports HOL-Library.BNF-Corec Lazy_List_Liminf
begin

```

A chain is a lazy lists 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 ⇒ 'a ⇒ bool) ⇒ 'a llist ⇒ bool **for** *R* :: 'a ⇒ 'a ⇒ bool **where**
chain_singleton: *chain R (LCons x LNil)*
| *chain_cons*: *chain R xs ⇒ R x (lhd xs) ⇒ chain R (LCons x xs)*

lemma

chain_LNil[simp]: $\neg \text{chain } R \text{ LNil}$ **and**
chain_not_lnull: *chain R xs ⇒ ¬ lnull xs*
by (*auto elim: chain.cases*)

lemma *chain_lappend*:

assumes

r_xs: *chain R xs* **and**

r_ys: *chain R ys* **and**

mid: *R (llast xs) (lhd ys)*

shows *chain R (lappend xs ys)*

proof (*cases lfinite xs*)

case *True*

then show *?thesis*

using *r_xs mid*

proof (*induct rule: lfinite.induct*)

case (*lfinite_LConsI xs x*)

note *fin = this(1)* **and** *ih = this(2)* **and** *r_xxs = this(3)* **and** *mid = this(4)*

show *?case*

proof (*cases xs = LNil*)

case *True*

then show *?thesis*

using *r_ys mid* **by** *simp (rule chain_cons)*

next

case *xs_nnil: False*

have *r_xs*: *chain R xs*

by (*metis chain_simps ltl_simps(2) r_xxs xs_nnil*)

have *mid'*: *R (llast xs) (lhd ys)*

by (*metis llast_LCons lnull_def mid xs_nnil*)

have *start*: *R x (lhd (lappend xs ys))*

by (*metis (no_types) chain_simps lhd_LCons lhd_lappend chain_not_lnull ltl_simps(2) r_xxs xs_nnil*)

show *?thesis*

unfolding *lappend_code(2)* **using** *ih[OF r_xs mid'] start* **by** (*rule chain_cons*)

qed

qed *simp*

qed (*simp add: r_xs lappend_inf*)

lemma *chain_length_pos*: *chain R xs ⇒ llength xs > 0*

by (*cases xs*) *simp+*

lemma *chain_ldropr*:

assumes *chain R xs* **and** *enat n < llength xs*

shows *chain R (ldropr n xs)*

using *assms*

by (*induct n arbitrary: xs, simp,*

metis chain.cases ldropr_eSuc_ltl ldropr_LNil ldropr_eq_LNil ltl_simps(2) not_less)

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 –

```

define ys where ys = ldropn j xs
have llength ys > 1
  unfolding ys_def using len
  by (metis One_nat_def funpow_swap1 ldropn_0 ldropn_def ldropn_eq_LNil ldropn_ltl not_less
      one_enat_def)
obtain y0 y1 ys' where
  ys: ys = LCons y0 (LCons y1 ys')
  unfolding ys_def by (metis Suc_ile_eq ldropn_Suc_conv_ldropn len less_imp_not_less not_less)
have chain R ys
  unfolding ys_def using Suc_ile_eq chain chain_ldropn len less_imp_le by blast
then have R y0 y1
  unfolding ys by (auto elim: chain.cases)
then show ?thesis
  using ys_def unfolding ys by (metis ldropn_Suc_conv_ldropn ldropn_eq_LConsD llist.inject)
qed

```

```

lemma infinite_chain_lnth_rel:
  assumes  $\neg$  lfinite c and chain r c
  shows r (lnth c i) (lnth c (Suc i))
  using assms chain_lnth_rel lfinite_conv_llength_enat by force

```

```

lemma lnth_rel_chain:
  assumes
     $\neg$  lnull xs and
     $\forall j. \text{enat } (j + 1) < \text{llength } xs \longrightarrow R (\text{lnth } xs \ j) (\text{lnth } xs \ (j + 1))$ 
  shows chain R xs
  using assms
proof (coinduction arbitrary: xs rule: chain.coinduct)
  case chain
  note nnul = this(1) and nth_chain = this(2)

```

```

  show ?case
  proof (cases lnull (ltl xs))
    case True
      have xs = LCons (lhd xs) LNil
        using nnul True by (simp add: llist.expand)
      then show ?thesis
        by blast
    next
      case nnul': False
      moreover have xs = LCons (lhd xs) (ltl xs)
        using nnul by simp
      moreover have
         $\forall j. \text{enat } (j + 1) < \text{llength } (\text{ltl } xs) \longrightarrow R (\text{lnth } (\text{ltl } xs) \ j) (\text{lnth } (\text{ltl } xs) \ (j + 1))$ 
        using nnul nth_chain
        by (metis Suc_eq_plus1 ldrop_eSuc_ltl ldropn_Suc_conv_ldropn ldropn_eq_LConsD lnth_ltl)
      moreover have R (lhd xs) (lhd (ltl xs))
        using nnul' nnul nth_chain[rule_format, of 0, simplified]
        by (metis ldropn_0 ldropn_Suc_conv_ldropn ldropn_eq_LConsD lhd_LCons_ltl lhd_conv_lnth
            lnth_Suc_LCons ltl_simps(2))
      ultimately show ?thesis
        by blast
  qed
qed

```

```

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)
  using assms
proof (coinduction arbitrary: xs)
  case chain
  then have  $(\exists y. xs = LCons \ y \ LNil) \vee (\exists ys \ x. xs = LCons \ x \ ys \wedge \text{chain } R \ ys \wedge R \ x \ (\text{lhd } ys))$ 
    using chain_simps[of R xs] by auto

```



```

then show ?case
proof
  assume  $\exists ys\ x. xs = LCons\ x\ ys \wedge chain\ R\ ys \wedge R\ x\ (lhd\ ys)$ 
  then have  $\exists ys\ x. lmap\ f\ xs = LCons\ x\ ys \wedge$ 
     $(\exists xs. ys = lmap\ f\ xs \wedge (\forall x\ y. R\ x\ y \longrightarrow R'\ (f\ x)\ (f\ y)) \wedge chain\ R\ xs) \wedge R'\ x\ (lhd\ ys)$ 
  using chain
  by (metis (no_types) lhd_LCons llist.distinct(1) llist.exhaust_sel llist.map_sel(1)
    lmap_eq_LNil chain_not_inull tl_lmap tl_simps(2))
  then show ?thesis
  by auto
qed auto
qed

lemma chain_mono:
  assumes  $\forall x\ y. R\ x\ y \longrightarrow R'\ x\ y$  and chain R xs
  shows chain R' xs
  using assms by (rule chain_lmap[of _ -  $\lambda x. x$ , unfolded llist.map_ident])

lemma lfinite_chain_imp_rtranclp_lhd_llast: lfinite xs  $\implies$  chain R xs  $\implies$  R** (lhd xs) (llast xs)
proof (induct rule: lfinite.induct)
  case (lfinite_LConsI xs x)
  note fin_xs = this(1) and ih = this(2) and r_x_xs = this(3)
  show ?case
  proof (cases xs = LNil)
    case xs_nnil: False
    then have r_xs: chain R xs
    using r_x_xs by (blast elim: chain.cases)
    then show ?thesis
    using ih[OF r_xs] xs_nnil r_x_xs
    by (metis chain.cases converse_rtranclp_into_rtranclp lhd_LCons llast_LCons chain_not_inull
      tl_simps(2))
  qed simp
qed simp

lemma tranclp_imp_exists_finite_chain_list:
   $R^{++}\ x\ y \implies \exists xs. xs \neq [] \wedge tl\ xs \neq [] \wedge chain\ R\ (l\list\_of\ xs) \wedge hd\ xs = x \wedge last\ xs = y$ 
proof (induct rule: tranclp.induct)
  case (r_into_trancl x y)
  note r_xy = this

  define xs where
    xs = [x, y]

  have xs  $\neq []$  and tl xs  $\neq []$  and chain R (l\list\_of xs) and hd xs = x and last xs = y
  unfolding xs_def using r_xy by (auto intro: chain.intros)
  then show ?case
  by blast

next
  case (trancl_into_trancl x y z)
  note rstar_xy = this(1) and ih = this(2) and r_yz = this(3)

  obtain xs where
    xs: xs  $\neq []$  tl xs  $\neq []$  chain R (l\list\_of xs) hd xs = x last xs = y
  using ih by blast
  define ys where
    ys = xs @ [z]

  have ys  $\neq []$  and tl ys  $\neq []$  and chain R (l\list\_of ys) and hd ys = x and last ys = z
  unfolding ys_def using xs r_yz
  by (auto simp: lappend_llist_of_llist_of[symmetric] intro: chain_singleton chain_lappend)
  then show ?case
  by blast
qed

```

inductive-cases *chain_consE*: *chain R (LCons x xs)*

inductive-cases *chain_nontrivE*: *chain R (LCons x (LCons y xs))*

primrec *prepend* **where**

prepend [] ys = ys

| *prepend (x # xs) ys = LCons x (prepend xs ys)*

lemma *prepend_butlast*:

xs ≠ [] ⇒ ¬ lnull ys ⇒ last xs = lhd ys ⇒ prepend (butlast xs) ys = prepend xs (ltl ys)

by (*induct xs*) *auto*

lemma *lnull_prepend[simp]*: *lnull (prepend xs ys) = (xs = [] ∧ lnull ys)*

by (*induct xs*) *auto*

lemma *lhd_prepend[simp]*: *lhd (prepend xs ys) = (if xs ≠ [] then hd xs else lhd ys)*

by (*induct xs*) *auto*

lemma *prepend_LNil[simp]*: *prepend xs LNil = llist_of xs*

by (*induct xs*) *auto*

lemma *lfinite_prepend[simp]*: *lfinite (prepend xs ys) ↔ lfinite ys*

by (*induct xs*) *auto*

lemma *llength_prepend[simp]*: *llength (prepend xs ys) = length xs + llength ys*

by (*induct xs*) (*auto simp: enat_0 iadd_Suc eSuc_enat[symmetric]*)

lemma *llast_prepend[simp]*: *¬ lnull ys ⇒ llast (prepend xs ys) = llast ys*

by (*induct xs*) (*auto simp: llast_LCons*)

lemma *prepend_prepend*: *prepend xs (prepend ys zs) = prepend (xs @ ys) zs*

by (*induct xs*) *auto*

lemma *chain_prepend*:

chain R (llist_of zs) ⇒ last zs = lhd xs ⇒ chain R xs ⇒ chain R (prepend zs (ltl xs))

by (*induct zs; cases xs*)

(*auto split: if_splits simp: lnull_def[symmetric] intro!: chain_cons elim!: chain_consE*)

lemma *lmap_prepend[simp]*: *lmap f (prepend xs ys) = prepend (map f xs) (lmap f ys)*

by (*induct xs*) *auto*

lemma *lset_prepend[simp]*: *lset (prepend xs ys) = set xs ∪ lset ys*

by (*induct xs*) *auto*

lemma *prepend_LCons*: *prepend xs (LCons y ys) = prepend (xs @ [y]) ys*

by (*induct xs*) *auto*

lemma *lnth_prepend*:

lnth (prepend xs ys) i = (if i < length xs then nth xs i else lnth ys (i - length xs))

by (*induct xs arbitrary: i*) (*auto simp: lnth_LCons' nth_Cons'*)

theorem *lfinite_less_induct[consumes 1, case_names less]*:

assumes *fin: lfinite xs*

and *step: ∧xs. lfinite xs ⇒ (∧zs. llength zs < llength xs ⇒ P zs) ⇒ P xs*

shows *P xs*

using *fin* **proof** (*induct the_enat (llength xs) arbitrary: xs rule: less_induct*)

case (*less xs*)

show *?case*

using *less(2)* **by** (*intro step[OF less(2)] less(1)*)

(*auto dest!: lfinite_llength_enat simp: eSuc_enat elim!: less_enatE llength_eq_enat_lfiniteD*)

qed

theorem *lfinite_prepend_induct[consumes 1, case_names LNil prepend]*:

```

assumes lfinite xs
  and LNil: P LNil
  and prepend:  $\bigwedge xs. lfinite\ xs \implies (\bigwedge zs. (\exists ys. xs = prepend\ ys\ zs \wedge ys \neq [])) \implies P\ zs \implies P\ xs$ 
shows P xs
using assms(1) proof (induct xs rule: lfinite_less_induct)
  case (less xs)
  from less(1) show ?case
  by (cases xs)
  (force simp: LNil neq_Nil_conv dest: lfinite_llength_enat intro!: prepend[of LCons - ] intro: less)+
qed

```

```

coinductive emb :: 'a llist  $\Rightarrow$  'a llist  $\Rightarrow$  bool where
  emb LNil xs
| emb xs ys  $\implies emb (LCons x xs) (prepend zs (LCons x ys))$ 

```

```

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 emb_prepend_coinduct[rotated, case_names emb]:
  assumes ( $\bigwedge x1\ x2. X\ x1\ x2 \implies$ 
    ( $\exists xs. x1 = LNil \wedge x2 = xs$ )
     $\vee (\exists xs\ ys\ x\ zs. x1 = LCons\ x\ xs \wedge x2 = prepend\ zs\ (LCons\ x\ ys)$ 
       $\wedge (prepend\_cong1\ (X\ xs)\ ys \vee emb\ xs\ ys)))$  (is  $\bigwedge x1\ x2. X\ x1\ x2 \implies ?bisim\ x1\ x2$ )
  shows  $X\ x1\ x2 \implies emb\ x1\ x2$ 
proof (erule emb.coinduct[OF prepend_cong1_base])
  fix xs zs
  assume prepend_cong1 (X xs) zs
  then show ?bisim xs zs
  by (induct zs rule: prepend_cong1.induct) (erule assms, force simp: prepend_prepend)
qed

```

```

context
begin

```

```

private coinductive chain' for R where
  chain' R (LCons x LNil)
| chain R (llist_of zs)  $\implies zs \neq [] \implies tl\ zs \neq [] \implies \neg lnull\ xs \implies last\ zs = lhd\ xs \implies$ 
  ys = ltl\ xs  $\implies chain'\ R\ xs \implies chain'\ R\ (prepend\ zs\ ys)$ 

```

```

private lemma chain_imp_chain': chain R xs  $\implies chain'\ R\ xs$ 
proof (coinduction arbitrary: xs rule: chain'.coinduct)
  case chain'
  then show ?case
  proof (cases rule: chain.cases)
  case (chain_cons zs z)
  then show ?thesis
  by (intro disjI2 (force intro: chain.intros exI[of - [z, lhd zs]] exI[of - zs]
    elim: chain.cases)
  qed simp
qed

```

```

private inductive-cases chain'_LConsE: chain' R (LCons x xs)

```

```

private lemma chain'_stepD1:
  assumes chain' R (LCons x (LCons y xs))
  shows chain' R (LCons y xs)
proof (cases xs)
  case [simp]: (LCons z zs)
  with assms show ?thesis
  proof (cases rule: chain'.cases)
  case (2 as ys xs)
  then show ?thesis

```

```

proof (cases tl (tl as))
  case Nil
  with 2 show ?thesis by (auto simp: neq_Nil_conv)
next
  case (Cons b bs)
  with 2 have chain' R (prepend (y # b # bs) xs)
  by (intro chain'.intros)
  (auto simp: chain_cons not_lnull_conv neq_Nil_conv elim: chain_nontrivE)
  with 2 Cons show ?thesis
  by (auto simp: neq_Nil_conv)
qed
qed
qed (simp only: chain'.intros(1))

private lemma chain'_stepD2: chain' R (LCons x (LCons y xs))  $\implies$  R x y
  by (erule chain'.cases) (auto simp: neq_Nil_conv elim!: chain_nontrivE split: if_splits)

private lemma chain'_imp_chain: chain' R xs  $\implies$  chain R xs
proof (coinduction arbitrary: xs rule: chain.coinduct)
  case chain
  then show ?case
  proof (cases rule: chain'.cases)
  case (2 ys zs xs)
  then show ?thesis
  proof (cases ltl zs)
  case LNil
  with chain 2 show ?thesis
  by (auto 0 4 simp: neq_Nil_conv not_lnull_conv elim: chain'_stepD1 chain'_stepD2)
  next
  case (LCons b bs)
  with chain 2 show ?thesis
  unfolding neq_Nil_conv not_lnull_conv
  by (elim exE) (auto elim: chain'_stepD1 chain_nontrivE)
  qed
qed simp
qed

private lemma chain_chain': chain = chain'
  unfolding fun_eq_iff by (metis chain_imp_chain' chain'_imp_chain)

lemma chain_prepend_coinduct[case_names chain]:
   $X x \implies (\bigwedge x. X x \implies$ 
   $(\exists z. x = LCons z LNil) \vee$ 
   $(\exists xs zs. x = prepend zs (ltl xs) \wedge zs \neq [] \wedge tl zs \neq [] \wedge \neg lnull xs \wedge last zs = lhd xs \wedge$ 
   $(X xs \vee chain R xs) \wedge chain R (llist_of zs))) \implies chain R x$ 
  by (subst chain_chain', erule chain'.coinduct) (auto simp: chain_chain')

end

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

private definition pick where
  pick x y = (SOME xs. xs  $\neq$  []  $\wedge$  tl xs  $\neq$  []  $\wedge$  chain R (llist_of xs)  $\wedge$  hd xs = x  $\wedge$  last xs = y)

private lemma pick[simp]:
  assumes R++ x y
  shows pick x y  $\neq$  []  $\wedge$  tl (pick x y)  $\neq$  []  $\wedge$  chain R (llist_of (pick x y))
  hd (pick x y) = x last (pick x y) = y
  unfolding pick_def using tranclp_imp_exists_finite_chain_list[THEN someI_ex, OF assms] by auto

private lemma butlast_pick[simp]: R++ x y  $\implies$  butlast (pick x y)  $\neq$  []

```

by (cases pick x y; cases tl (pick x y)) (auto dest: pick(2))

private friend-of-corec prepend where
 prepend xs ys = (case xs of [] =>
 (case ys of LNil => LNil | LCons x xs => LCons x xs) | x # xs' => LCons x (prepend xs' ys))
 by (simp split: list.splits llist.splits) transfer_prover

private corec wit where
 wit xs = (case xs of LCons x (LCons y xs) =>
 let zs = pick x y in LCons (hd zs) (prepend (butlast (tl zs)) (wit (LCons y xs)))) | _ => xs)

private lemma
 wit.LNil[simp]: wit LNil = LNil **and**
 wit.lsingleton[simp]: wit (LCons x LNil) = LCons x LNil **and**
 wit.LCons2: wit (LCons x (LCons y xs)) =
 (let zs = pick x y in LCons (hd zs) (prepend (butlast (tl zs)) (wit (LCons y xs))))
 by (subst wit.code; auto)+

private lemma wit_LCons: wit (LCons x xs) = (case xs of LNil => LCons x LNil | LCons y xs =>
 (let zs = pick x y in LCons (hd zs) (prepend (butlast (tl zs)) (wit (LCons y xs))))
 by (subst wit.code; auto split: llist.splits)+

private lemma lnull_wit[simp]: lnull (wit xs) <=> lnull xs
 by (subst wit.code) (auto split: list.splits simp: Let_def)

private lemma lhd_wit[simp]: chain R⁺⁺ xs ==> lhd (wit xs) = lhd xs
 by (erule chain.cases; subst wit.code) (auto split: llist.splits simp: Let_def)

private lemma butlast_alt: butlast xs = (if tl xs = [] then [] else hd xs # butlast (tl xs))
 by (cases xs) auto

private lemma wit_alt:
 chain R⁺⁺ xs ==> wit xs = (case xs of LCons x (LCons y xs) =>
 prepend (pick x y) (ltl (wit (LCons y xs))) | _ => xs)
 by (auto split: list.splits simp: prepend_butlast[symmetric] wit_LCons2 Let_def
 prepend_simps(2)[symmetric] butlast_alt[of pick - _]
 simp del: prepend_simps elim!: chain_nontrivE)

private lemma wit_alt2:
 chain R⁺⁺ xs ==> wit xs = (case xs of LCons x (LCons y xs) =>
 prepend (butlast (pick x y)) (wit (LCons y xs)) | _ => xs)
 by (auto split: list.splits simp: wit_LCons2 Let_def
 prepend_simps(2)[symmetric] butlast_alt[of pick - _]
 simp del: prepend_simps elim!: chain_nontrivE)

private lemma LNil_eq_iff_lnull: LNil = xs <=> lnull xs
 by (cases xs) auto

private lemma lfinite_wit[simp]:
 assumes chain R⁺⁺ xs
 shows lfinite (wit xs) <=> lfinite xs
proof
 assume lfinite (wit xs)
 from this assms show lfinite xs
proof (induct wit xs arbitrary: xs rule: lfinite_prepend_induct)
 case (prepend zs)
 then show ?case
proof (cases zs)
 case [simp]: (LCons x xs)
 then show ?thesis
proof (cases xs)
 case [simp]: LCons
 with prepend show ?thesis

```

      by (subst (asm) (2) wit_alt2) (force split: llist.splits elim!: chain_nontrivE)+
    qed simp
  qed simp
  qed (simp add: LNil_eq_iff_lnull)
next
  assume lfinite xs
  then show lfinite (wit xs)
  proof (induct xs rule: lfinite.induct)
    case (lfinite_LConsI xs x)
    then show ?case
    by (cases xs) (auto simp: wit_LCons Let_def)
  qed simp
qed

```

```

private lemma llast_wit[simp]:
  assumes chain R++ xs
  shows llast (wit xs) = llast xs
proof (cases lfinite xs)
  case True
  from this assms show ?thesis
  proof (induct rule: lfinite.induct)
    case (lfinite_LConsI xs x)
    then show ?case
    by (cases xs) (auto simp: wit_LCons2 llast_LCons elim: chain_nontrivE)
  qed auto
qed (auto simp: llast_linfiniteness assms)

```

```

lemma emb_wit[simp]: chain R++ xs  $\implies$  emb xs (wit xs)
proof (coinduction arbitrary: xs rule: emb_prepend_coinduct)
  case (emb xs)
  then show ?case
  proof (cases rule: chain.cases)
    case (chain_cons zs z)
    then show ?thesis
    by (subst (2) wit.code)
      (auto split: llist.splits intro!: exI[of _ []] exI[of _ - :: - llist]
        prepend_cong1_prepend[OF prepend_cong1_base])
  qed (auto intro!: exI[of _ LNil] exI[of _ []] emb.intros)
qed

```

```

lemma chain_tranclp_imp_exists_chain:
  chain R++ xs  $\implies$ 
   $\exists$  ys. chain R ys  $\wedge$  emb xs ys  $\wedge$  (lfinite ys  $\iff$  lfinite xs)  $\wedge$  lhd ys = lhd xs
   $\wedge$  llast ys = llast xs
proof (intro exI[of _ wit xs] conjI, coinduction arbitrary: xs rule: chain_prepend_coinduct)
  case chain
  then show ?case
  by (subst (1 2) wit_alt; assumption?) (erule chain.cases; force split: llist.splits)
qed auto

```

```

inductive-cases emb_LConsE: emb (LCons z zs) ys
inductive-cases emb_LNil2E: emb xs LNil

```

```

lemma emb_lset_mono[rotated]:  $x \in$  lset xs  $\implies$  emb xs ys  $\implies$   $x \in$  lset ys
  by (induct x xs arbitrary: ys rule: llist.set_induct) (auto elim!: emb_LConsE)

```

```

lemma emb_Ball_lset_antimono:
  assumes emb Xs Ys
  shows  $\forall Y \in$  lset Ys.  $x \in Y \implies \forall X \in$  lset Xs.  $x \in X$ 
  using emb_lset_mono[OF assms] by blast

```

```

lemma emb_lfinite_antimono[rotated]: lfinite ys  $\implies$  emb xs ys  $\implies$  lfinite xs
  by (induct ys arbitrary: xs rule: lfinite_prepend_induct)

```

(force elim!: emb_LNil2E simp: LNil_eq_iff_lnull prepend_LCons elim: emb.cases)+

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 $\forall j \geq i. x \in \text{lnth } Xs\ j$

using *assms* **proof** (*induct i arbitrary: Xs Ys rule: less_induct*)

case (*less i*)

then show *?case*

proof (*cases i*)

case 0

then show *?thesis*

using *emb_Ball_lset_antimono*[*OF less(2), of x*] *less(5)*

unfolding *Ball_def in_lset_conv_lnth simp_thms*

not_lfinite_llength[*OF less(3)*] *not_lfinite_llength*[*OF less(4)*] *enat_ord_code subset_eq*

by *blast*

next

case [*simp*]: (*Suc nat*)

from *less(2,3)* **obtain** *xs as b bs* **where**

[*simp*]: $Xs = LCons\ b\ xs\ Ys = \text{prepend}\ as\ (LCons\ b\ bs)$ **and** *emb xs bs*

by (*auto elim: emb.cases*)

have *IH*: $\forall k \geq j. x \in \text{lnth } xs\ k$ **if** $\forall k \geq j. x \in \text{lnth } bs\ k$ $j < i$ **for** *j*

using *that less(1)*[*OF _ (emb xs bs)*] *less(3,4)* **by** *auto*

from *less(5)* **have** $\forall k \geq i - \text{length } as - 1. x \in \text{lnth } xs\ k$

by (*intro IH allI*)

(*drule spec*[*of _ - + length as + 1*], *auto simp: lnth_prepend lnth_LCons'*)

then show *?thesis*

by (*auto simp: lnth_LCons'*)

qed

qed

lemma *emb_Liminf_llist_infinite*:

assumes *emb Xs Ys* **and** \neg *lfinite Xs*

shows *Liminf_llist Ys* \subseteq *Liminf_llist Xs*

proof -

from *assms* **have** \neg *lfinite Ys*

using *emb_lfinite_antimono* **by** *blast*

with *assms* **show** *?thesis*

unfolding *Liminf_llist_def* **by** (*auto simp: not_lfinite_llength dest: emb_Liminf_llist_mono_aux*)

qed

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

proof (*coinduction arbitrary: xs ys rule: emb.coinduct*)

case *emb*

show *?case*

proof (*cases xs*)

case *xs*: (*LCons x xs'*)

obtain *ysa0* **and** *zs0* **where**

ys: $ys = \text{prepend } zs0\ (LCons\ x\ ysa0)$ **and**

emb': *emb xs' ysa0*

using *emb_LConsE*[*OF emb[unfolded xs]*] **by** *metis*

let *?xa* = *f x*

let *?xsa* = *lmap f xs'*

let *?zs* = *map f zs0*

let *?ysa* = *lmap f ysa0*

have *lmap f xs* = *LCons ?xa ?xsa*

unfolding *xs* **by** *simp*

moreover **have** *lmap f ys* = *prepend ?zs (LCons ?xa ?ysa)*

unfolding *ys* **by** *simp*

moreover **have** $\exists xsa\ ysa. ?xsa = \text{lmap } f\ xsa \wedge ?ysa = \text{lmap } f\ ysa \wedge \text{emb } xsa\ ysa$

using *emb'* **by** *blast*

ultimately show *?thesis*
 by *blast*
 qed *simp*
 qed

end

lemma *chain_inf_llist_if_infinite_chain_function*:
 assumes $\forall i. r (f (Suc i)) (f i)$
 shows $\neg lfinite (inf_llist f) \wedge chain\ r^{-1-1} (inf_llist f)$
 using *assms* by (*simp add: lnth_rel_chain*)

lemma *infinite_chain_function_iff_infinite_chain_llist*:
 $(\exists f. \forall i. r (f (Suc i)) (f i)) \longleftrightarrow (\exists c. \neg lfinite\ c \wedge chain\ r^{-1-1}\ c)$
 using *chain_inf_llist_if_infinite_chain_function infinite_chain_lnth_rel* by *blast*

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

have $wfP\ r \longleftrightarrow wf\ \{(x, y). r\ x\ y\}$
 unfolding *wfP_def* by *auto*
 also have $\dots \longleftrightarrow (\nexists f. \forall i. (f (Suc i), f i) \in \{(x, y). r\ x\ y\})$
 using *wf_iff_no_infinite_down_chain* by *blast*
 also have $\dots \longleftrightarrow (\nexists f. \forall i. r (f (Suc i)) (f i))$
 by *auto*
 also have $\dots \longleftrightarrow (\nexists c. \neg lfinite\ c \wedge chain\ r^{-1-1}\ c)$
 using *infinite_chain_function_iff_infinite_chain_llist* by *blast*
 finally show *?thesis*
 by *auto*
 qed

4.2 Full Chains

coinductive *full_chain* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow bool$ for $R :: 'a \Rightarrow 'a \Rightarrow bool$ where
full_chain_singleton: $(\forall y. \neg R\ x\ y) \Longrightarrow full_chain\ R\ (LCons\ x\ LNil)$
| *full_chain_cons*: $full_chain\ R\ xs \Longrightarrow R\ x\ (lhd\ xs) \Longrightarrow full_chain\ R\ (LCons\ x\ xs)$

lemma
full_chain_LNil[simp]: $\neg full_chain\ R\ LNil$ and
full_chain_not_lnull: $full_chain\ R\ xs \Longrightarrow \neg lnull\ xs$
 by (*auto elim: full_chain.cases*)

lemma *full_chain_ldroprn*:
 assumes *full*: $full_chain\ R\ xs$ and *enat* $n < llength\ xs$
 shows $full_chain\ R\ (ldroprn\ n\ xs)$
 using *assms*
 by (*induct n arbitrary: xs, simp,*
metis full_chain.cases ldroprn_eSuc_ltl ldroprn_LNil ldroprn_eq_LNil ltl_simps(2) not_less)

lemma *full_chain_iff_chain*:
 $full_chain\ R\ xs \longleftrightarrow chain\ R\ xs \wedge (lfinite\ xs \longrightarrow (\forall y. \neg R\ (llast\ xs)\ y))$
 proof (*intro iffI conjI impI allI; (elim conjE)?*)
 assume *full*: $full_chain\ R\ xs$

show *chain*: $chain\ R\ xs$
 using *full* by (*coinduction arbitrary: xs*) (*auto elim: full_chain.cases*)

{
 fix *y*
 assume *lfinite xs*
 then obtain *n* where
suc_n: $Suc\ n = llength\ xs$
 by (*metis chain chain_length_pos lessE less_enatE lfinite_conv_llength_enat*)
 have $full_chain\ R\ (ldroprn\ n\ xs)$


```

    by (rule full_chain_ldropn[OF full]) (use suc.n Suc_ile.eq in force)
  moreover have ldropn n xs = LCons (llast xs) LNil
    using suc.n by (metis enat_le_plus_same(2) enat_ord_simps(2) gen_llength_def
      ldropn_Suc_conv_ldropn ldropn_all lessI llast_ldropn llast_singleton llength_code)
  ultimately show  $\neg R$  (llast xs) y
    by (auto elim: full_chain.cases)
}
next
assume
  chain R xs and
  lfinite xs  $\longrightarrow$  ( $\forall y. \neg R$  (llast xs) y)
then show full_chain R xs
  by (coinduction arbitrary: xs) (erule chain.cases, simp, metis lfinite_LConsI llast_LCons)
qed

lemma full_chain_imp_chain: full_chain R xs  $\implies$  chain R xs
  using full_chain_iff_chain by blast

lemma full_chain_length_pos: full_chain R xs  $\implies$  llength xs  $> 0$ 
  by (fact chain_length_pos[OF full_chain_imp_chain])

lemma full_chain_nth_rel:
  full_chain R xs  $\implies$  enat (Suc j)  $<$  llength xs  $\implies$  R (lnth xs j) (lnth xs (Suc j))
  by (fact chain_nth_rel[OF full_chain_imp_chain])

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. full_chain R ys  $\wedge$  emb xs ys  $\wedge$  lfinite ys = lfinite xs  $\wedge$  lhd ys = lhd xs
     $\wedge$  llast ys = llast xs
proof -
  obtain ys where ys:
    chain R ys emb xs ys lfinite ys = lfinite xs lhd ys = lhd xs llast ys = llast xs
  using full_chain_imp_chain[OF full] chain_tranclp_imp_exists_chain by blast
  have full_chain R ys
    using ys(1,3,5) full unfolding full_chain_iff_chain by auto
  then show ?thesis
    using ys(2-5) by auto
qed

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]*: $Pos\ (atm_of\ L) = L \longleftrightarrow is_pos\ L$
by (*cases L*) *simp+*

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

lemma *set_literal_atm_of*: $set_literal\ L = \{atm_of\ L\}$
by (*cases L*) *simp+*

lemma *ex_lit_cases*: $(\exists L. P\ L) \longleftrightarrow (\exists A. P\ (Pos\ A) \vee P\ (Neg\ A))$
by (*metis literal.exhaust*)

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 ..

end

lemma
uminus_Pos[simp]: $\neg\ Pos\ A = Neg\ A$ **and**
uminus_Neg[simp]: $\neg\ Neg\ A = Pos\ A$
unfolding *uminus_literal_def* **by** *simp_all*

lemma *atm_of_uminus[simp]*: $atm_of\ (\neg L) = atm_of\ L$
by (*case_tac L, auto*)

lemma *uminus_of_uminus_id[simp]*: $\neg\ (\neg\ (x :: 'v\ literal)) = x$
by (*simp add: uminus_literal_def*)

lemma *uminus_not_id[simp]*: $x \neq \neg\ (x :: 'v\ literal)$
by (*case_tac x, auto*)

lemma *uminus_not_id'[simp]*: $\neg\ x \neq (x :: 'v\ literal)$
by (*case_tac x, auto*)

lemma *uminus_eq_inj[iff]*: $\neg\ (a :: 'v\ literal) = \neg\ b \longleftrightarrow a = b$
by (*case_tac a; case_tac b*) *auto+*

lemma *uminus_lit_swap*: $(a :: 'a\ literal) = \neg\ b \longleftrightarrow \neg\ a = b$
by *auto*

lemma *is_pos_neg_not_is_pos*: $is_pos\ (\neg\ L) \longleftrightarrow \neg\ is_pos\ L$
by (*cases L*) *auto*

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

apply *intro_classes*

unfolding *less_literal_def less_eq_literal_def* **by** (*auto intro: order_trans simp: less_le_not_le*)

```

end

instantiation literal :: (order) order
begin

instance
  by intro_classes (auto simp: less_eq_literal_def intro: literal.expand)

end

lemma pos_less_neg[simp]: Pos A < Neg A
  unfolding less_literal_def by simp

lemma pos_less_pos_iff[simp]: Pos A < Pos B  $\longleftrightarrow$  A < B
  unfolding less_literal_def by simp

lemma pos_less_neg_iff[simp]: Pos A < Neg B  $\longleftrightarrow$  A  $\leq$  B
  unfolding less_literal_def by (auto simp: less_le_not_le)

lemma neg_less_pos_iff[simp]: Neg A < Pos B  $\longleftrightarrow$  A < B
  unfolding less_literal_def by simp

lemma neg_less_neg_iff[simp]: Neg A < Neg B  $\longleftrightarrow$  A < B
  unfolding less_literal_def by simp

lemma pos_le_neg[simp]: Pos A  $\leq$  Neg A
  unfolding less_eq_literal_def by simp

lemma pos_le_pos_iff[simp]: Pos A  $\leq$  Pos B  $\longleftrightarrow$  A  $\leq$  B
  unfolding less_eq_literal_def by (auto simp: less_le_not_le)

lemma pos_le_neg_iff[simp]: Pos A  $\leq$  Neg B  $\longleftrightarrow$  A < B
  unfolding less_eq_literal_def by (auto simp: less_imp_le)

lemma neg_le_pos_iff[simp]: Neg A  $\leq$  Pos B  $\longleftrightarrow$  A < B
  unfolding less_eq_literal_def by simp

lemma neg_le_neg_iff[simp]: Neg A  $\leq$  Neg B  $\longleftrightarrow$  A  $\leq$  B
  unfolding less_eq_literal_def by (auto simp: less_imp_le)

lemma leq_imp_less_eq_atm_of: L  $\leq$  M  $\implies$  atm_of L  $\leq$  atm_of M
  unfolding less_eq_literal_def using less_imp_le by blast

instantiation literal :: (linorder) linorder
begin

instance
  apply intro_classes
  unfolding less_eq_literal_def less_literal_def by auto

end

instantiation literal :: (wellorder) wellorder
begin

instance
proof intro_classes
  fix P :: 'a literal  $\Rightarrow$  bool and L :: 'a literal
  assume ih:  $\bigwedge L. (\bigwedge M. M < L \implies P M) \implies P L$ 
  have  $\bigwedge x. (\bigwedge y. y < x \implies P (Pos y) \wedge P (Neg y)) \implies P (Pos x) \wedge P (Neg x)$ 
    by (rule conjI[OF ih ih])
    (auto simp: less_literal_def atm_of_def split: literal.splits intro: ih)
  then have  $\bigwedge A. P (Pos A) \wedge P (Neg A)$ 

```

```

  by (rule less_induct) blast
then show P L
  by (cases L) simp+
qed

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 $Max_in_lits: C \neq \{\#\} \Longrightarrow Max_mset C \in\# C$
by *simp*

lemma $Max_atm_of_set_mset_commute: C \neq \{\#\} \Longrightarrow Max (atm_of ' set_mset C) = atm_of (Max_mset C)$
by (rule *mono_Max_commute[symmetric]*) (auto *simp: mono_def less_eq_literal_def*)

lemma $Max_pos_neg_less_multiset:$

assumes $max: Max_mset C = Pos A$ **and** $neg: Neg A \in\# D$

shows $C < D$

proof –

have $Max_mset C < Neg A$

using max **by** *simp*

then show *?thesis*

using neg **by** (*metis (no_types) Max_less_iff empty_iff ex_gt_imp_less_multiset finite_set_mset*)

qed

lemma $pos_Max_imp_neg_notin: Max_mset C = Pos A \Longrightarrow Neg A \notin\# C$
using $Max_pos_neg_less_multiset$ **by** *blast*

lemma $less_eq_Max_lit: C \neq \{\#\} \Longrightarrow C \leq D \Longrightarrow Max_mset C \leq Max_mset D$

proof (*unfold less_eq_multiset_HO*)

assume

$ne: C \neq \{\#\}$ **and**

$ex_gt: \forall x. count D x < count C x \longrightarrow (\exists y > x. count C y < count D y)$

from ne **have** $Max_mset C \in\# C$

by (*fast intro: Max_in_lits*)

then have $\exists l. l \in\# D \wedge \neg l < Max_mset C$

using ex_gt **by** (*metis count_greater_zero_iff count_inI less_not_sym*)

then have $\neg Max_mset D < Max_mset C$

by (*metis Max.coboundedI[OF finite_set_mset] le_less_trans*)

then show *?thesis*

by *simp*

qed

definition $atms_of :: 'a$ clause $\Rightarrow 'a$ set **where**

$atms_of C = atm_of ' set_mset C$

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

unfolding $atms_of_def$ **by** *simp*

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

unfolding $atms_of_def$ **by** *auto*

lemma *atms_of_add_mset*[simp]: $\text{atms_of } (\text{add_mset } a \ A) = \text{insert } (\text{atm_of } a) (\text{atms_of } A)$
unfolding *atms_of_def* **by** *auto*

lemma *atms_of_union_mset*[simp]: $\text{atms_of } (A \cup\# B) = \text{atms_of } A \cup \text{atms_of } B$
unfolding *atms_of_def* **by** *auto*

lemma *finite_atms_of*[iff]: $\text{finite } (\text{atms_of } C)$
by (*simp add: atms_of_def*)

lemma *atm_of_lit_in_atms_of*: $L \in\# C \implies \text{atm_of } L \in \text{atms_of } C$
by (*simp add: atms_of_def*)

lemma *atms_of_plus*[simp]: $\text{atms_of } (C + D) = \text{atms_of } C \cup \text{atms_of } D$
unfolding *atms_of_def* **by** *auto*

lemma *in_atms_of_minusD*: $x \in \text{atms_of } (A - B) \implies x \in \text{atms_of } A$
by (*auto simp: atms_of_def dest: in_diffD*)

lemma *pos_lit_in_atms_of*: $\text{Pos } A \in\# C \implies A \in \text{atms_of } C$
unfolding *atms_of_def* **by** *force*

lemma *neg_lit_in_atms_of*: $\text{Neg } A \in\# C \implies A \in \text{atms_of } C$
unfolding *atms_of_def* **by** *force*

lemma *atm_imp_pos_or_neg_lit*: $A \in \text{atms_of } C \implies \text{Pos } A \in\# C \vee \text{Neg } A \in\# C$
unfolding *atms_of_def image_def mem_Collect_eq* **by** (*metis Neg_atm_of_iff Pos_atm_of_iff*)

lemma *atm_iff_pos_or_neg_lit*: $A \in \text{atms_of } L \longleftrightarrow \text{Pos } A \in\# L \vee \text{Neg } A \in\# L$
by (*auto intro: pos_lit_in_atms_of neg_lit_in_atms_of dest: atm_imp_pos_or_neg_lit*)

lemma *atm_of_eq_atm_of*: $\text{atm_of } L = \text{atm_of } L' \longleftrightarrow (L = L' \vee L = -L')$
by (*cases L; cases L'*) *auto*

lemma *atm_of_in_atm_of_set_iff_in_set_or_uminus_in_set*: $\text{atm_of } L \in \text{atm_of } I \longleftrightarrow (L \in I \vee -L \in I)$
by (*auto intro: rev_image_eqI simp: atm_of_eq_atm_of*)

lemma *lits_subseteq_imp_atms_subseteq*: $\text{set_mset } C \subseteq \text{set_mset } D \implies \text{atms_of } C \subseteq \text{atms_of } D$
unfolding *atms_of_def* **by** *blast*

lemma *atms_empty_iff_empty*[iff]: $\text{atms_of } C = \{\} \longleftrightarrow C = \{\#\}$
unfolding *atms_of_def image_def Collect_empty_eq* **by** *auto*

lemma
atms_of_poss[simp]: $\text{atms_of } (\text{poss } AA) = \text{set_mset } AA$ **and**
atms_of_negs[simp]: $\text{atms_of } (\text{negs } AA) = \text{set_mset } AA$
unfolding *atms_of_def image_def* **by** *auto*

lemma *less_eq_Max_atms_of*: $C \neq \{\#\} \implies C \leq D \implies \text{Max } (\text{atms_of } C) \leq \text{Max } (\text{atms_of } D)$
unfolding *atms_of_def*
by (*metis Max_atm_of_set_mset_commute leq_imp_less_eq_atm_of less_eq_Max_lit less_eq_multiset_empty_right*)

lemma *le_multiset_Max_in_imp_Max*:
 $\text{Max } (\text{atms_of } D) = A \implies C \leq D \implies A \in \text{atms_of } C \implies \text{Max } (\text{atms_of } C) = A$
by (*metis Max_coboundedI[OF finite_atms_of] atms_of_def empty_iff eq_iff image_subsetI less_eq_Max_atms_of set_mset_empty subset_Cmpl_self_eq*)

lemma *atm_of_Max_lit*[simp]: $C \neq \{\#\} \implies \text{atm_of } (\text{Max_mset } C) = \text{Max } (\text{atms_of } C)$
unfolding *atms_of_def Max_atm_of_set_mset_commute* **..**

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

by (metis Neg_atm_of_iff Pos_atm_of_iff atm_of_Max_lit)

lemma *atms_less_imp_lit_less_pos*: $(\bigwedge B. B \in \text{atms_of } C \implies B < A) \implies L \in\# C \implies L < \text{Pos } A$
unfolding *atms_of_def less_literal_def* **by** *force*

lemma *atms_less_eq_imp_lit_less_eq_neg*: $(\bigwedge B. B \in \text{atms_of } C \implies B \leq A) \implies L \in\# C \implies L \leq \text{Neg } A$
unfolding *less_eq_literal_def* **by** (*simp add: atm_of_lit_in_atms_of*)

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** \models_l 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*]:
 $I \models_l \text{Pos } A \iff A \in I$
 $I \models_l \text{Neg } A \iff A \notin I$
unfolding *true_lit_def* **by** *auto*

lemma *true_lit_iff*[*iff*]: $I \models_l L \iff (\exists A. L = \text{Pos } A \wedge A \in I \vee L = \text{Neg } A \wedge A \notin I)$
by (*cases L*) *simp*+

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

lemma *true_cls_empty*[*iff*]: $\neg I \models \{\#\}$
unfolding *true_cls_def* **by** *simp*

lemma *true_cls_singleton*[*iff*]: $I \models \{\#L\} \iff I \models_l L$
unfolding *true_cls_def* **by** *simp*

lemma *true_cls_add_mset*[*iff*]: $I \models \text{add_mset } C D \iff I \models_l C \vee I \models D$
unfolding *true_cls_def* **by** *auto*

lemma *true_cls_union*[*iff*]: $I \models C + D \iff I \models C \vee I \models D$
unfolding *true_cls_def* **by** *auto*

lemma *true_cls_mono*: $\text{set_mset } C \subseteq \text{set_mset } D \implies I \models C \implies I \models D$
unfolding *true_cls_def subset_eq* **by** *metis*

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: $I \models C \implies \neg J \models C \implies \exists A \in J. \text{Neg } A \in\# C$
using *assms* **unfolding** *subset_iff true_cls_def* **by** (*metis literal.collapse true_lit_simps*)+

lemma *true_cls_replicate_mset*[*iff*]: $I \models \text{replicate_mset } n L \iff n \neq 0 \wedge I \models_l L$
by (*simp add: true_cls_def*)

lemma *pos_literal_in_imp_true_cls*[*intro*]: $\text{Pos } A \in\# C \implies A \in I \implies I \models C$
using *true_cls_def* **by** *blast*

lemma *neg_literal_notin_imp_true_cls*[intro]: $Neg A \in\# C \implies A \notin I \implies I \models C$
using *true_cls_def* **by** *blast*

lemma *pos_neg_in_imp_true*: $Pos A \in\# C \implies Neg A \in\# C \implies I \models C$
using *true_cls_def* **by** *blast*

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

lemma *true_cls_empty*[iff]: $I \models_s \{\}$
by (*simp add: true_cls_def*)

lemma *true_cls_singleton*[iff]: $I \models_s \{C\} \longleftrightarrow I \models C$
unfolding *true_cls_def* **by** *blast*

lemma *true_cls_insert*[iff]: $I \models_s insert C DD \longleftrightarrow I \models C \wedge I \models_s DD$
unfolding *true_cls_def* **by** *blast*

lemma *true_cls_union*[iff]: $I \models_s CC \cup DD \longleftrightarrow I \models_s CC \wedge I \models_s DD$
unfolding *true_cls_def* **by** *blast*

lemma *true_cls_mono*: $DD \subseteq CC \implies I \models_s CC \implies I \models_s DD$
by (*simp add: set_mp true_cls_def*)

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

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

lemma *true_cls_mset_empty*[iff]: $I \models_m \{\#\}$
unfolding *true_cls_mset_def* **by** *auto*

lemma *true_cls_mset_singleton*[iff]: $I \models_m \{\#C\# \} \longleftrightarrow I \models C$
by (*simp add: true_cls_mset_def*)

lemma *true_cls_mset_union*[iff]: $I \models_m CC + DD \longleftrightarrow I \models_m CC \wedge I \models_m DD$
unfolding *true_cls_mset_def* **by** *auto*

lemma *true_cls_mset_add_mset*[iff]: $I \models_m add_mset C CC \longleftrightarrow I \models C \wedge I \models_m CC$
unfolding *true_cls_mset_def* **by** *auto*

lemma *true_cls_mset_image_mset*[iff]: $I \models_m image_mset f A \longleftrightarrow (\forall x \in\# A. I \models f x)$
unfolding *true_cls_mset_def* **by** *auto*

lemma *true_cls_mset_mono*: $set_mset DD \subseteq set_mset CC \implies I \models_m CC \implies I \models_m DD$
unfolding *true_cls_mset_def subset_iff* **by** *auto*

lemma *true_cls_set_mset*[iff]: $I \models_s set_mset CC \longleftrightarrow I \models_m CC$
unfolding *true_cls_def true_cls_mset_def* **by** *auto*

lemma *true_cls_mset_true_cls*: $I \models_m CC \implies C \in\# CC \implies I \models C$
using *true_cls_mset_def* **by** *auto*

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  $leq: \forall i. f (Suc\ i) \leq f\ i$ 
  shows  $\exists l. \forall l' \geq l. f\ l' = f (Suc\ l')$ 
proof (rule ccontr)
  assume  $a: \nexists l. \forall l' \geq l. f\ l' = f (Suc\ l')$ 
  have  $\forall i. \exists i'. i' > i \wedge f\ i' < f\ i$ 
  proof
    fix  $i$ 
    from  $a$  have  $\exists l' \geq i. f\ l' \neq f (Suc\ l')$ 
    by auto
    then obtain  $l'$  where
       $l'_p: l' \geq i \wedge f\ l' \neq f (Suc\ l')$ 
    by metis
    then have  $f\ l' > f (Suc\ l')$ 
    using  $leq\ le\_eq\_less\_or\_eq$  by auto
    moreover have  $f\ i \geq f\ l'$ 
    using  $leq\ l'_p$  by (induction  $l'$  arbitrary: i) (blast intro: lift_Suc_antimono_le)+
    ultimately show  $\exists i' > i. f\ i' < f\ i$ 
    using  $l'_p\ less\_le\_trans$  by blast
  qed
  then obtain  $g\_sm :: nat \Rightarrow nat$  where
     $g\_sm\_p: \forall i. g\_sm\ i > i \wedge f (g\_sm\ i) < f\ i$ 
  by metis

  define  $c :: nat \Rightarrow nat$  where
     $\bigwedge n. c\ n = (g\_sm\ \wedge^{\wedge}\ n)\ 0$ 

  have  $f (c\ i) > f (c (Suc\ i))$  for  $i$ 
  by (induction  $i$ ) (auto simp: c_def\ g\_sm\_p)
  then have  $\forall i. (f \circ c)\ i > (f \circ c) (Suc\ i)$ 
  by auto
  then have  $\exists fc :: nat \Rightarrow nat. \forall i. fc\ i > fc (Suc\ i)$ 
  by metis
  then show False
  using wf_less_than by (simp add: wf_iff_no_infinite_down_chain)
qed

```

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  $\cdot a$  67) where
     $subst\_atm\_abbrev \equiv subst\_atm$ 

  abbreviation  $comp\_subst\_abbrev :: 's \Rightarrow 's \Rightarrow 's$  (infixl  $\odot$  67) where
     $comp\_subst\_abbrev \equiv comp\_subst$ 

  definition  $comp\_subst :: 's\ list \Rightarrow 's\ list \Rightarrow 's\ list$  (infixl  $\odot s$  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  $\cdot as$  67) where

```


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

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

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

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

definition

$$\text{subst_atm_mset_list} :: 'a \text{ multiset list} \Rightarrow 's \Rightarrow 'a \text{ multiset list} \text{ (infixl } \cdot \text{aml 67)}$$

where

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

definition

$$\text{subst_atm_mset_lists} :: 'a \text{ multiset list} \Rightarrow 's \text{ list} \Rightarrow 'a \text{ multiset list} \text{ (infixl } \cdot \cdot \text{aml 67)}$$

where

$$AAs \cdot \cdot \text{aml} \sigma s = \text{map2 } (\cdot \text{am}) AAs \sigma s$$

definition *subst_lit* :: 'a literal \Rightarrow 's \Rightarrow 'a literal (**infixl** ·l 67) **where**
 $L \cdot l \sigma = \text{map_literal } (\lambda A. A \cdot a \sigma) L$

lemma *atm_of_subst_lit[simp]*: $\text{atm_of } (L \cdot l \sigma) = \text{atm_of } L \cdot a \sigma$
unfolding *subst_lit_def* **by** (cases L) *simp+*

definition *subst_cls* :: 'a clause \Rightarrow 's \Rightarrow 'a clause (**infixl** · 67) **where**
 $AA \cdot \sigma = \text{image_mset } (\lambda A. A \cdot l \sigma) AA$

definition *subst_cls* :: 'a clause set \Rightarrow 's \Rightarrow 'a clause set (**infixl** ·cs 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** ·cl 67) **where**
 $Cs \cdot cl \sigma = \text{map } (\lambda A. A \cdot \sigma) Cs$

definition *subst_cls_lists* :: 'a clause list \Rightarrow 's list \Rightarrow 'a clause list (**infixl** ··cl 67) **where**
 $Cs \cdot \cdot \text{cl} \sigma s = \text{map2 } (\cdot) Cs \sigma s$

definition *subst_cls_mset* :: 'a clause multiset \Rightarrow 's \Rightarrow 'a clause multiset (**infixl** ·cm 67) **where**
 $CC \cdot cm \sigma = \text{image_mset } (\lambda A. A \cdot \sigma) CC$

lemma *subst_cls_add_mset[simp]*: $\text{add_mset } L C \cdot \sigma = \text{add_mset } (L \cdot l \sigma) (C \cdot \sigma)$
unfolding *subst_cls_def* **by** *simp*

lemma *subst_cls_mset_add_mset[simp]*: $\text{add_mset } C CC \cdot cm \sigma = \text{add_mset } (C \cdot \sigma) (CC \cdot cm \sigma)$
unfolding *subst_cls_mset_def* **by** *simp*

definition *generalizes_atm* :: 'a \Rightarrow 'a \Rightarrow bool **where**
 $\text{generalizes_atm } A B \iff (\exists \sigma. A \cdot a \sigma = B)$

definition *strictly_generalizes_atm* :: 'a \Rightarrow 'a \Rightarrow bool **where**
 $\text{strictly_generalizes_atm } A B \iff \text{generalizes_atm } A B \wedge \neg \text{generalizes_atm } B A$

definition *generalizes_lit* :: 'a literal \Rightarrow 'a literal \Rightarrow bool **where**
 $\text{generalizes_lit } L M \iff (\exists \sigma. L \cdot l \sigma = M)$

definition *strictly_generalizes_lit* :: 'a literal \Rightarrow 'a literal \Rightarrow bool **where**
 $\text{strictly_generalizes_lit } L M \iff \text{generalizes_lit } L M \wedge \neg \text{generalizes_lit } M L$

definition *generalizes_cls* :: 'a clause \Rightarrow 'a clause \Rightarrow bool **where**
 $\text{generalizes_cls } C D \iff (\exists \sigma. C \cdot \sigma = D)$

definition *strictly_generalizes_cls* :: 'a clause \Rightarrow 'a clause \Rightarrow bool **where**
strictly_generalizes_cls C D \longleftrightarrow *generalizes_cls* C D \wedge \neg *generalizes_cls* 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_cls* C D \wedge *generalizes_cls* 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_clsset* :: 'a clause set \Rightarrow bool **where**
is_ground_clsset 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_clsset* :: 'a clause set \Rightarrow 'a clause set **where**
grounding_of_clsset 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 $is_mgu :: 's \Rightarrow 'a \text{ set} \Rightarrow bool$ **where**
 $is_mgu \sigma AAA \iff is_unifiers \sigma AAA \wedge (\forall \tau. is_unifiers \tau AAA \longrightarrow (\exists \gamma. \tau = \sigma \odot \gamma))$

definition $var_disjoint :: 'a \text{ clause list} \Rightarrow bool$ **where**
 $var_disjoint Cs \iff$
 $(\forall \sigma s. length \sigma s = length Cs \longrightarrow (\exists \tau. \forall i < length Cs. \forall S. S \subseteq\# Cs ! i \longrightarrow S \cdot \sigma s ! i = S \cdot \tau))$

end

7.3 Substitution Lemmas

locale $substitution = substitution_ops \text{ subst_atm } id_subst \text{ comp_subst}$
for

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

$id_subst :: 's$ **and**

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

fixes

$atm_of_atms :: 'a \text{ list} \Rightarrow 'a$ **and**

$renamings_apart :: 'a \text{ clause list} \Rightarrow 's \text{ list}$

assumes

$subst_atm_id_subst[simp]: A \cdot a \text{ id_subst} = A$ **and**

$subst_atm_comp_subst[simp]: A \cdot a (\tau \odot \sigma) = (A \cdot a \tau) \cdot a \sigma$ **and**

$subst_ext: (\bigwedge A. A \cdot a \sigma = A \cdot a \tau) \implies \sigma = \tau$ **and**

$make_ground_subst: is_ground_cls (C \cdot \sigma) \implies \exists \tau. is_ground_subst \tau \wedge C \cdot \tau = C \cdot \sigma$ **and**

$renames_apart:$

$\bigwedge Cs. length (renamings_apart Cs) = length Cs \wedge$
 $(\forall \varrho \in set (renamings_apart Cs). is_renaming \varrho) \wedge$
 $var_disjoint (Cs \cdot\cdot cl (renamings_apart Cs))$ **and**

$atm_of_atms_subst:$

$\bigwedge As Bs. atm_of_atms As \cdot a \sigma = atm_of_atms Bs \iff map (\lambda A. A \cdot a \sigma) As = Bs$ **and**

$wf_strictly_generalizes_atm: wfP \text{ strictly_generalizes_atm}$

begin

lemma $subst_ext_iff: \sigma = \tau \iff (\forall A. A \cdot a \sigma = A \cdot a \tau)$

by ($blast \text{ intro: subst_ext}$)

7.3.1 Identity Substitution

lemma $id_subst_comp_subst[simp]: id_subst \odot \sigma = \sigma$

by ($rule \text{ subst_ext}$) $simp$

lemma $comp_subst_id_subst[simp]: \sigma \odot id_subst = \sigma$

by ($rule \text{ subst_ext}$) $simp$

lemma $id_subst_comp_subs[simp]: replicate (length \sigma s) id_subst \odot s \sigma s = \sigma s$

using $comp_subs_def$ **by** ($induction \sigma s$) $auto$

lemma $comp_subs_id_subst[simp]: \sigma s \odot s replicate (length \sigma s) id_subst = \sigma s$

using $comp_subs_def$ **by** ($induction \sigma s$) $auto$

lemma $subst_atms_id_subst[simp]: AA \cdot as \text{ id_subst} = AA$

unfolding $subst_atms_def$ **by** $simp$

lemma $subst_atmss_id_subst[simp]: AAA \cdot ass \text{ id_subst} = AAA$

unfolding $subst_atmss_def$ **by** $simp$

lemma $subst_atm_list_id_subst[simp]: As \cdot al \text{ id_subst} = As$

unfolding $subst_atm_list_def$ **by** $auto$

lemma $subst_atm_mset_id_subst[simp]: AA \cdot am \text{ id_subst} = AA$

unfolding $subst_atm_mset_def$ **by** $simp$

lemma $subst_atm_mset_list_id_subst[simp]: AAs \cdot aml \text{ id_subst} = AAs$

unfolding *subst_atm_mset_list_def* **by** *simp*

lemma *subst_atm_mset_lists_id_subst[simp]*: $AAs \cdot aml \text{ replicate } (\text{length } AAs) \text{ id_subst} = AAs$
unfolding *subst_atm_mset_lists_def* **by** (*induct AAs*) *auto*

lemma *subst_lit_id_subst[simp]*: $L \cdot l \text{ id_subst} = L$
unfolding *subst_lit_def* **by** (*simp add: literal.map_ident*)

lemma *subst_cls_id_subst[simp]*: $C \cdot \text{id_subst} = C$
unfolding *subst_cls_def* **by** *simp*

lemma *subst_cls_id_subst[simp]*: $CC \cdot cs \text{ id_subst} = CC$
unfolding *subst_cls_def* **by** *simp*

lemma *subst_cls_list_id_subst[simp]*: $Cs \cdot cl \text{ id_subst} = Cs$
unfolding *subst_cls_list_def* **by** *simp*

lemma *subst_cls_lists_id_subst[simp]*: $Cs \cdot cl \text{ replicate } (\text{length } Cs) \text{ id_subst} = Cs$
unfolding *subst_cls_lists_def* **by** (*induct Cs*) *auto*

lemma *subst_cls_mset_id_subst[simp]*: $CC \cdot cm \text{ id_subst} = CC$
unfolding *subst_cls_mset_def* **by** *simp*

7.3.2 Associativity of Composition

lemma *comp_subst_assoc[simp]*: $\sigma \odot (\tau \odot \gamma) = \sigma \odot \tau \odot \gamma$
by (*rule subst_ext*) *simp*

7.3.3 Compatibility of Substitution and Composition

lemma *subst_atms_comp_subst[simp]*: $AA \cdot as (\tau \odot \sigma) = AA \cdot as \tau \cdot as \sigma$
unfolding *subst_atms_def* **by** *auto*

lemma *subst_atmss_comp_subst[simp]*: $AAA \cdot ass (\tau \odot \sigma) = AAA \cdot ass \tau \cdot ass \sigma$
unfolding *subst_atmss_def* **by** *auto*

lemma *subst_atm_list_comp_subst[simp]*: $As \cdot al (\tau \odot \sigma) = As \cdot al \tau \cdot al \sigma$
unfolding *subst_atm_list_def* **by** *auto*

lemma *subst_atm_mset_comp_subst[simp]*: $AA \cdot am (\tau \odot \sigma) = AA \cdot am \tau \cdot am \sigma$
unfolding *subst_atm_mset_def* **by** *auto*

lemma *subst_atm_mset_list_comp_subst[simp]*: $AAs \cdot aml (\tau \odot \sigma) = (AAs \cdot aml \tau) \cdot aml \sigma$
unfolding *subst_atm_mset_list_def* **by** *auto*

lemma *subst_atm_mset_lists_comp_substs[simp]*: $AAs \cdot aml (\tau s \odot s \sigma s) = AAs \cdot aml \tau s \cdot aml \sigma s$
unfolding *subst_atm_mset_lists_def comp_substs_def map_zip_map map_zip_map2 map_zip_assoc*
by (*simp add: split_def*)

lemma *subst_lit_comp_subst[simp]*: $L \cdot l (\tau \odot \sigma) = L \cdot l \tau \cdot l \sigma$
unfolding *subst_lit_def* **by** (*auto simp: literal.map_comp o_def*)

lemma *subst_cls_comp_subst[simp]*: $C \cdot (\tau \odot \sigma) = C \cdot \tau \cdot \sigma$
unfolding *subst_cls_def* **by** *auto*

lemma *subst_clscomp_subst[simp]*: $CC \cdot cs (\tau \odot \sigma) = CC \cdot cs \tau \cdot cs \sigma$
unfolding *subst_cls_def* **by** *auto*

lemma *subst_cls_list_comp_subst[simp]*: $Cs \cdot cl (\tau \odot \sigma) = Cs \cdot cl \tau \cdot cl \sigma$
unfolding *subst_cls_list_def* **by** *auto*

lemma *subst_cls_lists_comp_substs[simp]*: $Cs \cdot cl (\tau s \odot s \sigma s) = Cs \cdot cl \tau s \cdot cl \sigma s$
unfolding *subst_cls_lists_def comp_substs_def map_zip_map map_zip_map2 map_zip_assoc*
by (*simp add: split_def*)

lemma *subst_cls_mset_comp_subst*[simp]: $CC \cdot cm (\tau \odot \sigma) = CC \cdot cm \tau \cdot cm \sigma$
unfolding *subst_cls_mset_def* **by** *auto*

7.3.4 “Commutativity” of Membership and Substitution

lemma *Melem_subst_atm_mset*[simp]: $A \in\# AA \cdot am \sigma \longleftrightarrow (\exists B. B \in\# AA \wedge A = B \cdot a \sigma)$
unfolding *subst_atm_mset_def* **by** *auto*

lemma *Melem_subst_cls*[simp]: $L \in\# C \cdot \sigma \longleftrightarrow (\exists M. M \in\# C \wedge L = M \cdot l \sigma)$
unfolding *subst_cls_def* **by** *auto*

lemma *Melem_subst_cls_mset*[simp]: $AA \in\# CC \cdot cm \sigma \longleftrightarrow (\exists BB. BB \in\# CC \wedge AA = BB \cdot \sigma)$
unfolding *subst_cls_mset_def* **by** *auto*

7.3.5 Signs and Substitutions

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

lemma *subst_lit_is_pos*[simp]: $is_pos (L \cdot l \sigma) = is_pos L$
unfolding *subst_lit_def* **by** *auto*

lemma *subst_minus*[simp]: $(- L) \cdot l \mu = - (L \cdot l \mu)$
by (*simp add: literal.map_sel subst_lit_def uminus_literal_def*)

7.3.6 Substitution on Literal(s)

lemma *eql_neg_lit_eql_atm*[simp]: $(Neg A' \cdot l \eta) = Neg A \longleftrightarrow A' \cdot a \eta = A$
by (*simp add: subst_lit_def*)

lemma *eql_pos_lit_eql_atm*[simp]: $(Pos A' \cdot l \eta) = Pos A \longleftrightarrow A' \cdot a \eta = A$
by (*simp add: subst_lit_def*)

lemma *subst_cls_negs*[simp]: $(negs AA) \cdot \sigma = negs (AA \cdot am \sigma)$
unfolding *subst_cls_def subst_lit_def subst_atm_mset_def* **by** *auto*

lemma *subst_cls_poss*[simp]: $(poss AA) \cdot \sigma = poss (AA \cdot am \sigma)$
unfolding *subst_cls_def subst_lit_def subst_atm_mset_def* **by** *auto*

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

proof –

have $atms_of (C \cdot \sigma) = set_mset (image_mset atm_of (image_mset (map_literal (\lambda A. A \cdot a \sigma)) C))$
unfolding *subst_cls_def subst_atms_def subst_lit_def atms_of_def* **by** *auto*
also have $\dots = set_mset (image_mset (\lambda A. A \cdot a \sigma) (image_mset atm_of C))$
by *simp (meson literal.map_sel)*
finally show $atms_of C \cdot as \sigma = atms_of (C \cdot \sigma)$
unfolding *subst_atms_def atms_of_def* **by** *auto*

qed

lemma *in_image_Neg_is_neg*[simp]: $L \cdot l \sigma \in Neg \text{ ' } AA \implies is_neg L$
by (*metis be_x_imageD literal.disc(2) literal.map_disc_iff subst_lit_def*)

lemma *subst_lit_in_negs_subst_is_neg*: $L \cdot l \sigma \in\# (negs AA) \cdot \tau \implies is_neg L$
by *simp*

lemma *subst_lit_in_negs_is_neg*: $L \cdot l \sigma \in\# negs AA \implies is_neg L$
by *simp*

7.3.7 Substitution on Empty

lemma *subst_atms_empty*[simp]: $\{\} \cdot as \sigma = \{\}$
unfolding *subst_atms_def* **by** *auto*

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

unfolding *subst_atmss_def* **by** *auto*

lemma *comp_substs_empty_iff[simp]*: $\sigma s \odot s \eta s = [] \iff \sigma s = [] \vee \eta s = []$
using *comp_substs_def map2_empty_iff* **by** *auto*

lemma *subst_atm_list_empty[simp]*: $[] \cdot al \sigma = []$
unfolding *subst_atm_list_def* **by** *auto*

lemma *subst_atm_mset_empty[simp]*: $\{\#\} \cdot am \sigma = \{\#\}$
unfolding *subst_atm_mset_def* **by** *auto*

lemma *subst_atm_mset_list_empty[simp]*: $[] \cdot aml \sigma = []$
unfolding *subst_atm_mset_list_def* **by** *auto*

lemma *subst_atm_mset_lists_empty[simp]*: $[] \cdot \cdot aml \sigma s = []$
unfolding *subst_atm_mset_lists_def* **by** *auto*

lemma *subst_cls_empty[simp]*: $\{\#\} \cdot \sigma = \{\#\}$
unfolding *subst_cls_def* **by** *auto*

lemma *subst_cls_empty[simp]*: $\{\} \cdot cs \sigma = \{\}$
unfolding *subst_cls_def* **by** *auto*

lemma *subst_cls_list_empty[simp]*: $[] \cdot cl \sigma = []$
unfolding *subst_cls_list_def* **by** *auto*

lemma *subst_cls_lists_empty[simp]*: $[] \cdot \cdot cl \sigma s = []$
unfolding *subst_cls_lists_def* **by** *auto*

lemma *subst_scls_mset_empty[simp]*: $\{\#\} \cdot cm \sigma = \{\#\}$
unfolding *subst_cls_mset_def* **by** *auto*

lemma *subst_atms_empty_iff[simp]*: $AA \cdot as \eta = \{\} \iff AA = \{\}$
unfolding *subst_atms_def* **by** *auto*

lemma *subst_atmss_empty_iff[simp]*: $AAA \cdot ass \eta = \{\} \iff AAA = \{\}$
unfolding *subst_atmss_def* **by** *auto*

lemma *subst_atm_list_empty_iff[simp]*: $As \cdot al \eta = [] \iff As = []$
unfolding *subst_atm_list_def* **by** *auto*

lemma *subst_atm_mset_empty_iff[simp]*: $AA \cdot am \eta = \{\#\} \iff AA = \{\#\}$
unfolding *subst_atm_mset_def* **by** *auto*

lemma *subst_atm_mset_list_empty_iff[simp]*: $AAs \cdot aml \eta = [] \iff AAs = []$
unfolding *subst_atm_mset_list_def* **by** *auto*

lemma *subst_atm_mset_lists_empty_iff[simp]*: $AAs \cdot \cdot aml \eta s = [] \iff (AAs = [] \vee \eta s = [])$
using *map2_empty_iff subst_atm_mset_lists_def* **by** *auto*

lemma *subst_cls_empty_iff[simp]*: $C \cdot \eta = \{\#\} \iff C = \{\#\}$
unfolding *subst_cls_def* **by** *auto*

lemma *subst_cls_empty_iff[simp]*: $CC \cdot cs \eta = \{\} \iff CC = \{\}$
unfolding *subst_cls_def* **by** *auto*

lemma *subst_cls_list_empty_iff[simp]*: $Cs \cdot cl \eta = [] \iff Cs = []$
unfolding *subst_cls_list_def* **by** *auto*

lemma *subst_cls_lists_empty_iff[simp]*: $Cs \cdot \cdot cl \eta s = [] \iff (Cs = [] \vee \eta s = [])$
using *map2_empty_iff subst_cls_lists_def* **by** *auto*

lemma *subst_cls_mset_empty_iff[simp]*: $CC \cdot cm \eta = \{\#\} \iff CC = \{\#\}$

unfolding *subst_cls_mset_def* by *auto*

7.3.8 Substitution on a Union

lemma *subst_atms_union[simp]*: $(AA \cup BB) \cdot as \sigma = AA \cdot as \sigma \cup BB \cdot as \sigma$
unfolding *subst_atms_def* by *auto*

lemma *subst_atmss_union[simp]*: $(AAA \cup BBB) \cdot ass \sigma = AAA \cdot ass \sigma \cup BBB \cdot ass \sigma$
unfolding *subst_atmss_def* by *auto*

lemma *subst_atm_list_append[simp]*: $(As @ Bs) \cdot al \sigma = As \cdot al \sigma @ Bs \cdot al \sigma$
unfolding *subst_atm_list_def* by *auto*

lemma *subst_atm_mset_union[simp]*: $(AA + BB) \cdot am \sigma = AA \cdot am \sigma + BB \cdot am \sigma$
unfolding *subst_atm_mset_def* by *auto*

lemma *subst_atm_mset_list_append[simp]*: $(AAs @ BBs) \cdot aml \sigma = AAs \cdot aml \sigma @ BBs \cdot aml \sigma$
unfolding *subst_atm_mset_list_def* by *auto*

lemma *subst_cls_union[simp]*: $(C + D) \cdot \sigma = C \cdot \sigma + D \cdot \sigma$
unfolding *subst_cls_def* by *auto*

lemma *subst_cls_union[simp]*: $(CC \cup DD) \cdot cs \sigma = CC \cdot cs \sigma \cup DD \cdot cs \sigma$
unfolding *subst_cls_def* by *auto*

lemma *subst_cls_list_append[simp]*: $(Cs @ Ds) \cdot cl \sigma = Cs \cdot cl \sigma @ Ds \cdot cl \sigma$
unfolding *subst_cls_list_def* by *auto*

lemma *subst_cls_mset_union[simp]*: $(CC + DD) \cdot cm \sigma = CC \cdot cm \sigma + DD \cdot cm \sigma$
unfolding *subst_cls_mset_def* by *auto*

7.3.9 Substitution on a Singleton

lemma *subst_atms_single[simp]*: $\{A\} \cdot as \sigma = \{A \cdot a \sigma\}$
unfolding *subst_atms_def* by *auto*

lemma *subst_atmss_single[simp]*: $\{AA\} \cdot ass \sigma = \{AA \cdot as \sigma\}$
unfolding *subst_atmss_def* by *auto*

lemma *subst_atm_list_single[simp]*: $[A] \cdot al \sigma = [A \cdot a \sigma]$
unfolding *subst_atm_list_def* by *auto*

lemma *subst_atm_mset_single[simp]*: $\{\#A\#\} \cdot am \sigma = \{\#A \cdot a \sigma\#\}$
unfolding *subst_atm_mset_def* by *auto*

lemma *subst_atm_mset_list[simp]*: $[AA] \cdot aml \sigma = [AA \cdot am \sigma]$
unfolding *subst_atm_mset_list_def* by *auto*

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

lemma *subst_cls_single[simp]*: $\{C\} \cdot cs \sigma = \{C \cdot \sigma\}$
unfolding *subst_cls_def* by *auto*

lemma *subst_cls_list_single[simp]*: $[C] \cdot cl \sigma = [C \cdot \sigma]$
unfolding *subst_cls_list_def* by *auto*

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

7.3.10 Substitution on (#)

lemma *subst_atm_list_Cons[simp]*: $(A \# As) \cdot al \sigma = A \cdot a \sigma \# As \cdot al \sigma$
unfolding *subst_atm_list_def* by *auto*

lemma *subst_atm_mset_list_Cons[simp]*: $(A \# As) \cdot aml \sigma = A \cdot am \sigma \# As \cdot aml \sigma$
unfolding *subst_atm_mset_list_def* **by** *auto*

lemma *subst_atm_mset_lists_Cons[simp]*: $(C \# Cs) \cdot \cdot aml (\sigma \# \sigma s) = C \cdot am \sigma \# Cs \cdot \cdot aml \sigma s$
unfolding *subst_atm_mset_lists_def* **by** *auto*

lemma *subst_cls_list_Cons[simp]*: $(C \# Cs) \cdot cl \sigma = C \cdot \sigma \# Cs \cdot cl \sigma$
unfolding *subst_cls_list_def* **by** *auto*

lemma *subst_cls_lists_Cons[simp]*: $(C \# Cs) \cdot \cdot cl (\sigma \# \sigma s) = C \cdot \sigma \# Cs \cdot \cdot cl \sigma s$
unfolding *subst_cls_lists_def* **by** *auto*

7.3.11 Substitution on *tl*

lemma *subst_atm_list_tl[simp]*: $tl (As \cdot al \eta) = tl As \cdot al \eta$
by (*induction As*) *auto*

lemma *subst_atm_mset_list_tl[simp]*: $tl (AAs \cdot aml \eta) = tl AAs \cdot aml \eta$
by (*induction AAs*) *auto*

7.3.12 Substitution on (!)

lemma *comp_substs_nth[simp]*:
 $length \tau s = length \sigma s \implies i < length \tau s \implies (\tau s \odot s \sigma s) ! i = (\tau s ! i) \odot (\sigma s ! i)$
by (*simp add: comp_substs_def*)

lemma *subst_atm_list_nth[simp]*: $i < length As \implies (As \cdot al \tau) ! i = As ! i \cdot a \tau$
unfolding *subst_atm_list_def* **using** *less_Suc_eq_0_disj nth_map* **by** *force*

lemma *subst_atm_mset_list_nth[simp]*: $i < length AAs \implies (AAs \cdot aml \eta) ! i = (AAs ! i) \cdot am \eta$
unfolding *subst_atm_mset_list_def* **by** *auto*

lemma *subst_atm_mset_lists_nth[simp]*:
 $length AAs = length \sigma s \implies i < length AAs \implies (AAs \cdot \cdot aml \sigma s) ! i = (AAs ! i) \cdot am (\sigma s ! i)$
unfolding *subst_atm_mset_lists_def* **by** *auto*

lemma *subst_cls_list_nth[simp]*: $i < length Cs \implies (Cs \cdot cl \tau) ! i = (Cs ! i) \cdot \tau$
unfolding *subst_cls_list_def* **using** *less_Suc_eq_0_disj nth_map* **by** (*induction Cs*) *auto*

lemma *subst_cls_lists_nth[simp]*:
 $length Cs = length \sigma s \implies i < length Cs \implies (Cs \cdot \cdot cl \sigma s) ! i = (Cs ! i) \cdot (\sigma s ! i)$
unfolding *subst_cls_lists_def* **by** *auto*

7.3.13 Substitution on Various Other Functions

lemma *subst_cls_image[simp]*: $image f X \cdot cs \sigma = \{f x \cdot \sigma \mid x. x \in X\}$
unfolding *subst_cls_def* **by** *auto*

lemma *subst_cls_mset_image_mset[simp]*: $image_mset f X \cdot cm \sigma = \{\# f x \cdot \sigma. x \in \# X \#\}$
unfolding *subst_cls_mset_def* **by** *auto*

lemma *mset_subst_atm_list_subst_atm_mset[simp]*: $mset (As \cdot al \sigma) = mset (As) \cdot am \sigma$
unfolding *subst_atm_list_def* *subst_atm_mset_def* **by** *auto*

lemma *mset_subst_cls_list_subst_cls_mset*: $mset (Cs \cdot cl \sigma) = (mset Cs) \cdot cm \sigma$
unfolding *subst_cls_mset_def* *subst_cls_list_def* **by** *auto*

lemma *sum_list_subst_cls_list_subst_cls[simp]*: $sum_list (Cs \cdot cl \eta) = sum_list Cs \cdot \eta$
unfolding *subst_cls_list_def* **by** (*induction Cs*) *auto*

lemma *set_mset_subst_cls_mset_subst_cls*: $set_mset (CC \cdot cm \mu) = (set_mset CC) \cdot cs \mu$
by (*simp add: subst_cls_mset_def* *subst_cls_def*)

lemma *Neg_Melem_subst_atm_subst_cls[simp]*: $Neg A \in\# C \implies Neg (A \cdot a \sigma) \in\# C \cdot \sigma$
by (*metis Melem_subst_cls eql_neg_lit_eql_atm*)

lemma *Pos_Melem_subst_atm_subst_cls[simp]*: $Pos A \in\# C \implies Pos (A \cdot a \sigma) \in\# C \cdot \sigma$
by (*metis Melem_subst_cls eql_pos_lit_eql_atm*)

lemma *in_atms_of_subst[simp]*: $B \in atms_of C \implies B \cdot a \sigma \in atms_of (C \cdot \sigma)$
by (*metis atms_of_subst_atms image_iff subst_atms_def*)

7.3.14 Renamings

lemma *is_renaming_id_subst[simp]*: *is_renaming id_subst*
unfolding *is_renaming_def* **by** *simp*

lemma *is_renamingD*: $is_renaming \sigma \implies (\forall A1 A2. A1 \cdot a \sigma = A2 \cdot a \sigma \longleftrightarrow A1 = A2)$
by (*metis is_renaming_def subst_atm_comp_subst subst_atm_id_subst*)

lemma *inv_renaming_cancel_r[simp]*: $is_renaming r \implies r \odot inv_renaming r = id_subst$
unfolding *inv_renaming_def is_renaming_def* **by** (*metis (mono_tags) someI_ex*)

lemma *inv_renaming_cancel_r_list[simp]*:
is_renaming_list rs $\implies rs \odot s \text{ map } inv_renaming rs = replicate (length rs) id_subst$
unfolding *is_renaming_list_def* **by** (*induction rs*) (*auto simp add: comp_substs_def*)

lemma *Nil_comp_substs[simp]*: $\[] \odot s = \[]$
unfolding *comp_substs_def* **by** *auto*

lemma *comp_substs_Nil[simp]*: $s \odot \[] = \[]$
unfolding *comp_substs_def* **by** *auto*

lemma *is_renaming_idempotent_id_subst*: $is_renaming r \implies r \odot r = r \implies r = id_subst$
by (*metis comp_subst_assoc comp_subst_id_subst inv_renaming_cancel_r*)

lemma *is_renaming_left_id_subst_right_id_subst*:
is_renaming r $\implies s \odot r = id_subst \implies r \odot s = id_subst$
by (*metis comp_subst_assoc comp_subst_id_subst is_renaming_def*)

lemma *is_renaming_closure*: $is_renaming r1 \implies is_renaming r2 \implies is_renaming (r1 \odot r2)$
unfolding *is_renaming_def* **by** (*metis comp_subst_assoc comp_subst_id_subst*)

lemma *is_renaming_inv_renaming_cancel_atm[simp]*: $is_renaming \varrho \implies A \cdot a \varrho \cdot a inv_renaming \varrho = A$
by (*metis inv_renaming_cancel_r subst_atm_comp_subst subst_atm_id_subst*)

lemma *is_renaming_inv_renaming_cancel_atms[simp]*: $is_renaming \varrho \implies AA \cdot as \varrho \cdot as inv_renaming \varrho = AA$
by (*metis inv_renaming_cancel_r subst_atms_comp_subst subst_atms_id_subst*)

lemma *is_renaming_inv_renaming_cancel_atmss[simp]*: $is_renaming \varrho \implies AAA \cdot ass \varrho \cdot ass inv_renaming \varrho = AAA$
by (*metis inv_renaming_cancel_r subst_atmss_comp_subst subst_atmss_id_subst*)

lemma *is_renaming_inv_renaming_cancel_atm_list[simp]*: $is_renaming \varrho \implies As \cdot al \varrho \cdot al inv_renaming \varrho = As$
by (*metis inv_renaming_cancel_r subst_atm_list_comp_subst subst_atm_list_id_subst*)

lemma *is_renaming_inv_renaming_cancel_atm_mset[simp]*: $is_renaming \varrho \implies AA \cdot am \varrho \cdot am inv_renaming \varrho = AA$
by (*metis inv_renaming_cancel_r subst_atm_mset_comp_subst subst_atm_mset_id_subst*)

lemma *is_renaming_inv_renaming_cancel_atm_mset_list[simp]*: $is_renaming \varrho \implies (AAs \cdot aml \varrho) \cdot aml inv_renaming \varrho = AAs$
by (*metis inv_renaming_cancel_r subst_atm_mset_list_comp_subst subst_atm_mset_list_id_subst*)

lemma *is_renaming_list_inv_renaming_cancel_atm_mset_lists[simp]*:
 $length AAs = length \varrho s \implies is_renaming_list \varrho s \implies AAs \cdot aml \varrho s \cdot aml \text{ map } inv_renaming \varrho s = AAs$
by (*metis inv_renaming_cancel_r_list subst_atm_mset_lists_comp_substs subst_atm_mset_lists_id_subst*)

lemma *is_renaming_inv_renaming_cancel_lit[simp]*: $is_renaming \varrho \implies (L \cdot l \varrho) \cdot l inv_renaming \varrho = L$

by (metis inv_renaming_cancel_r subst_lit_comp_subst subst_lit_id_subst)

lemma *is_renaming_inv_renaming_cancel_cls[simp]*: $is_renaming\ \varrho \implies C \cdot \varrho \cdot inv_renaming\ \varrho = C$
 by (metis inv_renaming_cancel_r subst_cls_comp_subst subst_cls_id_subst)

lemma *is_renaming_inv_renaming_cancel_cls[simp]*: $is_renaming\ \varrho \implies CC \cdot cs\ \varrho \cdot cs\ inv_renaming\ \varrho = CC$
 by (metis inv_renaming_cancel_r subst_cls_id_subst subst_clscomp_subst)

lemma *is_renaming_inv_renaming_cancel_cls_list[simp]*: $is_renaming\ \varrho \implies Cs \cdot cl\ \varrho \cdot cl\ inv_renaming\ \varrho = Cs$
 by (metis inv_renaming_cancel_r subst_cls_list_comp_subst subst_cls_list_id_subst)

lemma *is_renaming_list_inv_renaming_cancel_cls_list[simp]*:
 $length\ Cs = length\ \varrho s \implies is_renaming_list\ \varrho s \implies Cs \cdot cl\ \varrho s \cdot cl\ map\ inv_renaming\ \varrho s = Cs$
 by (metis inv_renaming_cancel_r_list subst_cls_lists_comp_substs subst_cls_lists_id_subst)

lemma *is_renaming_inv_renaming_cancel_cls_mset[simp]*: $is_renaming\ \varrho \implies CC \cdot cm\ \varrho \cdot cm\ inv_renaming\ \varrho = CC$
 by (metis inv_renaming_cancel_r subst_cls_mset_comp_subst subst_cls_mset_id_subst)

7.3.15 Monotonicity

lemma *subst_cls_mono*: $set_mset\ C \subseteq set_mset\ D \implies set_mset\ (C \cdot \sigma) \subseteq set_mset\ (D \cdot \sigma)$
 by force

lemma *subst_cls_mono_mset*: $C \subseteq\# D \implies C \cdot \sigma \subseteq\# D \cdot \sigma$
 unfolding *subst_cls_def* by (metis mset_subset_eq_exists_conv subst_cls_union)

lemma *subst_subset_mono*: $D \subset\# C \implies D \cdot \sigma \subset\# C \cdot \sigma$
 unfolding *subst_cls_def* by (simp add: image_mset_subset_mono)

7.3.16 Size after Substitution

lemma *size_subst[simp]*: $size\ (D \cdot \sigma) = size\ D$
 unfolding *subst_cls_def* by auto

lemma *subst_atm_list_length[simp]*: $length\ (As \cdot al\ \sigma) = length\ As$
 unfolding *subst_atm_list_def* by auto

lemma *length_subst_atm_mset_list[simp]*: $length\ (AAs \cdot aml\ \eta) = length\ AAs$
 unfolding *subst_atm_mset_list_def* by auto

lemma *subst_atm_mset_lists_length[simp]*: $length\ (AAs \cdot aml\ \sigma s) = min\ (length\ AAs)\ (length\ \sigma s)$
 unfolding *subst_atm_mset_lists_def* by auto

lemma *subst_cls_list_length[simp]*: $length\ (Cs \cdot cl\ \sigma) = length\ Cs$
 unfolding *subst_cls_list_def* by auto

lemma *comp_substs_length[simp]*: $length\ (\tau s \odot s\ \sigma s) = min\ (length\ \tau s)\ (length\ \sigma s)$
 unfolding *comp_substs_def* by auto

lemma *subst_cls_lists_length[simp]*: $length\ (Cs \cdot cl\ \sigma s) = min\ (length\ Cs)\ (length\ \sigma s)$
 unfolding *subst_cls_lists_def* by auto

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 *clarify*
 fix $\sigma s :: 's\ list$
 assume *a: length\ \sigma s = length\ Cs*
 then obtain τ where $\forall i < length\ Cs. \forall S. S \subseteq\# Cs\ !\ i \implies S \cdot \sigma s\ !\ i = S \cdot \tau$
 using *assms* unfolding *var_disjoint_def* by blast
 then have $\forall i < length\ Cs. (Cs\ !\ i) \cdot \sigma s\ !\ i = (Cs\ !\ i) \cdot \tau$
 by auto

then have $Cs \cdot cl \sigma s = Cs \cdot cl \tau$
using a by (*simp add: nth_equalityI*)
then show $\exists \tau. Cs \cdot cl \sigma s = Cs \cdot cl \tau$
by *auto*
qed

7.3.18 Ground Expressions and Substitutions

lemma *ex_ground_subst*: $\exists \sigma. is_ground_subst \sigma$
using *make_ground_subst*[*of* {#}]
by (*simp add: is_ground_cls_def*)

lemma *is_ground_cls_list_Cons*[*simp*]:
 $is_ground_cls_list (C \# Cs) = (is_ground_cls C \wedge is_ground_cls_list Cs)$
unfolding *is_ground_cls_list_def* **by** *auto*

Ground union lemma *is_ground_atms_union*[*simp*]: $is_ground_atms (AA \cup BB) \longleftrightarrow is_ground_atms AA \wedge is_ground_atms BB$
unfolding *is_ground_atms_def* **by** *auto*

lemma *is_ground_atm_mset_union*[*simp*]:
 $is_ground_atm_mset (AA + BB) \longleftrightarrow is_ground_atm_mset AA \wedge is_ground_atm_mset BB$
unfolding *is_ground_atm_mset_def* **by** *auto*

lemma *is_ground_cls_union*[*simp*]: $is_ground_cls (C + D) \longleftrightarrow is_ground_cls C \wedge is_ground_cls D$
unfolding *is_ground_cls_def* **by** *auto*

lemma *is_ground_cls_union*[*simp*]:
 $is_ground_clss (CC \cup DD) \longleftrightarrow is_ground_clss CC \wedge is_ground_clss DD$
unfolding *is_ground_cls_def* **by** *auto*

lemma *is_ground_cls_list_is_ground_cls_sum_list*[*simp*]:
 $is_ground_cls_list Cs \implies is_ground_cls (sum_list Cs)$
by (*meson in_mset_sum_list2 is_ground_cls_def is_ground_cls_list_def*)

Ground mono lemma *is_ground_cls_mono*: $C \subseteq\# D \implies is_ground_cls D \implies is_ground_cls C$
unfolding *is_ground_cls_def* **by** (*metis set_mset_mono subsetD*)

lemma *is_ground_cls_mono*: $CC \subseteq DD \implies is_ground_clss DD \implies is_ground_clss CC$
unfolding *is_ground_cls_def* **by** *blast*

lemma *grounding_of_cls_mono*: $CC \subseteq DD \implies grounding_of_clss CC \subseteq grounding_of_clss DD$
using *grounding_of_cls_def* **by** *auto*

lemma *sum_list_subseteq_mset_is_ground_cls_list*[*simp*]:
 $sum_list Cs \subseteq\# sum_list Ds \implies is_ground_cls_list Ds \implies is_ground_cls_list Cs$
by (*meson in_mset_sum_list is_ground_cls_def is_ground_cls_list_is_ground_cls_sum_list is_ground_cls_mono is_ground_cls_list_def*)

Substituting on ground expression preserves ground lemma *is_ground_comp_subst*[*simp*]: $is_ground_subst \sigma \implies is_ground_subst (\tau \odot \sigma)$
unfolding *is_ground_subst_def is_ground_atm_def* **by** *auto*

lemma *ground_subst_ground_atm*[*simp*]: $is_ground_subst \sigma \implies is_ground_atm (A \cdot a \sigma)$
by (*simp add: is_ground_subst_def*)

lemma *ground_subst_ground_lit*[*simp*]: $is_ground_subst \sigma \implies is_ground_lit (L \cdot l \sigma)$
unfolding *is_ground_lit_def subst_lit_def* **by** (*cases L*) *auto*

lemma *ground_subst_ground_cls*[*simp*]: $is_ground_subst \sigma \implies is_ground_cls (C \cdot \sigma)$
unfolding *is_ground_cls_def* **by** *auto*

lemma *ground_subst_ground_clss*[*simp*]: $is_ground_subst \sigma \implies is_ground_clss (CC \cdot cs \sigma)$

unfolding *is_ground_cls_def subst_cls_def* **by** *auto*

lemma *ground_subst_ground_cls_list[simp]*: $is_ground_subst\ \sigma \implies is_ground_cls_list\ (Cs \cdot cl\ \sigma)$
unfolding *is_ground_cls_list_def subst_cls_list_def* **by** *auto*

lemma *ground_subst_ground_cls_lists[simp]*:
 $\forall \sigma \in set\ \sigma s. is_ground_subst\ \sigma \implies is_ground_cls_list\ (Cs \cdot cl\ \sigma s)$
unfolding *is_ground_cls_list_def subst_cls_lists_def* **by** (*auto simp: set_zip*)

Substituting on ground expression has no effect **lemma** *is_ground_subst_atm[simp]*: $is_ground_atm\ A \implies A \cdot a\ \sigma = A$
unfolding *is_ground_atm_def* **by** *simp*

lemma *is_ground_subst_atms[simp]*: $is_ground_atms\ AA \implies AA \cdot as\ \sigma = AA$
unfolding *is_ground_atms_def subst_atms_def image_def* **by** *auto*

lemma *is_ground_subst_atm_mset[simp]*: $is_ground_atm_mset\ AA \implies AA \cdot am\ \sigma = AA$
unfolding *is_ground_atm_mset_def subst_atm_mset_def* **by** *auto*

lemma *is_ground_subst_atm_list[simp]*: $is_ground_atm_list\ As \implies As \cdot al\ \sigma = As$
unfolding *is_ground_atm_list_def subst_atm_list_def* **by** (*auto intro: nth_equalityI*)

lemma *is_ground_subst_atm_list_member[simp]*:
 $is_ground_atm_list\ As \implies i < length\ As \implies As\ !\ i \cdot a\ \sigma = As\ !\ i$
unfolding *is_ground_atm_list_def* **by** *auto*

lemma *is_ground_subst_lit[simp]*: $is_ground_lit\ L \implies L \cdot l\ \sigma = L$
unfolding *is_ground_lit_def subst_lit_def* **by** (*cases L*) *simp_all*

lemma *is_ground_subst_cls[simp]*: $is_ground_cls\ C \implies C \cdot \sigma = C$
unfolding *is_ground_cls_def subst_cls_def* **by** *simp*

lemma *is_ground_subst_cls_def[simp]*: $is_ground_cls\ C \implies C \cdot \sigma = C$
unfolding *is_ground_cls_def subst_cls_def* **by** *auto*

lemma *is_ground_subst_cls_lists[simp]*:
assumes $length\ P = length\ Cs$ **and** $is_ground_cls_list\ Cs$
shows $Cs \cdot cl\ P = Cs$
using *assms* **by** (*metis is_ground_cls_list_def is_ground_subst_cls min.idem nth_equalityI nth_mem subst_cls_lists_nth subst_cls_lists_length*)

lemma *is_ground_subst_lit_iff*: $is_ground_lit\ L \iff (\forall \sigma. L = L \cdot l\ \sigma)$
using *is_ground_atm_def is_ground_lit_def subst_lit_def* **by** (*cases L*) *auto*

lemma *is_ground_subst_cls_iff*: $is_ground_cls\ C \iff (\forall \sigma. C = C \cdot \sigma)$
by (*metis ex_ground_subst ground_subst_ground_cls is_ground_subst_cls*)

Members of ground expressions are ground **lemma** *is_ground_cls_as_atms*: $is_ground_cls\ C \iff (\forall A \in atms_of\ C. is_ground_atm\ A)$
by (*auto simp: atms_of_def is_ground_cls_def is_ground_lit_def*)

lemma *is_ground_cls_imp_is_ground_lit*: $L \in \# C \implies is_ground_cls\ C \implies is_ground_lit\ L$
by (*simp add: is_ground_cls_def*)

lemma *is_ground_cls_imp_is_ground_atm*: $A \in atms_of\ C \implies is_ground_cls\ C \implies is_ground_atm\ A$
by (*simp add: is_ground_cls_as_atms*)

lemma *is_ground_cls_is_ground_atms_atms_of[simp]*: $is_ground_cls\ C \implies is_ground_atms\ (atms_of\ C)$
by (*simp add: is_ground_cls_imp_is_ground_atm is_ground_atms_def*)

lemma *grounding_ground*: $C \in grounding_of_cls\ M \implies is_ground_cls\ C$
unfolding *grounding_of_cls_def grounding_of_cls_def* **by** *auto*

lemma *in_subset_eq_grounding_of_cls_is_ground_cls*[simp]:
 $C \in CC \implies CC \subseteq \text{grounding_of_class } DD \implies \text{is_ground_cls } C$
unfolding *grounding_of_cls_def grounding_of_cls_def* **by** *auto*

lemma *is_ground_cls_empty*[simp]: *is_ground_cls* $\{\#\}$
unfolding *is_ground_cls_def* **by** *simp*

lemma *grounding_of_cls_ground*: *is_ground_cls* $C \implies \text{grounding_of_cls } C = \{C\}$
unfolding *grounding_of_cls_def* **by** (*simp add: ex_ground_subst*)

lemma *grounding_of_cls_empty*[simp]: *grounding_of_cls* $\{\#\} = \{\{\#\}\}$
by (*simp add: grounding_of_cls_ground*)

7.3.19 Subsumption

lemma *subsumes_empty_left*[simp]: *subsumes* $\{\#\}$ C
unfolding *subsumes_def subst_cls_def* **by** *simp*

lemma *strictly_subsumes_empty_left*[simp]: *strictly_subsumes* $\{\#\}$ $C \longleftrightarrow C \neq \{\#\}$
unfolding *strictly_subsumes_def subsumes_def subst_cls_def* **by** *simp*

7.3.20 Unifiers

lemma *card_le_one_alt*: *finite* $X \implies \text{card } X \leq 1 \longleftrightarrow X = \{\} \vee (\exists x. X = \{x\})$
by (*induct rule: finite_induct*) *auto*

lemma *is_unifier_subst_atm_eqI*:
assumes *finite* AA
shows *is_unifier* σ $AA \implies A \in AA \implies B \in AA \implies A \cdot a \sigma = B \cdot a \sigma$
unfolding *is_unifier_def subst_atms_def card_le_one_alt*[*OF finite_imageI*][*OF assms*]
by (*metis equals0D imageI insert_iff*)

lemma *is_unifier_alt*:
assumes *finite* AA
shows *is_unifier* σ $AA \longleftrightarrow (\forall A \in AA. \forall B \in AA. A \cdot a \sigma = B \cdot a \sigma)$
unfolding *is_unifier_def subst_atms_def card_le_one_alt*[*OF finite_imageI*][*OF assms(1)*]
by (*rule iffI, metis empty_iff insert_iff insert_image, blast*)

lemma *is_unifiers_subst_atm_eqI*:
assumes *finite* AA *is_unifiers* σ AAA $AA \in AAA$ $A \in AA$ $B \in AA$
shows $A \cdot a \sigma = B \cdot a \sigma$
by (*metis assms is_unifiers_def is_unifier_subst_atm_eqI*)

theorem *is_unifiers_comp*:
is_unifiers σ (*set_mset* ' *set* (*map2 add_mset* As Bs) \cdot *ass* η) \longleftrightarrow
is_unifiers ($\eta \odot \sigma$) (*set_mset* ' *set* (*map2 add_mset* As Bs))
unfolding *is_unifiers_def is_unifier_def subst_atms_def* **by** *auto*

7.3.21 Most General Unifier

lemma *is_mgu_is_unifiers*: *is_mgu* σ $AAA \implies \text{is_unifiers } \sigma$ AAA
using *is_mgu_def* **by** *blast*

lemma *is_mgu_is_most_general*: *is_mgu* σ $AAA \implies \text{is_unifiers } \tau$ $AAA \implies \exists \gamma. \tau = \sigma \odot \gamma$
using *is_mgu_def* **by** *blast*

lemma *is_unifiers_is_unifier*: *is_unifiers* σ $AAA \implies AA \in AAA \implies \text{is_unifier } \sigma$ AA
using *is_unifiers_def* **by** *simp*

7.3.22 Generalization and Subsumption

lemma *variants_iff_subsumes*: *variants* C $D \longleftrightarrow \text{subsumes } C$ $D \wedge \text{subsumes } D$ C
proof

assume *variants* C D
then show *subsumes* C $D \wedge \text{subsumes } D$ C

```

unfolding variants_def generalizes_cls_def subsumes_def by (metis subset_mset.order.refl)
next
assume sub: subsumes C D  $\wedge$  subsumes D C
then have size C = size D
unfolding subsumes_def by (metis antisym size_mset_mono size_subst)
then show variants C D
using sub unfolding subsumes_def variants_def generalizes_cls_def
by (metis leD mset_subset.size size_mset_mono size_subst
      subset_mset.order.not_eq_order.implies_strict)
qed

lemma wf_strictly_generalizes_cls: wfP strictly_generalizes_cls
proof -
  {
assume  $\exists C\_at. \forall i. \text{strictly\_generalizes\_cls } (C\_at (Suc\ i)) (C\_at\ i)$ 
then obtain C_at :: nat  $\Rightarrow$  'a clause where
  sg_C:  $\bigwedge i. \text{strictly\_generalizes\_cls } (C\_at (Suc\ i)) (C\_at\ i)$ 
by blast

define n :: nat where
  n = size (C_at 0)

have sz_C: size (C_at i) = n for i
proof (induct i)
  case (Suc i)
then show ?case
using sg_C[of i] unfolding strictly_generalizes_cls_def generalizes_cls_def subst_cls_def
by (metis size_image_mset)
qed (simp add: n_def)

obtain  $\sigma\_at :: \text{nat} \Rightarrow 's$  where
  C_ $\sigma$ :  $\bigwedge i. \text{image\_mset } (\lambda L. L \cdot l\ \sigma\_at\ i) (C\_at (Suc\ i)) = C\_at\ i$ 
using sg_C[unfolded strictly_generalizes_cls_def generalizes_cls_def subst_cls_def] by metis

define Ls_at :: nat  $\Rightarrow$  'a literal list where
  Ls_at = rec_nat (SOME Ls. mset Ls = C_at 0)
    ( $\lambda i\ Lsi. \text{SOME } Ls. \text{mset } Ls = C\_at (Suc\ i) \wedge \text{map } (\lambda L. L \cdot l\ \sigma\_at\ i) Ls = Lsi$ )

have
  Ls_at_0: Ls_at 0 = (SOME Ls. mset Ls = C_at 0) and
  Ls_at_Suc:  $\bigwedge i. Ls\_at (Suc\ i) =$ 
    (SOME Ls. mset Ls = C_at (Suc i)  $\wedge$  map ( $\lambda L. L \cdot l\ \sigma\_at\ i$ ) Ls = Ls_at i)
unfolding Ls_at_def by simp+

have mset_Lt_at_0: mset (Ls_at 0) = C_at 0
unfolding Ls_at_0 by (rule someI_ex) (metis list_of_mset_ex)

have mset (Ls_at (Suc i)) = C_at (Suc i)  $\wedge$  map ( $\lambda L. L \cdot l\ \sigma\_at\ i$ ) (Ls_at (Suc i)) = Ls_at i
for i
proof (induct i)
  case 0
then show ?case
by (simp add: Ls_at_Suc, rule someI_ex,
      metis C_ $\sigma$  image_mset_of_subset_list mset_Lt_at_0)
next
  case Suc
then show ?case
by (subst (1 2) Ls_at_Suc) (rule someI_ex, metis C_ $\sigma$  image_mset_of_subset_list)
qed
note mset_Ls = this[THEN conjunct1] and Ls_ $\sigma$  = this[THEN conjunct2]

have len_Ls:  $\bigwedge i. \text{length } (Ls\_at\ i) = n$ 
by (metis mset_Ls mset_Lt_at_0 not0.implies_Suc size_mset sz_C)

```

```

have is_pos_Ls:  $\bigwedge i j. j < n \implies is\_pos (Ls\_at (Suc i) ! j) \longleftrightarrow is\_pos (Ls\_at i ! j)$ 
  using Ls_σ len_Ls by (metis literal.map_disc_iff nth_map subst_lit_def)

have Ls_τ_strict_lit:  $\bigwedge i \tau. map (\lambda L. L \cdot l \tau) (Ls\_at i) \neq Ls\_at (Suc i)$ 
  by (metis C_σ mset_Ls Ls_σ mset_map sg_C generalizes_cls_def strictly_generalizes_cls_def
    subst_cls_def)

have Ls_τ_strict_tm:
   $map ((\lambda t. t \cdot a \tau) \circ atm\_of) (Ls\_at i) \neq map atm\_of (Ls\_at (Suc i))$  for i  $\tau$ 
proof –
  obtain j :: nat where
    j_lt: j < n and
    j_τ:  $Ls\_at i ! j \cdot l \tau \neq Ls\_at (Suc i) ! j$ 
    using Ls_τ_strict_lit[of  $\tau$  i] len_Ls
    by (metis (no_types, lifting) length_map list_eq_iff_nth_eq nth_map)

  have  $atm\_of (Ls\_at i ! j) \cdot a \tau \neq atm\_of (Ls\_at (Suc i) ! j)$ 
    using j_τ is_pos_Ls[OF j_lt]
    by (metis (mono_guards) literal.expand literal.map_disc_iff literal.map_sel subst_lit_def)
  then show ?thesis
    using j_lt len_Ls by (metis nth_map o_apply)
qed

define tm_at :: nat  $\Rightarrow$  'a where
   $\bigwedge i. tm\_at i = atm\_of\_atms (map atm\_of (Ls\_at i))$ 

have  $\bigwedge i. generalizes\_atm (tm\_at (Suc i)) (tm\_at i)$ 
  unfolding tm_at_def generalizes_atm_def atm_of_atms_subst
  using Ls_σ[THEN arg_cong, of map atm_of] by (auto simp: comp_def)
moreover have  $\bigwedge i. \neg generalizes\_atm (tm\_at i) (tm\_at (Suc i))$ 
  unfolding tm_at_def generalizes_atm_def atm_of_atms_subst by (simp add: Ls_τ_strict_tm)
ultimately have  $\bigwedge i. strictly\_generalizes\_atm (tm\_at (Suc i)) (tm\_at i)$ 
  unfolding strictly_generalizes_atm_def by blast
then have False
  using wf_strictly_generalizes_atm[unfolded wfP_def wf_iff_no_infinite_down_chain] by blast
}
then show wfP (strictly_generalizes_cls :: 'a clause  $\Rightarrow$  -  $\Rightarrow$  -)
  unfolding wfP_def by (blast intro: wf_iff_no_infinite_down_chain[THEN iffD2])
qed

lemma strict_subset_subst_strictly_subsumes:
  assumes cη_sub:  $C \cdot \eta \subset\# D$ 
  shows strictly_subsumes C D
  by (metis cη_sub leD mset_subset_size size_mset_mono size_subst strictly_subsumes_def
    subset_mset.dual_order.strict_implies_order substitution_ops.subsumes_def)

lemma subsumes_trans:  $subsumes C D \implies subsumes D E \implies subsumes C E$ 
  unfolding subsumes_def
  by (metis (no_types) subset_mset.order.trans subst_cls_comp_subst subst_cls_mono_mset)

lemma subset_strictly_subsumes:  $C \subset\# D \implies strictly\_subsumes C D$ 
  using strict_subset_subst_strictly_subsumes[of C id_subst] by auto

lemma strictly_subsumes_neq:  $strictly\_subsumes D' D \implies D' \neq D \cdot \sigma$ 
  unfolding strictly_subsumes_def subsumes_def by blast

lemma strictly_subsumes_has_minimum:
  assumes CC  $\neq \{\}$ 
  shows  $\exists C \in CC. \forall D \in CC. \neg strictly\_subsumes D C$ 
proof (rule ccontr)
  assume  $\neg (\exists C \in CC. \forall D \in CC. \neg strictly\_subsumes D C)$ 
  then have  $\forall C \in CC. \exists D \in CC. strictly\_subsumes D C$ 

```

```

  by blast
then obtain f where
  f-p:  $\forall C \in CC. f C \in CC \wedge \text{strictly\_subsumes } (f C) C$ 
  by metis
from assms obtain C where
  C-p:  $C \in CC$ 
  by auto

define c :: nat  $\Rightarrow$  'a clause where
   $\wedge n. c n = (f \wedge^n) C$ 

have incc:  $c i \in CC$  for i
  by (induction i) (auto simp: c_def f-p C-p)
have ps:  $\forall i. \text{strictly\_subsumes } (c (Suc i)) (c i)$ 
  using incc f-p unfolding c_def by auto
have  $\forall i. \text{size } (c i) \geq \text{size } (c (Suc i))$ 
  using ps unfolding strictly_subsumes_def subsumes_def by (metis size_mset_mono size_subst)
then have lte:  $\forall i. (\text{size} \circ c) i \geq (\text{size} \circ c) (Suc i)$ 
  unfolding comp_def .
then have  $\exists l. \forall l' \geq l. \text{size } (c l') = \text{size } (c (Suc l'))$ 
  using f_Suc_decr_eventually_const comp_def by auto
then obtain l where
  l-p:  $\forall l' \geq l. \text{size } (c l') = \text{size } (c (Suc l'))$ 
  by metis
then have  $\forall l' \geq l. \text{strictly\_generalizes\_cls } (c (Suc l')) (c l')$ 
  using ps unfolding strictly_generalizes_cls_def generalizes_cls_def
  by (metis size_subst less_irrefl strictly_subsumes_def mset_subset_size
    subset_mset_def subsumes_def strictly_subsumes_neq)
then have  $\forall i. \text{strictly\_generalizes\_cls } (c (Suc i + l)) (c (i + l))$ 
  unfolding strictly_generalizes_cls_def generalizes_cls_def by auto
then have  $\exists f. \forall i. \text{strictly\_generalizes\_cls } (f (Suc i)) (f i)$ 
  by (rule exI[of _  $\lambda x. c (x + l)$ ])
then show False
  using wf_strictly_generalizes_cls
  wf_iff_no_infinite_down_chain[of  $\{(x, y). \text{strictly\_generalizes\_cls } x y\}$ ]
  unfolding wfP_def by auto
qed

end

```

7.4 Most General Unifiers

```

locale mgu = substitution subst_atm id_subst comp_subst atm_of_atms renamings_apart
for
  subst_atm :: 'a  $\Rightarrow$  's  $\Rightarrow$  'a and
  id_subst :: 's and
  comp_subst :: 's  $\Rightarrow$  's  $\Rightarrow$  's and
  atm_of_atms :: 'a list  $\Rightarrow$  'a and
  renamings_apart :: 'a literal multiset list  $\Rightarrow$  's list +
fixes
  mgu :: 'a set set  $\Rightarrow$  's option
assumes
  mgu_sound:  $\text{finite } AAA \Longrightarrow (\forall AA \in AAA. \text{finite } AA) \Longrightarrow \text{mgu } AAA = \text{Some } \sigma \Longrightarrow \text{is\_mgu } \sigma \text{ } AAA$  and
  mgu_complete:
     $\text{finite } AAA \Longrightarrow (\forall AA \in AAA. \text{finite } AA) \Longrightarrow \text{is\_unifiers } \sigma \text{ } AAA \Longrightarrow \exists \tau. \text{mgu } AAA = \text{Some } \tau$ 
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:  $\text{length } As = n$  and
    aaslen:  $\text{length } AAs = n$  and

```



```

    mgu: Some  $\sigma = \text{mgu } (\text{set\_mset } ' \text{ set } (\text{map2 } \text{add\_mset } \text{As } \text{AAs}))$  and
    i_lt:  $i < n$  and
    a_in:  $A \in \# \text{AAs } ! i$ 
  shows  $A \cdot a \sigma = \text{As } ! i \cdot a \sigma$ 
proof -
  from mgu have is_mgu  $\sigma$  (set_mset ' set (map2 add_mset As AAs))
    using mgu_sound by auto
  then have is_unifiers  $\sigma$  (set_mset ' set (map2 add_mset As AAs))
    using is_mgu_is_unifiers by auto
  then have is_unifier  $\sigma$  (set_mset (add_mset (As ! i) (AAs ! i)))
    using i_lt aslen aaslen unfolding is_unifiers_def is_unifier_def
    by simp (metis length_zip min.idem nth_mem nth_zip prod.case set_mset_add_mset_insert)
  then show ?thesis
    using aslen aaslen a_in is_unifier_subst_atm_eqI
    by (metis finite_set_mset insertCI set_mset_add_mset_insert)
qed

end

end

```

8 Refutational Inference Systems

```

theory Inference_System
  imports Herbrand.Interpretation
begin

```

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

8.1 Preliminaries

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

```

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

```

```

abbreviation prem_of :: 'a inference  $\Rightarrow$  'a clause multiset where
  prem_of  $\gamma \equiv \text{side\_prem\_of } \gamma + \{\#\text{main\_prem\_of } \gamma\}$ 

```

```

abbreviation concls_of :: 'a inference set  $\Rightarrow$  'a clause set where
  concls_of  $\Gamma \equiv \text{concl\_of } ' \Gamma$ 

```

```

definition infer_from :: 'a clause set  $\Rightarrow$  'a inference  $\Rightarrow$  bool where
  infer_from  $CC \ \gamma \iff \text{set\_mset } (\text{prems\_of } \gamma) \subseteq CC$ 

```

```

locale inference_system =
  fixes  $\Gamma :: 'a \text{ inference set}$ 
begin

```

```

definition inferences_from :: 'a clause set  $\Rightarrow$  'a inference set where
  inferences_from  $CC = \{\gamma. \gamma \in \Gamma \wedge \text{infer\_from } CC \ \gamma\}$ 

```

```

definition inferences_between :: 'a clause set  $\Rightarrow$  'a clause  $\Rightarrow$  'a inference set where
  inferences_between  $CC \ C = \{\gamma. \gamma \in \Gamma \wedge \text{infer\_from } (CC \cup \{C\}) \ \gamma \wedge C \in \# \text{prems\_of } \gamma\}$ 

```

```

lemma inferences_from_mono:  $CC \subseteq DD \implies \text{inferences\_from } CC \subseteq \text{inferences\_from } DD$ 
  unfolding inferences_from_def infer_from_def by fast

```

```

definition saturated :: 'a clause set  $\Rightarrow$  bool where
  saturated  $N \iff \text{concls\_of } (\text{inferences\_from } N) \subseteq N$ 

```

```

lemma saturatedD:
  assumes
    satur: saturated N and
    inf: Infer CC D E ∈ Γ and
    cc_subs_n: set_mset CC ⊆ N and
    d_in_n: D ∈ N
  shows E ∈ N
proof –
  have Infer CC D E ∈ inferences_from N
  unfolding inferences_from_def infer_from_def using inf cc_subs_n d_in_n by simp
  then have E ∈ concls_of (inferences_from N)
  unfolding image_iff by (metis inference.sel(3))
  then show E ∈ N
  using satur unfolding saturated_def by blast
qed

```

end

Satisfiability preservation is a weaker requirement than soundness.

```

locale sat_preserving_inference_system = inference_system +
  assumes Γ_sat_preserving: satisfiable N ⇒ satisfiable (N ∪ concls_of (inferences_from N))

```

```

locale sound_inference_system = inference_system +
  assumes Γ_sound: Infer CC D E ∈ Γ ⇒ I ⊨m CC ⇒ I ⊨ D ⇒ I ⊨ E
begin

```

```

lemma Γ_sat_preserving:
  assumes sat_n: satisfiable N
  shows satisfiable (N ∪ concls_of (inferences_from N))
proof –
  obtain I where i: I ⊨s N
  using sat_n by blast
  then have ∧ CC D E. Infer CC D E ∈ Γ ⇒ set_mset CC ⊆ N ⇒ D ∈ N ⇒ I ⊨ E
  using Γ_sound unfolding true_cls_def true_cls_mset_def by (simp add: subset_eq)
  then have ∧ γ. γ ∈ Γ ⇒ infer_from N γ ⇒ I ⊨ concl_of γ
  unfolding infer_from_def by (case_tac γ) clarsimp
  then have I ⊨s concls_of (inferences_from N)
  unfolding inferences_from_def image_def true_cls_def infer_from_def by blast
  then have I ⊨s N ∪ concls_of (inferences_from N)
  using i by simp
  then show ?thesis
  by blast
qed

```

```

sublocale sat_preserving_inference_system
  by unfold_locales (erule Γ_sat_preserving)

```

end

```

locale reductive_inference_system = inference_system Γ for Γ :: ('a :: wellorder) inference set +
  assumes Γ_reductive: γ ∈ Γ ⇒ concl_of γ < main_prem_of γ

```

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 Γ for Γ :: ('a :: wellorder) inference set +
  fixes L_of :: 'a clause set ⇒ 'a interp
  assumes Γ_counterex_reducing:
    {#} ∉ N ⇒ D ∈ N ⇒ ¬ L_of N ⊨ D ⇒ (∧ C. C ∈ N ⇒ ¬ L_of N ⊨ C ⇒ D ≤ C) ⇒
    ∃ CC E. set_mset CC ⊆ N ∧ L_of N ⊨m CC ∧ Infer CC D E ∈ Γ ∧ ¬ L_of N ⊨ E ∧ E < D

```

```

begin
lemma ex_min_counterex:
  fixes  $N :: ('a :: wellorder) \text{ clause set}$ 
  assumes  $\neg I \models N$ 
  shows  $\exists C \in N. \neg I \models C \wedge (\forall D \in N. D < C \longrightarrow I \models D)$ 
proof -
  obtain  $C$  where  $C \in N$  and  $\neg I \models C$ 
  using assms unfolding true_cls_def by auto
  then have  $c\_in: C \in \{C \in N. \neg I \models C\}$ 
  by blast
  show ?thesis
  using wf_eq_minimal[THEN iffD1, rule_format, OF wf_less_multiset c_in] by blast
qed

```

```

theorem saturated_model:
  assumes
    satur: saturated N and
    ec_ni_n: \{\#\} \notin N
  shows  $L\ of\ N \models N$ 
proof -
  have ec_ni_n: \{\#\} \notin N
  using ec_ni_n by auto

  {
    assume  $\neg L\ of\ N \models N$ 
    then obtain  $D$  where
      d_in_n: D \in N and
      d_cex: \neg L\ of\ N \models D and
      d_min: \bigwedge C. C \in N \implies C < D \implies L\ of\ N \models C
      by (meson ex_min_counterex)
    then obtain  $CC\ E$  where
      cc_subs_n: set_mset CC \subseteq N and
      inf_e: Infer CC D E \in \Gamma and
      e_cex: \neg L\ of\ N \models E and
      e_lt_d: E < D
      using  $\Gamma\_counterex\_reducing[OF\ ec\_ni\_n]$  not_less by metis
    from cc_subs_n inf_e have  $E \in N$ 
      using d_in_n satur by (blast dest: saturatedD)
    then have False
      using e_cex e_lt_d d_min not_less by blast
  }
  then show ?thesis
  by satx
qed

```

Cf. Corollary 3.10:

```

corollary saturated_complete:  $saturated\ N \implies \neg\ satisfiable\ N \implies \{\#\} \in N$ 
  using saturated_model by blast

```

end

8.3 Compactness

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

```

context inference_system
begin

```

```

inductive-set saturate :: 'a clause set  $\Rightarrow$  'a clause set' for  $CC ::$  'a clause set' where

```

base: $C \in CC \implies C \in \text{saturate } CC$
| *step*: $\text{Infer } CC' D E \in \Gamma \implies (\bigwedge C'. C' \in \# CC' \implies C' \in \text{saturate } CC) \implies D \in \text{saturate } CC \implies E \in \text{saturate } CC$

lemma *saturate_mono*: $C \in \text{saturate } CC \implies CC \subseteq DD \implies C \in \text{saturate } DD$
by (*induct rule*: *saturate.induct*) (*auto intro*: *saturate.intros*)

lemma *saturated_saturate*[*simp*, *intro*]: *saturated* (*saturate N*)
unfolding *saturated_def inferences_from_def infer_from_def image_def*
by *clarify* (*rename_tac x*, *case_tac x*, *auto elim!*: *saturate.step*)

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

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

case (*base C*)

then have $\{C\} \subseteq CC$ **and** *finite* $\{C\}$ **and** $C \in \text{saturate } \{C\}$

by (*auto intro*: *saturate.intros*)

then show *?case*

by *blast*

next

case (*step CC' D E*)

obtain *DD_of* **where**

$\bigwedge C. C \in \# CC' \implies DD_of C \subseteq CC \wedge \text{finite } (DD_of C) \wedge C \in \text{saturate } (DD_of C)$

using *step(3)* **by** *metis*

then have

$(\bigcup C \in \text{set_mset } CC'. DD_of C) \subseteq CC$

finite $(\bigcup C \in \text{set_mset } CC'. DD_of C) \wedge \text{set_mset } CC' \subseteq \text{saturate } (\bigcup C \in \text{set_mset } CC'. DD_of C)$

by (*auto intro*: *saturate_mono*)

then obtain *DD* **where**

d_sub: $DD \subseteq CC$ **and** *d_fin*: *finite* *DD* **and** *in_sat_d*: $\text{set_mset } CC' \subseteq \text{saturate } DD$

by *blast*

obtain *EE* **where**

e_sub: $EE \subseteq CC$ **and** *e_fin*: *finite* *EE* **and** *in_sat_ee*: $D \in \text{saturate } EE$

using *step(5)* **by** *blast*

have $DD \cup EE \subseteq CC$

using *d_sub e_sub step(1)* **by** *fast*

moreover have *finite* $(DD \cup EE)$

using *d_fin e_fin* **by** *fast*

moreover have $E \in \text{saturate } (DD \cup EE)$

using *in_sat_d in_sat_ee step.hyps(1)*

by (*blast intro*: *inference_system.saturate.step saturate_mono*)

ultimately show *?case*

by *blast*

qed

end

context *sound_inference_system*

begin

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

by (*induct rule*: *saturate.induct*) (*auto simp*: *true_cls_mset_def true_cls_def* *Γ_sound*)

end

context *sat_preserving_inference_system*

begin

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

theorem *saturate_sat_preserving*: *satisfiable* $CC \implies \text{satisfiable } (\text{saturate } CC)$

oops

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  $\neg$  satisfiable N  $\longleftrightarrow$  ( $\exists$  DD  $\subseteq$  N. finite DD  $\wedge$   $\neg$  satisfiable DD)  
proof  
  assume  $\neg$  satisfiable N  
  then have  $\{\#\} \in$  saturate N  
    using saturated_complete saturated_saturate saturate_base unfolding true_cls_def by meson  
  then have  $\exists$  DD  $\subseteq$  N. finite DD  $\wedge$   $\{\#\} \in$  saturate DD  
    using saturate_finite by fastforce  
  then show  $\exists$  DD  $\subseteq$  N. finite DD  $\wedge$   $\neg$  satisfiable DD  
    using saturate_sound by auto  
next  
  assume  $\exists$  DD  $\subseteq$  N. finite DD  $\wedge$   $\neg$  satisfiable DD  
  then show  $\neg$  satisfiable N  
    by (blast intro: true_cls_mono)  
qed
```

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 $\lambda.$ $\{\}$ as instantiation for the S parameter.

```
locale selection =  
  fixes S :: 'a clause  $\Rightarrow$  'a clause  
  assumes  
    S_selects_subseteq:  $S$  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) clause  $\Rightarrow$  'a clause  
begin
```

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

```
context  
  fixes N :: 'a clause set  
begin
```

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

by auto
termination by (rule termination[OF wf, simplified])

declare production.simps [simp del]

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

lemma production_unfold:
 production C = {A. C \in N \wedge C \neq {#} \wedge Max_mset C = Pos A \wedge \neg interp C \models C \wedge S C = {#}}
unfolding interp_def **by** (rule production.simps)

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

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

lemma producesD: produces C A \implies C \in N \wedge C \neq {#} \wedge Pos A = Max_mset C \wedge \neg interp C \models C \wedge S C = {#}
unfolding production_unfold **by** auto

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

lemma interp_subseteq_Interp[simp]: interp C \subseteq Interp C
by (simp add: Interp_def)

lemma Interp_as_UNION: Interp C = ($\bigcup D \in \{D. D \leq C\}. \text{production } D$)
unfolding Interp_def interp_def less_eq_multiset_def **by** fast

lemma productive_not_empty: productive C \implies C \neq {#}
unfolding production_unfold **by** simp

lemma productive_imp_produces_Max_literal: productive C \implies produces C (atm_of (Max_mset C))
unfolding production_unfold **by** (auto simp del: atm_of_Max_lit)

lemma productive_imp_produces_Max_atom: productive C \implies produces C (Max (atms_of C))
unfolding atms_of_def Max_atm_of_set_mset_commute[OF productive_not_empty]
by (rule productive_imp_produces_Max_literal)

lemma produces_imp_Max_literal: produces C A \implies A = atm_of (Max_mset C)
using productive_imp_produces_Max_literal **by** auto

lemma produces_imp_Max_atom: produces C A \implies A = Max (atms_of C)
using producesD produces_imp_Max_literal **by** auto

lemma produces_imp_Pos_in_lits: produces C A \implies Pos A \in # C
by (simp add: producesD)

lemma productive_in_N: productive C \implies C \in N
unfolding production_unfold **by** simp

lemma produces_imp_atms_leq: produces C A \implies B \in atms_of C \implies B \leq A
using Max.coboundedI produces_imp_Max_atom **by** blast

lemma produces_imp_neg_notin_lits: produces C A \implies \neg Neg A \in # C
by (simp add: pos_Max_imp_neg_notin producesD)

lemma less_eq_imp_interp_subseteq_interp: C \leq D \implies interp C \subseteq interp D
unfolding interp_def **by** auto (metis order.strict_trans2)

lemma less_eq_imp_interp_subseteq_Interp: C \leq D \implies interp C \subseteq Interp D
unfolding Interp_def **using** less_eq_imp_interp_subseteq_interp **by** blast

lemma *less_imp_production_subseteq_interp*: $C < D \implies \text{production } C \subseteq \text{interp } D$
unfolding *interp_def* **by** *fast*

lemma *less_eq_imp_production_subseteq_Interp*: $C \leq D \implies \text{production } C \subseteq \text{Interp } D$
unfolding *Interp_def* **using** *less_imp_production_subseteq_interp*
by (*metis le_imp_less_or_eq le_supI1 sup_ge2*)

lemma *less_imp_Interp_subseteq_interp*: $C < D \implies \text{Interp } C \subseteq \text{interp } D$
by (*simp add: Interp_def less_eq_imp_interp_subseteq_interp less_imp_production_subseteq_interp*)

lemma *less_eq_imp_Interp_subseteq_Interp*: $C \leq D \implies \text{Interp } C \subseteq \text{Interp } D$
using *Interp_def less_eq_imp_interp_subseteq_Interp less_eq_imp_production_subseteq_Interp* **by** *auto*

lemma *not_Interp_to_interp_imp_less*: $A \notin \text{Interp } C \implies A \in \text{interp } D \implies C < D$
using *less_eq_imp_interp_subseteq_Interp not_less* **by** *blast*

lemma *not_interp_to_interp_imp_less*: $A \notin \text{interp } C \implies A \in \text{interp } D \implies C < D$
using *less_eq_imp_interp_subseteq_interp not_less* **by** *blast*

lemma *not_Interp_to_Interp_imp_less*: $A \notin \text{Interp } C \implies A \in \text{Interp } D \implies C < D$
using *less_eq_imp_Interp_subseteq_Interp not_less* **by** *blast*

lemma *not_interp_to_Interp_imp_le*: $A \notin \text{interp } C \implies A \in \text{Interp } D \implies C \leq D$
using *less_imp_Interp_subseteq_interp not_less* **by** *blast*

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

lemma *interp_subseteq_INTERP*: $\text{interp } C \subseteq \text{INTERP}$
unfolding *interp_def INTERP_def* **by** (*auto simp: production_unfold*)

lemma *production_subseteq_INTERP*: $\text{production } C \subseteq \text{INTERP}$
unfolding *INTERP_def* **using** *production_unfold* **by** *blast*

lemma *Interp_subseteq_INTERP*: $\text{Interp } C \subseteq \text{INTERP}$
by (*simp add: Interp_def interp_subseteq_INTERP production_subseteq_INTERP*)

lemma *produces_imp_in_interp*:
assumes *a_in_c*: $\text{Neg } A \in \# C$ **and** *d*: *produces* $D A$
shows $A \in \text{interp } C$
by (*metis Interp_def Max_pos_neg_less_multiset UnCI a_in_c d not_interp_to_Interp_imp_le not_less producesD singletonI*)

lemma *neg_notin_Interp_not_produce*: $\text{Neg } A \in \# C \implies A \notin \text{Interp } D \implies C \leq D \implies \neg \text{produces } D'' A$
using *less_eq_imp_interp_subseteq_Interp produces_imp_in_interp* **by** *blast*

lemma *in_production_imp_produces*: $A \in \text{production } C \implies \text{produces } C A$
using *productive_imp_produces_Max_atom* **by** *fastforce*

lemma *not_produces_imp_notin_production*: $\neg \text{produces } C A \implies A \notin \text{production } C$
using *in_production_imp_produces* **by** *blast*

lemma *not_produces_imp_notin_interp*: $(\bigwedge D. \neg \text{produces } D A) \implies A \notin \text{interp } C$
unfolding *interp_def* **by** (*fast intro!: in_production_imp_produces*)

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: $\text{Interp } D \models C$ **and**
subs: $\text{interp } D' \subseteq (\bigcup C \in CC. \text{production } C)$

shows $(\bigcup C \in CC. \text{production } C) \models C$
proof (cases $\exists A. \text{Pos } A \in \# C \wedge A \in \text{Interp } D$)
case *True*
then obtain A **where** $a_{in_c}: \text{Pos } A \in \# C$ **and** $a_{at_d}: A \in \text{Interp } D$
by *blast*
from a_{at_d} **have** $A \in \text{interp } D'$
using $d_{lt_d'}$ *less_imp_Interp_subseteq_interp* **by** *blast*
then show *?thesis*
using $subs\ a_{in_c}$ **by** (*blast dest: contra_subsetD*)
next
case *False*
then obtain A **where** $a_{in_c}: \text{Neg } A \in \# C$ **and** $A \notin \text{Interp } D$
using c_{at_d} *unfolding true_cls_def* **by** *blast*
then have $\bigwedge D''. \neg \text{produces } D'' A$
using c_{le_d} *neg_notin_Interp_not_produce* **by** *simp*
then show *?thesis*
using $a_{in_c}\ subs\ not_produces_imp_notin_production$ **by** *auto*
qed

lemma *Interp_imp_interp*: $C \leq D \implies D < D' \implies \text{Interp } D \models C \implies \text{interp } D' \models C$
using *interp_def Interp_imp_general* **by** *simp*

lemma *Interp_imp_Interp*: $C \leq D \implies D \leq D' \implies \text{Interp } D \models C \implies \text{Interp } D' \models C$
using *Interp_as_UNION interp_subseteq_Interp Interp_imp_general* **by** (*metis antisym_conv2*)

lemma *Interp_imp_INTERP*: $C \leq D \implies \text{Interp } D \models C \implies \text{INTERP} \models C$
using *INTERP_def interp_subseteq_INTERP Interp_imp_general[OF le_multiset_right_total]* **by** *simp*

lemma *interp_imp_general*:
assumes
 $c_{le_d}: C \leq D$ **and**
 $d_{le_d'}: D \leq D'$ **and**
 $c_{at_d}: \text{interp } D \models C$ **and**
 $subs: \text{interp } D' \subseteq (\bigcup C \in CC. \text{production } C)$
shows $(\bigcup C \in CC. \text{production } C) \models C$
proof (cases $\exists A. \text{Pos } A \in \# C \wedge A \in \text{interp } D$)
case *True*
then obtain A **where** $a_{in_c}: \text{Pos } A \in \# C$ **and** $a_{at_d}: A \in \text{interp } D$
by *blast*
from a_{at_d} **have** $A \in \text{interp } D'$
using $d_{le_d'}$ *less_eq_imp_interp_subseteq_interp* **by** *blast*
then show *?thesis*
using $subs\ a_{in_c}$ **by** (*blast dest: contra_subsetD*)
next
case *False*
then obtain A **where** $a_{in_c}: \text{Neg } A \in \# C$ **and** $A \notin \text{interp } D$
using c_{at_d} *unfolding true_cls_def* **by** *blast*
then have $\bigwedge D''. \neg \text{produces } D'' A$
using c_{le_d} **by** (*auto dest: produces_imp_in_interp less_eq_imp_interp_subseteq_interp*)
then show *?thesis*
using $a_{in_c}\ subs\ not_produces_imp_notin_production$ **by** *auto*
qed

lemma *interp_imp_interp*: $C \leq D \implies D \leq D' \implies \text{interp } D \models C \implies \text{interp } D' \models C$
using *interp_def interp_imp_general* **by** *simp*

lemma *interp_imp_Interp*: $C \leq D \implies D \leq D' \implies \text{interp } D \models C \implies \text{Interp } D' \models C$
using *Interp_as_UNION interp_subseteq_Interp[of D'] interp_imp_general* **by** *simp*

lemma *interp_imp_INTERP*: $C \leq D \implies \text{interp } D \models C \implies \text{INTERP} \models C$
using *INTERP_def interp_subseteq_INTERP interp_imp_general linear* **by** *metis*

lemma *productive_imp_not_interp*: $\text{productive } C \implies \neg \text{interp } C \models C$

unfolding *production_unfold* by *simp*

This corresponds to Lemma 3.3:

lemma *productive_imp_Interp*:
 assumes *productive C*
 shows *Interp C* \models *C*
proof –
 obtain *A* **where** *a*: *produces C A*
 using *assms productive_imp_produces_Max_atom* **by** *blast*
 then have *a.in.c*: *Pos A* \in $\#$ *C*
 by (*rule produces_imp_Pos.in.lits*)
 moreover have *A* \in *Interp C*
 using *a less_eq_imp_production_subseteq_Interp* **by** *blast*
 ultimately show *?thesis*
 by *fast*
qed

lemma *productive_imp_INTERP*: *productive C* \implies *INTERP* \models *C*
 by (*fast intro: productive_imp_Interp Interp_imp_INTERP*)

This corresponds to Lemma 3.5:

lemma *max_pos_imp_Interp*:
 assumes *C* \in *N* **and** *C* \neq $\{\#\}$ **and** *Max.mset C* = *Pos A* **and** *S C* = $\{\#\}$
 shows *Interp C* \models *C*
proof (*cases productive C*)
 case *True*
 then show *?thesis*
 by (*fast intro: productive_imp_Interp*)
next
 case *False*
 then have *interp C* \models *C*
 using *assms unfolding production_unfold* **by** *simp*
 then show *?thesis*
 unfolding *Interp_def* **using** *False* **by** *auto*
qed

The following results correspond to Lemma 3.6:

lemma *max_atm_imp_Interp*:
 assumes
 c.in.n: *C* \in *N* **and**
 pos.in: *Pos A* \in $\#$ *C* **and**
 max_atm: *A* = *Max (atms.of C)* **and**
 s.c.e: *S C* = $\{\#\}$
 shows *Interp C* \models *C*
proof (*cases Neg A* \in $\#$ *C*)
 case *True*
 then show *?thesis*
 using *pos.in pos_neg_in_imp_true* **by** *metis*
next
 case *False*
 moreover have *ne*: *C* \neq $\{\#\}$
 using *pos.in* **by** *auto*
 ultimately have *Max.mset C* = *Pos A*
 using *max_atm* **using** *Max.in.lits Max_lit_eq_pos_or_neg_Max_atm* **by** *metis*
 then show *?thesis*
 using *ne c.in.n s.c.e* **by** (*blast intro: max_pos_imp_Interp*)
qed

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 *Max (atms.of D')* < *Max (atms.of D)* **and**
 d'.at.d: \neg *Interp D* \models *D'* **and**

$d.lt.c: D < C$ and
 $subs: interp\ C \subseteq (\bigcup C \in CC. production\ C)$
shows $\neg (\bigcup C \in CC. production\ C) \models D'$
proof –
{
 assume $cc_blw_d': (\bigcup C \in CC. production\ C) \models D'$
 have $Interp\ D \subseteq (\bigcup C \in CC. production\ C)$
 using $less_imp_Interp_subsetq_interp\ d.lt.c\ subs$ **by** $blast$
 then obtain A **where** $a_in_d': Pos\ A \in \# D'$ **and** $a_blw_cc: A \in (\bigcup C \in CC. production\ C)$
 using $cc_blw_d'\ d'_at_d\ false_to_true_imp_ex_pos$ **by** $metis$
 from a_in_d' **have** $a_at_d: A \notin Interp\ D$
 using d'_at_d **by** $fast$
 from a_blw_cc **obtain** C' **where** $prod_c': production\ C' = \{A\}$
 by $(fast\ intro!: in_production_imp_produces)$
 have $max_c': Max\ (atms_of\ C') = A$
 using $prod_c'\ productive_imp_produces_Max_atom$ **by** $force$
 have $leq_dc': D \leq C'$
 using $a_at_d\ d'_at_d\ prod_c'$ **by** $(auto\ simp: Interp_def\ intro: not_interp_to_Interp_imp_le)$
 then have $D' \leq C'$
 using $d'_le_d\ order_trans$ **by** $blast$
 then have $max_d': Max\ (atms_of\ D') = A$
 using $a_in_d'\ max_c'$ **by** $(fast\ intro: pos_lit_in_atms_of\ le_multiset_Max_in_imp_Max)$

 {
 assume $D' \in N \wedge S\ D' = \{\#\}$
 then have $Interp\ D' \models D'$
 using $a_in_d'\ max_d'$ **by** $(blast\ intro: max_atm_imp_Interp)$
 then have $Interp\ D \models D'$
 using d'_le_d **by** $(auto\ intro: Interp_imp_Interp\ simp: less_eq_multiset_def)$
 then have $False$
 using d'_at_d **by** $satx$
 }
 moreover
 {
 assume $Max\ (atms_of\ D') < Max\ (atms_of\ D)$
 then have $False$
 using $max_d'\ leq_dc'\ max_c'\ d'_le_d$
 by $(metis\ le_imp_less_or_eq\ le_multiset_empty_right\ less_eq_Max_atms_of\ less_imp_not_less)$
 }
 ultimately have $False$
 using $in_n_or_max_gt$ **by** $satx$
}
then show $?thesis$
 by $satx$
qed

lemma $not_Interp_imp_not_interp:$

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

using $interp_def\ not_Interp_imp_general$ **by** $simp$

lemma $not_Interp_imp_not_Interp:$

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

using $Interp_as_UNION\ interp_subsetq_Interp\ not_Interp_imp_general$ **by** $metis$

lemma $not_Interp_imp_not_INTERP:$

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

using $INTERP_def\ interp_subsetq_INTERP\ not_Interp_imp_general[OF\ _ _ _ le_multiset_right_total]$
by $simp$

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

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** $\neg \text{interp } D \models C$
using n **unfolding** d c interp_def **by** auto

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** $S = \lambda_. \{\#\}$
by *unfold_locales auto*

inductive *unord_resolve* :: ‘a clause \Rightarrow ‘a clause \Rightarrow ‘a clause \Rightarrow bool **where**
 $\text{unord_resolve } (C + \text{replicate_mset } (\text{Suc } n) (\text{Pos } A)) (\text{add_mset } (\text{Neg } A) D) (C + D)$

lemma *unord_resolve_sound*: $\text{unord_resolve } C D E \implies I \models C \implies I \models D \implies I \models E$
using *unord_resolve.cases* **by** *fastforce*

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
 $\text{ec_ni_n}: \{\#\} \notin N$ **and**
 $\text{c_in_n}: C \in N$ **and**
 $\text{c_ce}: \neg \text{INTERP } N \models C$ **and**
 $\text{c_min}: \bigwedge D. D \in N \implies \neg \text{INTERP } N \models D \implies C \leq D$
obtains $D E$ **where**
 $D \in N$
 $\text{INTERP } N \models D$
 $\text{productive } N D$
 $\text{unord_resolve } D C E$
 $\neg \text{INTERP } N \models E$
 $E < C$

proof –

have $\text{c_ne}: C \neq \{\#\}$
using c_in_n ec_ni_n **by** *blast*
have $\exists A. A \in \text{atms_of } C \wedge A = \text{Max } (\text{atms_of } C)$
using c_ne **by** (*blast intro: Max_in_lits atm_of_Max_lit atm_of_lit_in_atms_of*)

then have $\exists A. \text{Neg } A \in\# C$
using *c_ne c.in.n c.cex c.min Max.in.lits Max.lit.eq_pos_or_neg_Max.atm max_pos_imp_Interp*
Interp_imp_INTERP **by** *metis*
then obtain A **where** $\text{neg_a_in_c}: \text{Neg } A \in\# C$
by *blast*
then obtain C' **where** $c: C = \text{add_mset } (\text{Neg } A) C'$
using *insert_DiffM* **by** *metis*
have $A \in \text{INTERP } N$
using *neg_a_in_c c.cex[unfolded true_cls_def]* **by** *fast*
then obtain D **where** $d0: \text{produces } N D A$
unfolding *INTERP_def* **by** (*metis UN_E not_produces_imp_notin_production*)
have $\text{prod_d}: \text{productive } N D$
unfolding $d0$ **by** *simp*
then have $d.in.n: D \in N$
using *productive_in_N* **by** *fast*
have $d.true: \text{INTERP } N \models D$
using *prod_d productive_imp_INTERP* **by** *blast*

obtain D' AAA **where**
 $d: D = D' + AAA$ **and**
 $d': D' = \{\#L \in\# D. L \neq \text{Pos } A\# \}$ **and**
 $aa: AAA = \{\#L \in\# D. L = \text{Pos } A\# \}$
using *multiset_partition_union_commute* **by** *metis*
have $d'_subs: \text{set.mset } D' \subseteq \text{set.mset } D$
unfolding d' **by** *auto*
have $\neg \text{Neg } A \in\# D$
using $d0$ **by** (*blast dest: produces_imp_neg_notin_lits*)
then have $\text{neg_a_ni_d}': \neg \text{Neg } A \in\# D'$
using d'_subs **by** *auto*
have $a.ni.d': A \notin \text{atms_of } D'$
using $d' \text{ neg_a_ni_d}'$ **by** (*auto dest: atm_imp_pos_or_neg_lit*)
have $\exists n. AAA = \text{replicate_mset } (\text{Suc } n) (\text{Pos } A)$
using $aa d0 \text{ not0_implies_Suc produces_imp_Pos_in_lits[of } N]$
by (*simp add: filter_eq_replicate_mset del: replicate_mset_Suc*)
then have $\text{res.e}: \text{unord_resolve } D C (D' + C')$
unfolding $c d$ **by** (*fastforce intro: unord_resolve.intros*)

have $d'_le.d: D' \leq D$
unfolding d **by** *simp*
have $a.max.d: A = \text{Max } (\text{atms_of } D)$
using $d0 \text{ productive_imp_produces_Max_atom}$ **by** *auto*
then have $D' \neq \{\#\} \implies \text{Max } (\text{atms_of } D') \leq A$
using $d'_le.d$ **by** (*blast intro: less_eq_Max_atms_of*)
moreover have $D' \neq \{\#\} \implies \text{Max } (\text{atms_of } D') \neq A$
using $a.ni.d' \text{ Max.in}$ **by** (*blast intro: atms_empty_iff_empty[THEN iffD1]*)
ultimately have $\text{max_d}'_lt.a: D' \neq \{\#\} \implies \text{Max } (\text{atms_of } D') < A$
using *dual_order.strict_iff_order* **by** *blast*

have $\neg \text{interp } N D \models D$
using $d0 \text{ productive_imp_not_interp}$ **by** *blast*
then have $\neg \text{Interp } N D \models D'$
unfolding $d0 d' \text{ Interp_def true_cls_def}$ **by** (*auto simp: true_lit_def simp del: not_gr_zero*)
then have $\neg \text{INTERP } N \models D'$
using $a.max.d d'_le.d \text{ max_d}'_lt.a \text{ not_Interp_imp_not_INTERP}$ **by** *blast*
moreover have $\neg \text{INTERP } N \models C'$
using *c_cex unfolding c* **by** *simp*
ultimately have $e.cex: \neg \text{INTERP } N \models D' + C'$
by *simp*

have $\bigwedge B. B \in \text{atms_of } D' \implies B \leq A$
using $d0 d'_subs \text{ contra_subsetD lits_subsetq_imp_atms_subsetq produces_imp_atms_leq}$ **by** *metis*
then have $\bigwedge L. L \in\# D' \implies L < \text{Neg } A$
using $\text{neg_a_ni_d}' \text{ antisym_conv1 atms.less_eq_imp_lit.less_eq_neg}$ **by** *metis*

```

then have lt_cex:  $D' + C' < C$ 
  by (force intro: add commute simp: c less_multisetDM intro: exI[of - {#Neg A#}])

from d_in_n d_true prod_d res_e e_cex lt_cex show ?thesis ..
qed

```

10.2 Inference System

Lemma 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 = {Infer {#C#} D E | C D E. unord_resolve C D E}

```

sublocale *unord_Γ_sound_counterex_reducing?*:

```

sound_counterex_reducing_inference_system unord_Γ INTERP

```

proof *unfold_locales*

```

fix D E and N :: ('b :: wellorder) clause set

```

```

assume {#}  $\notin N$  and  $D \in N$  and  $\neg \text{INTERP } N \models D$  and  $\bigwedge C. C \in N \implies \neg \text{INTERP } N \models C \implies D \leq C$ 

```

```

then obtain C E where

```

```

  c_in_n:  $C \in N$  and

```

```

  c_true:  $\text{INTERP } N \models C$  and

```

```

  res_e: unord_resolve C D E and

```

```

  e_cex:  $\neg \text{INTERP } N \models E$  and

```

```

  e_lt_d:  $E < D$ 

```

```

  using unord_resolve_counterex_reducing by (metis (no_types))

```

```

from c_in_n have set_mset {#C#}  $\subseteq N$ 

```

```

  by auto

```

```

moreover have Infer {#C#} D E  $\in \text{unord}_\Gamma$ 

```

```

  unfolding unord_Γ_def using res_e by blast

```

```

ultimately show

```

```

   $\exists CC E. \text{set\_mset } CC \subseteq N \wedge \text{INTERP } N \models_m CC \wedge \text{Infer } CC D E \in \text{unord}_\Gamma \wedge \neg \text{INTERP } N \models E \wedge E < D$ 

```

```

  using c_in_n c_true e_cex e_lt_d by blast

```

next

```

fix CC D E and I :: 'b interp

```

```

assume Infer CC D E  $\in \text{unord}_\Gamma$  and  $I \models_m CC$  and  $I \models D$ 

```

```

then show  $I \models E$ 

```

```

  by (clarsimp simp: unord_Γ_def true_cls_mset_def) (erule unord_resolve_sound, auto)

```

qed

```

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 \equiv 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

using *eligible.intros ground_resolution_with_selection.eligible.cases ground_resolution_with_selection.axioms* **by** blast

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 ($\bigcup \#$ 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

using *res_e*

proof (*cases rule*: *ord_resolve.cases*)

case (*ord_resolve* n Cs D)

note DA = *this*(1) **and** e = *this*(2) **and** cas.len = *this*(3) **and** cs.len = *this*(4) **and**

as.len = *this*(6) **and** cas = *this*(8) **and** aas.ne = *this*(9) **and** a.eq = *this*(10)

show ?thesis

proof (*cases* $\forall A \in$ set As. A \in I)

case True

then have \neg I \models negs (mset As)

unfolding *true_cls_def* **by** fastforce

then have I \models D

using *d_true* DA **by** fast

then show ?thesis

unfolding e **by** blast

next

case False

then obtain i **where**

a_in_aa: i < n **and**

a_false: As ! i \notin I

using cas.len as.len **by** (*metis in_set_conv_nth*)

```

have  $\neg I \models \text{poss } (AAs ! i)$ 
  using a_false a_eq aas_ne a_in_aa unfolding true_cls_def by auto
moreover have  $I \models CAs ! i$ 
  using a_in_aa cc_true unfolding true_cls_mset_def using cas_len by auto
ultimately have  $I \models Cs ! i$ 
  using cas a_in_aa by auto
then show ?thesis
  using a_in_aa cs_len unfolding e true_cls_def
  by (meson in_Union_mset_iff nth_mem_mset union_iff)
qed
qed

```

```

lemma filter_neg_atm_of_S:  $\{\#Neg (atm\_of L). L \in \# S C\# \} = S C$ 
  by (simp add: S_selects_neg lits)

```

This corresponds to Lemma 3.13:

```

lemma ord_resolve_reductive:
  assumes ord_resolve CAs DA AAs As E
  shows  $E < DA$ 
  using assms
proof (cases rule: ord_resolve.cases)
  case (ord_resolve n Cs D)
  note  $DA = \text{this}(1)$  and  $e = \text{this}(2)$  and  $cas\_len = \text{this}(3)$  and  $cs\_len = \text{this}(4)$  and
     $ai\_len = \text{this}(6)$  and  $nz = \text{this}(7)$  and  $cas = \text{this}(8)$  and  $maxim = \text{this}(12)$ 

```

```

show ?thesis
proof (cases  $\bigcup \# mset Cs = \{\#\}$ )
  case True
  have  $\text{negs } (mset As) \neq \{\#\}$ 
    using nz ai_len by auto
  then show ?thesis
    unfolding True e DA by auto
next
  case False

```

```

define max_A_of-Cs where  $max\_A\_of\_Cs = Max (atms\_of (\bigcup \# mset Cs))$ 

```

```

have
  mc_in:  $max\_A\_of\_Cs \in atms\_of (\bigcup \# mset Cs)$  and
  mc_max:  $\bigwedge B. B \in atms\_of (\bigcup \# mset Cs) \implies B \leq max\_A\_of\_Cs$ 
  using max_A_of-Cs_def False by auto

```

```

then have  $\exists C\_max \in set Cs. max\_A\_of\_Cs \in atms\_of (C\_max)$ 
  by (metis atm_imp_pos_or_neg_lit in_Union_mset_iff neg_lit_in_atms_of pos_lit_in_atms_of set_mset_mset)

```

```

then obtain max_i where
  cm_in_cas:  $max\_i < length CAs$  and
  mc_in_cm:  $max\_A\_of\_Cs \in atms\_of (Cs ! max\_i)$ 
  using in_set_conv_nth[of _ CAs] by (metis cas_len cs_len in_set_conv_nth)

```

```

define CA_max where  $CA\_max = CAs ! max\_i$ 
define A_max where  $A\_max = As ! max\_i$ 
define C_max where  $C\_max = Cs ! max\_i$ 

```

```

have mc_lt_ma:  $max\_A\_of\_Cs < A\_max$ 
  using maxim cm_in_cas mc_in_cm cas_len unfolding strictly_maximal_wrt_def A_max_def by auto

```

```

then have ucas_ne_neg_aa:  $(\bigcup \# mset Cs) \neq \text{negs } (mset As)$ 
  using mc_in mc_max mc_lt_ma cm_in_cas cas_len ai_len unfolding A_max_def
  by (metis atms_of_negs nth_mem set_mset_mset leD)
moreover have ucas_lt_ma:  $\forall B \in atms\_of (\bigcup \# mset Cs). B < A\_max$ 
  using mc_max mc_lt_ma by fastforce
moreover have  $\neg Neg A\_max \in \# (\bigcup \# mset Cs)$ 

```

```

    using ucas_lt_ma neg_lit_in_atms_of[of A_max  $\cup$  # mset Cs] by auto
  moreover have Neg A_max  $\in$  # negs (mset As)
    using cm_in_cas cas_len ai_len A_max_def by auto
  ultimately have ( $\cup$  # mset Cs) < negs (mset As)
    unfolding less_multiset_HO
    by (metis (no_types) atms_less_eq_imp_lit_less_eq_neg count_greater_zero_iff
        count_inI le_imp_less_or_eq less_imp_not_less not_le)
  then show ?thesis
    unfolding e DA by auto
qed
qed

```

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 –

```

  have d_ne: DA  $\neq$  {#}
    using d_in_n ec_ni_n by blast
  have  $\exists$  As. As  $\neq$  []  $\wedge$  negs (mset As)  $\leq$  # DA  $\wedge$  eligible As DA
proof (cases S DA = {#})
  assume s_d_e: S DA = {#}

```

```

define A where A = Max (atms_of DA)
define As where As = [A]
define D where D = DA - {#Neg A #}

```

```

  have na_in_d: Neg A  $\in$  # DA
    unfolding A_def using s_d_e d_ne d_in_n d_cex d_min
    by (metis Max_in_lits Max_lit_eq_pos_or_neg_Max_atm max_pos_imp_Interp Interp_imp_INTERP)
  then have das: DA = D + negs (mset As) unfolding D_def As_def by auto
  moreover from na_in_d have negs (mset As)  $\subseteq$  # DA
    by (simp add: As_def)
  moreover have As ! 0 = Max (atms_of (D + negs (mset As)))
    using A_def As_def das by auto
  then have eligible As DA
    using eligible s_d_e As_def das maximal_wrt_def by auto
  ultimately show ?thesis
    using As_def by blast

```

next

```

  assume s_d_e: S DA  $\neq$  {#}
define As :: 'a list where
  As = list_of_mset {#atm_of L. L  $\in$  # S DA#}
define D :: 'a clause where
  D = DA - negs {#atm_of L. L  $\in$  # S DA#}

```

```

  have As  $\neq$  [] unfolding As_def using s_d_e
    by (metis image_mset_is_empty_iff list_of_mset_empty)
  moreover have da_sub_as: negs {#atm_of L. L  $\in$  # S DA#}  $\subseteq$  # DA
    using S_selects_subseteq by (auto simp: filter_neg_atm_of_S)
  then have negs (mset As)  $\subseteq$  # DA
    unfolding As_def by auto

```



```

moreover have das:  $DA = D + \text{negs } (\text{mset } As)$ 
  using da_sub_as unfolding D.def As_def by auto
moreover have  $S DA = \text{negs } \{\#atm\_of L. L \in \# S DA\}$ 
  by (auto simp: filter_neg_atm_of_S)
then have  $S DA = \text{negs } (\text{mset } As)$ 
  unfolding As_def by auto
then have eligible As DA
  unfolding das using eligible by auto
ultimately show ?thesis
  by blast
qed
then obtain As :: 'a list where
  as_ne:  $As \neq []$  and
  negs_as_le_d:  $\text{negs } (\text{mset } As) \leq \# DA$  and
  s_d: eligible As DA
  by blast

define D :: 'a clause where
   $D = DA - \text{negs } (\text{mset } As)$ 

have set As  $\subseteq INTERP N$ 
  using d_cex negs_as_le_d by force
then have prod_ex:  $\forall A \in \text{set } As. \exists D. \text{produces } N D A$ 
  unfolding INTERP_def
  by (metis (no_types, lifting) INTERP_def subsetCE UN_E not_produces_imp_notin_production)
then have  $\bigwedge A. \exists D. \text{produces } N D A \longrightarrow A \in \text{set } As$ 
  using ec_ni_n by (auto intro: productive_in_N)
then have  $\bigwedge A. \exists D. \text{produces } N D A \longleftrightarrow A \in \text{set } As$ 
  using prod_ex by blast
then obtain CA_of where c_of0:  $\bigwedge A. \text{produces } N (CA\_of A) A \longleftrightarrow A \in \text{set } As$ 
  by metis
then have prod_c0:  $\forall A \in \text{set } As. \text{produces } N (CA\_of A) A$ 
  by blast

define C_of where
   $\bigwedge A. C\_of A = \{\#L \in \# CA\_of A. L \neq Pos A\}$ 
define Aj_of where
   $\bigwedge A. Aj\_of A = \text{image\_mset } atm\_of \{\#L \in \# CA\_of A. L = Pos A\}$ 

have pospos:  $\bigwedge LL A. \{\#\text{Pos } (atm\_of x). x \in \#\{\#L \in \# LL. L = Pos A\}\} = \{\#L \in \# LL. L = Pos A\}$ 
  by (metis (mono_tags, lifting) image_filter_cong literal.sel(1) multiset.map_ident)
have ca_of_c_of_Aj_of:  $\bigwedge A. CA\_of A = C\_of A + \text{poss } (Aj\_of A)$ 
  using pospos[of - CA_of -] by (simp add: C_of_def Aj_of_def add commute multiset_partition)

define n :: nat where
   $n = \text{length } As$ 
define Cs :: 'a clause list where
   $Cs = \text{map } C\_of As$ 
define AAs :: 'a multiset list where
   $AAs = \text{map } Aj\_of As$ 
define CAs :: 'a literal multiset list where
   $CAs = \text{map } CA\_of As$ 

have m_nz:  $\bigwedge A. A \in \text{set } As \implies Aj\_of A \neq \{\#\}$ 
  unfolding Aj_of_def using prod_c0 produces_imp_Pos_in_lits
  by (metis (full_types) filter_mset_empty_conv image_mset_is_empty_iff)

have prod_c: productive N CA if ca_in:  $CA \in \text{set } CAs$  for CA
proof -
  obtain i where i_p:  $i < \text{length } CAs$   $CAs ! i = CA$ 
  using ca_in by (meson in_set_conv_nth)
  have production N (CA_of (As ! i)) = \{As ! i\}
  using i_p CAs_def prod_c0 by auto

```

```

then show productive N CA
  using i_p CAs_def by auto
qed
then have cs_subst_n: set CAs  $\subseteq$  N
  using productive_in_N by auto
have cs_true: INTERP N  $\models_m$  mset CAs
  unfolding true_cls_mset_def using prod_c productive_imp_INTERP by auto

have  $\bigwedge A. A \in \text{set } As \implies \neg \text{Neg } A \in \# \text{ CA\_of } A$ 
  using prod_c0 produces_imp_neg_notin_lits by auto
then have a_ni_c':  $\bigwedge A. A \in \text{set } As \implies A \notin \text{atms\_of } (C\_of A)$ 
  unfolding C_of_def using atm_imp_pos_or_neg_lit by force
have c'_le_c:  $\bigwedge A. C\_of A \leq CA\_of A$ 
  unfolding C_of_def by (auto intro: subset_eq_imp_le_multiset)
have a_max_c:  $\bigwedge A. A \in \text{set } As \implies A = \text{Max } (\text{atms\_of } (CA\_of A))$ 
  using prod_c0 productive_imp_produces_Max_atom[of N] by auto
then have  $\bigwedge A. A \in \text{set } As \implies C\_of A \neq \{\#\} \implies \text{Max } (\text{atms\_of } (C\_of A)) \leq A$ 
  using c'_le_c by (metis less_eq_Max_atms_of)
moreover have  $\bigwedge A. A \in \text{set } As \implies C\_of A \neq \{\#\} \implies \text{Max } (\text{atms\_of } (C\_of A)) \neq A$ 
  using a_ni_c' Max_in by (metis (no_types) atms_empty_iff_empty finite_atms_of)
ultimately have max_c'_lt_a:  $\bigwedge A. A \in \text{set } As \implies C\_of A \neq \{\#\} \implies \text{Max } (\text{atms\_of } (C\_of A)) < A$ 
  by (metis order.strict_iff_order)

have le_cs_as: length CAs = length As
  unfolding CAs_def by simp

have length CAs = n
  by (simp add: le_cs_as n_def)
moreover have length Cs = n
  by (simp add: Cs_def n_def)
moreover have length AAs = n
  by (simp add: AAs_def n_def)
moreover have length As = n
  using n_def by auto
moreover have n  $\neq$  0
  by (simp add: as_ne n_def)
moreover have  $\forall i. i < \text{length } AAs \longrightarrow (\forall A \in \# AAs ! i. A = As ! i)$ 
  using AAs_def Aj_of_def by auto

have  $\bigwedge x B. \text{production } N (CA\_of x) = \{x\} \implies B \in \# CA\_of x \implies B \neq \text{Pos } x \implies \text{atm\_of } B < x$ 
  by (metis atm_of_lit_in_atms_of insert_not_empty le_imp_less_or_eq Pos_atm_of_iff
    Neg_atm_of_iff pos_neg_in_imp_true produces_imp_Pos_in_lits produces_imp_atms_leq
    productive_imp_not_interp)
then have  $\bigwedge B A. A \in \text{set } As \implies B \in \# CA\_of A \implies B \neq \text{Pos } A \implies \text{atm\_of } B < A$ 
  using prod_c0 by auto
have  $\forall i. i < \text{length } AAs \longrightarrow AAs ! i \neq \{\#\}$ 
  unfolding AAs_def using m_nz by simp
have  $\forall i < n. CAs ! i = Cs ! i + \text{poss } (AAs ! i)$ 
  unfolding CAs_def Cs_def AAs_def using ca_of_c_of_aj_of by (simp add: n_def)

moreover have  $\forall i < n. AAs ! i \neq \{\#\}$ 
  using  $\langle \forall i < \text{length } AAs. AAs ! i \neq \{\#\} \rangle$  calculation(3) by blast
moreover have  $\forall i < n. \forall A \in \# AAs ! i. A = As ! i$ 
  by (simp add:  $\langle \forall i < \text{length } AAs. \forall A \in \# AAs ! i. A = As ! i \rangle$  calculation(3))
moreover have eligible As DA
  using s_d by auto
then have eligible As (D + negs (mset As))
  using D_def negs_as_le_d by auto
moreover have  $\bigwedge i. i < \text{length } AAs \implies \text{strictly\_maximal\_wrt } (As ! i) ((Cs ! i))$ 
  by (simp add: C_of_def Cs_def  $\langle \bigwedge x B. [\text{production } N (CA\_of x) = \{x\}; B \in \# CA\_of x; B \neq \text{Pos } x] \implies \text{atm\_of } B < x \rangle$ 
    atms_of_def calculation(3) n_def prod_c0 strictly_maximal_wrt_def)

have  $\forall i < n. \text{strictly\_maximal\_wrt } (As ! i) (Cs ! i)$ 

```

by (simp add: $\langle \bigwedge i. i < \text{length } AAs \implies \text{strictly_maximal_wrt } (As \ ! \ i) \ (Cs \ ! \ i) \rangle$ calculation(3))
moreover have $\forall CA \in \text{set } CAs. S \ CA = \{\#\}$
 using prod_c producesD productive_imp_produces_Max_literal by blast
have $\forall CA \in \text{set } CAs. S \ CA = \{\#\}$
 using $\langle \forall CA \in \text{set } CAs. S \ CA = \{\#\} \rangle$ by simp
then have $\forall i < n. S \ (CAs \ ! \ i) = \{\#\}$
 using $\langle \text{length } CAs = n \rangle$ nth_mem by blast
ultimately have res_e: ord_resolve CAs (D + negs (mset As)) AAs As ($\bigcup \#$ mset Cs + D)
 using ord_resolve by auto

have $\bigwedge A. A \in \text{set } As \implies \neg \text{interp } N \ (CA_of \ A) \models CA_of \ A$
 by (simp add: prod_c0 producesD)
then have $\bigwedge A. A \in \text{set } As \implies \neg \text{Interp } N \ (CA_of \ A) \models C_of \ A$
 unfolding prod_c0 C_of_def Interp_def true_cls_def using true_lit_def not_gr_zero prod_c0
 by auto
then have c'_at_n: $\bigwedge A. A \in \text{set } As \implies \neg \text{INTERP } N \models C_of \ A$
 using a_max_c c'_le_c max_c'_lt_a not_Interp_imp_not_INTERP unfolding true_cls_def
 by (metis true_cls_def true_cls_empty)

have $\neg \text{INTERP } N \models \bigcup \# \text{ mset } Cs$
 unfolding Cs_def true_cls_def using c'_at_n by fastforce
moreover have $\neg \text{INTERP } N \models D$
 using d_cex by (metis D_def add_diff_cancel_right' negs_as_le_d subset_mset.add_diff_assoc2
 true_cls_def union_iff)
ultimately have e_cex: $\neg \text{INTERP } N \models \bigcup \# \text{ mset } Cs + D$
 by simp

have $\text{set } CAs \subseteq N$
 by (simp add: cs_subs_n)
moreover have $\text{INTERP } N \models_m \text{ mset } CAs$
 by (simp add: cs_true)
moreover have $\bigwedge CA. CA \in \text{set } CAs \implies \text{productive } N \ CA$
 by (simp add: prod_c)
moreover have ord_resolve CAs DA AAs As ($\bigcup \#$ mset Cs + D)
 using D_def negs_as_le_d res_e by auto
moreover have $\neg \text{INTERP } N \models \bigcup \# \text{ mset } Cs + D$
 using e_cex by simp
moreover have ($\bigcup \# \text{ mset } Cs + D$) < DA
 using calculation(4) ord_resolve_reductive by auto
ultimately show thesis
 ..
qed

lemma ord_resolve_atms_of_concl_subset:
assumes ord_resolve CAs DA AAs As E
shows $\text{atms_of } E \subseteq (\bigcup C \in \text{set } CAs. \text{atms_of } C) \cup \text{atms_of } DA$
using assms
proof (cases rule: ord_resolve.cases)
case (ord_resolve n Cs D)
note DA = this(1) **and** e = this(2) **and** cas_len = this(3) **and** cs_len = this(4) **and** cas = this(8)

have $\forall i < n. \text{set_mset } (Cs \ ! \ i) \subseteq \text{set_mset } (CAs \ ! \ i)$
 using cas by auto
then have $\forall i < n. Cs \ ! \ i \subseteq \# \bigcup \# \text{ mset } CAs$
 by (metis cas cas_len mset_subset_eq_add_left nth_mem_mset sum_mset.remove union_assoc)
then have $\forall C \in \text{set } Cs. C \subseteq \# \bigcup \# \text{ mset } CAs$
 using cs_len in_set_conv_nth[of _ Cs] by auto
then have $\text{set_mset } (\bigcup \# \text{ mset } Cs) \subseteq \text{set_mset } (\bigcup \# \text{ mset } CAs)$
 by auto (meson in_mset_sum_list2 mset_subset_eqD)
then have $\text{atms_of } (\bigcup \# \text{ mset } Cs) \subseteq \text{atms_of } (\bigcup \# \text{ mset } CAs)$
 by (meson lits_subseteq_imp_atms_subseteq mset_subset_eqD subsetI)
moreover have $\text{atms_of } (\bigcup \# \text{ mset } CAs) = (\bigcup CA \in \text{set } CAs. \text{atms_of } CA)$
 by (intro set_eqI iffI, simp_all,

```

    metis in_mset_sum_list2 atm_imp_pos_or_neg_lit neg_lit_in_atms_of pos_lit_in_atms_of,
    metis in_mset_sum_list atm_imp_pos_or_neg_lit neg_lit_in_atms_of pos_lit_in_atms_of)
ultimately have atms_of (∪# mset Cs) ⊆ (∪ CA ∈ set CAs. atms_of CA)
by auto
moreover have atms_of D ⊆ atms_of DA
using DA by auto
ultimately show ?thesis
unfolding e by auto
qed

```

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 | 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 *unfold_locales*

fix DA :: 'a clause **and** N :: 'a clause set

assume {#} ∉ N **and** DA ∈ N **and** ¬ INTERP N ⊨ DA **and** ∧C. C ∈ N ⇒ ¬ INTERP N ⊨ C ⇒ DA ≤ C

then obtain CAs AAs As E **where**

```
dd_sset_n: set CAs ⊆ N and
dd_true: INTERP N ⊨m mset CAs and
res_e: ord_resolve CAs DA AAs As E and
e_cex: ¬ INTERP N ⊨ E and
e_lt_c: E < DA
```

using ord_resolve_counterex_reducing[of N DA thesis] **by auto**

have Infer (mset CAs) DA E ∈ ord_Γ

using res_e **unfolding** ord_Γ_def **by** (metis (mono_tags, lifting) mem_Collect_eq)

then show ∃ CC E. set_mset CC ⊆ N ∧ INTERP N ⊨_m CC ∧ Infer CC DA E ∈ ord_Γ
 ∧ ¬ INTERP N ⊨ E ∧ E < DA

using dd_sset_n dd_true e_cex e_lt_c **by** (metis set_mset_mset)

qed (*auto simp: ord_Γ_def intro: ord_resolve_sound ord_resolve_reductive*)

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. R_f denotes \mathcal{R}_F ; R_i denotes \mathcal{R}_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

$Ri_subset_Gamma: Ri\ N \subseteq \Gamma$ **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 $\text{saturated_upto} :: 'a \text{ clause set} \Rightarrow \text{bool}$ **where**
 $\text{saturated_upto } N \longleftrightarrow \text{inferences_from } (N - Rf\ N) \subseteq Ri\ N$

inductive $\text{derive} :: 'a \text{ clause set} \Rightarrow 'a \text{ clause set} \Rightarrow \text{bool}$ (**infix** \triangleright 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 $\text{derive_subset}: M \triangleright N \Longrightarrow N \subseteq M \cup \text{concls_of } (\text{inferences_from } M)$
by ($\text{meson } \text{Diff_subset_conv } \text{derive.cases}$)

end

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

begin

lemma $\text{deriv_sat_preserving}$:

assumes

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

shows $\text{satisfiable } (\text{Sup_l1ist } Ns)$

proof –

have $\text{ns0}: \text{lth } Ns\ 0 = \text{lhd } Ns$
using deriv **by** ($\text{metis } \text{chain_not_lnull } \text{lhd_conv_lth}$)

have $\text{len_ns}: \text{l1ength } Ns > 0$
using deriv **by** ($\text{case_tac } Ns$) simp+

{

fix DD

assume $\text{fin}: \text{finite } DD$ **and** $\text{sset_lun}: DD \subseteq \text{Sup_l1ist } Ns$

then obtain k **where** $\text{dd_sset}: DD \subseteq \text{Sup_upto_l1ist } Ns\ k$
using $\text{finite_Sup_l1ist_imp_Sup_upto_l1ist}$ **by** blast

have $\text{satisfiable } (\text{Sup_upto_l1ist } Ns\ k)$

proof ($\text{induct } k$)

case 0

then show $?case$
using $\text{len_ns } \text{ns0 } \text{sat_n0}$ **unfolding** $\text{Sup_upto_l1ist_def } \text{true_clss_def}$ **by** auto

next

case ($\text{Suc } k$)

show $?case$

proof ($\text{cases } \text{enat } (\text{Suc } k) \geq \text{l1ength } Ns$)

case True

then have $\text{Sup_upto_l1ist } Ns\ k = \text{Sup_upto_l1ist } Ns\ (\text{Suc } k)$
unfolding $\text{Sup_upto_l1ist_def}$ **using** $\text{le_Suc_eq } \text{not_less}$ **by** blast

then show $?thesis$
using Suc **by** simp

next

case False

then have $\text{lth } Ns\ k \triangleright \text{lth } Ns\ (\text{Suc } k)$
using deriv **by** ($\text{auto } \text{simp}: \text{chain_lth_rel}$)

then have $\text{lth } Ns\ (\text{Suc } k) \subseteq \text{lth } Ns\ k \cup \text{concls_of } (\text{inferences_from } (\text{lth } Ns\ k))$
by ($\text{rule } \text{derive_subset}$)

moreover have $\text{lth } Ns\ k \subseteq \text{Sup_upto_l1ist } Ns\ k$
unfolding $\text{Sup_upto_l1ist_def}$ **using** $\text{False } \text{Suc_ile_eq } \text{linear}$ **by** blast

```

ultimately have  $lnth\ Ns\ (Suc\ k)$ 
   $\subseteq Sup\_upto\_llist\ Ns\ k \cup\ concls\_of\ (inferences\_from\ (Sup\_upto\_llist\ Ns\ k))$ 
  by clarsimp (metis UnCI UnE image_Un inferences_from_mono le_iff_sup)
moreover have  $Sup\_upto\_llist\ Ns\ (Suc\ k) = Sup\_upto\_llist\ Ns\ k \cup\ lnth\ Ns\ (Suc\ k)$ 
  unfolding Sup\_upto\_llist_def using False by (force elim: le_SucE)
moreover have
   $satisfiable\ (Sup\_upto\_llist\ Ns\ k \cup\ concls\_of\ (inferences\_from\ (Sup\_upto\_llist\ Ns\ k)))$ 
  using Suc  $\Gamma\_sat\_preserving$  unfolding sat\_preserving\_inference\_system_def by simp
ultimately show ?thesis
  by (metis le_iff_sup true_cls_union)
qed
qed
then have satisfiable DD
  using dd_sset unfolding Sup\_upto\_llist_def by (blast intro: true_cls_mono)
}
then show ?thesis
  using ground\_resolution\_without\_selection.clausal\_logic\_compact[THEN iffD1] by metis
qed

```

This corresponds to Lemma 4.2:

lemma

assumes *deriv: chain* (\triangleright) *Ns*

shows

Rf_Sup_subset_Rf_Liminf: $Rf\ (Sup_llist\ Ns) \subseteq Rf\ (Liminf_llist\ Ns)$ **and**

Ri_Sup_subset_Ri_Liminf: $Ri\ (Sup_llist\ Ns) \subseteq Ri\ (Liminf_llist\ Ns)$ **and**

sat_deriv_Liminf_iff: $satisfiable\ (Liminf_llist\ Ns) \longleftrightarrow satisfiable\ (lhd\ Ns)$

proof –

{

fix *C i j*

assume

c_in: $C \in lnth\ Ns\ i$ **and**

c_ni: $C \notin Rf\ (Sup_llist\ Ns)$ **and**

j: $j \geq i$ **and**

j': $enat\ j < llength\ Ns$

from *c_ni* **have** *c_ni'*: $\bigwedge i. enat\ i < llength\ Ns \implies C \notin Rf\ (lnth\ Ns\ i)$

using *Rf_mono lnth_subset_Sup_llist Sup_llist_def* by (*blast dest: contra_subsetD*)

have $C \in lnth\ Ns\ j$

using *j j'*

proof (*induct j*)

case *0*

then show *?case*

using *c_in* by *blast*

next

case (*Suc k*)

then show *?case*

proof (*cases i < Suc k*)

case *True*

have $i \leq k$

using *True* by *linarith*

moreover have $enat\ k < llength\ Ns$

using *Suc.premis(2) Suc_ile_eq* by (*blast intro: dual_order.strict.implies_order*)

ultimately have *c_in_k*: $C \in lnth\ Ns\ k$

using *Suc.hyps* by *blast*

have *rel*: $lnth\ Ns\ k \triangleright lnth\ Ns\ (Suc\ k)$

using *Suc.premis deriv* by (*auto simp: chain_lnth_rel*)

then show *?thesis*

using *c_in_k c_ni' Suc.premis(2)* by *cases auto*

next

case *False*

then show *?thesis*

using *Suc c_in* by *auto*

qed

qed

```

}
then have lu_ll: Sup_llist Ns - Rf (Sup_llist Ns)  $\subseteq$  Liminf_llist Ns
  unfolding Sup_llist_def Liminf_llist_def by blast
have rf: Rf (Sup_llist Ns - Rf (Sup_llist Ns))  $\subseteq$  Rf (Liminf_llist Ns)
  using lu_ll Rf_mono by simp
have ri: Ri (Sup_llist Ns - Rf (Sup_llist Ns))  $\subseteq$  Ri (Liminf_llist Ns)
  using lu_ll Ri_mono by simp
show Rf (Sup_llist Ns)  $\subseteq$  Rf (Liminf_llist Ns)
  using rf Rf_indep by blast
show Ri (Sup_llist Ns)  $\subseteq$  Ri (Liminf_llist Ns)
  using ri Ri_indep by blast

show satisfiable (Liminf_llist Ns)  $\longleftrightarrow$  satisfiable (lhd Ns)
proof
  assume satisfiable (lhd Ns)
  then have satisfiable (Sup_llist Ns)
    using deriv deriv_sat_preserving by simp
  then show satisfiable (Liminf_llist Ns)
    using true_cls_mono[OF Liminf_llist_subset_Sup_llist] by blast
next
  assume satisfiable (Liminf_llist Ns)
  then have satisfiable (Sup_llist Ns - Rf (Sup_llist Ns))
    using true_cls_mono[OF lu_ll] by blast
  then have satisfiable (Sup_llist Ns)
    using Rf_sat by blast
  then show satisfiable (lhd Ns)
    using deriv true_cls_mono lhd_subset_Sup_llist chain_not_lnull by metis
qed
qed

```

lemma

assumes *chain* (\triangleright) *Ns*

shows

Rf_Liminf_eq_Rf_Sup: *Rf (Liminf_llist Ns)* = *Rf (Sup_llist Ns)* **and**

Ri_Liminf_eq_Ri_Sup: *Ri (Liminf_llist Ns)* = *Ri (Sup_llist Ns)*

using *assms*

by (*auto simp*: *Rf_Sup_subset_Rf_Liminf* *Rf_mono* *Ri_Sup_subset_Ri_Liminf* *Ri_mono* *Liminf_llist_subset_Sup_llist* *subset_antisym*)

end

The assumption below corresponds to condition R4 of Definition 4.1.

locale *effective_redundancy_criterion* = *redundancy_criterion* +

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

begin

definition *fair_cls_seq* :: '*a* clause set llist \Rightarrow bool **where**

fair_cls_seq Ns \longleftrightarrow (let *N'* = *Liminf_llist Ns* - *Rf (Liminf_llist Ns)* in

concls_of (inferences_from N' - Ri N') \subseteq *Sup_llist Ns* \cup *Rf (Sup_llist Ns)*)

end

locale *sat_preserving_effective_redundancy_criterion* =

sat_preserving_inference_system Γ :: ('*a* :: wellorder) *inference set* +

effective_redundancy_criterion

begin

sublocale *sat_preserving_redundancy_criterion*

..

The result below corresponds to Theorem 4.3.

theorem *fair_derive_saturated_upto*:

assumes

```

    deriv: chain (▷) Ns and
    fair: fair_clsseq Ns
shows saturated_upto (Liminf_llist Ns)
unfolding saturated_upto_def
proof
  fix γ
  let ?N' = Liminf_llist Ns - Rf (Liminf_llist Ns)
  assume γ: γ ∈ inferences_from ?N'
  show γ ∈ Ri (Liminf_llist Ns)
  proof (cases γ ∈ Ri ?N')
    case True
    then show ?thesis
      using Ri_mono by blast
  next
  case False
  have concl_of (inferences_from ?N' - Ri ?N') ⊆ Sup_llist Ns ∪ Rf (Sup_llist Ns)
    using fair unfolding fair_clsseq_def Let_def .
  then have concl_of γ ∈ Sup_llist Ns ∪ Rf (Sup_llist Ns)
    using False γ by auto
  moreover
  {
    assume concl_of γ ∈ Sup_llist Ns
    then have γ ∈ Ri (Sup_llist Ns)
      using γ Ri_effective_inferences_from_def by blast
    then have γ ∈ Ri (Liminf_llist Ns)
      using deriv Ri_Sup_subset_Ri_Liminf by fast
  }
  moreover
  {
    assume concl_of γ ∈ Rf (Sup_llist Ns)
    then have concl_of γ ∈ Rf (Liminf_llist Ns)
      using deriv Rf_Sup_subset_Rf_Liminf by blast
    then have γ ∈ Ri (Liminf_llist Ns)
      using γ Ri_effective_inferences_from_def by auto
  }
  ultimately show γ ∈ Ri (Liminf_llist Ns)
    by blast
qed
qed
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 ⇒ 'a clause set where
  Rf _ = {}

```

```

definition Ri :: 'a clause set ⇒ 'a inference set where
  Ri N = {γ. γ ∈ Γ ∧ concl_of γ ∈ N}

```

```

sublocale effective_redundancy_criterion Γ Rf Ri
  by unfold_locales (auto simp: Rf_def Ri_def)

```

```

lemma saturated_upto_iff: saturated_upto N ⟷ concl_of (inferences_from N) ⊆ N
  unfolding saturated_upto_def inferences_from_def Rf_def Ri_def by auto

```

```

end

```

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

```

lemma redundancy_criterion_standard_extension:

```


assumes $\Gamma \subseteq \Gamma'$ **and** *redundancy_criterion* Γ *Rf Ri*
shows *redundancy_criterion* Γ' *Rf* $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))$
using *assms* **unfolding** *redundancy_criterion_def* **by** (*intro conjI*) ((*auto simp: rev_subsetD*)[5], *sat*)

lemma *redundancy_criterion_standard_extension_saturated_upto_iff*:
assumes $\Gamma \subseteq \Gamma'$ **and** *redundancy_criterion* Γ *Rf Ri*
shows *redundancy_criterion.saturated_upto* Γ *Rf Ri M* \longleftrightarrow
redundancy_criterion.saturated_upto Γ' *Rf* $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))\ M$
using *assms* *redundancy_criterion.saturated_upto_def* *redundancy_criterion.saturated_upto_def*
redundancy_criterion_standard_extension
unfolding *inference_system.inferences_from_def* **by** *blast*

lemma *redundancy_criterion_standard_extension_effective*:
assumes $\Gamma \subseteq \Gamma'$ **and** *effective_redundancy_criterion* Γ *Rf Ri*
shows *effective_redundancy_criterion* Γ' *Rf* $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))$
using *assms* *redundancy_criterion_standard_extension*[*of* Γ]
unfolding *effective_redundancy_criterion_def* *effective_redundancy_criterion_axioms_def* **by** *auto*

lemma *redundancy_criterion_standard_extension_fair_iff*:
assumes $\Gamma \subseteq \Gamma'$ **and** *effective_redundancy_criterion* Γ *Rf Ri*
shows *effective_redundancy_criterion.fair_clsseq* Γ' *Rf* $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))\ Ns$ \longleftrightarrow
effective_redundancy_criterion.fair_clsseq Γ *Rf Ri Ns*
using *assms* *redundancy_criterion_standard_extension_effective*[*of* Γ Γ' *Rf Ri*]
effective_redundancy_criterion.fair_clsseq_def[*of* Γ *Rf Ri Ns*]
effective_redundancy_criterion.fair_clsseq_def[*of* Γ' *Rf* $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))\ Ns$]
unfolding *inference_system.inferences_from_def* *Let_def* **by** *auto*

theorem *redundancy_criterion_standard_extension_fair_derive_saturated_upto*:
assumes
subs: $\Gamma \subseteq \Gamma'$ **and**
red: *redundancy_criterion* Γ *Rf Ri* **and**
red': *sat_preserving_effective_redundancy_criterion* Γ' *Rf* $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))$ **and**
deriv: *chain* (*redundancy_criterion.derive* Γ' *Rf*) *Ns* **and**
fair: *effective_redundancy_criterion.fair_clsseq* Γ' *Rf* $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))\ Ns$
shows *redundancy_criterion.saturated_upto* Γ *Rf Ri* (*Liminf_llist* *Ns*)
proof –
have *redundancy_criterion.saturated_upto* Γ' *Rf* $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))$ (*Liminf_llist* *Ns*)
by (*rule* *sat_preserving_effective_redundancy_criterion.fair_derive_saturated_upto*
[*OF red' deriv fair*])
then show *?thesis*
by (*rule* *redundancy_criterion_standard_extension_saturated_upto_iff*[*THEN iffD2*, *OF subs red*])
qed

end

13 The Standard Redundancy Criterion

theory *Standard_Redundancy*
imports *Proving_Process*
begin

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

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

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

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

$$Rf N = \{C. \exists DD. \text{set_mset } DD \subseteq N \wedge (\forall I. I \models_m DD \longrightarrow I \models C) \wedge (\forall D. D \in\# DD \longrightarrow D < C)\}$$

definition $Ri :: 'a \text{ clause set} \Rightarrow 'a \text{ inference set}$ **where**

$$Ri N = \{\gamma \in \Gamma. \text{redundant_infer } N \ \gamma\}$$

lemma *tautology_redundant*:

assumes $Pos \ A \in\# \ C$

assumes $Neg \ A \in\# \ C$

shows $C \in Rf \ N$

proof –

have $\text{set_mset } \{\#\} \subseteq N \wedge (\forall I. I \models_m \{\#\} \longrightarrow I \models C) \wedge (\forall D. D \in\# \{\#\} \longrightarrow D < C)$

using *assms* **by** *auto*

then show $C \in Rf \ N$

unfolding *Rf_def* **by** *blast*

qed

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

unfolding *Rf_def* **by** *force*

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

lemma *Rf_mono*: $N \subseteq N' \Longrightarrow Rf \ N \subseteq Rf \ N'$

unfolding *Rf_def* **by** *auto*

lemma *wlog_non_Rf*:

assumes $ex: \exists DD. \text{set_mset } DD \subseteq N \wedge (\forall I. I \models_m DD + CC \longrightarrow I \models E) \wedge (\forall D'. D' \in\# DD \longrightarrow D' < D)$

shows $\exists DD. \text{set_mset } DD \subseteq N - Rf \ N \wedge (\forall I. I \models_m DD + CC \longrightarrow I \models E) \wedge (\forall D'. D' \in\# DD \longrightarrow D' < D)$

proof –

from *ex* **obtain** $DD0$ **where**

$dd0: DD0 \in \{DD. \text{set_mset } DD \subseteq N \wedge (\forall I. I \models_m DD + CC \longrightarrow I \models E) \wedge (\forall D'. D' \in\# DD \longrightarrow D' < D)\}$

by *blast*

have $\exists DD. \text{set_mset } DD \subseteq N \wedge (\forall I. I \models_m DD + CC \longrightarrow I \models E) \wedge (\forall D'. D' \in\# DD \longrightarrow D' < D) \wedge$

$(\forall DD'. \text{set_mset } DD' \subseteq N \wedge (\forall I. I \models_m DD' + CC \longrightarrow I \models E) \wedge (\forall D'. D' \in\# DD' \longrightarrow D' < D) \longrightarrow DD \leq DD')$

using *wf_eq_minimal*[*THEN iffD1, rule_format, OF wf_less_multiset dd0*]

unfolding *not_le*[*symmetric*] **by** *blast*

then obtain DD **where**

$dd_subs_n: \text{set_mset } DD \subseteq N$ **and**

$ddcc_imp_e: \forall I. I \models_m DD + CC \longrightarrow I \models E$ **and**

$dd_lt_d: \forall D'. D' \in\# DD \longrightarrow D' < D$ **and**

$d_min: \forall DD'. \text{set_mset } DD' \subseteq N \wedge (\forall I. I \models_m DD' + CC \longrightarrow I \models E) \wedge (\forall D'. D' \in\# DD' \longrightarrow D' < D) \longrightarrow DD \leq DD'$

by *blast*

have $\forall Da. Da \in\# DD \longrightarrow Da \notin Rf \ N$

proof *clarify*

fix Da

assume

$da_in_dd: Da \in\# DD$ **and**

$da_rf: Da \in Rf \ N$

from *da_rf* **obtain** DD' **where**

$dd'_subs_n: \text{set_mset } DD' \subseteq N$ **and**

$dd'_imp_da: \forall I. I \models_m DD' \longrightarrow I \models Da$ **and**

$dd'_lt_da: \forall D'. D' \in\# DD' \longrightarrow D' < Da$

unfolding *Rf_def* **by** *blast*

define DDa **where**

$$DDa = DD - \{\#Da\# \} + DD'$$

have $\text{set_mset } DDa \subseteq N$

unfolding *DDa_def* **using** *dd_subs_n dd'_subs_n*

by (*meson contra_subsetD in_diffD subsetI union_iff*)
moreover have $\forall I. I \models_m DDa + CC \longrightarrow I \models E$
 using *dd'_imp_da ddc_imp_e da_in_dd unfolding DDa_def true_cls_mset_def*
 by (*metis in_remove1_mset_neq union_iff*)
moreover have $\forall D'. D' \in\# DDa \longrightarrow D' < D$
 using *dd_lt_d dd'_lt_da da_in_dd unfolding DDa_def*
 by (*metis insert_DiffM2 order.strict_trans union_iff*)
moreover have $DDa < DD$
 unfolding *DDa_def*
 by (*meson da_in_dd dd'_lt_da mset.lt_single_right_iff single_subset_iff union_le_diff_plus*)
ultimately show *False*
 using *d_min unfolding less_eq_multiset_def* by (*auto intro!: antisym*)
qed
then show *?thesis*
 using *dd_subs_n ddc_imp_e dd_lt_d* by *auto*
qed

lemma *Rf_imp_ex_non_Rf*:
assumes $C \in Rf\ N$
shows $\exists CC. set_mset\ CC \subseteq N - Rf\ N \wedge (\forall I. I \models_m CC \longrightarrow I \models C) \wedge (\forall C'. C' \in\# CC \longrightarrow C' < C)$
using *assms* by (*auto simp: Rf_def intro: wlog_non_Rf[of _ {#}], simplified*)

lemma *Rf_subs_Rf_diff_Rf*: $Rf\ N \subseteq Rf\ (N - Rf\ N)$

proof
fix C
assume $c_rf: C \in Rf\ N$
then obtain CC **where**
cc_subs: set_mset CC $\subseteq N - Rf\ N$ and
cc_imp_c: $\forall I. I \models_m CC \longrightarrow I \models C$ and
cc_lt_c: $\forall C'. C' \in\# CC \longrightarrow C' < C$
using *Rf_imp_ex_non_Rf* by *blast*
have $\forall D. D \in\# CC \longrightarrow D \notin Rf\ N$
using *cc_subs* by (*simp add: subset_iff*)
then have *cc_nr*:
 $\bigwedge C\ DD. C \in\# CC \implies set_mset\ DD \subseteq N \implies \forall I. I \models_m DD \longrightarrow I \models C \implies \exists D. D \in\# DD \wedge \sim D < C$
unfolding *Rf_def* by *auto metis*
have $set_mset\ CC \subseteq N$
using *cc_subs* by *auto*
then have $set_mset\ CC \subseteq N - \{C. \exists DD. set_mset\ DD \subseteq N \wedge (\forall I. I \models_m DD \longrightarrow I \models C) \wedge (\forall D. D \in\# DD \longrightarrow D < C)\}$
using *cc_nr* by *auto*
then show $C \in Rf\ (N - Rf\ N)$
using *cc_imp_c cc_lt_c unfolding Rf_def* by *auto*
qed

lemma *Rf_eq_Rf_diff_Rf*: $Rf\ N = Rf\ (N - Rf\ N)$
by (*metis Diff_subset Rf_mono Rf_subs_Rf_diff_Rf subset_antisym*)

The following results correspond to Lemma 4.6.

lemma *Ri_mono*: $N \subseteq N' \implies Ri\ N \subseteq Ri\ N'$
unfolding *Ri_def* by *auto*

lemma *Ri_subs_Ri_diff_Rf*: $Ri\ N \subseteq Ri\ (N - Rf\ N)$

proof
fix γ
assume $\gamma_ri: \gamma \in Ri\ N$
then obtain $CC\ D\ E$ **where** $\gamma: \gamma = Infer\ CC\ D\ E$
by (*cases γ*)
have $cc: CC = side_prems_of\ \gamma$ **and** $d: D = main_prem_of\ \gamma$ **and** $e: E = concl_of\ \gamma$
unfolding γ by *simp_all*
obtain DD **where**
 $set_mset\ DD \subseteq N$ **and** $\forall I. I \models_m DD + CC \longrightarrow I \models E$ **and** $\forall C. C \in\# DD \longrightarrow C < D$
using γ_ri **unfolding** *Ri_def cc d e* by *blast*

then obtain DD' **where**
set.mset $DD' \subseteq N - Rf\ N$ **and** $\forall I. I \models_m DD' + CC \longrightarrow I \models E$ **and** $\forall D'. D' \in\# DD' \longrightarrow D' < D$
using *wlog_non_Rf* **by** *atomize_elim blast*
then show $\gamma \in Ri\ (N - Rf\ N)$
using γ_{ri} **unfolding** *Ri_def d cc e* **by** *blast*
qed

lemma *Ri_eq_Ri_diff_Rf*: $Ri\ N = Ri\ (N - Rf\ N)$
by (*metis Diff_subset Ri_mono Ri_subs_Ri_diff_Rf subset_antisym*)

lemma *Ri_subset_Gamma*: $Ri\ N \subseteq \Gamma$
unfolding *Ri_def* **by** *blast*

lemma *Rf_indep*: $N' \subseteq Rf\ N \implies Rf\ N \subseteq Rf\ (N - N')$
by (*metis Diff_cancel Diff_eq_empty_iff Diff_mono Rf_eq_Rf_diff_Rf Rf_mono*)

lemma *Ri_indep*: $N' \subseteq Rf\ N \implies Ri\ N \subseteq Ri\ (N - N')$
by (*metis Diff_mono Ri_eq_Ri_diff_Rf Ri_mono order_refl*)

lemma *Rf_model*:
assumes $I \models_s N - Rf\ N$
shows $I \models_s N$
proof –
have $I \models_s Rf\ (N - Rf\ N)$
unfolding *true_cls_def*
by (*subst Rf_def, simp add: true_cls_mset_def, metis assms subset_eq true_cls_def*)
then have $I \models_s Rf\ N$
using *Rf_subs_Rf_diff_Rf true_cls_mono* **by** *blast*
then show *?thesis*
using *assms* **by** (*metis Un_Diff_cancel true_cls_union*)
qed

lemma *Rf_sat*: *satisfiable* $(N - Rf\ N) \implies$ *satisfiable* N
by (*metis Rf_model*)

The following corresponds to Theorem 4.7:

sublocale *redundancy_criterion* $\Gamma\ Rf\ Ri$
by *unfold_locales (rule Ri_subset_Gamma, (elim Rf_mono Ri_mono Rf_indep Ri_indep Rf_sat)+)*

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 $\gamma \in Ri\ N$
proof –
obtain $CC\ D\ E$ **where**
 $\gamma: \gamma = Infer\ CC\ D\ E$
by (*cases* γ)
then have $cc: CC = side_prems_of\ \gamma$ **and** $d: D = main_prem_of\ \gamma$ **and** $e: E = concl_of\ \gamma$
unfolding γ **by** *simp_all*
note $e_{in_n_un_rf_n} = concl_of_in_n_un_rf_n[folded\ e]$

{
assume $E \in N$
moreover have $E < D$
using $\Gamma_reductive\ e\ d\ in_gamma$ **by** *auto*

ultimately have
set.mset $\{\#E\# \} \subseteq N$ and $\forall I. I \models_m \{\#E\# \} + CC \longrightarrow I \models E$ and $\forall D'. D' \in \# \{\#E\# \} \longrightarrow D' < D$
by *simp.all*
then have *redundant_infer* $N \ \gamma$
using *cc d e* **by** *blast*
}
moreover
{
assume $E \in Rf \ N$
then obtain DD **where**
dd.sset: $set.mset \ DD \subseteq N$ and
dd.imp.e: $\forall I. I \models_m DD \longrightarrow I \models E$ and
dd.lt.e: $\forall C'. C' \in \# \ DD \longrightarrow C' < E$
unfolding *Rf.def* **by** *blast*
from *dd.lt.e* **have** $\forall Da. Da \in \# \ DD \longrightarrow Da < D$
using *d e in.γ Γ_reductive less_trans* **by** *blast*
then have *redundant_infer* $N \ \gamma$
using *dd.sset dd.imp.e cc d e* **by** *blast*
}
ultimately show $\gamma \in Ri \ N$
using *in.γ e.in.n.un.rf.n* **unfolding** *Ri.def* **by** *blast*
qed

sublocale *effective_redundancy_criterion* $\Gamma \ Rf \ Ri$
unfolding *effective_redundancy_criterion_def*
by (*intro conjI redundancy_criterion_axioms, unfold_locales, rule Ri_effective*)

lemma *contradiction_Rf*: $\{\#\} \in N \implies Ri \ N = \Gamma$
unfolding *Ri_def* **using** $\Gamma_reductive \ le_multiset_empty_right$
by (*force intro: exI[of - $\{\#\#\}$] le_multiset_empty_left*)

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 (*rule ccontr*)
assume *ec.ni.n*: $\{\#\} \notin N$

define M **where**
 $M = N - Rf \ N$

have *ec.ni.m*: $\{\#\} \notin M$
unfolding *M_def* **using** *ec.ni.n* **by** *fast*

have *L.of* $M \models_s M$
proof (*rule ccontr*)
assume \neg *L.of* $M \models_s M$
then obtain D **where**
d.in.m: $D \in M$ and
d.cex: \neg *L.of* $M \models D$ and
d.min: $\bigwedge C. C \in M \implies C < D \implies L.of \ M \models C$
using *ex_min_counterex* **by** *meson*
then obtain $\gamma \ CC \ E$ **where**
 γ : $\gamma = Infer \ CC \ D \ E$ and
cc.subs.m: $set.mset \ CC \subseteq M$ and

```

cc_true: L_of M  $\models$  CC and
 $\gamma$ _in:  $\gamma \in \Gamma$  and
e_cex:  $\neg$  L_of M  $\models$  E and
e_lt_d: E < D
using  $\Gamma$ _counterex_reducing[OF ec_ni_m] not_less by metis
have cc: CC = side_prem_of  $\gamma$  and d: D = main_prem_of  $\gamma$  and e: E = concl_of  $\gamma$ 
unfolding  $\gamma$  by simp_all
have  $\gamma \in Ri$  N
by (rule set_mp[OF satur[unfolding saturated_upto_def inferences_from_def infer_from_def]])
(simp add:  $\gamma$ _in d_in_m cc_subs_m cc[symmetric] d[symmetric] M_def[symmetric])
then have  $\gamma \in Ri$  M
unfolding M_def using Ri_indep by fast
then obtain DD where
dd_subs_m: set_mset DD  $\subseteq$  M and
dd_cc_imp_d:  $\forall I. I \models DD + CC \longrightarrow I \models E$  and
dd_lt_d:  $\forall C. C \in \# DD \longrightarrow C < D$ 
unfolding Ri_def cc d e by blast
from dd_subs_m dd_lt_d have L_of M  $\models$  DD
using d_min unfolding true_cls_mset_def by (metis contra_subsetD)
then have L_of M  $\models$  E
using dd_cc_imp_d cc_true by auto
then show False
using e_cex by auto
qed
then have L_of M  $\models$  N
using M_def Rf_model by blast
then show False
using unsat by blast
qed

```

```

theorem saturated_upto_complete:
assumes saturated_upto N
shows  $\neg$  satisfiable N  $\longleftrightarrow$  {#}  $\in$  N
using assms saturated_upto_complete_if_true_cls_def by auto

```

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 atm_of_atms renamings_apart 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 σ) (B \cdot a σ)

begin

14.1 Library

lemma *Bex_cartesian_product*: $(\exists xy \in A \times B. P \ xy) \equiv (\exists x \in A. \exists y \in B. P \ (x, y))$
by *simp*

lemma *length_sorted_list_of_multiset*[*simp*]: $\text{length} \ (\text{sorted_list_of_multiset} \ A) = \text{size} \ A$
by (*metis mset_sorted_list_of_multiset size_mset*)

lemma *eql_map_neg_lit_eql_atm*:
assumes *map* $(\lambda L. L \cdot l \ \eta) \ (\text{map} \ \text{Neg} \ As') = \text{map} \ \text{Neg} \ As$
shows $As' \cdot al \ \eta = As$
using *assms* by (*induction As' arbitrary: As*) *auto*

lemma *instance_list*:
assumes *negs* $(\text{mset} \ As) = SDA' \cdot \eta$
shows $\exists As'. \text{negs} \ (\text{mset} \ As') = SDA' \wedge As' \cdot al \ \eta = As$
proof –
from *assms* have *negL*: $\forall L \in \# \ SDA'. \text{is_neg} \ L$
using *Melem_subst_cls subst_lit_in_negs_is_neg* by *metis*

from *assms* have $\{\#L \cdot l \ \eta. L \in \# \ SDA'\} = \text{mset} \ (\text{map} \ \text{Neg} \ As)$
using *subst_cls_def* by *auto*
then have $\exists NAs'. \text{map} \ (\lambda L. L \cdot l \ \eta) \ NAs' = \text{map} \ \text{Neg} \ As \wedge \text{mset} \ NAs' = SDA'$
using *image_mset_of_subset_list*[of $\lambda L. L \cdot l \ \eta \ SDA' \ \text{map} \ \text{Neg} \ As$] by *auto*
then obtain *As'_p* where *As'_p*:
 $\text{map} \ (\lambda L. L \cdot l \ \eta) \ (\text{map} \ \text{Neg} \ As') = \text{map} \ \text{Neg} \ As \wedge \text{mset} \ (\text{map} \ \text{Neg} \ As') = SDA'$
by (*metis (no_types, lifting) Neg_atm_of_iff negL ex_map_conv set_mset_mset*)

have $\text{negs} \ (\text{mset} \ As') = SDA'$
using *As'_p* by *auto*
moreover have $\text{map} \ (\lambda L. L \cdot l \ \eta) \ (\text{map} \ \text{Neg} \ As') = \text{map} \ \text{Neg} \ As$
using *As'_p* by *auto*
then have $As' \cdot al \ \eta = As$
using *eql_map_neg_lit_eql_atm* by *auto*
ultimately show *?thesis*
by *blast*

qed

context

fixes *S* :: 'a clause \Rightarrow 'a clause

begin

14.2 Calculus

The following corresponds to Figure 4.

definition *maximal_wrt* :: 'a \Rightarrow 'a literal multiset \Rightarrow bool **where**
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 literal multiset \Rightarrow bool **where**
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*: *strictly_maximal_wrt* *A C* \Longrightarrow *maximal_wrt* *A C*
unfolding *maximal_wrt_def* *strictly_maximal_wrt_def* by *auto*

inductive *eligible* :: 's \Rightarrow 'a list \Rightarrow 'a clause \Rightarrow bool **where**

eligible:

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

eligible σ 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:

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

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

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

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

$n \neq 0 \Longrightarrow$

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

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

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

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

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

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

$\text{ord_resolve } CAs \ (D + \text{negs } (\text{mset } As)) \ AAs \ As \ \sigma \ (((\bigcup \# \ \text{mset } Cs) + D) \cdot \sigma)$

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:

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

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

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

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

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

$q = \text{hd } (\text{renamings_apart } (DA \ \# \ CAs)) \Longrightarrow$

$qs = \text{tl } (\text{renamings_apart } (DA \ \# \ CAs)) \Longrightarrow$

$\text{ord_resolve } (CAs \ \cdot\text{-cl } qs) \ (DA \cdot q) \ (AAs \ \cdot\text{-aml } qs) \ (As \cdot\text{-al } q) \ \sigma \ E \Longrightarrow$

$\text{ord_resolve_rename } CAs \ DA \ AAs \ As \ \sigma \ E$

lemma *ord_resolve_empty_main_prem:* $\neg \text{ord_resolve } Cs \ \{\#\} \ AAs \ As \ \sigma \ E$

by (*simp add: ord_resolve.simps*)

lemma *ord_resolve_rename_empty_main_prem:* $\neg \text{ord_resolve_rename } Cs \ \{\#\} \ AAs \ As \ \sigma \ E$

by (*simp add: ord_resolve_empty_main_prem ord_resolve_rename.simps*)

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: $I \models_m \text{mset } CAs \cdot \text{cm } \sigma \cdot \text{cm } \eta$ **and**

d_inst_true: $I \models DA \cdot \sigma \cdot \eta$ **and**

ground_subst_η: $\text{is_ground_subst } \eta$

shows $I \models E \cdot \eta$

using *res_e*

proof (*cases rule: ord_resolve.cases*)

case (*ord_resolve n Cs D*)

note *da* = *this(1)* **and** *e* = *this(2)* **and** *cas_len* = *this(3)* **and** *cs_len* = *this(4)* **and**

aas_len = *this(5)* **and** *as_len* = *this(6)* **and** *cas* = *this(8)* **and** *mgu* = *this(10)* **and**

len = *this(1)*

have *len:* $\text{length } CAs = \text{length } As$

using *as_len cas_len* **by** *auto*

have *is_ground_subst* ($\sigma \odot \eta$)

using *ground_subst_η* **by** (*rule is_ground_comp_subst*)

then have *cc_true:* $I \models_m \text{mset } CAs \cdot \text{cm } \sigma \cdot \text{cm } \eta$ **and** *d_true:* $I \models DA \cdot \sigma \cdot \eta$


```

using cc_inst_true d_inst_true by auto

from mgu have unif:  $\forall i < n. \forall A \in \#AAs ! i. A \cdot a \sigma = As ! i \cdot a \sigma$ 
using mgu_unifier as_len aas_len by blast

show  $I \models E \cdot \eta$ 
proof (cases  $\forall A \in set As. A \cdot a \sigma \cdot a \eta \in I$ )
  case True
  then have  $\neg I \models negs (mset As) \cdot \sigma \cdot \eta$ 
  unfolding true_cls_def[of I] by auto
  then have  $I \models D \cdot \sigma \cdot \eta$ 
  using d_true da by auto
  then show ?thesis
  unfolding e by auto
next
  case False
  then obtain i where a_in_aa:  $i < length CAs$  and a_false:  $(As ! i) \cdot a \sigma \cdot a \eta \notin I$ 
  using da len by (metis in_set_conv_nth)
  define C where  $C \equiv Cs ! i$ 
  define BB where  $BB \equiv AAs ! i$ 
  have c_cf':  $C \subseteq \# \cup \# mset CAs$ 
  unfolding C_def using a_in_aa cas cas_len
  by (metis less_subset_eq_Union_mset mset_subset_eq_add_left subset_mset.order.trans)
  have c_in_cc:  $C + poss BB \in \# mset CAs$ 
  using C_def BB_def a_in_aa cas_len in_set_conv_nth cas by fastforce
  {
  fix B
  assume  $B \in \# BB$ 
  then have  $B \cdot a \sigma = (As ! i) \cdot a \sigma$ 
  using unif a_in_aa cas_len unfolding BB_def by auto
  }
  then have  $\neg I \models poss BB \cdot \sigma \cdot \eta$ 
  using a_false by (auto simp: true_cls_def)
  moreover have  $I \models (C + poss BB) \cdot \sigma \cdot \eta$ 
  using c_in_cc cc_true true_cls_mset_true_cls[of I mset CAs ·cm  $\sigma$  ·cm  $\eta$ ] by force
  ultimately have  $I \models C \cdot \sigma \cdot \eta$ 
  by simp
  then show ?thesis
  unfolding e subst_cls_union using c_cf' C_def a_in_aa cas_len cs_len
  by (metis (no_types, lifting) mset_subset_eq_add_left nth_mem_mset set_mset_mono sum_mset.remove true_cls_mono
  subst_cls_mono)
qed
qed

```

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 σ E and

$\varrho_s = tl (renamings_apart (DA \# CAs))$ and

$\varrho = hd (renamings_apart (DA \# CAs))$ and

$I \models_m (mset (CAs \cdot cl \varrho_s)) \cdot cm \sigma \cdot cm \eta$ and

$I \models DA \cdot \varrho \cdot \sigma \cdot \eta$ and

is_ground_subst η

shows $I \models E \cdot \eta$

using *assms* by (cases rule: *ord_resolve_rename.cases*) (fast intro: *ord_resolve_ground_inst_sound*)

Here follows the soundness theorem for the resolution rule.

theorem *ord_resolve_sound*:

assumes

res_e: ord_resolve CAs DA AAs As σ E and

cc_d_true: $\bigwedge \sigma. is_ground_subst \sigma \implies I \models_m (mset CAs + \{\#DA\}) \cdot cm \sigma$ and

ground_subst_ η : is_ground_subst η

shows $I \models E \cdot \eta$
proof (use *res.e* in *(cases rule: ord_resolve.cases)*)
case (*ord_resolve n Cs D*)
note $da = \text{this}(1)$ **and** $e = \text{this}(2)$ **and** $cas_len = \text{this}(3)$ **and** $cs_len = \text{this}(4)$
and $aas_len = \text{this}(5)$ **and** $as_len = \text{this}(6)$ **and** $cas = \text{this}(8)$ **and** $mgu = \text{this}(10)$
have *ground_subst_σ_η*: *is_ground_subst* ($\sigma \odot \eta$)
using *ground_subst_η* **by** (*rule is_ground_comp_subst*)
have *cas_true*: $I \models_m \text{mset } CAs \cdot cm \sigma \cdot cm \eta$
using *cc_d.true ground_subst_σ_η* **by** *fastforce*
have *da_true*: $I \models DA \cdot \sigma \cdot \eta$
using *cc_d.true ground_subst_σ_η* **by** *fastforce*
show $I \models E \cdot \eta$
using *ord_resolve_ground_inst_sound[OF res.e cas_true da_true]* *ground_subst_η* **by** *auto*
qed

lemma *subst_sound*:
assumes
 $\bigwedge \sigma. \text{is_ground_subst } \sigma \implies I \models (C \cdot \sigma)$ **and**
is_ground_subst η
shows $I \models (C \cdot \varrho) \cdot \eta$
using *assms is_ground_comp_subst subst_cls_comp_subst* **by** *metis*

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: *is_ground_subst* η
shows $I \models_m \text{mset } (CAs \cdot cl P) \cdot cm \eta$
proof –
from *true_cas* **have** $\bigwedge CA. CA \in \# \text{mset } CAs \implies (\bigwedge \sigma. \text{is_ground_subst } \sigma \implies I \models CA \cdot \sigma)$
unfolding *true_cls_mset_def* **by** *force*
then **have** $\forall i < \text{length } CAs. \forall \sigma. \text{is_ground_subst } \sigma \longrightarrow (I \models CAs ! i \cdot \sigma)$
using *in_set_conv_nth* **by** *auto*
then **have** *true_cp*: $\forall i < \text{length } CAs. \forall \sigma. \text{is_ground_subst } \sigma \longrightarrow I \models CAs ! i \cdot P ! i \cdot \sigma$
using *subst_sound len* **by** *auto*

{
fix *CA*
assume $CA \in \# \text{mset } (CAs \cdot cl P)$
then **obtain** *i* **where**
 $i.x: i < \text{length } (CAs \cdot cl P) \text{ } CA = (CAs \cdot cl P) ! i$
by (*metis in_mset_conv_nth*)
then **have** $\forall \sigma. \text{is_ground_subst } \sigma \longrightarrow I \models CA \cdot \sigma$
using *true_cp* **unfolding** *subst_cls_lists_def* **by** (*simp add: len*)
}
then **show** *?thesis*
using *assms* **unfolding** *true_cls_mset_def* **by** *auto*
qed

Here follows the soundness theorem for the resolution rule with renaming.

lemma *ord_resolve_rename_sound*:
assumes
res.e: *ord_resolve_rename CAs DA AAs As σ 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: *is_ground_subst* η
shows $I \models E \cdot \eta$
using *res.e*
proof (*cases rule: ord_resolve_rename.cases*)
case (*ord_resolve_rename n ρ qs*)
note $qs = \text{this}(7)$ **and** $res = \text{this}(8)$
have *len*: $\text{length } qs = \text{length } CAs$
using *qs renames_apart* **by** *auto*
have $\bigwedge \sigma. \text{is_ground_subst } \sigma \implies I \models_m (\text{mset } (CAs \cdot cl qs) + \{\#DA \cdot \varrho\# \}) \cdot cm \sigma$

```

using subst_sound_scl[OF len, of I] subst_sound cc_d_true by auto
then show  $I \models E \cdot \eta$ 
using ground_subst_η ord_resolve_sound[OF res] by simp
qed

```

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  $\sigma = \sigma' \wedge E = E'$ 
using assms
proof (cases rule: ord_resolve.cases[case_product ord_resolve.cases], intro conjI)
case (ord_resolve_ord_resolve CAs n Cs AAs As  $\sigma''$  DA CAs' n' Cs' AAs' As'  $\sigma'''$  DA')
note res = this(1-17) and res' = this(18-34)

show  $\sigma = \sigma'$ 
using res(3-5,14) res'(3-5,14) by (metis option.inject)

have Cs = Cs'
using res(1,3,7,8,12) res'(1,3,7,8,12) by (metis add_right_imp_eq nth_equalityI)
moreover have DA = DA'
using res(2,4) res'(2,4) by fastforce
ultimately show E = E'
using res(5,6) res'(5,6)  $\sigma$  by blast
qed

```

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  $\sigma = \sigma' \wedge E = E'$ 
using assms unfolding ord_resolve_rename.simps using ord_resolve_unique by meson

```

lemma *ord_resolve_max_side_premis*: $\text{ord_resolve } CAs \ DA \ AAs \ As \ \sigma \ E \implies \text{length } CAs \leq \text{size } DA$
by (auto elim!: ord_resolve.cases)

lemma *ord_resolve_rename_max_side_premis*:

```

ord_resolve_rename CAs DA AAs As  $\sigma$  E  $\implies \text{length } CAs \leq \text{size } DA$ 
by (elim ord_resolve_rename.cases, drule ord_resolve_max_side_premis, simp add: renames_apart)

```

14.5 Inference System

definition *ord_FO_Γ* :: 'a inference set **where**

```

ord_FO_Γ = {Infer (mset CAs) DA E | CAs DA AAs As  $\sigma$  E. ord_resolve_rename CAs DA AAs As  $\sigma$  E}

```

interpretation *ord_FO_resolution*: inference_system ord_FO_Γ .

lemma *exists_compose*: $\exists x. P(f x) \implies \exists y. P y$
by meson

lemma *finite_ord_FO_resolution_inferences_between*:

```

assumes fin_cc: finite CC
shows finite (ord_FO_resolution.inferences_between CC C)
proof -
let ?CCC = CC  $\cup$  {C}

```

```

define all_AA where all_AA = ( $\bigcup D \in ?CCC. \text{atms\_of } D$ )
define max_ary where max_ary = Max (size ' ?CCC)
define CAS where CAS = {CAs. CAs  $\in$  lists ?CCC  $\wedge$  length CAs  $\leq$  max_ary}
define AS where AS = {As. As  $\in$  lists all_AA  $\wedge$  length As  $\leq$  max_ary}
define AAs where AAs = {AAs. AAs  $\in$  lists (mset ' AS)  $\wedge$  length AAs  $\leq$  max_ary}

```

```

note defs = all_AA_def max_ary_def CAS_def AS_def AAS_def

let ?infer_of =
  λCAS DA AAs As. Infer (mset CAS) DA (THE E. ∃σ. ord_resolve_rename CAS DA AAs As σ E)

let ?Z = {γ | CAS DA AAs As σ E γ. γ = Infer (mset CAS) DA E
  ∧ ord_resolve_rename CAS DA AAs As σ E ∧ infer_from ?CCC γ ∧ C ∈# prems_of γ}
let ?Y = {Infer (mset CAS) DA E | CAS DA AAs As σ E.
  ord_resolve_rename CAS DA AAs As σ E ∧ set CAS ∪ {DA} ⊆ ?CCC}
let ?X = {?infer_of CAS DA AAs As | CAS DA AAs As. CAS ∈ CAS ∧ DA ∈ ?CCC ∧ AAs ∈ AAS ∧ As ∈ AS}
let ?W = CAS × ?CCC × AAS × AS

have fin_w: finite ?W
  unfolding defs using fin_cc by (simp add: finite_lists_length_le lists_eq_set)

have ?Z ⊆ ?Y
  by (force simp: infer_from_def)
also have ... ⊆ ?X
proof -
  {
    fix CAS DA AAs As σ E
    assume
      res_e: ord_resolve_rename CAS DA AAs As σ E and
      da_in: DA ∈ ?CCC and
      cas_sub: set CAS ⊆ ?CCC

    have E = (THE E. ∃σ. ord_resolve_rename CAS DA AAs As σ E)
      ∧ CAS ∈ CAS ∧ AAs ∈ AAS ∧ As ∈ AS (is ?e ∧ ?cas ∧ ?aas ∧ ?as)
    proof (intro conjI)
      show ?e
        using res_e ord_resolve_rename_unique by (blast intro: the_equality[symmetric])
    next
      show ?cas
        unfolding CAS_def max_ary_def using cas_sub
          ord_resolve_rename_max_side_premis[OF res_e] da_in fin_cc
        by (auto simp add: Max_ge_iff)
    next
      show ?aas
        using res_e
    proof (cases rule: ord_resolve_rename.cases)
      case (ord_resolve_rename n ρ ρs)
        note len_cas = this(1) and len_aas = this(2) and len_as = this(3) and
          aas_sub = this(4) and as_sub = this(5) and res_e' = this(8)

        show ?thesis
          unfolding AAS_def
        proof (clarify, intro conjI)
          show AAs ∈ lists (mset 'AS)
            unfolding AS_def image_def
          proof clarsimp
            fix AA
            assume AA ∈ set AAs
            then obtain i where
              i_lt: i < n and
              aa: AA = AAs ! i
            by (metis in_set_conv_nth len_aas)

          have casi_in: CAS ! i ∈ ?CCC
            using i_lt len_cas cas_sub nth_mem by blast

          have pos_aa_sub: poss AA ⊆# CAS ! i
            using aa aas_sub i_lt by blast
          then have set_mset AA ⊆ atms_of (CAS ! i)
  }

```

```

    by (metis atms_of_poss lits_subseteq_imp_atms_subseteq set_mset_mono)
  also have aa_sub: ...  $\subseteq$  all_AA
    unfolding all_AA_def using casi_in by force
  finally have aa_sub: set_mset AA  $\subseteq$  all_AA
  .

  have size AA = size (poss AA)
    by simp
  also have ...  $\leq$  size (CAs ! i)
    by (rule size_mset_mono[OF pos_aa_sub])
  also have ...  $\leq$  max_ary
    unfolding max_ary_def using fin_cc casi_in by auto
  finally have sz_aa: size AA  $\leq$  max_ary
  .

  let ?As' = sorted_list_of_multiset AA

  have ?As'  $\in$  lists all_AA
    using aa_sub by auto
  moreover have length ?As'  $\leq$  max_ary
    using sz_aa by simp
  moreover have AA = mset ?As'
    by simp
  ultimately show  $\exists xa. xa \in$  lists all_AA  $\wedge$  length xa  $\leq$  max_ary  $\wedge$  AA = mset xa
    by blast
qed
next
  have length AAs = length As
    unfolding len_aas len_as ..
  also have ...  $\leq$  size DA
    using as_sub size_mset_mono by fastforce
  also have ...  $\leq$  max_ary
    unfolding max_ary_def using fin_cc da_in by auto
  finally show length AAs  $\leq$  max_ary
  .
qed
qed
next
  show ?as
    unfolding AS_def
  proof (clarify, intro conjI)
    have set As  $\subseteq$  atms_of DA
      using res_e[simplified ord_resolve_rename.simps]
      by (metis atms_of_negs lits_subseteq_imp_atms_subseteq set_mset_mono set_mset_mset)
    also have ...  $\subseteq$  all_AA
      unfolding all_AA_def using da_in by blast
    finally show As  $\in$  lists all_AA
      unfolding lists_eq_set by simp
  next
    have length As  $\leq$  size DA
      using res_e[simplified ord_resolve_rename.simps]
      ord_resolve_rename_max_side_premis[OF res_e] by auto
    also have size DA  $\leq$  max_ary
      unfolding max_ary_def using fin_cc da_in by auto
    finally show length As  $\leq$  max_ary
  .
qed
qed
}
then show ?thesis
  by simp fast
qed
also have ...  $\subseteq$   $(\lambda(CAs, DA, AAs, As). ?infer\_of\ CAs\ DA\ AAs\ As)$  ' ?W

```

unfolding *image_def Bex-cartesian_product* **by** *fast*
finally show *?thesis*
unfolding *inference_system.inferences_between_def ord_FO_Γ_def mem_Collect_eq*
by (*fast intro: rev_finite_subset[OF finite_imageI[OF fn_w]]*)
qed

lemma *ord_FO_resolution_inferences_between_empty_empty:*
ord_FO_resolution.inferences_between {} {#} = {}
unfolding *ord_FO_resolution.inferences_between_def inference_system.inferences_between_def*
infer_from_def ord_FO_Γ_def
using *ord_resolve_rename_empty_main_prem* **by** *auto*

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*
by (*rule select*)

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 grounding_of_class M*
obtains *D σ where*

D \in M \wedge C = D \cdot σ \wedge S_M C = S D \cdot σ \wedge is_ground_subst σ
proof (*atomize_elim, unfold S_M_def eqTrueI[OF assms] if_True, rule someI_ex*)
from *assms* **show** $\exists C' D \sigma. D \in M \wedge C = D \cdot \sigma \wedge C' = S D \cdot \sigma \wedge is_ground_subst \sigma$
by (*auto simp: grounding_of_class_def grounding_of_cls_def*)
qed

lemma *S_M_not_grounding_of_class: C \notin grounding_of_class M \implies S_M C = S C*
unfolding *S_M_def* **by** *simp*

lemma *S_M_selects_subseteq: S_M C \subseteq # C*
by (*metis S_M_grounding_of_class S_M_not_grounding_of_class S_selects_subseteq subst_cls_mono_mset*)

lemma *S_M_selects_neg_lits: L \in # S_M C \implies is_neg L*
by (*metis Melem_subst_cls S_M_grounding_of_class S_M_not_grounding_of_class S_selects_neg_lits*
subst_lit_is_neg)

end

end

The following corresponds to Lemma 4.12:

lemma *map2_add_mset_map:*
assumes *length AAs' = n and length As' = n*
shows *map2 add_mset (As' \cdot al η) (AAs' \cdot aml η) = map2 add_mset As' AAs' \cdot aml η*
using *assms*
proof (*induction n arbitrary: AAs' As'*)
case (*Suc n*)
then have *map2 add_mset (tl (As' \cdot al η)) (tl (AAs' \cdot aml η)) = map2 add_mset (tl As') (tl AAs') \cdot aml η*
by *simp*
moreover

have $Succ: length (As' \cdot al \ \eta) = Suc \ n \ length (AAs' \cdot aml \ \eta) = Suc \ n$
using $Suc(3) \ Suc(2)$ **by** *auto*
then have $length (tl (As' \cdot al \ \eta)) = n \ length (tl (AAs' \cdot aml \ \eta)) = n$
by *auto*
then have $length (map2 \ add_mset (tl (As' \cdot al \ \eta)) (tl (AAs' \cdot aml \ \eta))) = n$
 $length (map2 \ add_mset (tl \ As') (tl \ AAs') \cdot aml \ \eta) = n$
using $Suc(2,3)$ **by** *auto*
ultimately have $\forall i < n. tl (map2 \ add_mset ((As' \cdot al \ \eta)) ((AAs' \cdot aml \ \eta))) ! i =$
 $tl (map2 \ add_mset (As') (AAs') \cdot aml \ \eta) ! i$
using $Suc(2,3) \ Succ$ **by** (*simp add: map2_tl map_tl subst_atm_mset_list_def del: subst_atm_list_tl*)
moreover have $nn: length (map2 \ add_mset ((As' \cdot al \ \eta)) ((AAs' \cdot aml \ \eta))) = Suc \ n$
 $length (map2 \ add_mset (As') (AAs') \cdot aml \ \eta) = Suc \ n$
using $Succ \ Suc$ **by** *auto*
ultimately have $\forall i. i < Suc \ n \longrightarrow i > 0 \longrightarrow$
 $map2 \ add_mset (As' \cdot al \ \eta) (AAs' \cdot aml \ \eta) ! i = (map2 \ add_mset \ As' \ AAs' \cdot aml \ \eta) ! i$
by (*auto simp: subst_atm_mset_list_def gr0_conv_Suc subst_atm_mset_def*)
moreover have $add_mset (hd \ As' \cdot a \ \eta) (hd \ AAs' \cdot am \ \eta) = add_mset (hd \ As') (hd \ AAs') \cdot am \ \eta$
unfolding $subst_atm_mset_def$ **by** *auto*
then have $(map2 \ add_mset (As' \cdot al \ \eta) (AAs' \cdot aml \ \eta)) ! 0 = (map2 \ add_mset (As') (AAs') \cdot aml \ \eta) ! 0$
using Suc **by** (*simp add: Succ(2) subst_atm_mset_def*)
ultimately have $\forall i < Suc \ n. (map2 \ add_mset (As' \cdot al \ \eta) (AAs' \cdot aml \ \eta)) ! i =$
 $(map2 \ add_mset (As') (AAs') \cdot aml \ \eta) ! i$
using Suc **by** *auto*
then show *?case*
using $nn \ list_eq_iff_nth_eq$ **by** *metis*
qed *auto*

lemma $maximal_wrt_subst: maximal_wrt (A \cdot a \ \sigma) (C \cdot \sigma) \Longrightarrow maximal_wrt \ A \ C$
unfolding $maximal_wrt_def$ **using** $in_atms_of_subst \ less_atm_stable$ **by** *blast*

lemma $strictly_maximal_wrt_subst: strictly_maximal_wrt (A \cdot a \ \sigma) (C \cdot \sigma) \Longrightarrow strictly_maximal_wrt \ A \ C$
unfolding $strictly_maximal_wrt_def$ **using** $in_atms_of_subst \ less_atm_stable$ **by** *blast*

lemma $ground_resolvent_subset:$

assumes

$gr_cas: is_ground_cls_list \ CAs$ **and**

$gr_da: is_ground_cls \ DA$ **and**

$res_e: ord_resolve \ S \ CAs \ DA \ AAs \ As \ \sigma \ E$

shows $E \subseteq \# (\bigcup \# \ mset \ CAs) + DA$

using res_e

proof (*cases rule: ord_resolve.cases*)

case ($ord_resolve \ n \ Cs \ D$)

note $da = this(1)$ **and** $e = this(2)$ **and** $cas_len = this(3)$ **and** $cs_len = this(4)$

and $aas_len = this(5)$ **and** $as_len = this(6)$ **and** $cas = this(8)$ **and** $mgu = this(10)$

then have $cs_sub_cas: \bigcup \# \ mset \ Cs \subseteq \# \bigcup \# \ mset \ CAs$

using $subseq_list_Union_mset \ cas_len \ cs_len$ **by** *force*

then have $cs_sub_cas: \bigcup \# \ mset \ Cs \subseteq \# \bigcup \# \ mset \ CAs$

using $subseq_list_Union_mset \ cas_len \ cs_len$ **by** *force*

then have $gr_cs: is_ground_cls_list \ Cs$

using gr_cas **by** *simp*

have $d_sub_da: D \subseteq \# \ DA$

by (*simp add: da*)

then have $gr_d: is_ground_cls \ D$

using $gr_da \ is_ground_cls_mono$ **by** *auto*

have $is_ground_cls (\bigcup \# \ mset \ Cs + D)$

using $gr_cs \ gr_d$ **by** *auto*

with e **have** $E = (\bigcup \# \ mset \ Cs + D)$

by *auto*

then show *?thesis*

using $cs_sub_cas \ d_sub_da$ **by** (*auto simp: subset_mset.add_mono*)

qed

lemma *ord_resolve_obtain_clauses*:

assumes

res.e: *ord_resolve* (*S M S M*) *CAs DA AAs As* σ *E* **and**

select: *selection S* **and**

grounding: $\{DA\} \cup \text{set } CAs \subseteq \text{grounding_of_clss } M$ **and**

n: *length CAs* = *n* **and**

d: *DA* = *D* + *negs (mset As)* **and**

c: $(\forall i < n. CAs ! i = Cs ! i + \text{poss } (AAs ! i))$ *length Cs* = *n* *length AAs* = *n*

obtains *DA0 η 0 CAs0 η s0 As0 AAs0 D0 Cs0* **where**

length CAs0 = *n*

length η s0 = *n*

DA0 \in *M*

DA0 \cdot *η 0* = *DA*

S DA0 \cdot *η 0* = *S M S M DA*

$\forall CA0 \in \text{set } CAs0. CA0 \in M$

CAs0 \cdot *cl η s0* = *CAs*

map S CAs0 \cdot *cl η s0* = *map (S M S M) CAs*

is_ground_subst η 0

is_ground_subst_list η s0

As0 \cdot *al η 0* = *As*

AAs0 \cdot *aml η s0* = *AAs*

length As0 = *n*

D0 \cdot *η 0* = *D*

DA0 = *D0* + (*negs (mset As0)*)

S M S M (D + negs (mset As)) \neq $\{\#\}$ \implies *negs (mset As0)* = *S DA0*

length Cs0 = *n*

Cs0 \cdot *cl η s0* = *Cs*

$\forall i < n. CAs0 ! i = Cs0 ! i + \text{poss } (AAs0 ! i)$

length AAs0 = *n*

using *res.e*

proof (*cases rule: ord_resolve.cases*)

case (*ord_resolve n_twin Cs_twins D_twin*)

note *da* = *this(1)* **and** *e* = *this(2)* **and** *cas* = *this(8)* **and** *mgu* = *this(10)* **and** *eligible* = *this(11)*

from *ord_resolve* **have** *n_twin* = *n* *D_twin* = *D*

using *n d* **by** *auto*

moreover **have** *Cs_twins* = *Cs*

using *c cas n calculation(1)* \langle *length Cs_twins* = *n_twin* \rangle **by** (*auto simp add: nth_equalityI*)

ultimately

have *nz*: *n* \neq 0 **and** *cs_len*: *length Cs* = *n* **and** *aas_len*: *length AAs* = *n* **and** *as_len*: *length As* = *n*

and *da*: *DA* = *D* + *negs (mset As)* **and** *eligible*: *eligible (S M S M) σ As (D + negs (mset As))*

and *cas*: $\forall i < n. CAs ! i = Cs ! i + \text{poss } (AAs ! i)$

using *ord_resolve* **by** *force+*

note *n* = \langle *n* \neq 0 \rangle \langle *length CAs* = *n* \rangle \langle *length Cs* = *n* \rangle \langle *length AAs* = *n* \rangle \langle *length As* = *n* \rangle

interpret *S*: *selection S* **by** (*rule select*)

— Obtain FO side premises

have $\forall CA \in \text{set } CAs. \exists CA0 \eta c0. CA0 \in M \wedge CA0 \cdot \eta c0 = CA \wedge S CA0 \cdot \eta c0 = S M S M CA \wedge \text{is_ground_subst } \eta c0$

using *grounding S M grounding_of_clss select* **by** (*metis (no_types) le_supE subset_iff*)

then have $\forall i < n. \exists CA0 \eta c0. CA0 \in M \wedge CA0 \cdot \eta c0 = (CAs ! i) \wedge S CA0 \cdot \eta c0 = S M S M (CAs ! i) \wedge \text{is_ground_subst } \eta c0$

using *n* **by** *force*

then obtain *η s0f CAs0f* **where** *f-p*:

$\forall i < n. CAs0f i \in M$

$\forall i < n. (CAs0f i) \cdot (\eta s0f i) = (CAs ! i)$

$\forall i < n. S (CAs0f i) \cdot (\eta s0f i) = S M S M (CAs ! i)$

$\forall i < n. \text{is_ground_subst } (\eta s0f i)$

using *n* **by** (*metis (no_types)*)

define *η s0* **where**

η s0 = *map η s0f [0 ..<n]*


```

define CAs0 where
  CAs0 = map CAs0f [0 ..<n]

have length ηs0 = n length CAs0 = n
  unfolding ηs0_def CAs0_def by auto
note n = ⟨length ηs0 = n⟩ ⟨length CAs0 = n⟩ n

— The properties we need of the FO side premises
have CAs0.in_M:  $\forall CA0 \in \text{set } CAs0. CA0 \in M$ 
  unfolding CAs0_def using f_p(1) by auto
have CAs0.to_CAs: CAs0  $\cdot\text{cl}$  ηs0 = CAs
  unfolding CAs0_def ηs0_def using f_p(2) by (auto simp: n intro: nth_equalityI)
have SCAs0.to_SMCAs: (map S CAs0)  $\cdot\text{cl}$  ηs0 = map (S_M S M) CAs
  unfolding CAs0_def ηs0_def using f_p(3) n by (force intro: nth_equalityI)
have sub_ground:  $\forall \eta c0 \in \text{set } \eta s0. \text{is\_ground\_subst } \eta c0$ 
  unfolding ηs0_def using f_p n by force
then have is_ground_subst_list ηs0
  using n unfolding is_ground_subst_list_def by auto

— Split side premises CAs0 into Cs0 and AAs0
obtain AAs0 Cs0 where AAs0_Cs0_p:
  AAs0  $\cdot\text{aml}$  ηs0 = AAs length Cs0 = n Cs0  $\cdot\text{cl}$  ηs0 = Cs
   $\forall i < n. CAs0 ! i = Cs0 ! i + \text{poss } (AAs0 ! i) \text{ length } AAs0 = n$ 
proof —
  have  $\forall i < n. \exists AA0. AA0 \cdot \text{am } \eta s0 ! i = AAs ! i \wedge \text{poss } AA0 \subseteq\# CAs0 ! i$ 
  proof (rule, rule)
    fix i
    assume i < n
    have CAs0 ! i  $\cdot \eta s0 ! i = CAs ! i$ 
      using ⟨i < n⟩ ⟨CAs0  $\cdot\text{cl}$  ηs0 = CAs⟩ n by force
    moreover have poss (AAs ! i)  $\subseteq\# CAs ! i$ 
      using ⟨i < n⟩ cas by auto
    ultimately obtain poss_AA0 where
      nn: poss_AA0  $\cdot \eta s0 ! i = \text{poss } (AAs ! i) \wedge \text{poss\_AA0} \subseteq\# CAs0 ! i$ 
      using cas image_mset_of_subset unfolding subst_cls_def by metis
    then have l:  $\forall L \in\# \text{poss\_AA0}. \text{is\_pos } L$ 
      unfolding subst_cls_def by (metis Melem_subst_cls imageE literal.disc(1))
      literal.map_disc_iff set_image_mset subst_cls_def subst_lit_def)

define AA0 where
  AA0 = image_mset atm_of poss_AA0

have na: poss AA0 = poss_AA0
  using l unfolding AA0_def by auto
then have AA0  $\cdot \text{am } \eta s0 ! i = AAs ! i$ 
  using nn by (metis (mono_tags) literal.inject(1) multiset.inj_map_strong subst_cls_poss)
moreover have poss AA0  $\subseteq\# CAs0 ! i$ 
  using na nn by auto
ultimately show  $\exists AA0. AA0 \cdot \text{am } \eta s0 ! i = AAs ! i \wedge \text{poss } AA0 \subseteq\# CAs0 ! i$ 
  by blast
qed
then obtain AAs0f where
  AAs0f_p:  $\forall i < n. AAs0f i \cdot \text{am } \eta s0 ! i = AAs ! i \wedge (\text{poss } (AAs0f i)) \subseteq\# CAs0 ! i$ 
  by metis

define AAs0 where AAs0 = map AAs0f [0 ..<n]

then have length AAs0 = n
  by auto
note n = n ⟨length AAs0 = n⟩

from AAs0_def have  $\forall i < n. AAs0 ! i \cdot \text{am } \eta s0 ! i = AAs ! i$ 
  using AAs0f_p by auto

```

then have $AA0_AAs: AAs0 \cdot aml \eta s0 = AAs$
using n **by** $(auto \text{ intro: } nth_equalityI)$

from $AA0_def$ **have** $AA0_in_CA0: \forall i < n. poss (AA0 ! i) \subseteq\# CA0 ! i$
using $AA0f_p$ **by** $auto$

define $Cs0$ **where**
 $Cs0 = map2 (-) CA0 (map poss AAs0)$

have $length\ Cs0 = n$
using $Cs0_def\ n$ **by** $auto$
note $n = n \langle length\ Cs0 = n \rangle$

have $\forall i < n. CA0 ! i = Cs0 ! i + poss (AA0 ! i)$
using $AA0_in_CA0\ Cs0_def\ n$ **by** $auto$
then have $Cs0 \cdot cl \eta s0 = Cs$
using $\langle CA0 \cdot cl \eta s0 = CA \rangle AAs0_AAs\ cas\ n$ **by** $(auto \text{ intro: } nth_equalityI)$

show $?thesis$
using $that$
 $\langle AAs0 \cdot aml \eta s0 = AAs \rangle \langle Cs0 \cdot cl \eta s0 = Cs \rangle \langle \forall i < n. CA0 ! i = Cs0 ! i + poss (AA0 ! i) \rangle$
 $\langle length\ AAs0 = n \rangle \langle length\ Cs0 = n \rangle$
by $blast$

qed

— Obtain FO main premise
have $\exists DA0\ \eta0. DA0 \in M \wedge DA = DA0 \cdot \eta0 \wedge S\ DA0 \cdot \eta0 = S_M\ S\ M\ DA \wedge is_ground_subst\ \eta0$
using $grounding\ S_M_grounding_of_clss\ select$ **by** $(metis\ le_supE\ singletonI\ subsetCE)$
then obtain $DA0\ \eta0$ **where**
 $DA0.\eta0.p: DA0 \in M \wedge DA = DA0 \cdot \eta0 \wedge S\ DA0 \cdot \eta0 = S_M\ S\ M\ DA \wedge is_ground_subst\ \eta0$
by $auto$

— The properties we need of the FO main premise
have $DA0_in_M: DA0 \in M$
using $DA0.\eta0.p$ **by** $auto$
have $DA0_to_DA: DA0 \cdot \eta0 = DA$
using $DA0.\eta0.p$ **by** $auto$
have $SDA0_to_SMDA: S\ DA0 \cdot \eta0 = S_M\ S\ M\ DA$
using $DA0.\eta0.p$ **by** $auto$
have $is_ground_subst\ \eta0$
using $DA0.\eta0.p$ **by** $auto$

— Split main premise DA0 into D0 and As0
obtain $D0\ As0$ **where** $D0As0.p:$
 $As0 \cdot al\ \eta0 = As\ length\ As0 = n\ D0 \cdot \eta0 = D\ DA0 = D0 + (negs\ (mset\ As0))$
 $S_M\ S\ M\ (D + negs\ (mset\ As)) \neq \{\#\} \implies negs\ (mset\ As0) = S\ DA0$
proof –

{

assume $a: S_M\ S\ M\ (D + negs\ (mset\ As)) = \{\#\} \wedge length\ As = (Suc\ 0)$
 $\wedge maximal_wrt\ (As ! 0 \cdot a\ \sigma) ((D + negs\ (mset\ As)) \cdot \sigma)$
then have $as: mset\ As = \{\#As ! 0\#\}$
by $(auto \text{ intro: } nth_equalityI)$
then have $negs\ (mset\ As) = \{\#Neg\ (As ! 0)\#\}$
by $(simp\ add: \langle mset\ As = \{\#As ! 0\#\} \rangle)$
then have $DA = D + \{\#Neg\ (As ! 0)\#\}$
using da **by** $auto$
then obtain L **where** $L \in\# DA0 \wedge L \cdot l\ \eta0 = Neg\ (As ! 0)$
using $DA0_to_DA$ **by** $(metis\ Melem_subst_cls\ mset_subset_eq_add_right\ single_subset_iff)$
then have $Neg\ (atm_of\ L) \in\# DA0 \wedge Neg\ (atm_of\ L) \cdot l\ \eta0 = Neg\ (As ! 0)$
by $(metis\ Neg_atm_of_iff\ literal.sel(2)\ subst.lit.is.pos)$
then have $[atm_of\ L] \cdot al\ \eta0 = As \wedge negs\ (mset\ [atm_of\ L]) \subseteq\# DA0$
using $as\ subst.lit_def$ **by** $auto$
then have $\exists As0. As0 \cdot al\ \eta0 = As \wedge negs\ (mset\ As0) \subseteq\# DA0$
 $\wedge (S_M\ S\ M\ (D + negs\ (mset\ As)) \neq \{\#\} \implies negs\ (mset\ As0) = S\ DA0)$

```

    using a by blast
  }
  moreover
  {
    assume  $S\_M\ S\ M\ (D + \text{negs}\ (mset\ As)) = \text{negs}\ (mset\ As)$ 
    then have  $\text{negs}\ (mset\ As) = S\ DA0 \cdot \eta0$ 
      using da  $\langle S\ DA0 \cdot \eta0 = S\_M\ S\ M\ DA \rangle$  by auto
    then have  $\exists As0. \text{negs}\ (mset\ As0) = S\ DA0 \wedge As0 \cdot al\ \eta0 = As$ 
      using instance_list[of As S DA0  $\eta0$ ] S.S.selects_neg_lits by auto
    then have  $\exists As0. As0 \cdot al\ \eta0 = As \wedge \text{negs}\ (mset\ As0) \subseteq\# DA0$ 
       $\wedge (S\_M\ S\ M\ (D + \text{negs}\ (mset\ As)) \neq \{\#\} \longrightarrow \text{negs}\ (mset\ As0) = S\ DA0)$ 
      using S.S.selects_subseteq by auto
  }
  ultimately have  $\exists As0. As0 \cdot al\ \eta0 = As \wedge (\text{negs}\ (mset\ As0)) \subseteq\# DA0$ 
     $\wedge (S\_M\ S\ M\ (D + \text{negs}\ (mset\ As)) \neq \{\#\} \longrightarrow \text{negs}\ (mset\ As0) = S\ DA0)$ 
    using eligible_unfolding_eligible_simps by auto
  then obtain As0 where
    As0_p:  $As0 \cdot al\ \eta0 = As \wedge \text{negs}\ (mset\ As0) \subseteq\# DA0$ 
     $\wedge (S\_M\ S\ M\ (D + \text{negs}\ (mset\ As)) \neq \{\#\} \longrightarrow \text{negs}\ (mset\ As0) = S\ DA0)$ 
    by blast
  then have length_As0 = n
    using as_len by auto
  note n = n this

  have As0_al_eta0 = As
    using As0_p by auto

  define D0 where
    D0 = DA0 - negs (mset As0)
  then have DA0 = D0 + negs (mset As0)
    using As0_p by auto
  then have D0_eta0 = D
    using DA0_to_DA da As0_p by auto

  have S_M_S_M (D + negs (mset As))  $\neq \{\#\} \implies \text{negs}\ (mset\ As0) = S\ DA0$ 
    using As0_p by blast
  then show ?thesis
    using that  $\langle As0 \cdot al\ \eta0 = As \rangle \langle D0 \cdot \eta0 = D \rangle \langle DA0 = D0 + (\text{negs}\ (mset\ As0)) \rangle \langle \text{length}\ As0 = n \rangle$ 
    by metis
qed

show ?thesis
  using that[OF n(2,1) DA0_in_M DA0_to_DA SDA0_to_SMDA CAs0_in_M CAs0_to_CAs SCAs0_to_SMCAs
     $\langle is\_ground\_subst\ \eta0 \rangle \langle is\_ground\_subst\_list\ \eta s0 \rangle \langle As0 \cdot al\ \eta0 = As \rangle$ 
     $\langle AAs0 \cdot aml\ \eta s0 = AAs \rangle$ 
     $\langle \text{length}\ As0 = n \rangle$ 
     $\langle D0 \cdot \eta0 = D \rangle$ 
     $\langle DA0 = D0 + (\text{negs}\ (mset\ As0)) \rangle$ 
     $\langle S\_M\ S\ M\ (D + \text{negs}\ (mset\ As)) \neq \{\#\} \implies \text{negs}\ (mset\ As0) = S\ DA0 \rangle$ 
     $\langle \text{length}\ Cs0 = n \rangle$ 
     $\langle Cs0 \cdot cl\ \eta s0 = Cs \rangle$ 
     $\langle \forall i < n. CAs0\ !\ i = Cs0\ !\ i + \text{poss}\ (AAs0\ !\ i) \rangle$ 
     $\langle \text{length}\ AAs0 = n \rangle]$ 
  by auto
qed

lemma
  assumes Pos A  $\in\# C$ 
  shows A  $\in\ \text{atms\_of}\ C$ 
  using assms
  by (simp add: atm_iff_pos_or_neg_lit)

lemma 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: *ord_resolve* (*S_M S M*) *CAs DA AAs As σ E* **and**

select: *selection S* **and**

grounding: $\{DA\} \cup \text{set } CAs \subseteq \text{grounding_of_class } M$

obtains $\eta_s \eta \eta_2 CAs_0 DA_0 AAs_0 As_0 E_0 \tau$ **where**

is_ground_subst η

is_ground_subst_list η_s

is_ground_subst η_2

ord_resolve_rename *S CAs_0 DA_0 AAs_0 As_0 τ E_0*

$CAs_0 \cdot cl \eta_s = CAs DA_0 \cdot \eta = DA E_0 \cdot \eta_2 = E$

$\{DA_0\} \cup \text{set } CAs_0 \subseteq M$

using *res_e*

proof (*cases rule: ord_resolve.cases*)

case (*ord_resolve n Cs D*)

note *da = this(1)* **and** *e = this(2)* **and** *cas_len = this(3)* **and** *cs_len = this(4)* **and**

aas_len = this(5) **and** *as_len = this(6)* **and** *nz = this(7)* **and** *cas = this(8)* **and**

aas_not_empty = this(9) **and** *mgu = this(10)* **and** *eligible = this(11)* **and** *str_max = this(12)* **and**

sel_empty = this(13)

have *sel_ren_list_inv*:

$\bigwedge \varrho_s Cs. \text{length } \varrho_s = \text{length } Cs \implies \text{is_renaming_list } \varrho_s \implies \text{map } S (Cs \cdot cl \varrho_s) = \text{map } S Cs \cdot cl \varrho_s$

using *sel_stable unfolding is_renaming_list_def* **by** (*auto intro: nth_equality1*)

note $n = \langle n \neq 0 \rangle \langle \text{length } CAs = n \rangle \langle \text{length } Cs = n \rangle \langle \text{length } AAs = n \rangle \langle \text{length } As = n \rangle$

interpret *S*: *selection S* **by** (*rule select*)

obtain *DA_0 η_0 CAs_0 η_s_0 As_0 AAs_0 D_0 Cs_0* **where** *as_0*:

length CAs_0 = n

length $\eta_s_0 = n$

DA_0 $\in M$

DA_0 $\cdot \eta_0 = DA$

S DA_0 $\cdot \eta_0 = S_M S M DA$

$\forall CA_0 \in \text{set } CAs_0. CA_0 \in M$

CAs_0 $\cdot cl \eta_s_0 = CAs$

map S CAs_0 $\cdot cl \eta_s_0 = \text{map } (S_M S M) CAs$

is_ground_subst η_0

is_ground_subst_list η_s_0

As_0 $\cdot al \eta_0 = As$

AAs_0 $\cdot aml \eta_s_0 = AAs$

length As_0 = n

D_0 $\cdot \eta_0 = D$

DA_0 = D_0 + (negs (mset As_0))

S_M S M (D + negs (mset As)) $\neq \{\#\} \implies \text{negs (mset As_0) = S DA_0}$

length Cs_0 = n

Cs_0 $\cdot cl \eta_s_0 = Cs$

$\forall i < n. CAs_0 ! i = Cs_0 ! i + \text{poss } (AAs_0 ! i)$

length AAs_0 = n

using *ord_resolve_obtain_clauses*[*of S M CAs DA, OF res_e select grounding n(2) $\langle DA = D + \text{negs (mset As) \rangle$*

$\langle \forall i < n. CAs ! i = Cs ! i + \text{poss } (AAs ! i) \rangle \langle \text{length } Cs = n \rangle \langle \text{length } AAs = n \rangle$, *of thesis*] **by** *blast*

note $n = \langle \text{length } CAs_0 = n \rangle \langle \text{length } \eta_s_0 = n \rangle \langle \text{length } As_0 = n \rangle \langle \text{length } AAs_0 = n \rangle \langle \text{length } Cs_0 = n \rangle n$

have *length (renamings_apart (DA_0 # CAs_0)) = Suc n*

using *n renames_apart* **by** *auto*

note $n = \text{this } n$

define ϱ **where**

$\varrho = \text{hd } (\text{renamings_apart } (DA_0 \# CAs_0))$

define ϱ_s **where**

```

     $\rho s = tl (renamings\_apart (DA0 \# CAs0))$ 
define  $DA0'$  where
     $DA0' = DA0 \cdot \rho$ 
define  $D0'$  where
     $D0' = D0 \cdot \rho$ 
define  $As0'$  where
     $As0' = As0 \cdot al \ \rho$ 
define  $CAs0'$  where
     $CAs0' = CAs0 \cdot cl \ \rho s$ 
define  $Cs0'$  where
     $Cs0' = Cs0 \cdot cl \ \rho s$ 
define  $AAs0'$  where
     $AAs0' = AAs0 \cdot aml \ \rho s$ 
define  $\eta0'$  where
     $\eta0' = inv\_renaming \ \rho \odot \eta0$ 
define  $\eta s0'$  where
     $\eta s0' = map \ inv\_renaming \ \rho s \odot s \ \eta s0$ 

have  $renames\_DA0$ :  $is\_renaming \ \rho$ 
using  $renames\_apart$  unfolding  $\rho\_def$ 
by ( $metis \ length\_greater\_0\_conv \ list.exhaust\_sel \ list.set\_intros(1) \ list.simps(3)$ )

have  $renames\_CAs0$ :  $is\_renaming\_list \ \rho s$ 
using  $renames\_apart$  unfolding  $\rho s\_def$ 
by ( $metis \ is\_renaming\_list\_def \ length\_greater\_0\_conv \ list.set\_sel(2) \ list.simps(3)$ )

have  $length \ \rho s = n$ 
unfolding  $\rho s\_def$  using  $n$  by  $auto$ 
note  $n = n$  ( $length \ \rho s = n$ )
have  $length \ As0' = n$ 
unfolding  $As0'\_def$  using  $n$  by  $auto$ 
have  $length \ CAs0' = n$ 
using  $as0(1)$   $n$  unfolding  $CAs0'\_def$  by  $auto$ 
have  $length \ Cs0' = n$ 
unfolding  $Cs0'\_def$  using  $n$  by  $auto$ 
have  $length \ AAs0' = n$ 
unfolding  $AAs0'\_def$  using  $n$  by  $auto$ 
have  $length \ \eta s0' = n$ 
using  $as0(2)$   $n$  unfolding  $\eta s0'\_def$  by  $auto$ 
note  $n =$  ( $length \ CAs0' = n$ ) ( $length \ \eta s0' = n$ ) ( $length \ As0' = n$ ) ( $length \ AAs0' = n$ ) ( $length \ Cs0' = n$ )  $n$ 

have  $DA0'\_DA$ :  $DA0' \cdot \eta0' = DA$ 
using  $as0(4)$  unfolding  $\eta0'\_def \ DA0'\_def$  using  $renames\_DA0$  by  $simp$ 
have  $D0'\_D$ :  $D0' \cdot \eta0' = D$ 
using  $as0(14)$  unfolding  $\eta0'\_def \ D0'\_def$  using  $renames\_DA0$  by  $simp$ 
have  $As0'\_As$ :  $As0' \cdot al \ \eta0' = As$ 
using  $as0(11)$  unfolding  $\eta0'\_def \ As0'\_def$  using  $renames\_DA0$  by  $auto$ 
have  $S \ DA0' \cdot \eta0' = S\_M \ S \ M \ DA$ 
using  $as0(5)$  unfolding  $\eta0'\_def \ DA0'\_def$  using  $renames\_DA0$   $sel\_stable$  by  $auto$ 
have  $CAs0'\_CAs$ :  $CAs0' \cdot cl \ \eta s0' = CAs$ 
using  $as0(7)$  unfolding  $CAs0'\_def \ \eta s0'\_def$  using  $renames\_apart \ renames\_CAs0 \ n$  by  $auto$ 
have  $Cs0'\_Cs$ :  $Cs0' \cdot cl \ \eta s0' = Cs$ 
using  $as0(18)$  unfolding  $Cs0'\_def \ \eta s0'\_def$  using  $renames\_apart \ renames\_CAs0 \ n$  by  $auto$ 
have  $AAs0'\_AAs$ :  $AAs0' \cdot aml \ \eta s0' = AAs$ 
using  $as0(12)$  unfolding  $\eta s0'\_def \ AAs0'\_def$  using  $renames\_CAs0$  using  $n$  by  $auto$ 
have  $map \ S \ CAs0' \cdot cl \ \eta s0' = map \ (S\_M \ S \ M) \ CAs$ 
unfolding  $CAs0'\_def \ \eta s0'\_def$  using  $as0(8)$   $n$   $renames\_CAs0 \ sel\_ren\_list\_inv$  by  $auto$ 

have  $DA0'\_split$ :  $DA0' = D0' + negs \ (mset \ As0')$ 
using  $as0(15)$   $DA0'\_def \ D0'\_def \ As0'\_def$  by  $auto$ 
then have  $D0'\_subset\_DA0'$ :  $D0' \subseteq\# \ DA0'$ 
by  $auto$ 
from  $DA0'\_split$  have  $negs\_As0'\_subset\_DA0'$ :  $negs \ (mset \ As0') \subseteq\# \ DA0'$ 

```

by auto
have $CA0'_{split}: \forall i < n. CA0' ! i = Cs0' ! i + poss (AAs0' ! i)$
using $as0(19) CA0'_{def} Cs0'_{def} AAs0'_{def} n$ **by auto**
then have $\forall i < n. Cs0' ! i \subseteq\# CA0' ! i$
by auto
from $CA0'_{split}$ **have** $poss_AAs0'_{subset_CA0'}: \forall i < n. poss (AAs0' ! i) \subseteq\# CA0' ! i$
by auto
then have $AAs0'_{in_atms_of_CA0'}: \forall i < n. \forall A \in\# AAs0' ! i. A \in atms_of (CA0' ! i)$
by (*auto simp add: atm_iff_pos_or_neg_lit*)

have $as0'$:
 $S_M S M (D + negs (mset As)) \neq \{\#\} \implies negs (mset As0') = S DA0'$
proof –
assume $a: S_M S M (D + negs (mset As)) \neq \{\#\}$
then have $negs (mset As0) \cdot \varrho = S DA0 \cdot \varrho$
using $as0(16)$ **unfolding** ϱ_def **by metis**
then show $negs (mset As0') = S DA0'$
using $As0'_{def} DA0'_{def}$ **using** $sel_stable[of \varrho DA0]$ $renames_DA0$ **by auto**
qed

have $vd: var_disjoint (DA0' \# CA0')$
unfolding $DA0'_{def} CA0'_{def}$ **using** $renames_apart[of DA0 \# CA0]$
unfolding $\varrho_def \varrho s_def$
by (*metis length_greater_0_conv list.exhaust_sel n(6) substitution.subst_cls_lists_Cons substitution.axioms zero_less_Suc*)

– Introduce ground substitution
from $vd DA0'_{DA} CA0'_{CAs}$ **have** $\exists \eta. \forall i < Suc n. \forall S. S \subseteq\# (DA0' \# CA0') ! i \longrightarrow S \cdot (\eta0' \# \eta s0') ! i = S \cdot \eta$
unfolding $var_disjoint_def$ **using** n **by auto**
then obtain η **where** $\eta_p: \forall i < Suc n. \forall S. S \subseteq\# (DA0' \# CA0') ! i \longrightarrow S \cdot (\eta0' \# \eta s0') ! i = S \cdot \eta$
by auto
have $\eta_p_lit: \forall i < Suc n. \forall L. L \in\# (DA0' \# CA0') ! i \longrightarrow L \cdot l (\eta0' \# \eta s0') ! i = L \cdot l \eta$
proof (*rule, rule, rule, rule*)
fix $i :: nat$ **and** $L :: 'a literal$
assume $a:$
 $i < Suc n$
 $L \in\# (DA0' \# CA0') ! i$
then have $\forall S. S \subseteq\# (DA0' \# CA0') ! i \longrightarrow S \cdot (\eta0' \# \eta s0') ! i = S \cdot \eta$
using η_p **by auto**
then have $\{\# L \#\} \cdot (\eta0' \# \eta s0') ! i = \{\# L \#\} \cdot \eta$
using a **by** (*meson single_subset_iff*)
then show $L \cdot l (\eta0' \# \eta s0') ! i = L \cdot l \eta$ **by auto**
qed
have $\eta_p_atm: \forall i < Suc n. \forall A. A \in atms_of ((DA0' \# CA0') ! i) \longrightarrow A \cdot a (\eta0' \# \eta s0') ! i = A \cdot a \eta$
proof (*rule, rule, rule, rule*)
fix $i :: nat$ **and** $A :: 'a$
assume $a:$
 $i < Suc n$
 $A \in atms_of ((DA0' \# CA0') ! i)$
then obtain L **where** $L_p: atm_of L = A \wedge L \in\# (DA0' \# CA0') ! i$
unfolding $atms_of_def$ **by auto**
then have $L \cdot l (\eta0' \# \eta s0') ! i = L \cdot l \eta$
using $\eta_p_lit a$ **by auto**
then show $A \cdot a (\eta0' \# \eta s0') ! i = A \cdot a \eta$
using L_p **unfolding** $subst_lit_def$ **by** (*cases L*) **auto**
qed

have $DA0'_{DA}: DA0' \cdot \eta = DA$
using $DA0'_{DA} \eta_p$ **by auto**
have $D0' \cdot \eta = D$ **using** $\eta_p D0'_{D} D0'_{subset_DA0'}$ **by auto**
have $As0' \cdot al \eta = As$

```

proof (rule nth_equalityI)
  show length (As0' · al η) = length As
    using n by auto
next
  show  $\forall i < \text{length } (As0' \cdot al \eta). (As0' \cdot al \eta) ! i = As ! i$ 
  proof (rule, rule)
    fix i :: nat
    assume a:  $i < \text{length } (As0' \cdot al \eta)$ 
    have A_eq:  $\forall A. A \in \text{atms\_of } DA0' \longrightarrow A \cdot a \eta0' = A \cdot a \eta$ 
      using η-p_atm n by force
    have As0' ! i ∈ atms_of DA0'
      using negs_As0'_subset_DA0' unfolding atms_of_def
      using a n by force
    then have As0' ! i · a η0' = As0' ! i · a η
      using A_eq by simp
    then show (As0' · al η) ! i = As ! i
      using As0'_As ⟨length As0' = n⟩ a by auto
  qed
qed

have S DA0' · η = S_M S M DA
  using ⟨S DA0' · η0' = S_M S M DA⟩ η-p S.S_selects_subseteq by auto

from η-p have η-p_CAs0':  $\forall i < n. (CAs0' ! i) \cdot (\eta s0' ! i) = (CAs0' ! i) \cdot \eta$ 
  using n by auto
then have CAs0' · cl ηs0' = CAs0' · cl η
  using n by (auto intro: nth_equalityI)
then have CAs0' · η-fo-CAs: CAs0' · cl η = CAs
  using CAs0'_CAs η-p n by auto

from η-p have  $\forall i < n. S (CAs0' ! i) \cdot \eta s0' ! i = S (CAs0' ! i) \cdot \eta$ 
  using S.S_selects_subseteq n by auto
then have map S CAs0' · cl ηs0' = map S CAs0' · cl η
  using n by (auto intro: nth_equalityI)
then have SCAs0' · η-fo-SMCAs: map S CAs0' · cl η = map (S_M S M) CAs
  using ⟨map S CAs0' · cl ηs0' = map (S_M S M) CAs⟩ by auto

have Cs0' · cl η = Cs
proof (rule nth_equalityI)
  show length (Cs0' · cl η) = length Cs
    using n by auto
next
  show  $\forall i < \text{length } (Cs0' \cdot cl \eta). (Cs0' \cdot cl \eta) ! i = Cs ! i$ 
  proof (rule, rule)
    fix i
    assume i < length (Cs0' · cl η)
    then have a:  $i < n$ 
      using n by force
    have (Cs0' · cl ηs0') ! i = Cs ! i
      using Cs0'_Cs a n by force
    moreover
    have η-p_CAs0':  $\forall S. S \subseteq \# CAs0' ! i \longrightarrow S \cdot \eta s0' ! i = S \cdot \eta$ 
      using η-p a by force
    have Cs0' ! i · ηs0' ! i = (Cs0' · cl η) ! i
      using η-p_CAs0' ⟨ $\forall i < n. Cs0' ! i \subseteq \# CAs0' ! i$ ⟩ a n by force
    then have (Cs0' · cl ηs0') ! i = (Cs0' · cl η) ! i
      using a n by force
    ultimately show (Cs0' · cl η) ! i = Cs ! i
      by auto
  qed
qed

have AAs0'_AAs: AAs0' · aml η = AAs

```

```

proof (rule nth_equalityI)
  show length (AAs0' · aml η) = length AAs
    using n by auto
next
  show  $\forall i < \text{length } (AAs0' \cdot \text{aml } \eta). (AAs0' \cdot \text{aml } \eta) ! i = AAs ! i$ 
  proof (rule, rule)
    fix i :: nat
    assume a:  $i < \text{length } (AAs0' \cdot \text{aml } \eta)$ 
    then have  $i < n$ 
      using n by force
    then have  $\forall A. A \in \text{atms\_of } ((DA0' \# CA0') ! \text{Suc } i) \longrightarrow A \cdot a (\eta0' \# \eta s0') ! \text{Suc } i = A \cdot a \eta$ 
      using  $\eta\_p\_atm$  n by force
    then have  $A\_eq: \forall A. A \in \text{atms\_of } (CA0' ! i) \longrightarrow A \cdot a \eta s0' ! i = A \cdot a \eta$ 
      by auto
    have  $AAs\_CA0': \forall A \in \# AAs0' ! i. A \in \text{atms\_of } (CA0' ! i)$ 
      using  $AAs0'\_in\_atms\_of\_CA0'$  unfolding  $\text{atms\_of\_def}$ 
      using a n by force
    then have  $AAs0' ! i \cdot am \eta s0' ! i = AAs0' ! i \cdot am \eta$ 
      unfolding  $\text{subst\_atm\_mset\_def}$  using  $A\_eq$  unfolding  $\text{subst\_atm\_mset\_def}$  by auto
    then show  $(AAs0' \cdot \text{aml } \eta) ! i = AAs ! i$ 
      using  $AAs0'\_AAs$   $\langle \text{length } AAs0' = n \rangle \langle \text{length } \eta s0' = n \rangle$  a by auto
  qed
qed

```

— Obtain MGU and substitution

```

obtain  $\tau \varphi$  where  $\tau \varphi$ :
  Some  $\tau = \text{mgu } (\text{set\_mset } ' \text{set } (\text{map2 } \text{add\_mset } As0' AAs0'))$ 
   $\tau \odot \varphi = \eta \odot \sigma$ 
proof –
  have  $uu: \text{is\_unifiers } \sigma (\text{set\_mset } ' \text{set } (\text{map2 } \text{add\_mset } (As0' \cdot al \eta) (AAs0' \cdot aml \eta)))$ 
    using  $\text{mgu } \text{mgu\_sound } \text{is\_mgu\_def}$  unfolding  $\langle AAs0' \cdot aml \eta = AAs \rangle$  using  $\langle As0' \cdot al \eta = As \rangle$  by auto
  have  $\eta \sigma uni: \text{is\_unifiers } (\eta \odot \sigma) (\text{set\_mset } ' \text{set } (\text{map2 } \text{add\_mset } As0' AAs0'))$ 
  proof –
    have  $\text{set\_mset } ' \text{set } (\text{map2 } \text{add\_mset } As0' AAs0' \cdot aml \eta) =$ 
       $\text{set\_mset } ' \text{set } (\text{map2 } \text{add\_mset } As0' AAs0') \cdot \text{ass } \eta$ 
      unfolding  $\text{subst\_atmss\_def } \text{subst\_atm\_mset\_list\_def}$  using  $\text{subst\_atm\_mset\_def } \text{subst\_atms\_def}$ 
      by ( $\text{simp } \text{add}: \text{image\_image } \text{subst\_atm\_mset\_def } \text{subst\_atms\_def}$ )
    then have  $\text{is\_unifiers } \sigma (\text{set\_mset } ' \text{set } (\text{map2 } \text{add\_mset } As0' AAs0') \cdot \text{ass } \eta)$ 
      using uu by ( $\text{auto } \text{simp}: n \text{ map2\_add\_mset\_map}$ )
    then show  $?thesis$ 
      using  $\text{is\_unifiers\_comp}$  by auto
  qed
  then obtain  $\tau$  where
     $\tau\_p: \text{Some } \tau = \text{mgu } (\text{set\_mset } ' \text{set } (\text{map2 } \text{add\_mset } As0' AAs0'))$ 
    using  $\text{mgu\_complete}$ 
    by ( $\text{metis } (\text{mono\_tags}, \text{hide\_lams}) \text{List.finite\_set } \text{finite\_imageI } \text{finite\_set\_mset } \text{image\_iff}$ )
  moreover then obtain  $\varphi$  where  $\varphi\_p: \tau \odot \varphi = \eta \odot \sigma$ 
    by ( $\text{metis } (\text{mono\_tags}, \text{hide\_lams}) \text{finite\_set } \eta \sigma uni \text{finite\_imageI } \text{finite\_set\_mset } \text{image\_iff}$ 
       $\text{mgu\_sound } \text{set\_mset\_mset } \text{substitution\_ops.is\_mgu\_def}$ )
  ultimately show  $thesis$ 
    using that by auto
qed

```

— Lifting eligibility

```

have  $\text{eligible0': eligible } S \tau As0' (D0' + \text{negs } (\text{mset } As0'))$ 
proof –
  have  $S\_M S M (D + \text{negs } (\text{mset } As)) = \text{negs } (\text{mset } As) \vee S\_M S M (D + \text{negs } (\text{mset } As)) = \{\#\} \wedge$ 
     $\text{length } As = 1 \wedge \text{maximal\_wrt } (As ! 0 \cdot a \sigma) ((D + \text{negs } (\text{mset } As)) \cdot \sigma)$ 
    using  $\text{eligible}$  unfolding  $\text{eligible.simps}$  by auto
  then show  $?thesis$ 
proof
  assume  $S\_M S M (D + \text{negs } (\text{mset } As)) = \text{negs } (\text{mset } As)$ 
  then have  $S\_M S M (D + \text{negs } (\text{mset } As)) \neq \{\#\}$ 

```


using n **by force**
then have $S (D0' + \text{negs } (\text{mset } As0')) = \text{negs } (\text{mset } As0')$
using $as0' \text{ DA0'_{split}}$ **by auto**
then show $?thesis$
unfolding $\text{eligible.simps[simplified]}$ **by auto**
next
assume $asm: S_M S M (D + \text{negs } (\text{mset } As)) = \{\#\} \wedge \text{length } As = 1 \wedge$
 $\text{maximal_wrt } (As ! 0 \cdot a \sigma) ((D + \text{negs } (\text{mset } As)) \cdot \sigma)$
then have $S (D0' + \text{negs } (\text{mset } As0')) = \{\#\}$
using $\langle D0' \cdot \eta = D \rangle[\text{symmetric}] \langle As0' \cdot al \eta = As \rangle[\text{symmetric}] \langle S (DA0') \cdot \eta = S_M S M (DA) \rangle$
 $\text{da } DA0'_{split} \text{ subst_cls_empty_iff}$ **by metis**
moreover from asm **have** $l: \text{length } As0' = 1$
using $\langle As0' \cdot al \eta = As \rangle$ **by auto**
moreover from asm **have** $\text{maximal_wrt } (As0' ! 0 \cdot a (\tau \odot \varphi)) ((D0' + \text{negs } (\text{mset } As0')) \cdot (\tau \odot \varphi))$
using $\langle As0' \cdot al \eta = As \rangle \langle D0' \cdot \eta = D \rangle$ **using** $l \tau \varphi$ **by auto**
then have $\text{maximal_wrt } (As0' ! 0 \cdot a \tau \cdot a \varphi) ((D0' + \text{negs } (\text{mset } As0')) \cdot \tau \cdot \varphi)$
by auto
then have $\text{maximal_wrt } (As0' ! 0 \cdot a \tau) ((D0' + \text{negs } (\text{mset } As0')) \cdot \tau)$
using maximal_wrt_subst **by blast**
ultimately show $?thesis$
unfolding $\text{eligible.simps[simplified]}$ **by auto**
qed
qed

— Lifting maximality

have $\text{maximality}: \forall i < n. \text{strictly_maximal_wrt } (As0' ! i \cdot a \tau) (Cs0' ! i \cdot \tau)$

proof –

from str_max **have** $\forall i < n. \text{strictly_maximal_wrt } ((As0' \cdot al \eta) ! i \cdot a \sigma) ((Cs0' \cdot cl \eta) ! i \cdot \sigma)$
using $\langle As0' \cdot al \eta = As \rangle \langle Cs0' \cdot cl \eta = Cs \rangle$ **by simp**
then have $\forall i < n. \text{strictly_maximal_wrt } (As0' ! i \cdot a (\tau \odot \varphi)) (Cs0' ! i \cdot (\tau \odot \varphi))$
using $n \tau \varphi$ **by simp**
then have $\forall i < n. \text{strictly_maximal_wrt } (As0' ! i \cdot a \tau \cdot a \varphi) (Cs0' ! i \cdot \tau \cdot \varphi)$
by auto
then show $\forall i < n. \text{strictly_maximal_wrt } (As0' ! i \cdot a \tau) (Cs0' ! i \cdot \tau)$
using $\text{strictly_maximal_wrt_subst } \tau \varphi$ **by blast**
qed

— Lifting nothing being selected

have $\text{nothing_selected}: \forall i < n. S (CAs0' ! i) = \{\#\}$

proof –

have $\forall i < n. (\text{map } S CAs0' \cdot cl \eta) ! i = \text{map } (S_M S M) CAs ! i$
by $(\text{simp add}: \langle \text{map } S CAs0' \cdot cl \eta = \text{map } (S_M S M) CAs \rangle)$
then have $\forall i < n. S (CAs0' ! i) \cdot \eta = S_M S M (CAs ! i)$
using n **by auto**
then have $\forall i < n. S (CAs0' ! i) \cdot \eta = \{\#\}$
using $\text{sel_empt } \langle \forall i < n. S (CAs0' ! i) \cdot \eta = S_M S M (CAs ! i) \rangle$ **by auto**
then show $\forall i < n. S (CAs0' ! i) = \{\#\}$
using $\text{subst_cls_empty_iff}$ **by blast**
qed

— Lifting AAs0's non-emptiness

have $\forall i < n. AAs0' ! i \neq \{\#\}$
using $n \text{ aas_not_empt } \langle AAs0' \cdot aml \eta = AAs \rangle$ **by auto**

— Resolve the lifted clauses

define $E0'$ **where**

$E0' = ((\bigcup \# \text{mset } Cs0') + D0') \cdot \tau$

have $\text{res_e0'}: \text{ord_resolve } S CAs0' DA0' AAs0' As0' \tau E0'$
using $\text{ord_resolve.intros[of } CAs0' n Cs0' AAs0' As0' \tau S D0',$
 $OF \text{ } \dots \dots \langle \forall i < n. AAs0' ! i \neq \{\#\} \rangle \tau \varphi(1) \text{ eligible0'}$
 $\langle \forall i < n. \text{strictly_maximal_wrt } (As0' ! i \cdot a \tau) (Cs0' ! i \cdot \tau) \rangle \langle \forall i < n. S (CAs0' ! i) = \{\#\} \rangle]$

unfolding $E0'_{def}$ **using** $DA0'_{split}$ $n \langle \forall i < n. CAs0' ! i = Cs0' ! i + poss (AAs0' ! i) \rangle$ **by** *blast*

— Prove resolvent instantiates to ground resolvent

have $e0'\varphi e: E0' \cdot \varphi = E$

proof —

have $E0' \cdot \varphi = ((\bigcup \# mset Cs0') + D0') \cdot (\tau \odot \varphi)$

unfolding $E0'_{def}$ **by** *auto*

also have $\dots = (\bigcup \# mset Cs0' + D0') \cdot (\eta \odot \sigma)$

using $\tau\varphi$ **by** *auto*

also have $\dots = (\bigcup \# mset Cs + D) \cdot \sigma$

using $\langle Cs0' \cdot cl \eta = Cs \rangle \langle D0' \cdot \eta = D \rangle$ **by** *auto*

also have $\dots = E$

using e **by** *auto*

finally show $e0'\varphi e: E0' \cdot \varphi = E$

qed

— Replace φ with a true ground substitution

obtain $\eta2$ **where**

$ground_eta2: is_ground_subst \eta2 E0' \cdot \eta2 = E$

proof —

have $is_ground_cls_list CAs is_ground_cls DA$

using $grounding$ $grounding_ground$ **unfolding** $is_ground_cls_list_def$ **by** *auto*

then have $is_ground_cls E$

using res_e $ground_resolvent_subset$ **by** ($force$ $intro: is_ground_cls_mono$)

then show *thesis*

using $that$ $e0'\varphi e$ $make_ground_subst$ **by** *auto*

qed

— Wrap up the proof

have $ord_resolve S (CAs0 \cdot cl \varrho s) (DA0 \cdot \varrho) (AAs0 \cdot aml \varrho s) (As0 \cdot al \varrho) \tau E0'$

using res_e0' $As0'_{def}$ ϱ_def $AAs0'_{def}$ ϱs_def $DA0'_{def}$ ϱ_def $CAs0'_{def}$ ϱs_def **by** *simp*

moreover have $\forall i < n. poss (AAs0 ! i) \subseteq \# CAs0 ! i$

using $as0(19)$ **by** *auto*

moreover have $negs (mset As0) \subseteq \# DA0$

using $local.as0(15)$ **by** *auto*

ultimately have $ord_resolve_rename S CAs0 DA0 AAs0 As0 \tau E0'$

using $ord_resolve_rename[of CAs0 n AAs0 As0 DA0 \varrho \varrho s S \tau E0'] \varrho_def \varrho s_def n$ **by** *auto*

then show *thesis*

using $that[of \eta0 \eta s0 \eta2 CAs0 DA0] \langle is_ground_subst \eta0 \rangle \langle is_ground_subst_list \eta s0 \rangle$

$\langle is_ground_subst \eta2 \rangle \langle CAs0 \cdot cl \eta s0 = CAs \rangle \langle DA0 \cdot \eta0 = DA \rangle \langle E0' \cdot \eta2 = E \rangle \langle DA0 \in M \rangle$

$\langle \forall CA \in set CAs0. CA \in M \rangle$ **by** *blast*

qed

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 $is_least :: (nat \Rightarrow bool) \Rightarrow nat \Rightarrow bool$ **where**

$is_least P n \longleftrightarrow P n \wedge (\forall n' < n. \neg P n')$

lemma $least_exists: P n \Longrightarrow \exists n. is_least P n$

using $exists_least_iff$ **unfolding** is_least_def **by** *auto*

The following corresponds to page 42 and 43 of Section 4.3, from the explanation of RP to Lemma 4.10.

type-synonym $'a \text{ state} = 'a \text{ clause set} \times 'a \text{ clause set} \times 'a \text{ clause set}$

locale $FO_resolution_prover =$

$FO_resolution \text{ subst_atm } id_subst \text{ comp_subst } renamings_apart \text{ atm_of_atms } mgu \text{ less_atm} +$
 $selection \ S$

for

$S :: ('a :: wellorder) \text{ clause} \Rightarrow 'a \text{ clause} \text{ and}$
 $subst_atm :: 'a \Rightarrow 's \Rightarrow 'a \text{ and}$
 $id_subst :: 's \text{ and}$
 $comp_subst :: 's \Rightarrow 's \Rightarrow 's \text{ and}$
 $renamings_apart :: 'a \text{ clause list} \Rightarrow 's \text{ list} \text{ and}$
 $atm_of_atms :: 'a \text{ list} \Rightarrow 'a \text{ and}$
 $mgu :: 'a \text{ set set} \Rightarrow 's \text{ option} \text{ and}$
 $less_atm :: 'a \Rightarrow 'a \Rightarrow bool +$

assumes

$sel_stable: \bigwedge \varrho \ C. \ is_renaming \ \varrho \Longrightarrow S \ (C \cdot \varrho) = S \ C \cdot \varrho \text{ and}$
 $less_atm_ground: \ is_ground_atm \ A \Longrightarrow is_ground_atm \ B \Longrightarrow less_atm \ A \ B \Longrightarrow A < B$

begin

fun $N_of_state :: 'a \text{ state} \Rightarrow 'a \text{ clause set} \text{ where}$

$N_of_state \ (N, P, Q) = N$

fun $P_of_state :: 'a \text{ state} \Rightarrow 'a \text{ clause set} \text{ where}$

$P_of_state \ (N, P, Q) = P$

O denotes relation composition in Isabelle, so the formalization uses Q instead.

fun $Q_of_state :: 'a \text{ state} \Rightarrow 'a \text{ clause set} \text{ where}$

$Q_of_state \ (N, P, Q) = Q$

definition $clss_of_state :: 'a \text{ state} \Rightarrow 'a \text{ clause set} \text{ where}$

$clss_of_state \ St = N_of_state \ St \cup P_of_state \ St \cup Q_of_state \ St$

abbreviation $grounding_of_state :: 'a \text{ state} \Rightarrow 'a \text{ clause set} \text{ where}$

$grounding_of_state \ St \equiv grounding_of_clss \ (clss_of_state \ St)$

interpretation $ord_FO_resolution: \text{inference_system } ord_FO.\Gamma \ S .$

The following inductive predicate formalizes the resolution prover in Figure 5.

inductive $RP :: 'a \text{ state} \Rightarrow 'a \text{ state} \Rightarrow bool \text{ (infix } \rightsquigarrow 50) \text{ where}$

$tautology_deletion: \text{Neg } A \in \# \ C \Longrightarrow \text{Pos } A \in \# \ C \Longrightarrow (N \cup \{C\}, P, Q) \rightsquigarrow (N, P, Q)$
 $| \text{forward_subsumption}: \ D \in P \cup Q \Longrightarrow \text{subsumes } D \ C \Longrightarrow (N \cup \{C\}, P, Q) \rightsquigarrow (N, P, Q)$
 $| \text{backward_subsumption}_P: \ D \in N \Longrightarrow \text{strictly_subsumes } D \ C \Longrightarrow (N, P \cup \{C\}, Q) \rightsquigarrow (N, P, Q)$
 $| \text{backward_subsumption}_Q: \ D \in N \Longrightarrow \text{strictly_subsumes } D \ C \Longrightarrow (N, P, Q \cup \{C\}) \rightsquigarrow (N, P, Q)$
 $| \text{forward_reduction}: \ D + \{\#L'\#\} \in P \cup Q \Longrightarrow - \ L = L' \cdot l \ \sigma \Longrightarrow D \cdot \sigma \subseteq \# \ C \Longrightarrow$
 $(N \cup \{C + \{\#L\#\}\}, P, Q) \rightsquigarrow (N \cup \{C\}, P, Q)$
 $| \text{backward_reduction}_P: \ D + \{\#L'\#\} \in N \Longrightarrow - \ L = L' \cdot l \ \sigma \Longrightarrow D \cdot \sigma \subseteq \# \ C \Longrightarrow$
 $(N, P \cup \{C + \{\#L\#\}\}, Q) \rightsquigarrow (N, P \cup \{C\}, Q)$
 $| \text{backward_reduction}_Q: \ D + \{\#L'\#\} \in N \Longrightarrow - \ L = L' \cdot l \ \sigma \Longrightarrow D \cdot \sigma \subseteq \# \ C \Longrightarrow$
 $(N, P, Q \cup \{C + \{\#L\#\}\}) \rightsquigarrow (N, P \cup \{C\}, Q)$
 $| \text{clause_processing}: \ (N \cup \{C\}, P, Q) \rightsquigarrow (N, P \cup \{C\}, Q)$
 $| \text{inference_computation}: \ N = \text{concls_of } (ord_FO_resolution.\text{inferences_between } Q \ C) \Longrightarrow$
 $(\{\}, P \cup \{C\}, Q) \rightsquigarrow (N, P, Q \cup \{C\})$

lemma $final_RP: \neg (\{\}, \{\}, Q) \rightsquigarrow St$

by $(\text{auto elim: } RP.\text{cases})$

definition $Sup_state :: 'a \text{ state llist} \Rightarrow 'a \text{ state} \text{ where}$

$Sup_state \ Sts =$
 $(Sup_llist \ (\text{lmap } N_of_state \ Sts), \text{Sup_llist } (\text{lmap } P_of_state \ Sts),$
 $\text{Sup_llist } (\text{lmap } Q_of_state \ Sts))$

definition $Liminf_state :: 'a \text{ state llist} \Rightarrow 'a \text{ state} \text{ where}$

```

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)
using Sts by (simp_all add: Liminf_state_def lfinite_Liminf_llist llast_lmap)

```

lemma *Liminf_state_fin*: $Liminf_state\ Sts' = Liminf_state\ Sts$

```

using N_of_Liminf_state_fin P_of_Liminf_state_fin Q_of_Liminf_state_fin
by (simp add: Liminf_state_def)

```

end

context

```

  fixes Sts Sts' :: 'a state llist
  assumes Sts:  $\neg$  lfinite Sts emb Sts Sts'

```

begin

lemma

```

  N_of_Liminf_state_inf: N_of_state (Liminf_state Sts')  $\subseteq$  N_of_state (Liminf_state Sts) and
  P_of_Liminf_state_inf: P_of_state (Liminf_state Sts')  $\subseteq$  P_of_state (Liminf_state Sts) and
  Q_of_Liminf_state_inf: Q_of_state (Liminf_state Sts')  $\subseteq$  Q_of_state (Liminf_state Sts)
using Sts by (simp_all add: Liminf_state_def emb_Liminf_llist.infinite emb_lmap)

```

lemma *cls_of_Liminf_state_inf*:

```

  cls_of_state (Liminf_state Sts')  $\subseteq$  cls_of_state (Liminf_state Sts)
unfolding cls_of_state_def
using N_of_Liminf_state_inf P_of_Liminf_state_inf Q_of_Liminf_state_inf by blast

```

end

definition *fair_state_seq* :: 'a state llist \Rightarrow bool **where**

```

  fair_state_seq Sts  $\longleftrightarrow$  N_of_state (Liminf_state Sts) = {}  $\wedge$  P_of_state (Liminf_state Sts) = {}

```

The following formalizes Lemma 4.10.

context

```

  fixes
    Sts :: 'a state llist
  assumes
    deriv: chain ( $\rightsquigarrow$ ) Sts and
    empty_Q0: Q_of_state (lhd Sts) = {}

```

begin

lemmas *lhd_lmap_Sts* = $lmap.sel(1)[OF\ chain_not_lnull[OF\ deriv]]$

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

```

  S_Q = S.M S (Q_of_state (Liminf_state Sts))

```

interpretation *sq*: selection *S_Q*

```

unfolding S_Q_def using S_M_selects_subseteq S_M_selects_neg_lits selection_axioms
by unfold_locales auto

```

interpretation *gr*: ground_resolution_with_selection *S_Q*

```

by unfold_locales

```

interpretation *sr*: *standard_redundancy_criterion_reductive gr.ord.Γ*
by *unfold_locales*

interpretation *sr*: *standard_redundancy_criterion_countersx_reducing gr.ord.Γ*
ground_resolution_with_selection.INTERP S_Q
by *unfold_locales*

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*
unfolding *ground_sound.Γ_def* **using** *gr.ord.Γ_def gr.ord_resolve_sound* **by** *fastforce*

lemma *sound_ground_sound.Γ*: *sound_inference_system ground_sound.Γ*
unfolding *sound_inference_system_def ground_sound.Γ_def* **by** *auto*

lemma *sat_preserving_ground_sound.Γ*: *sat_preserving_inference_system ground_sound.Γ*
using *sound_ground_sound.Γ sat_preserving_inference_system.intro*
sound_inference_system.Γ_sat_preserving **by** *blast*

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
unfolding *sat_preserving_redundancy_criterion_def sr_ext.Ri_def*
using *sat_preserving_ground_sound.Γ redundancy_criterion_standard_extension gd_ord.Γ_ngd_ord.Γ*
sr_redundancy_criterion_axioms **by** *auto*

lemma *strict_subset_subsumption_redundant_clause*:
assumes
sub: $D \cdot \sigma \subset\# C$ **and**
ground_σ: *is_ground_subst σ*
shows $C \in sr.Rf$ (*grounding_of_cls D*)
proof -
from *sub* **have** $\forall I. I \models D \cdot \sigma \longrightarrow I \models C$
unfolding *true_cls_def* **by** *blast*
moreover **have** $C > D \cdot \sigma$
using *sub* **by** (*simp add: subset_imp_less_mset*)
moreover **have** $D \cdot \sigma \in grounding_of_cls D$
using *ground_σ* **by** (*metis (mono_tags, lifting) mem_Collect_eq substitution_ops_grounding_of_cls_def*)
ultimately **have** $set_mset \{\#D \cdot \sigma\} \subseteq grounding_of_cls D$
 $(\forall I. I \models_m \{\#D \cdot \sigma\} \longrightarrow I \models C)$
 $(\forall D'. D' \in\# \{\#D \cdot \sigma\} \longrightarrow D' < C)$
by *auto*
then **show** *?thesis*
using *sr.Rf_def* **by** *blast*
qed

lemma *strict_subset_subsumption_redundant_cls*:
assumes
 $D \cdot \sigma \subset\# C$ **and**
is_ground_subst σ **and**
 $D \in CC$
shows $C \in sr.Rf$ (*grounding_of_cls CC*)
using *assms*
proof -
have $C \in sr.Rf$ (*grounding_of_cls D*)
using *strict_subset_subsumption_redundant_clause assms* **by** *auto*
then **show** *?thesis*

using *assms* **unfolding** *clss_of_state_def* *grounding_of_clss_def*
by (*metis* (*no_types*) *sr.Rf_mono* *sup_ge1* *SUP_absorb* *contra_subsetD*)
qed

lemma *strict_subset_subsumption_grounding_redundant_clss*:

assumes

$D\sigma_subset.C: D \cdot \sigma \subset\# C$ **and**

$D_in_St: D \in CC$

shows $grounding_of_cls\ C \subseteq sr.Rf\ (grounding_of_clss\ CC)$

proof

fix $C\mu$

assume $C\mu \in grounding_of_cls\ C$

then obtain μ **where**

$\mu.p: C\mu = C \cdot \mu \wedge is_ground_subst\ \mu$

unfolding *grounding_of_cls_def* **by** *auto*

have $D\sigma\mu C\mu: D \cdot \sigma \cdot \mu \subset\# C \cdot \mu$

using $D\sigma_subset.C$ *subst_subset_mono* **by** *auto*

then show $C\mu \in sr.Rf\ (grounding_of_clss\ CC)$

using $\mu.p$ *strict_subset_subsumption_redundant_clss*[*of* $D\ \sigma \odot \mu\ C \cdot \mu$] D_in_St

unfolding *clss_of_state_def* **by** *auto*

qed

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

fix $C\sigma'$

assume $C\sigma' \in grounding_of_cls\ C$

then obtain σ' **where**

$C\sigma': C \cdot \sigma' = C\sigma' is_ground_subst\ \sigma'$

unfolding *grounding_of_cls_def* **by** *auto*

then have $C \cdot \sigma' = D \cdot \sigma \cdot \sigma' \wedge is_ground_subst\ (\sigma \odot \sigma')$

using *assms* **by** *auto*

then show $C\sigma' \in grounding_of_cls\ D$

unfolding *grounding_of_cls_def* **using** $C\sigma'(1)$ **by** *force*

qed

lemma *derive_if_remove_subsumed*:

assumes

$D \in clss_of_state\ St$ **and**

subsumes $D\ C$

shows $sr_ext.derive\ (grounding_of_state\ St \cup grounding_of_cls\ C)\ (grounding_of_state\ St)$

proof –

from *assms* **obtain** σ **where**

$D \cdot \sigma = C \vee D \cdot \sigma \subset\# C$

by (*auto simp: subsumes_def subset_mset_def*)

then have $D \cdot \sigma = C \vee D \cdot \sigma \subset\# C$

by (*simp add: subset_mset_def*)

then show *?thesis*

proof

assume $D \cdot \sigma = C$

then have $grounding_of_cls\ C \subseteq grounding_of_cls\ D$

using *subst_cls_eq_grounding_of_cls_subset_eq* **by** *simp*

then have $(grounding_of_state\ St \cup grounding_of_cls\ C) = grounding_of_state\ St$

using *assms* **unfolding** *clss_of_state_def* *grounding_of_clss_def* **by** *auto*

then show *?thesis*

by (*auto intro: sr_ext.derive.intros*)

next

assume $a: D \cdot \sigma \subset\# C$

then have $grounding_of_cls\ C \subseteq sr.Rf\ (grounding_of_state\ St)$

using *strict_subset_subsumption_grounding_redundant_clss* *assms* **by** *auto*

then show *?thesis*

unfolding *clss_of_state_def* *grounding_of_clss_def* **by** (*force intro: sr_ext.derive.intros*)

qed
qed

lemma *reduction_in_concls_of*:

assumes

$C\mu \in \text{grounding_of_cls } C$ **and**
 $D + \{\#L'\#\} \in CC$ **and**
 $- L = L' \cdot l \sigma$ **and**
 $D \cdot \sigma \subseteq\# C$

shows $C\mu \in \text{concls_of } (sr_ext.inferences_from (\text{grounding_of_clss } (CC \cup \{C + \{\#L\#\})))$

proof –

from *assms*

obtain μ **where**

$\mu_p: C\mu = C \cdot \mu \wedge \text{is_ground_subst } \mu$
unfolding *grounding_of_cls_def* **by** *auto*

define γ **where**

$\gamma = \text{Infer } \{\#(C + \{\#L\#\}) \cdot \mu\# \} ((D + \{\#L'\#\}) \cdot \sigma \cdot \mu) (C \cdot \mu)$

have $(D + \{\#L'\#\}) \cdot \sigma \cdot \mu \in \text{grounding_of_clss } (CC \cup \{C + \{\#L\#\})$

unfolding *grounding_of_clss_def* *grounding_of_cls_def*

by (*rule UN_I*[of $D + \{\#L'\#\}$], *use* *assms*(2) *clss_of_state_def* **in** *simp*,
metis (*mono_tags*, *lifting*) μ_p *is_ground_comp_subst* *mem_Collect_eq* *subst_cls_comp_subst*)

moreover **have** $(C + \{\#L\#\}) \cdot \mu \in \text{grounding_of_clss } (CC \cup \{C + \{\#L\#\})$

using μ_p **unfolding** *grounding_of_clss_def* *grounding_of_cls_def* **by** *auto*

moreover **have** $\forall I. I \models D \cdot \sigma \cdot \mu + \{\#-(L \cdot l \mu)\#\} \longrightarrow I \models C \cdot \mu + \{\#L \cdot l \mu\#\} \longrightarrow I \models D \cdot \sigma \cdot \mu + C \cdot \mu$

by *auto*

then **have** $\forall I. I \models (D + \{\#L'\#\}) \cdot \sigma \cdot \mu \longrightarrow I \models (C + \{\#L\#\}) \cdot \mu \longrightarrow I \models D \cdot \sigma \cdot \mu + C \cdot \mu$

using *assms*

by (*metis* *add_mset_add_single* *subst_cls_add_mset* *subst_cls_union* *subst_minus*)

then **have** $\forall I. I \models (D + \{\#L'\#\}) \cdot \sigma \cdot \mu \longrightarrow I \models (C + \{\#L\#\}) \cdot \mu \longrightarrow I \models C \cdot \mu$

using *assms* **by** (*metis* (*no_types*, *lifting*) *subset_mset.le_iff_add* *subst_cls_union* *true_cls_union*)

then **have** $\forall I. I \models m \{\#(D + \{\#L'\#\}) \cdot \sigma \cdot \mu\# \} \longrightarrow I \models (C + \{\#L\#\}) \cdot \mu \longrightarrow I \models C \cdot \mu$

by (*meson* *true_cls_mset_singleton*)

ultimately **have** $\gamma \in sr_ext.inferences_from (\text{grounding_of_clss } (CC \cup \{C + \{\#L\#\}))$

unfolding *sr_ext.inferences_from_def* **unfolding** *ground_sound_\Gamma_def* *infer_from_def* γ_def **by** *auto*

then **have** $C \cdot \mu \in \text{concls_of } (sr_ext.inferences_from (\text{grounding_of_clss } (CC \cup \{C + \{\#L\#\}))$

using *image_iff* **unfolding** γ_def **by** *fastforce*

then **show** $C\mu \in \text{concls_of } (sr_ext.inferences_from (\text{grounding_of_clss } (CC \cup \{C + \{\#L\#\}))$

using μ_p **by** *auto*

qed

lemma *reduction_derivable*:

assumes

$D + \{\#L'\#\} \in CC$ **and**
 $- L = L' \cdot l \sigma$ **and**
 $D \cdot \sigma \subseteq\# C$

shows $sr_ext.derive (\text{grounding_of_clss } (CC \cup \{C + \{\#L\#\})) (\text{grounding_of_clss } (CC \cup \{C\}))$

proof –

from *assms* **have** $\text{grounding_of_clss } (CC \cup \{C\}) - \text{grounding_of_clss } (CC \cup \{C + \{\#L\#\})$

$\subseteq \text{concls_of } (sr_ext.inferences_from (\text{grounding_of_clss } (CC \cup \{C + \{\#L\#\}))$

using *reduction_in_concls_of* **unfolding** *grounding_of_clss_def* *clss_of_state_def*

by *auto*

moreover

have $\text{grounding_of_cls } (C + \{\#L\#\}) \subseteq sr.Rf (\text{grounding_of_clss } (CC \cup \{C\}))$

using *strict_subset_subsumption_grounding_redundant_clss*[of C *id_subst*]

by *auto*

then **have** $\text{grounding_of_clss } (CC \cup \{C + \{\#L\#\}) - \text{grounding_of_clss } (CC \cup \{C\})$

$\subseteq sr.Rf (\text{grounding_of_clss } (CC \cup \{C\}))$

unfolding *clss_of_state_def* *grounding_of_clss_def* **by** *auto*

ultimately **show** $sr_ext.derive (\text{grounding_of_clss } (CC \cup \{C + \{\#L\#\})) (\text{grounding_of_clss } (CC \cup \{C\}))$

using *sr_ext.derive.intros*[of *grounding_of_cls* $(CC \cup \{C\})$]

```

      grounding_of_cls (CC ∪ {C + {#L#}})
    by auto
qed

```

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 (induction rule: RP.induct)
  case (tautology_deletion A C N P Q)
  {
    fix C  $\sigma$ 
    assume C  $\sigma \in$  grounding_of_cls C
    then obtain  $\sigma$  where
      C  $\sigma = C \cdot \sigma$ 
    unfolding grounding_of_cls_def by auto
    then have Neg (A  $\cdot$  a  $\sigma$ )  $\in$   $\#$  C  $\sigma \wedge$  Pos (A  $\cdot$  a  $\sigma$ )  $\in$   $\#$  C  $\sigma$ 
    using tautology_deletion Neg_Melem_subst_atm_subst_cls Pos_Melem_subst_atm_subst_cls by auto
    then have C  $\sigma \in$  sr.Rf (grounding_of_state (N, P, Q))
    using sr.tautology_redundant by auto
  }
then have grounding_of_state (N ∪ {C}, P, Q) – grounding_of_state (N, P, Q)
   $\subseteq$  sr.Rf (grounding_of_state (N, P, Q))
  unfolding cls_of_state_def grounding_of_cls_def by auto
moreover have grounding_of_state (N, P, Q) – grounding_of_state (N ∪ {C}, P, Q) = {}
  unfolding cls_of_state_def grounding_of_cls_def by auto
ultimately show ?case
  using sr_ext.derive.intros[of grounding_of_state (N, P, Q) grounding_of_state (N ∪ {C}, P, Q)]
  by auto
next
  case (forward_subsumption D P Q C N)
  then show ?case
    using derive_if_remove_subsumed[of D (N, P, Q) C] unfolding grounding_of_cls_def cls_of_state_def
    by (simp add: sup_commute sup_left_commute)
next
  case (backward_subsumption_P D N C P Q)
  then show ?case
    using derive_if_remove_subsumed[of D (N, P, Q) C] strictly_subsumes_def unfolding grounding_of_cls_def
cls_of_state_def
    by (simp add: sup_commute sup_left_commute)
next
  case (backward_subsumption_Q D N C P Q)
  then show ?case
    using derive_if_remove_subsumed[of D (N, P, Q) C] strictly_subsumes_def unfolding grounding_of_cls_def
cls_of_state_def
    by (simp add: sup_commute sup_left_commute)
next
  case (forward_reduction D L' P Q L  $\sigma$  C N)
  then show ?case
    using reduction_derivable[of _ _ N ∪ P ∪ Q] unfolding cls_of_state_def by force
next
  case (backward_reduction_P D L' N L  $\sigma$  C P Q)
  then show ?case
    using reduction_derivable[of _ _ N ∪ P ∪ Q] unfolding cls_of_state_def by force
next
  case (backward_reduction_Q D L' N L  $\sigma$  C P Q)
  then show ?case
    using reduction_derivable[of _ _ N ∪ P ∪ Q] unfolding cls_of_state_def by force
next
  case (clause_processing N C P Q)
  then show ?case
    unfolding cls_of_state_def using sr_ext.derive.intros by auto
next
  case (inference_computation N Q C P)

```



```

{
  fix Eμ
  assume Eμ ∈ grounding_of_cls N
  then obtain μ E where
    E-μ-p: Eμ = E · μ ∧ E ∈ N ∧ is_ground_subst μ
    unfolding grounding_of_cls_def grounding_of_cls_def by auto
  then have E_concl: E ∈ concls_of (ord_FO_resolution.inferences_between Q C)
    using inference_computation by auto
  then obtain γ where
    γ-p: γ ∈ ord_FO_Γ S ∧ infer_from (Q ∪ {C}) γ ∧ C ∈ # prems_of γ ∧ concl_of γ = E
    unfolding ord_FO_resolution.inferences_between_def by auto
  then obtain CC CAs D AAs As σ where
    γ-p2: γ = Infer CC D E ∧ ord_resolve_rename S CAs D AAs As σ E ∧ mset CAs = CC
    unfolding ord_FO_Γ_def by auto
  define ρ where
    ρ = hd (renamings_apart (D # CAs))
  define ρs where
    ρs = tl (renamings_apart (D # CAs))
  define γ_ground where
    γ_ground = Infer (mset (CAs ..cl ρs) · cm σ · cm μ) (D · ρ · σ · μ) (E · μ)
  have ∀ I. I ⊨m mset (CAs ..cl ρs) · cm σ · cm μ → I ⊨ D · ρ · σ · μ → I ⊨ E · μ
    using ord_resolve_rename_ground_inst_sound[of ----- μ] ρ-def ρs-def E-μ-p γ-p2
    by auto
  then have γ_ground ∈ {Infer cc d e | cc d e. ∀ I. I ⊨m cc → I ⊨ d → I ⊨ e}
    unfolding γ_ground_def by auto
  moreover have set_mset (prems_of γ_ground) ⊆ grounding_of_state ({}, P ∪ {C}, Q)
  proof -
    have D = C ∨ D ∈ Q
      unfolding γ_ground_def using E-μ-p γ-p2 γ-p unfolding infer_from_def
      unfolding cls_of_state_def grounding_of_cls_def
      unfolding grounding_of_cls_def
      by simp
    then have D · ρ · σ · μ ∈ grounding_of_cls C ∨ (∃ x ∈ Q. D · ρ · σ · μ ∈ grounding_of_cls x)
      using E-μ-p
      unfolding grounding_of_cls_def
      by (metis (mono_tags, lifting) is_ground_comp_subst mem_Collect_eq subst_cls_comp_subst)
    then have (D · ρ · σ · μ ∈ grounding_of_cls C ∨
      (∃ x ∈ P. D · ρ · σ · μ ∈ grounding_of_cls x) ∨
      (∃ x ∈ Q. D · ρ · σ · μ ∈ grounding_of_cls x))
      by metis
    moreover have ∀ i < length (CAs ..cl ρs · cl σ · cl μ). ((CAs ..cl ρs · cl σ · cl μ) ! i) ∈
      {C · σ | σ. is_ground_subst σ} ∪
      ((∪ C ∈ P. {C · σ | σ. is_ground_subst σ}) ∪ (∪ C ∈ Q. {C · σ | σ. is_ground_subst σ}))
    proof (rule, rule)
      fix i
      assume i < length (CAs ..cl ρs · cl σ · cl μ)
      then have a: i < length CAs ∧ i < length ρs
        by simp
      moreover from a have CAs ! i ∈ {C} ∪ Q
        using γ-p2 γ-p unfolding infer_from_def
        by (metis (no_types, lifting) Un_subset_iff inference.sel(1) set_mset_union
          sup_commute nth_mem_mset subsetCE)
      ultimately have (CAs ..cl ρs · cl σ · cl μ) ! i ∈
        {C · σ | σ. is_ground_subst σ} ∨
        ((CAs ..cl ρs · cl σ · cl μ) ! i ∈ (∪ C ∈ P. {C · σ | σ. is_ground_subst σ}) ∨
        (CAs ..cl ρs · cl σ · cl μ) ! i ∈ (∪ C ∈ Q. {C · σ | σ. is_ground_subst σ}))
      unfolding γ_ground_def using E-μ-p γ-p2 γ-p unfolding infer_from_def
      unfolding cls_of_state_def grounding_of_cls_def
      unfolding grounding_of_cls_def
      apply -
      apply (cases CAs ! i = C)
      subgoal
        apply (rule disjI1)
    end
  end
}

```

```

    apply (rule Set.CollectI)
    apply (rule_tac x=(qs ! i) ⊙ σ ⊙ μ in exI)
    using qs_def using renames_apart apply (auto;fail)
  done
subgoal
  apply (rule disjI2)
  apply (rule disjI2)
  apply (rule_tac a=CAs ! i in UN_I)
  subgoal
    apply blast
  done
  subgoal
    apply (rule Set.CollectI)
    apply (rule_tac x=(qs ! i) ⊙ σ ⊙ μ in exI)
    using qs_def using renames_apart apply (auto;fail)
  done
done
done
then show (CAs · cl qs · cl σ · cl μ) ! i ∈ {C · σ | σ. is_ground_subst σ} ∪
  ((⋃ C ∈ P. {C · σ | σ. is_ground_subst σ}) ∪ (⋃ C ∈ Q. {C · σ | σ. is_ground_subst σ}))
  by blast
qed
then have ∀ x ∈ # mset (CAs · cl qs · cl σ · cl μ). x ∈ {C · σ | σ. is_ground_subst σ} ∪
  ((⋃ C ∈ P. {C · σ | σ. is_ground_subst σ}) ∪ (⋃ C ∈ Q. {C · σ | σ. is_ground_subst σ}))
  by (metis (lifting) in_set_conv_nth set_mset_mset)
then have set_mset (mset (CAs · cl qs) · cm σ · cm μ) ⊆
  grounding_of_cls C ∪ grounding_of_cls P ∪ grounding_of_cls Q
  unfolding grounding_of_cls_def grounding_of_cls_def
  using mset_subst_cls_list_subst_cls_mset by auto
ultimately show ?thesis
  unfolding γ_ground_def cls_of_state_def grounding_of_cls_def by auto
qed
ultimately have E · μ ∈ concls_of (sr_ext.inferences_from (grounding_of_state ({}, P ∪ {C}, Q)))
  unfolding sr_ext.inferences_from_def inference_system.inferences_from_def ground_sound_Γ_def infer_from_def
  using γ_ground_def by (metis (no.types, lifting) imageI inference.sel(3) mem_Collect_eq)
then have Eμ ∈ concls_of (sr_ext.inferences_from (grounding_of_state ({}, P ∪ {C}, Q)))
  using E.μ-p by auto
}
then have grounding_of_state (N, P, Q ∪ {C}) – grounding_of_state ({}, P ∪ {C}, Q)
  ⊆ concls_of (sr_ext.inferences_from (grounding_of_state ({}, P ∪ {C}, Q)))
  unfolding cls_of_state_def grounding_of_cls_def by auto
moreover have grounding_of_state ({}, P ∪ {C}, Q) – grounding_of_state (N, P, Q ∪ {C}) = {}
  unfolding cls_of_state_def grounding_of_cls_def by auto
ultimately show ?case
  using sr_ext.derive.intros[of (grounding_of_state (N, P, Q ∪ {C}))
    (grounding_of_state ({}, P ∪ {C}, Q))] by auto
qed

```

A useful consequence:

theorem *RP_model*:

$St \rightsquigarrow St' \implies I \models_s \text{grounding_of_state } St' \iff I \models_s \text{grounding_of_state } St$

proof (*drule RP_ground_derive, erule sr_ext.derive.cases, hypsubst*)

let

$?gSt = \text{grounding_of_state } St$ **and**

$?gSt' = \text{grounding_of_state } St'$

assume

deduct: $?gSt' - ?gSt \subseteq \text{concls_of } (sr_ext.inferences_from ?gSt)$ (**is** $- \subseteq ?\text{concls}$) **and**

delete: $?gSt - ?gSt' \subseteq sr.Rf ?gSt'$

show $I \models_s ?gSt' \iff I \models_s ?gSt$

proof

assume *bef*: $I \models_s ?gSt$

```

then have  $I \models_s ?concls$ 
  unfolding ground_sound. $\Gamma\_def$  inference_system.inferences_from_def true_cls_def true_cls_mset_def
  by (auto simp add: image_def infer_from_def dest!: spec[of _ I])
then have  $diff: I \models_s ?gSt' - ?gSt$ 
  using deduct by (blast intro: true_cls_mono)
then show  $I \models_s ?gSt'$ 
  using bef unfolding true_cls_def by blast
next
assume  $aft: I \models_s ?gSt'$ 
have  $I \models_s ?gSt' \cup sr.Rf ?gSt'$ 
  by (rule sr.Rf_model) (metis aft sr.Rf_mono[OF Un_upper1] Diff_eq_empty_iff Diff_subset
    Un_Diff true_cls_mono true_cls_union)
then have  $I \models_s sr.Rf ?gSt'$ 
  using true_cls_union by blast
then have  $diff: I \models_s ?gSt - ?gSt'$ 
  using delete by (blast intro: true_cls_mono)
then show  $I \models_s ?gSt$ 
  using aft unfolding true_cls_def by blast
qed
qed

```

Another formulation of the part of Lemma 4.10 that states we have a theorem proving process:

```

lemma RP_ground_derive_chain:
  chain sr_ext.derive (lmap grounding_of_state Sts)
  using deriv RP_ground_derive by (simp add: chain_lmap[of (~)])

```

The following is used to prove Lemma 4.11:

```

lemma in_Sup_llist_in_nth:  $C \in Sup\_llist\ Gs \implies \exists j. enat\ j < llength\ Gs \wedge C \in lnth\ Gs\ j$ 
  unfolding Sup_llist_def by auto
  — Note: Gs is called Ns in the chapter

```

```

lemma Sup_llist_grounding_of_state_ground:
  assumes  $C \in Sup\_llist\ (lmap\ grounding\_of\_state\ Sts)$ 
  shows is_ground_cls C

```

```

proof —
  have  $\exists j. enat\ j < llength\ (lmap\ grounding\_of\_state\ Sts) \wedge C \in lnth\ (lmap\ grounding\_of\_state\ Sts)\ j$ 
    using assms in_Sup_llist_in_nth by metis
  then obtain  $j$  where
     $enat\ j < llength\ (lmap\ grounding\_of\_state\ Sts)$ 
     $C \in lnth\ (lmap\ grounding\_of\_state\ Sts)\ j$ 
    by blast
  then show ?thesis
    unfolding grounding_of_cls_def grounding_of_cls_def by auto
qed

```

```

lemma Liminf_grounding_of_state_ground:
   $C \in Liminf\_llist\ (lmap\ grounding\_of\_state\ Sts) \implies is\_ground\_cls\ C$ 
  using Liminf_llist_subset_Sup_llist[of lmap grounding_of_state Sts]
  Sup_llist_grounding_of_state_ground
  by blast

```

```

lemma in_Sup_llist_in_Sup_state:
  assumes  $C \in Sup\_llist\ (lmap\ grounding\_of\_state\ Sts)$ 
  shows  $\exists D\ \sigma. D \in clss\_of\_state\ (Sup\_state\ Sts) \wedge D \cdot \sigma = C \wedge is\_ground\_subst\ \sigma$ 
proof —
  from assms obtain  $i$  where
     $i\_p: enat\ i < llength\ Sts \wedge C \in lnth\ (lmap\ grounding\_of\_state\ Sts)\ i$ 
    using in_Sup_llist_in_nth by fastforce
  then obtain  $D\ \sigma$  where
     $D \in clss\_of\_state\ (lnth\ Sts\ i) \wedge D \cdot \sigma = C \wedge is\_ground\_subst\ \sigma$ 
    using assms unfolding grounding_of_cls_def grounding_of_cls_def by fastforce
  then have  $D \in clss\_of\_state\ (Sup\_state\ Sts) \wedge D \cdot \sigma = C \wedge is\_ground\_subst\ \sigma$ 
    using  $i\_p$  unfolding Sup_state_def clss_of_state_def

```

by (metis (no_types, lifting) UnCI UnE contra_subsetD N_of_state.simps P_of_state.simps
 Q_of_state.simps llength_lmap lnth_lmap lnth_subset_Sup_llist)
then show ?thesis
 by auto
qed

lemma

N_of_state_Liminf: $N_of_state (Liminf_state\ Sts) = Liminf_llist (lmap\ N_of_state\ Sts)$ **and**
P_of_state_Liminf: $P_of_state (Liminf_state\ Sts) = Liminf_llist (lmap\ P_of_state\ Sts)$
unfolding *Liminf_state_def* **by** auto

lemma *eventually_removed_from_N*:

assumes
d_in: $D \in N_of_state (lnth\ Sts\ i)$ **and**
fair: *fair_state_seq* *Sts* **and**
i_Sts: *enat* $i < llength\ Sts$
shows $\exists l. D \in N_of_state (lnth\ Sts\ l) \wedge D \notin N_of_state (lnth\ Sts\ (Suc\ l)) \wedge i \leq l \wedge enat (Suc\ l) < llength\ Sts$
proof (rule *ccontr*)
assume $a: \neg ?thesis$
have $i \leq l \implies enat\ l < llength\ Sts \implies D \in N_of_state (lnth\ Sts\ l)$ **for** l
using *d_in* **by** (induction l , blast, metis *a* *Suc_ile_eq* *le_SucE* *less_imp_le*)
then have $D \in Liminf_llist (lmap\ N_of_state\ Sts)$
unfolding *Liminf_llist_def* **using** *i_Sts* **by** auto
then show *False*
using *fair* **unfolding** *fair_state_seq_def* **by** (simp add: *N_of_state_Liminf*)
qed

lemma *eventually_removed_from_P*:

assumes
d_in: $D \in P_of_state (lnth\ Sts\ i)$ **and**
fair: *fair_state_seq* *Sts* **and**
i_Sts: *enat* $i < llength\ Sts$
shows $\exists l. D \in P_of_state (lnth\ Sts\ l) \wedge D \notin P_of_state (lnth\ Sts\ (Suc\ l)) \wedge i \leq l \wedge enat (Suc\ l) < llength\ Sts$
proof (rule *ccontr*)
assume $a: \neg ?thesis$
have $i \leq l \implies enat\ l < llength\ Sts \implies D \in P_of_state (lnth\ Sts\ l)$ **for** l
using *d_in* **by** (induction l , blast, metis *a* *Suc_ile_eq* *le_SucE* *less_imp_le*)
then have $D \in Liminf_llist (lmap\ P_of_state\ Sts)$
unfolding *Liminf_llist_def* **using** *i_Sts* **by** auto
then show *False*
using *fair* **unfolding** *fair_state_seq_def* **by** (simp add: *P_of_state_Liminf*)
qed

lemma *instance_if_subsumed_and_in_limit*:

assumes
ns: $Gs = lmap\ grounding_of_state\ Sts$ **and**
c: $C \in Liminf_llist\ Gs - sr.Rf (Liminf_llist\ Gs)$ **and**
 $d: D \in N_of_state (lnth\ Sts\ i) \cup P_of_state (lnth\ Sts\ i) \cup Q_of_state (lnth\ Sts\ i)$
 $enat\ i < llength\ Sts\ subsumes\ D\ C$
shows $\exists \sigma. D \cdot \sigma = C \wedge is_ground_subst\ \sigma$
proof –
let $?Ps = \lambda i. P_of_state (lnth\ Sts\ i)$
let $?Qs = \lambda i. Q_of_state (lnth\ Sts\ i)$

have *ground_C*: *is_ground_cls* C
using *c* **using** *Liminf_grounding_of_state_ground* *ns* **by** auto

have *derivns*: *chain* *sr_ext.derive* Gs
using *RP_ground_derive_chain* *deriv* *ns* **by** auto

have $\exists \sigma. D \cdot \sigma = C$

```

proof (rule ccontr)
  assume  $\nexists \sigma. D \cdot \sigma = C$ 
  moreover from  $d(\beta)$  obtain  $\tau\_proto$  where
     $D \cdot \tau\_proto \subseteq\# C$  unfolding subsumes_def
    by blast
  then obtain  $\tau$  where
     $\tau\_p: D \cdot \tau \subseteq\# C \wedge is\_ground\_subst \tau$ 
    using ground_C by (metis is_ground_cls_mono make_ground_subst subset_mset.order_refl)
  ultimately have  $subsub: D \cdot \tau \subseteq\# C$ 
    using subset_mset.le_imp_less_or_eq by auto
  moreover have is_ground_subst  $\tau$ 
    using  $\tau\_p$  by auto
  moreover have  $D \in clss\_of\_state (lnth Sts i)$ 
    using  $d$  unfolding clss_of_state_def by auto
  ultimately have  $C \in sr.Rf (grounding\_of\_state (lnth Sts i))$ 
    using strict_subset_subsumption_redundant_cls by auto
  then have  $C \in sr.Rf (Sup\_llist Gs)$ 
    using  $d$  ns by (metis contra_subsetD llength_lmap lnth_lmap lnth_subset_Sup_llist sr.Rf_mono)
  then have  $C \in sr.Rf (Liminf\_llist Gs)$ 
    unfolding  $ns$  using local_sr_ext.Rf_Sup_subset_Rf_Liminf derivns ns by auto
  then show False
    using  $c$  by auto
qed
then obtain  $\sigma$  where
   $D \cdot \sigma = C \wedge is\_ground\_subst \sigma$ 
  using ground_C by (metis make_ground_subst)
then show ?thesis
  by auto
qed

lemma from_Q_to_Q_inf:
  assumes
    fair: fair_state_seq  $Sts$  and
    ns:  $Gs = lmap \text{grounding\_of\_state } Sts$  and
    c:  $C \in Liminf\_llist Gs - sr.Rf (Liminf\_llist Gs)$  and
    d:  $D \in Q\_of\_state (lnth Sts i) \text{ enat } i < llength Sts \text{ subsumes } D C$  and
    d_least:  $\forall E \in \{E. E \in (clss\_of\_state (Sup\_state Sts)) \wedge \text{subsumes } E C\}. \neg \text{strictly\_subsumes } E D$ 
  shows  $D \in Q\_of\_state (Liminf\_state Sts)$ 

proof -
  let  $?Ps = \lambda i. P\_of\_state (lnth Sts i)$ 
  let  $?Qs = \lambda i. Q\_of\_state (lnth Sts i)$ 

  have ground_C: is_ground_cls  $C$ 
    using  $c$  using Liminf_grounding_of_state_ground ns by auto

  have derivns: chain sr_ext.derive  $Gs$ 
    using RP_ground_derive_chain deriv ns by auto

  have  $\exists \sigma. D \cdot \sigma = C \wedge is\_ground\_subst \sigma$ 
    using instance_if_subsumed_and_in_limit ns c d by blast
  then obtain  $\sigma$  where
     $\sigma: D \cdot \sigma = C \wedge is\_ground\_subst \sigma$ 
    by auto

  have in_Sts.in_Sts_Suc:
     $\forall l \geq i. \text{enat } (Suc l) < llength Sts \longrightarrow D \in Q\_of\_state (lnth Sts l) \longrightarrow D \in Q\_of\_state (lnth Sts (Suc l))$ 
proof (rule, rule, rule, rule)
  fix  $l$ 
  assume
    len:  $i \leq l$  and
    llen:  $\text{enat } (Suc l) < llength Sts$  and
    d_in_q:  $D \in Q\_of\_state (lnth Sts l)$ 

```

```

have lnth Sts l  $\rightsquigarrow$  lnth Sts (Suc l)
  using llen deriv chain_lnth_rel by blast
then show  $D \in Q\_of\_state$  (lnth Sts (Suc l))
proof (cases rule: RP.cases)
  case (backward_subsumption_Q D' N D_removed P Q)
  moreover
  {
    assume D_removed = D
    then obtain D_subsumes where
      D_subsumes_p: D_subsumes  $\in$  N  $\wedge$  strictly_subsumes D_subsumes D
    using backward_subsumption_Q by auto
    moreover from D_subsumes_p have subsumes D_subsumes C
    using d subsumes_trans unfolding strictly_subsumes_def by blast
    moreover from backward_subsumption_Q have D_subsumes  $\in$  class_of_state (Sup.state Sts)
    using D_subsumes_p llen
    by (metis (no_types) UnI1 class_of_state_def N_of_state.simps llength_lmap lnth_lmap
      lnth_subset_Sup_llist rev_subsetD Sup.state_def)
    ultimately have False
    using d_least unfolding subsumes_def by auto
  }
ultimately show ?thesis
  using d_in_q by auto
next
case (backward_reduction_Q E L' N L  $\sigma$  D' P Q)
{
  assume D' + {#L#} = D
  then have D'_p: strictly_subsumes D' D  $\wedge$  D'  $\in$  ?Ps (Suc l)
    using subset_strictly_subsumes[of D' D] backward_reduction_Q by auto
  then have subc: subsumes D' C
    using d(3) subsumes_trans unfolding strictly_subsumes_def by auto
  from D'_p have D'  $\in$  class_of_state (Sup.state Sts)
    using llen by (metis (no_types) UnI1 class_of_state_def P_of_state.simps llength_lmap
      lnth_lmap lnth_subset_Sup_llist subsetCE sup_ge2 Sup.state_def)
  then have False
    using d_least D'_p subc by auto
}
then show ?thesis
  using backward_reduction_Q d_in_q by auto
qed (use d_in_q in auto)
qed
have D_in_Sts:  $D \in Q\_of\_state$  (lnth Sts l) and D_in_Sts_Suc:  $D \in Q\_of\_state$  (lnth Sts (Suc l))
  if Li:  $l \geq i$  and enat:  $enat$  (Suc l)  $<$  llength Sts for l
proof -
  show  $D \in Q\_of\_state$  (lnth Sts l)
    using Li enat
    apply (induction l - i arbitrary: l)
    subgoal using d by auto
    subgoal using d(1) in_Sts_in_Sts_Suc
      by (metis (no_types, lifting) Suc_ile_eq add_Suc_right add_diff_cancel_left' le_SucE
        le_Suc_ex less_imp_le)
    done
  then show  $D \in Q\_of\_state$  (lnth Sts (Suc l))
    using Li enat in_Sts_in_Sts_Suc by blast
qed
have  $i \leq x \implies enat$  x  $<$  llength Sts  $\implies D \in Q\_of\_state$  (lnth Sts x) for x
  apply (cases x)
  subgoal using d(1) by (auto intro!: exI[of _ i] simp: less_Suc_eq)
  subgoal for x'
    using d(1) D_in_Sts_Suc[of x'] by (cases (i  $\leq$  x')) (auto simp: not_less_eq_eq)
  done
then have  $D \in Liminf\_llist$  (lmap Q_of_state Sts)
  unfolding Liminf_llist_def by (auto intro!: exI[of _ i] simp: d)
then show ?thesis

```

unfolding *Liminf_state_def* by auto
qed

lemma *from_P_to_Q*:

assumes

fair: *fair_state_seq* *Sts* and

ns: *Gs* = *lmap* *grounding_of_state* *Sts* and

c: *C* ∈ *Liminf_llist* *Gs* - *sr.Rf* (*Liminf_llist* *Gs*) and

d: *D* ∈ *P_of_state* (*lnth* *Sts* *i*) enat *i* < *llength* *Sts* subsumes *D* *C* and

d_least: $\forall E \in \{E. E \in (\text{class_of_state } (\text{Sup_state } \text{Sts})) \wedge \text{subsumes } E \ C\}. \neg \text{strictly_subsumes } E \ D$

shows $\exists l. D \in Q_of_state$ (*lnth* *Sts* *l*) ∧ enat *l* < *llength* *Sts*

proof -

let *?Ns* = $\lambda i. N_of_state$ (*lnth* *Sts* *i*)

let *?Ps* = $\lambda i. P_of_state$ (*lnth* *Sts* *i*)

let *?Qs* = $\lambda i. Q_of_state$ (*lnth* *Sts* *i*)

have *ground_C*: *is_ground_cls* *C*

using *c* using *Liminf_grounding_of_state_ground* *ns* by auto

have *derivns*: *chain* *sr_ext.derive* *Gs*

using *RP_ground_derive_chain* *deriv* *ns* by auto

have $\exists \sigma. D \cdot \sigma = C \wedge \text{is_ground_subst } \sigma$

using *instance_if_subsumed_and_in_limit* *ns* *c* *d* by blast

then obtain σ where

$\sigma: D \cdot \sigma = C$ *is_ground_subst* σ

by auto

obtain *l* where

l_p: *D* ∈ *P_of_state* (*lnth* *Sts* *l*) ∧ *D* ∉ *P_of_state* (*lnth* *Sts* (*Suc* *l*)) ∧ *i* ≤ *l* ∧ enat (*Suc* *l*) < *llength* *Sts*

using *fair* using *eventually_removed_from_P* *d* unfolding *ns* by auto

then have *LGs*: enat (*Suc* *l*) < *llength* *Gs*

using *ns* by auto

from *l_p* have *lnth* *Sts* *l* \rightsquigarrow *lnth* *Sts* (*Suc* *l*)

using *deriv* using *chain_lnth_rel* by auto

then show *?thesis*

proof (cases rule: *RP.cases*)

case (*backward_subsumption_P* *D'* *N* *D_twin* *P* *Q*)

note *lhs* = *this*(1,2) and *D'_p* = *this*(3,4)

then have *twins*: *D_twin* = *D* *?Ns* (*Suc* *l*) = *N* *?Ns* *l* = *N* *?Ps* (*Suc* *l*) = *P*

?Ps *l* = *P* ∪ {*D_twin*} *?Qs* (*Suc* *l*) = *Q* *?Qs* *l* = *Q*

using *l_p* by auto

note *D'_p* = *D'_p*[*unfolded* *twins*(1)]

then have *subc*: *subsumes* *D'* *C*

unfolding *strictly_subsumes_def* *subsumes_def* using σ

by (*metis* *subst_cls_comp_subst* *subst_cls_mono_mset*)

from *D'_p* have *D'* ∈ *class_of_state* (*Sup_state* *Sts*)

unfolding *twins*(2)[*symmetric*] using *l_p*

by (*metis* (*no_types*) *UnI1* *class_of_state_def* *N_of_state.simps* *llength_lmap* *lnth_lmap*

lnth_subset_Sup_llist *subsetCE* *Sup_state_def*)

then have *False*

using *d_least* *D'_p* *subc* by auto

then show *?thesis*

by auto

next

case (*backward_reduction_P* *E* *L'* *N* *L* σ *D'* *P* *Q*)

then have *twins*: *D'* + {*#L#*} = *D* *?Ns* (*Suc* *l*) = *N* *?Ns* *l* = *N* *?Ps* (*Suc* *l*) = *P* ∪ {*D'*}

?Ps *l* = *P* ∪ {*D'* + {*#L#*}} *?Qs* (*Suc* *l*) = *Q* *?Qs* *l* = *Q*

using *l_p* by auto

then have *D'_p*: *strictly_subsumes* *D'* *D* ∧ *D'* ∈ *?Ps* (*Suc* *l*)

using *subset_strictly_subsumes*[of *D'* *D*] by auto

then have *subc*: *subsumes* *D'* *C*

using *d*(3) *subsumes_trans* unfolding *strictly_subsumes_def* by auto

```

from  $D'_p$  have  $D' \in \text{class\_of\_state } (Sup\_state\ Sts)$ 
  using  $L_p$  by (metis (no\_types) UnI1 class\_of\_state\_def P\_of\_state.simps llength\_lmap lnth\_lmap
    lnth\_subset\_Sup\_l1list subsetCE sup\_ge2 Sup\_state\_def)
then have False
  using d\_least D'_p subc by auto
then show ?thesis
  by auto
next
  case (inference\_computation N Q D\_twin P)
then have  $D\_twin = D \ ?Ps\ (Suc\ l) = P \ ?Ps\ l = P \cup \{D\_twin\}$ 
   $\ ?Qs\ (Suc\ l) = Q \cup \{D\_twin\} \ ?Qs\ l = Q$ 
  using  $L_p$  by auto
then show ?thesis
  using  $d\ \sigma\ L_p$  by auto
qed (use L_p in auto)
qed

```

```

lemma variants_sym: variants D D'  $\longleftrightarrow$  variants D' D
  unfolding variants\_def by auto

```

```

lemma variants_imp_exists_substitution: variants D D'  $\implies \exists \sigma. D \cdot \sigma = D'$ 
  unfolding variants\_iff\_subsumes subsumes\_def
  by (meson strictly\_subsumes\_def subset\_mset\_def strict\_subset\_subst\_strictly\_subsumes subsumes\_def)

```

```

lemma properly\_subsume\_variants:
  assumes strictly\_subsumes E D and variants D D'
  shows strictly\_subsumes E D'

```

```

proof -
  from assms obtain  $\sigma\ \sigma'$  where
     $\sigma_{\sigma'_p}: D \cdot \sigma = D' \wedge D' \cdot \sigma' = D$ 
  using variants\_imp\_exists\_substitution variants\_sym by metis

```

```

from assms obtain  $\sigma''$  where
   $E \cdot \sigma'' \subseteq\# D$ 
  unfolding strictly\_subsumes\_def subsumes\_def by auto
then have  $E \cdot \sigma'' \cdot \sigma \subseteq\# D \cdot \sigma$ 
  using subst\_cls\_mono\_mset by blast
then have  $E \cdot (\sigma'' \odot \sigma) \subseteq\# D'$ 
  using  $\sigma_{\sigma'_p}$  by auto
moreover from assms have  $n: (\exists \sigma. D \cdot \sigma \subseteq\# E)$ 
  unfolding strictly\_subsumes\_def subsumes\_def by auto
have  $\exists \sigma. D' \cdot \sigma \subseteq\# E$ 

```

```

proof
  assume  $\exists \sigma'''. D' \cdot \sigma''' \subseteq\# E$ 
  then obtain  $\sigma'''$  where
     $D' \cdot \sigma''' \subseteq\# E$ 
  by auto
  then have  $D \cdot (\sigma \odot \sigma''') \subseteq\# E$ 
  using  $\sigma_{\sigma'_p}$  by auto
  then show False
  using  $n$  by metis
qed
ultimately show ?thesis
  unfolding strictly\_subsumes\_def subsumes\_def by metis
qed

```

```

lemma neg\_properly\_subsume\_variants:
  assumes  $\neg$  strictly\_subsumes E D and variants D D'
  shows  $\neg$  strictly\_subsumes E D'
  using assms properly\_subsume\_variants variants\_sym by auto

```

```

lemma from\_N\_to\_P\_or\_Q:
  assumes

```



```

fair: fair_state_seq Sts and
ns: Gs = lmap grounding_of_state Sts and
c: C ∈ Liminf_llist Gs - sr.Rf (Liminf_llist Gs) and
d: D ∈ N_of_state (lnth Sts i) enat i < llength Sts subsumes D C and
d.least: ∀ E ∈ {E. E ∈ (class_of_state (Sup_state Sts)) ∧ subsumes E C}. ¬ strictly_subsumes E D
shows ∃ l D' σ'. D' ∈ P_of_state (lnth Sts l) ∪ Q_of_state (lnth Sts l) ∧
  enat l < llength Sts ∧
  (∀ E ∈ {E. E ∈ (class_of_state (Sup_state Sts)) ∧ subsumes E C}. ¬ strictly_subsumes E D') ∧
  D' · σ' = C ∧ is_ground_subst σ' ∧ subsumes D' C
proof -
let ?Ns = λi. N_of_state (lnth Sts i)
let ?Ps = λi. P_of_state (lnth Sts i)
let ?Qs = λi. Q_of_state (lnth Sts i)

have ground_C: is_ground_cls C
  using c using Liminf_grounding_of_state_ground ns by auto

have derivns: chain sr_ext.derive Gs
  using RP_ground.derive_chain deriv ns by auto

have ∃ σ. D · σ = C ∧ is_ground_subst σ
  using instance_if_subsumed_and_in_limit ns c d by blast
then obtain σ where
  σ: D · σ = C is_ground_subst σ
  by auto

from c have no_taut: ¬ (∃ A. Pos A ∈# C ∧ Neg A ∈# C)
  using sr.tautology_redundant by auto

have ∃ l. D ∈ N_of_state (lnth Sts l) ∧ D ∉ N_of_state (lnth Sts (Suc l)) ∧ i ≤ l ∧ enat (Suc l) < llength Sts
  using fair using eventually_removed_from_N d unfolding ns by auto
then obtain l where
  l_p: D ∈ N_of_state (lnth Sts l) ∧ D ∉ N_of_state (lnth Sts (Suc l)) ∧ i ≤ l ∧ enat (Suc l) < llength Sts
  by auto
then have l_Gs: enat (Suc l) < llength Gs
  using ns by auto
from l_p have lnth Sts l ∼ lnth Sts (Suc l)
  using deriv using chain_lnth_rel by auto
then show ?thesis
proof (cases rule: RP.cases)
  case (tautology_deletion A D_twin N P Q)
  then have D_twin = D
  using l_p by auto
  then have Pos (A · a σ) ∈# C ∧ Neg (A · a σ) ∈# C
  using tautology_deletion(3,4) σ
  by (metis Melem_subst_cls eql_neg_lit_eql_atm eql_pos_lit_eql_atm)
  then have False
  using no_taut by metis
  then show ?thesis
  by blast
next
  case (forward_subsumption D' P Q D_twin N)
  note lrhs = this(1,2) and D'_p = this(3,4)
  then have twins: D_twin = D ?Ns (Suc l) = N ?Ns l = N ∪ {D_twin} ?Ps (Suc l) = P
    ?Ps l = P ?Qs (Suc l) = Q ?Qs l = Q
  using l_p by auto
  note D'_p = D'_p[unfolded twins(1)]
  from D'_p(2) have subs: subsumes D' C
  using d(3) by (blast intro: subsumes_trans)
  moreover have D' ∈ class_of_state (Sup_state Sts)
  using twins D'_p l_p unfolding class_of_state_def Sup_state_def
  by simp (metis (no_types) contra_subsetD llength_lmap lnth_lmap lnth_subset_Sup_llist)
  ultimately have ¬ strictly_subsumes D' D

```

```

    using d_least by auto
  then have subsumes D D'
    unfolding strictly_subsumes_def using D'_p by auto
  then have v: variants D D'
    using D'_p unfolding variants_iff_subsumes by auto
  then have mini:  $\forall E \in \{E \in \text{cls\_of\_state } (\text{Sup\_state } \text{Sts}). \text{subsumes } E \ C\}. \neg \text{strictly\_subsumes } E \ D'$ 
    using d_least D'_p neg_properly_subsume_variants[of _ D D'] by auto

  from v have  $\exists \sigma'. D' \cdot \sigma' = C$ 
    using  $\sigma$  variants_imp_exists_substitution variants_sym by (metis subst_cls_comp_subst)
  then have  $\exists \sigma'. D' \cdot \sigma' = C \wedge \text{is\_ground\_subst } \sigma'$ 
    using ground_C by (meson make_ground_subst refl)
  then obtain  $\sigma'$  where
     $\sigma'_p: D' \cdot \sigma' = C \wedge \text{is\_ground\_subst } \sigma'$ 
    by metis

  show ?thesis
    using D'_p twins L_p subs mini  $\sigma'_p$  by auto
next
  case (forward_reduction E L' P Q L  $\sigma$  D' N)
  then have twins:  $D' + \{\#L\# \} = D \ ?Ns \ (Suc \ l) = N \cup \{D'\} \ ?Ns \ l = N \cup \{D' + \{\#L\# \}$ 
     $\ ?Ps \ (Suc \ l) = P \ ?Ps \ l = P \ ?Qs \ (Suc \ l) = Q \ ?Qs \ l = Q$ 
    using L_p by auto
  then have D'_p: strictly_subsumes D' D  $\wedge D' \in \ ?Ns \ (Suc \ l)$ 
    using subset_strictly_subsumes[of D' D] by auto
  then have subc: subsumes D' C
    using d(3) subsumes_trans unfolding strictly_subsumes_def by blast
  from D'_p have  $D' \in \text{cls\_of\_state } (\text{Sup\_state } \text{Sts})$ 
    using L_p by (metis (no_types) Un11 cls_of_state_def N_of_state.simps llength_lmap lnth_lmap
    lnth_subset_Sup_llist subsetCE Sup_state_def)
  then have False
    using d_least D'_p subc by auto
  then show ?thesis
    by auto
next
  case (clause_processing N D_twin P Q)
  then have twins:  $D\_twin = D \ ?Ns \ (Suc \ l) = N \ ?Ns \ l = N \cup \{D\} \ ?Ps \ (Suc \ l) = P \cup \{D\}$ 
     $\ ?Ps \ l = P \ ?Qs \ (Suc \ l) = Q \ ?Qs \ l = Q$ 
    using L_p by auto
  then show ?thesis
    using d  $\sigma$  L_p d_least by blast
qed (use L_p in auto)
qed

lemma eventually_in_Qinf:
  assumes
    D_p:  $D \in \text{cls\_of\_state } (\text{Sup\_state } \text{Sts})$ 
    subsumes D C  $\forall E \in \{E. E \in (\text{cls\_of\_state } (\text{Sup\_state } \text{Sts})) \wedge \text{subsumes } E \ C\}. \neg \text{strictly\_subsumes } E \ D$  and
    fair: fair_state_seq Sts and

    ns:  $Gs = \text{lmap } \text{grounding\_of\_state } \text{Sts}$  and
    c:  $C \in \text{Liminf\_llist } Gs - \text{sr.Rf } (\text{Liminf\_llist } Gs)$  and
    ground_C: is_ground_cls C
  shows  $\exists D' \ \sigma'. D' \in \text{Q\_of\_state } (\text{Liminf\_state } \text{Sts}) \wedge D' \cdot \sigma' = C \wedge \text{is\_ground\_subst } \sigma'$ 
proof -
  let  $\ ?Ns = \lambda i. N\_of\_state \ (lnth \ \text{Sts} \ i)$ 
  let  $\ ?Ps = \lambda i. P\_of\_state \ (lnth \ \text{Sts} \ i)$ 
  let  $\ ?Qs = \lambda i. Q\_of\_state \ (lnth \ \text{Sts} \ i)$ 

  from D_p obtain i where
    i_p:  $i < \text{llength } \text{Sts} \ D \in \ ?Ns \ i \vee D \in \ ?Ps \ i \vee D \in \ ?Qs \ i$ 
    unfolding cls_of_state_def Sup_state_def
    by simp_all (metis (no_types) in_Sup_llist_in_nth llength_lmap lnth_lmap)

```

have *derivns*: *chain sr_ext.derive Gs using RP_ground_derive_chain deriv ns by auto*

have $\exists \sigma. D \cdot \sigma = C \wedge is_ground_subst \sigma$
using *instance-if-subsumed-and-in-limit[OF ns c] D-p i-p by blast*
then obtain σ **where**
 $\sigma: D \cdot \sigma = C$ *is_ground_subst* σ
by *blast*

{
assume $a: D \in ?Ns\ i$
then obtain $D' \sigma' l$ **where** D'_p :
 $D' \in ?Ps\ l \cup ?Qs\ l$
 $D' \cdot \sigma' = C$
 $enat\ l < llength\ Sts$
 $is_ground_subst\ \sigma'$
 $\forall E \in \{E. E \in (clss_of_state\ (Sup_state\ Sts)) \wedge subsumes\ E\ C\}. \neg\ strictly_subsumes\ E\ D'$
 $subsumes\ D'\ C$
using *from_N_to_P_or_Q deriv fair ns c i-p(1) D-p(2) D-p(3) by blast*
then obtain l' **where**
 $l'_p: D' \in ?Qs\ l'\ l' < llength\ Sts$
using *from_P_to_Q[OF fair ns c - D'_p(3) D'_p(6) D'_p(5)] by blast*
then have $D' \in Q_of_state\ (Liminf_state\ Sts)$
using *from_Q_to_Q_inf[OF fair ns c - l'_p(2)] D'_p by auto*
then have *?thesis*
using D'_p **by** *auto*
}

moreover
{
assume $a: D \in ?Ps\ i$
then obtain l' **where**
 $l'_p: D \in ?Qs\ l'\ l' < llength\ Sts$
using *from_P_to_Q[OF fair ns c a i-p(1) D-p(2) D-p(3)] by auto*
then have $D \in Q_of_state\ (Liminf_state\ Sts)$
using *from_Q_to_Q_inf[OF fair ns c l'-p(1) l'-p(2)] D-p(3) \sigma(1) \sigma(2) D-p(2) by auto*
then have *?thesis*
using $D_p\ \sigma$ **by** *auto*
}

moreover
{
assume $a: D \in ?Qs\ i$
then have $D \in Q_of_state\ (Liminf_state\ Sts)$
using *from_Q_to_Q_inf[OF fair ns c a i-p(1)] \sigma D-p(2,3) by auto*
then have *?thesis*
using $D_p\ \sigma$ **by** *auto*
}

ultimately show *?thesis*
using i_p **by** *auto*

qed

The following corresponds to Lemma 4.11:

lemma *fair_imp_Liminf_minus_Rf_subset_ground_Liminf_state*:

assumes

fair: *fair_state_seq Sts* **and**

ns: $Gs = lmap\ grounding_of_state\ Sts$

shows $Liminf_llist\ Gs - sr.Rf\ (Liminf_llist\ Gs) \subseteq grounding_of_clss\ (Q_of_state\ (Liminf_state\ Sts))$

proof

let $?Ns = \lambda i. N_of_state\ (lnth\ Sts\ i)$

let $?Ps = \lambda i. P_of_state\ (lnth\ Sts\ i)$

let $?Qs = \lambda i. Q_of_state\ (lnth\ Sts\ i)$

have $SQinf: clss_of_state\ (Liminf_state\ Sts) = Liminf_llist\ (lmap\ Q_of_state\ Sts)$

using *fair unfolding fair_state_seq_def Liminf_state_def clss_of_state_def by auto*

```

fix C
assume C_p: C ∈ Liminf_llist Gs — sr.Rf (Liminf_llist Gs)
then have C ∈ Sup_llist Gs
  using Liminf_llist_subset_Sup_llist[of Gs] by blast
then obtain D_proto where
  D_proto ∈ clss_of_state (Sup_state Sts) ∧ subsumes D_proto C
  using in_Sup_llist_in_Sup_state unfolding ns subsumes_def by blast
then obtain D where
  D_p: D ∈ clss_of_state (Sup_state Sts)
  subsumes D C
  ∀ E ∈ {E. E ∈ clss_of_state (Sup_state Sts) ∧ subsumes E C}. ¬ strictly_subsumes E D
  using strictly_subsumes_has_minimum[of {E. E ∈ clss_of_state (Sup_state Sts) ∧ subsumes E C}]
  by auto

have ground_C: is_ground_cls C
  using C_p using Liminf_grounding_of_state_ground ns by auto

have ∃ D' σ'. D' ∈ Q_of_state (Liminf_state Sts) ∧ D' · σ' = C ∧ is_ground_subst σ'
  using eventually_in_Qinf[of D C Gs] using D_p(1) D_p(2) D_p(3) fair ns C_p ground_C by auto
then obtain D' σ' where
  D'_p: D' ∈ Q_of_state (Liminf_state Sts) ∧ D' · σ' = C ∧ is_ground_subst σ'
  by blast
then have D' ∈ clss_of_state (Liminf_state Sts)
  by (simp add: clss_of_state_def)
then have C ∈ grounding_of_state (Liminf_state Sts)
  unfolding grounding_of_cls_def grounding_of_cls_def using D'_p by auto
then show C ∈ grounding_of_cls (Q_of_state (Liminf_state Sts))
  using SQinf clss_of_state_def fair fair_state_seq_def by auto
qed

```

The following corresponds to (one direction of) Theorem 4.13:

lemma *ground_subclauses*:

```

assumes
  ∀ i < length CAs. CAs ! i = Cs ! i + poss (AAs ! i) and
  length Cs = length CAs and
  is_ground_cls_list CAs
shows is_ground_cls_list Cs
unfolding is_ground_cls_list_def
by (metis assms in_set_conv_nth is_ground_cls_list_def is_ground_cls_union)

```

lemma *subsetq_Liminf_state_eventually_always*:

```

fixes CC
assumes
  finite CC and
  CC ≠ {} and
  CC ⊆ Q_of_state (Liminf_state Sts)
shows ∃ j. enat j < llength Sts ∧ (∀ j' ≥ enat j. j' < llength Sts → CC ⊆ Q_of_state (lnth Sts j'))
proof —
  from assms(3) have ∀ C ∈ CC. ∃ j. enat j < llength Sts ∧
    (∀ j' ≥ enat j. j' < llength Sts → C ∈ Q_of_state (lnth Sts j'))
  unfolding Liminf_state_def Liminf_llist_def by force
  then obtain f where
    f_p: ∀ C ∈ CC. f C < llength Sts ∧ (∀ j' ≥ enat (f C). j' < llength Sts → C ∈ Q_of_state (lnth Sts j'))
  by moura

  define j :: nat where
    j = Max (f ` CC)

```

```

have enat j < llength Sts
  unfolding j_def using f_p assms(1)
  by (metis (mono_tags) Max_in assms(2) finite_imageI imageE image_is_empty)
moreover have ∀ C j'. C ∈ CC → enat j ≤ j' → j' < llength Sts → C ∈ Q_of_state (lnth Sts j')

```

```

proof (intro allI impI)
  fix C :: 'a clause and j' :: nat
  assume a: C ∈ CC enat j ≤ enat j' enat j' < llength Sts
  then have f C ≤ j'
    unfolding j-def using assms(1) Max.bounded-iff by auto
  then show C ∈ Q_of_state (lnth Sts j')
    using f-p a by auto
qed
ultimately show ?thesis
  by auto
qed

lemma empty_clause_in_Q_of_Liminf_state:
  assumes
    empty_in: {#} ∈ Liminf_llist (lmap grounding_of_state Sts) and
    fair: fair_state_seq Sts
  shows {#} ∈ Q_of_state (Liminf_state Sts)
proof -
  define Gs :: 'a clause set llist where
    ns: Gs = lmap grounding_of_state Sts
  from empty_in have in_Liminf_not_Rf: {#} ∈ Liminf_llist Gs - sr.Rf (Liminf_llist Gs)
  unfolding ns sr.Rf-def by auto
  then have {#} ∈ grounding_of_clss (Q_of_state (Liminf_state Sts))
  using fair_imp_Liminf_minus_Rf_subset_ground_Liminf_state[OF fair ns] by auto
  then show ?thesis
    unfolding grounding_of_clss_def grounding_of_cls_def by auto
qed

lemma grounding_of_state_Liminf_state_subseteq:
  grounding_of_state (Liminf_state Sts) ⊆ Liminf_llist (lmap grounding_of_state Sts)
proof
  fix C :: 'a clause
  assume C ∈ grounding_of_state (Liminf_state Sts)
  then obtain D σ where
    D_σ_p: D ∈ clss_of_state (Liminf_state Sts) D · σ = C is_ground_subst σ
  unfolding clss_of_state_def grounding_of_clss_def grounding_of_cls_def by auto
  then have ii: D ∈ Liminf_llist (lmap N_of_state Sts) ∨
    D ∈ Liminf_llist (lmap P_of_state Sts) ∨
    D ∈ Liminf_llist (lmap Q_of_state Sts)
  unfolding clss_of_state_def Liminf_state_def by simp
  then have C ∈ Liminf_llist (lmap grounding_of_clss (lmap N_of_state Sts)) ∨
    C ∈ Liminf_llist (lmap grounding_of_clss (lmap P_of_state Sts)) ∨
    C ∈ Liminf_llist (lmap grounding_of_clss (lmap Q_of_state Sts))
  unfolding Liminf_llist_def grounding_of_clss_def grounding_of_cls_def
  apply -
  apply (erule disjE)
  subgoal
    apply (rule disjI1)
    using D_σ_p by auto
  subgoal
    apply (erule HOL.disjE)
  subgoal
    apply (rule disjI2)
    apply (rule disjI1)
    using D_σ_p by auto
  subgoal
    apply (rule disjI2)
    apply (rule disjI2)
    using D_σ_p by auto
  done
done
  then show C ∈ Liminf_llist (lmap grounding_of_state Sts)
    unfolding Liminf_llist_def clss_of_state_def grounding_of_clss_def by auto

```

qed

theorem *RP_sound*:

assumes $\{\#\} \in \text{cls_of_state } (\text{Liminf_state } \text{Sts})$
shows $\neg \text{satisfiable } (\text{grounding_of_state } (\text{lhd } \text{Sts}))$

proof –

from *assms* **have** $\{\#\} \in \text{grounding_of_state } (\text{Liminf_state } \text{Sts})$
unfolding *grounding_of_cls_def* **by** (*force intro: ex_ground_subst*)
then **have** $\{\#\} \in \text{Liminf_llist } (\text{lmap } \text{grounding_of_state } \text{Sts})$
using *grounding_of_state_Liminf_state_subseteq* **by** *auto*
then **have** $\neg \text{satisfiable } (\text{Liminf_llist } (\text{lmap } \text{grounding_of_state } \text{Sts}))$
using *true_cls_def* **by** *auto*
then **have** $\neg \text{satisfiable } (\text{lhd } (\text{lmap } \text{grounding_of_state } \text{Sts}))$
using *sr_ext.sat.deriv.Liminf_iff RP_ground.derive_chain* **by** *metis*
then **show** *?thesis*
unfolding *lhd_lmap_Sts* .

qed

lemma *ground_ord_resolve_ground*:

assumes
CAs_p: *gr_ord_resolve 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 –

have $a1: \text{atms_of } E \subseteq (\bigcup CA \in \text{set } CAs. \text{atms_of } CA) \cup \text{atms_of } DA$
using *gr_ord_resolve_atms_of_concl_subset[of CAs DA - - E]* *CAs_p* **by** *auto*
{
fix $L :: 'a \text{ literal}$
assume $L \in \# E$
then **have** $\text{atm_of } L \in \text{atms_of } E$
by (*meson atm_of_lit_in_atms_of*)
then **have** $\text{is_ground_atm } (\text{atm_of } L)$
using $a1$ *ground_cas ground_da is_ground_cls_imp_is_ground_atm is_ground_cls_list_def*
by *auto*
}
then **show** *?thesis*
unfolding *is_ground_cls_def is_ground_lit_def* **by** *simp*

qed

theorem *RP_saturated_if_fair*:

assumes *fair*: *fair_state_seq Sts*
shows *sr.saturated_upto* (*Liminf_llist* (*lmap* *grounding_of_state Sts*))

proof –

define $Gs :: 'a \text{ clause set llist}$ **where**
 $ns: Gs = \text{lmap } \text{grounding_of_state } \text{Sts}$

let $?N = \lambda i. \text{grounding_of_state } (\text{lnth } \text{Sts } i)$

let $?Ns = \lambda i. \text{N_of_state } (\text{lnth } \text{Sts } i)$

let $?Ps = \lambda i. \text{P_of_state } (\text{lnth } \text{Sts } i)$

let $?Qs = \lambda i. \text{Q_of_state } (\text{lnth } \text{Sts } i)$

have *ground_ns_in_ground_limit_st*:

Liminf_llist $Gs - \text{sr.Rf } (\text{Liminf_llist } Gs) \subseteq \text{grounding_of_cls } (\text{Q_of_state } (\text{Liminf_state } \text{Sts}))$
using *fair deriv fair_imp_Liminf_minus_Rf_subset_ground_Liminf_state ns* **by** *blast*

have *derivns*: *chain sr_ext.derive Gs*

using *RP_ground.derive_chain deriv ns* **by** *auto*

{
fix $\gamma :: 'a \text{ inference}$
assume $\gamma_p: \gamma \in \text{gr_ord } \Gamma$

```

let ?CC = side_prem_of  $\gamma$ 
let ?DA = main_prem_of  $\gamma$ 
let ?E = concl_of  $\gamma$ 
assume a: set_mset ?CC  $\cup$  {?DA}
   $\subseteq$  Liminf_llist (lmap grounding_of_state Sts) – sr.Rf (Liminf_llist (lmap grounding_of_state Sts))

have ground_ground_Liminf: is_ground_cls (Liminf_llist (lmap grounding_of_state Sts))
  using Liminf_grounding_of_state_ground unfolding is_ground_cls_def by auto

have ground_cc: is_ground_cls (set_mset ?CC)
  using a ground_ground_Liminf is_ground_cls_def by auto

have ground_da: is_ground_cls ?DA
  using a grounding_ground_singletonI ground_ground_Liminf
  by (simp add: Liminf_grounding_of_state_ground)

from  $\gamma$ -p obtain CAs AAs As where
  CAs_p: gr.ord_resolve CAs ?DA AAs As ?E  $\wedge$  mset CAs = ?CC
  unfolding gr.ord_def by auto

have DA_CAs_in_ground_Liminf:
  {?DA}  $\cup$  set CAs  $\subseteq$  grounding_of_cls (Q_of_state (Liminf_state Sts))
  using a CAs_p unfolding cls_of_state_def using fair unfolding fair_state_seq_def
  by (metis (no_types, lifting) Un.empty_left ground_ns_in_ground_limit_st a cls_of_state_def
    ns set_mset.mset subset_trans sup_commute)

then have ground_cas: is_ground_cls_list CAs
  using CAs_p unfolding is_ground_cls_list_def by auto

then have ground_e: is_ground_cls ?E
  using ground_ord_resolve_ground CAs_p ground_da by auto

have  $\exists$  AAs As  $\sigma$ . ord_resolve (S_M S (Q_of_state (Liminf_state Sts))) CAs ?DA AAs As  $\sigma$  ?E
  using CAs_p[THEN conjunct1]
proof (cases rule: gr.ord_resolve.cases)
  case (ord_resolve n Cs D)
  note DA = this(1) and e = this(2) and cas_len = this(3) and cs_len = this(4) and
    aas_len = this(5) and as_len = this(6) and nz = this(7) and cas = this(8) and
    aas_not_empty = this(9) and as_aas = this(10) and eligibility = this(11) and
    str_max = this(12) and sel_empty = this(13)

  have len_aas_len_as: length AAs = length As
    using aas_len as_len by auto

  from as_aas have  $\forall i < n$ .  $\forall A \in \#$  add_mset (As ! i) (AAs ! i). A = As ! i
    using ord_resolve by simp
  then have  $\forall i < n$ . card (set_mset (add_mset (As ! i) (AAs ! i)))  $\leq$  Suc 0
    using all_the_same by metis
  then have  $\forall i < \text{length AAs}$ . card (set_mset (add_mset (As ! i) (AAs ! i)))  $\leq$  Suc 0
    using aas_len by auto
  then have  $\forall AA \in \text{set (map2 add_mset As AAs)}$ . card (set_mset AA)  $\leq$  Suc 0
    using set_map2_ex[of AAs As add_mset, OF len_aas_len_as] by auto
  then have is_unifiers_id_subst (set_mset ' set (map2 add_mset As AAs))
    unfolding is_unifiers_def is_unifier_def by auto
  moreover have finite (set_mset ' set (map2 add_mset As AAs))
    by auto
  moreover have  $\forall AA \in \text{set_mset ' set (map2 add_mset As AAs)}$ . finite AA
    by auto
  ultimately obtain  $\sigma$  where
     $\sigma$ -p: Some  $\sigma$  = mgu (set_mset ' set (map2 add_mset As AAs))
    using mgu_complete by metis

  have ground_elig: gr.eligible As (D + negs (mset As))

```

```

using ord_resolve by simp
have ground_cs:  $\forall i < n. is\_ground\_cls (Cs ! i)$ 
  using ord_resolve(8) ord_resolve(3,4) ground_cas
  using ground_subclauses[of CAs Cs AAs] unfolding is_ground_cls_list_def by auto
have ground_set_as:  $is\_ground\_atms (set As)$ 
  using ord_resolve(1) ground_da
  by (metis atms_of_negs is_ground_cls_union set_mset_mset is_ground_cls_is_ground_atms_atms_of)
then have ground_mset_as:  $is\_ground\_atm\_mset (mset As)$ 
  unfolding is_ground_atm_mset_def is_ground_atms_def by auto
have ground_as:  $is\_ground\_atm\_list As$ 
  using ground_set_as is_ground_atm_list_def is_ground_atms_def by auto
have ground_d:  $is\_ground\_cls D$ 
  using ground_da ord_resolve by simp

from as.len nz have atms_of  $D \cup set As \neq \{\}$  finite (atms_of  $D \cup set As$ )
  by auto
then have Max (atms_of  $D \cup set As$ )  $\in atms\_of D \cup set As$ 
  using Max.in by metis
then have is_ground_Max:  $is\_ground\_atm (Max (atms\_of D \cup set As))$ 
  using ground_d ground_mset_as is_ground_cls_imp_is_ground_atm
  unfolding is_ground_atm_mset_def by auto
then have Max $\sigma$ _is_Max:  $\forall \sigma. Max (atms\_of D \cup set As) \cdot a \sigma = Max (atms\_of D \cup set As)$ 
  by auto

have ann1: maximal_wrt (Max (atms_of  $D \cup set As$ )) ( $D + negs (mset As)$ )
  unfolding maximal_wrt_def
  by clarsimp (metis Max_less_iff UnCI (atms_of  $D \cup set As \neq \{\}$ )
  (finite (atms_of  $D \cup set As$ )) ground_d ground_set_as infinite_growing is_ground_Max
  is_ground_atms_def is_ground_cls_imp_is_ground_atm less_atm_ground)

from ground_elig have ann2:
  Max (atms_of  $D \cup set As$ )  $\cdot a \sigma = Max (atms\_of D \cup set As)$ 
   $D \cdot \sigma + negs (mset As \cdot am \sigma) = D + negs (mset As)$ 
  using is_ground_Max ground_mset_as ground_d by auto

from ground_elig have fo_elig:
  eligible ( $S\_M S (Q\_of\_state (Liminf\_state Sts))$ )  $\sigma As (D + negs (mset As))$ 
  unfolding gr_eligible.simps eligible.simps gr_maximal_wrt_def using ann1 ann2
  by (auto simp: S_Q_def)

have l:  $\forall i < n. gr\_strictly\_maximal\_wrt (As ! i) (Cs ! i)$ 
  using ord_resolve by simp
then have  $\forall i < n. strictly\_maximal\_wrt (As ! i) (Cs ! i)$ 
  unfolding gr_strictly_maximal_wrt_def strictly_maximal_wrt_def
  using ground_as[unfolded is_ground_atm_list_def] ground_cs as_len less_atm_ground
  by clarsimp (fastforce simp: is_ground_cls_as_atms)+

then have ll:  $\forall i < n. strictly\_maximal\_wrt (As ! i \cdot a \sigma) (Cs ! i \cdot \sigma)$ 
  by (simp add: ground_as ground_cs as_len)

have m:  $\forall i < n. S\_Q (CAs ! i) = \{\#\}$ 
  using ord_resolve by simp

have ground_e:  $is\_ground\_cls (\bigcup \#mset Cs + D)$ 
  using ground_d ground_cs ground_e e by simp
show ?thesis
  using ord_resolve.intros[OF cas.len cs.len aas.len as_len nz cas aas_not_empty  $\sigma\_p$  fo_elig ll] m DA e ground_e
  unfolding S_Q_def by auto
qed
then obtain AAs As  $\sigma$  where
   $\sigma\_p$ : ord_resolve ( $S\_M S (Q\_of\_state (Liminf\_state Sts))$ ) CAs ?DA AAs As  $\sigma$  ?E
  by auto
then obtain  $\eta s' \eta' \eta 2' CAs' DA' AAs' As' \tau' E'$  where  $s\_p$ :

```



```

is_ground_subst η'
is_ground_subst_list ηs'
is_ground_subst η2'
ord_resolve_rename S CAs' DA' AAs' As' τ' E'
CAs' ..cl ηs' = CAs
DA' · η' = ?DA
E' · η2' = ?E
{DA'} ∪ set CAs' ⊆ Q_of_state (Liminf_state Sts)
using ord_resolve_rename_lifting[OF sel_stable, of Q_of_state (Liminf_state Sts) CAs ?DA]
σ_p selection_axioms DA.CAs_in_ground_Liminf by metis
from this(8) have ∃j. enat j < llength Sts ∧ (set CAs' ∪ {DA'} ⊆ ?Qs j)
unfolding Liminf_llist_def
using subseteq_Liminf_state_eventually_always[of {DA'} ∪ set CAs'] by auto
then obtain j where
j-p: is_least (λj. enat j < llength Sts ∧ set CAs' ∪ {DA'} ⊆ ?Qs j) j
using least_exists[of λj. enat j < llength Sts ∧ set CAs' ∪ {DA'} ⊆ ?Qs j] by force
then have j-p': enat j < llength Sts set CAs' ∪ {DA'} ⊆ ?Qs j
unfolding is_least_def by auto
then have jn0: j ≠ 0
using empty_Q0 by (metis bot_eq_sup_iff gr_implies_not_zero insert_not_empty llength_lnull
lnth_0_conv_lhd sup.orderE)
then have j_adds_CAs': ¬ set CAs' ∪ {DA'} ⊆ ?Qs (j - 1) set CAs' ∪ {DA'} ⊆ ?Qs j
using j-p unfolding is_least_def
apply (metis (no_types) One_nat_def Suc_diff_Suc Suc_ile_eq diff_diff_cancel diff_zero
less_imp_le less_one neq0_conv zero_less_diff)
using j-p'(2) by blast
have lnth Sts (j - 1) ~ lnth Sts j
using j-p'(1) jn0 deriv_chain_lnth_rel[of _ _ j - 1] by force
then obtain C' where C'-p:
?Ns (j - 1) = {}
?Ps (j - 1) = ?Ps j ∪ {C'}
?Qs j = ?Qs (j - 1) ∪ {C'}
?Ns j = concls_of (ord_FO_resolution.inferences_between (?Qs (j - 1)) C')
C' ∈ set CAs' ∪ {DA'}
C' ∉ ?Qs (j - 1)
using j_adds_CAs' by (induction rule: RP.cases) auto
have E' ∈ ?Ns j
proof -
have E' ∈ concls_of (ord_FO_resolution.inferences_between (Q_of_state (lnth Sts (j - 1))) C')
unfolding infer_from_def ord_FO_Γ_def unfolding inference_system.inferences_between_def
apply (rule_tac x = Infer (mset CAs') DA' E' in image_eqI)
subgoal by auto
subgoal
using s-p(4)
unfolding infer_from_def
apply (rule ord_resolve_rename.cases)
using s-p(4)
using C'-p(3) C'-p(5) j-p'(2) apply force
done
done
then show ?thesis
using C'-p(4) by auto
qed
then have E' ∈ class_of_state (lnth Sts j)
using j-p' unfolding class_of_state_def by auto
then have ?E ∈ grounding_of_state (lnth Sts j)
using s-p(7) s-p(3) unfolding grounding_of_class_def grounding_of_cls_def by force
then have γ ∈ sr.Ri (grounding_of_state (lnth Sts j))
using sr.Ri_effective γ-p by auto
then have γ ∈ sr_ext_Ri (?N j)
unfolding sr_ext_Ri_def by auto
then have γ ∈ sr_ext_Ri (Sup_llist (lmap grounding_of_state Sts))
using j-p' contra_subsetD llength_lmap lnth_lmap lnth_subset_Sup_llist sr_ext.Ri_mono by metis

```

```

then have  $\gamma \in sr\_ext\_Ri$  (Liminf_llist (lmap grounding_of_state Sts))
  using sr_ext.Ri_Sup_subset_Ri_Liminf[of Gs] derivns ns by blast
}
then have sr_ext.saturated_upto (Liminf_llist (lmap grounding_of_state Sts))
  unfolding sr_ext.saturated_upto_def sr_ext.inferences_from_def infer_from_def sr_ext.Ri_def
  by auto
then show ?thesis
  using gd_ord. $\Gamma$ _ngd_ord. $\Gamma$  sr.redundancy_criterion_axioms
  redundancy_criterion_standard_extension_saturated_upto_iff[of gr_ord. $\Gamma$ ]
  unfolding sr_ext.Ri_def by auto
qed

```

corollary *RP_complete_if_fair*:

```

assumes
  fair: fair_state_seq Sts and
  unsat:  $\neg$  satisfiable (grounding_of_state (lhd Sts))
shows  $\{\#\} \in Q\_of\_state$  (Liminf_state Sts)
proof -
  have  $\neg$  satisfiable (Liminf_llist (lmap grounding_of_state Sts))
    unfolding sr_ext.sat_deriv_Liminf_iff[OF RP_ground_derive_chain]
    by (rule unsat[folded lhd_lmap_Sts[of grounding_of_state]])
  moreover have sr.saturated_upto (Liminf_llist (lmap grounding_of_state Sts))
    by (rule RP_saturated_if_fair[OF fair, simplified])
  ultimately have  $\{\#\} \in Liminf_llist$  (lmap grounding_of_state Sts)
    using sr.saturated_upto_complete_if by auto
  then show ?thesis
    using empty_clause_in_Q_of_Liminf_state fair by auto
qed

```

end

end

end