A Verified Functional Implementation of Bachmair and Ganzinger's Ordered Resolution Prover

Anders Schlichtkrull, Jasmin Christian Blanchette, and Dmitriy Traytel

March 17, 2025

Abstract

This Isabelle/HOL formalization refines the abstract ordered resolution prover presented in Section 4.3 of Bachmair and Ganzinger's "Resolution Theorem Proving" chapter in the *Handbook of Automated Reasoning*. The result is a functional implementation of a first-order prover.

Contents

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 starts from an existing formalization of Bachmair and Ganzinger's chapter, up to and including Section 4.3. It refines the abstract ordered resolution prover presented in Section 4.3 to obtain an executable, functional implementation of a first-order prover. Figure 1 shows the corresponding Isabelle theory structure.

We refer to the following conference paper for details:

> Anders Schlichtkrull, Jasmin Christian Blanchette, Dmitriy Traytel:
> A verified prover based on ordered resolution.
> CPP 2019: 152-165
> http://matryoshka.gforge.inria.fr/pubs/fun_rp_paper.pdf

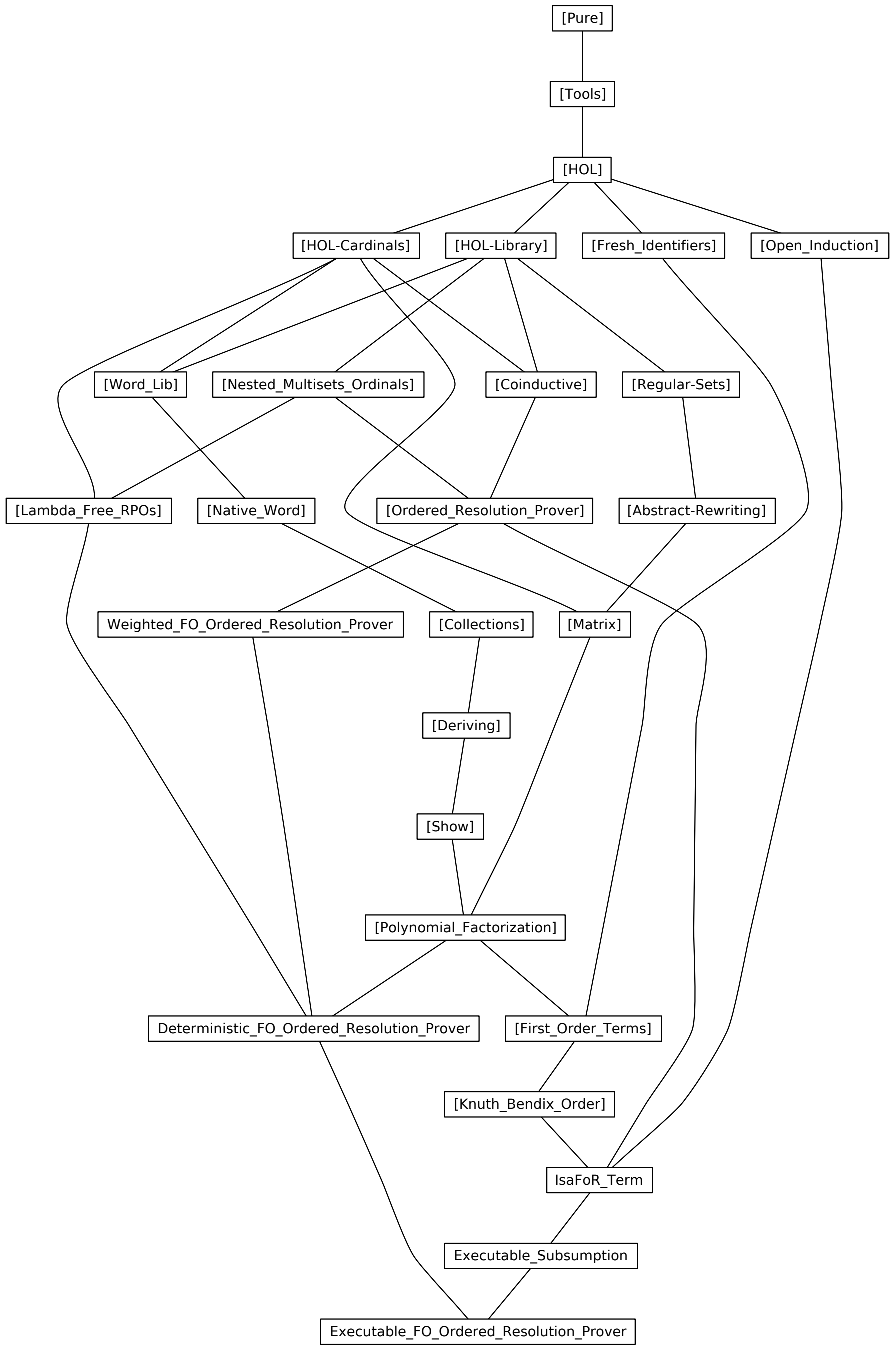

Figure 1: Theory dependency graph

2 A Fair Ordered Resolution Prover for First-Order Clauses with Weights

The *weighted_RP* prover introduced below operates on finite multisets of clauses and organizes the multiset of processed clauses as a priority queue to ensure that inferences are performed in a fair manner, to guarantee completeness.

theory *Weighted_FO_Ordered_Resolution_Prover*
  imports *Ordered_Resolution_Prover.FO_Ordered_Resolution_Prover*
begin

type-synonym *'a wclause* = *'a clause* × *nat*
type-synonym *'a wstate* = *'a wclause multiset* × *'a wclause multiset* × *'a wclause multiset* × *nat*

fun *state_of_wstate* :: *'a wstate* ⇒ *'a state* **where**
  state_of_wstate (*N*, *P*, *Q*, *n*) =
   (*set_mset* (*image_mset fst N*), *set_mset* (*image_mset fst P*), *set_mset* (*image_mset fst Q*))

locale *weighted_FO_resolution_prover* =
  FO_resolution_prover S subst_atm id_subst comp_subst renamings_apart atm_of_atms mgu less_atm
  for
    S :: (*'a* :: *wellorder*) *clause* ⇒ *'a clause* **and**
    subst_atm :: *'a* ⇒ *'s* ⇒ *'a* **and**
    id_subst :: *'s* **and**
    comp_subst :: *'s* ⇒ *'s* ⇒ *'s* **and**
    renamings_apart :: *'a clause list* ⇒ *'s list* **and**
    atm_of_atms :: *'a list* ⇒ *'a* **and**
    mgu :: *'a set set* ⇒ *'s option* **and**
    less_atm :: *'a* ⇒ *'a* ⇒ *bool* +
  fixes
    weight :: *'a clause* × *nat* ⇒ *nat*
  assumes
    weight_mono: *i* < *j* ⟹ *weight* (*C*, *i*) < *weight* (*C*, *j*)
begin

abbreviation *clss_of_wstate* :: *'a wstate* ⇒ *'a clause set* **where**
  clss_of_wstate St ≡ *clss_of_state* (*state_of_wstate St*)

abbreviation *N_of_wstate* :: *'a wstate* ⇒ *'a clause set* **where**
  N_of_wstate St ≡ *N_of_state* (*state_of_wstate St*)

abbreviation *P_of_wstate* :: *'a wstate* ⇒ *'a clause set* **where**
  P_of_wstate St ≡ *P_of_state* (*state_of_wstate St*)

abbreviation *Q_of_wstate* :: *'a wstate* ⇒ *'a clause set* **where**
  Q_of_wstate St ≡ *Q_of_state* (*state_of_wstate St*)

fun *wN_of_wstate* :: *'a wstate* ⇒ *'a wclause multiset* **where**
  wN_of_wstate (*N*, *P*, *Q*, *n*) = *N*

fun *wP_of_wstate* :: *'a wstate* ⇒ *'a wclause multiset* **where**
  wP_of_wstate (*N*, *P*, *Q*, *n*) = *P*

fun *wQ_of_wstate* :: *'a wstate* ⇒ *'a wclause multiset* **where**
  wQ_of_wstate (*N*, *P*, *Q*, *n*) = *Q*

fun *n_of_wstate* :: *'a wstate* ⇒ *nat* **where**
  n_of_wstate (*N*, *P*, *Q*, *n*) = *n*

lemma *of_wstate_split*[*simp*]:
  (*wN_of_wstate St*, *wP_of_wstate St*, *wQ_of_wstate St*, *n_of_wstate St*) = *St*
  ⟨*proof*⟩

abbreviation *grounding_of_wstate* :: *'a wstate* ⇒ *'a clause set* **where**
grounding_of_wstate St ≡ *grounding_of_state* (*state_of_wstate St*)

abbreviation *Liminf_wstate* :: *'a wstate llist* ⇒ *'a state* **where**
Liminf_wstate Sts ≡ *Liminf_state* (*lmap state_of_wstate Sts*)

lemma *timestamp_le_weight*: *n* ≤ *weight* (*C*, *n*)
⟨*proof*⟩

inductive *weighted_RP* :: *'a wstate* ⇒ *'a wstate* ⇒ *bool* (**infix** ‹$\leadsto_w$› *50*) **where**
tautology_deletion: *Neg A* ∈# *C* ⟹ *Pos A* ∈# *C* ⟹ (*N* + {#(*C*, *i*)#}, *P*, *Q*, *n*) $\leadsto_w$ (*N*, *P*, *Q*, *n*)
| *forward_subsumption*: *D* ∈# *image_mset fst* (*P* + *Q*) ⟹ *subsumes D C* ⟹
(*N* + {#(*C*, *i*)#}, *P*, *Q*, *n*) $\leadsto_w$ (*N*, *P*, *Q*, *n*)
| *backward_subsumption_P*: *D* ∈# *image_mset fst N* ⟹ *C* ∈# *image_mset fst P* ⟹
strictly_subsumes D C ⟹ (*N*, *P*, *Q*, *n*) $\leadsto_w$ (*N*, {#(*E*, *k*) ∈# *P*. *E* ≠ *C*#}, *Q*, *n*)
| *backward_subsumption_Q*: *D* ∈# *image_mset fst N* ⟹ *strictly_subsumes D C* ⟹
(*N*, *P*, *Q* + {#(*C*, *i*)#}, *n*) $\leadsto_w$ (*N*, *P*, *Q*, *n*)
| *forward_reduction*: *D* + {#*L'*#} ∈# *image_mset fst* (*P* + *Q*) ⟹ − *L* = *L'* ·*l* *σ* ⟹ *D* · *σ* ⊆# *C* ⟹
(*N* + {#(*C* + {#*L*#}, *i*)#}, *P*, *Q*, *n*) $\leadsto_w$ (*N* + {#(*C*, *i*)#}, *P*, *Q*, *n*)
| *backward_reduction_P*: *D* + {#*L'*#} ∈# *image_mset fst N* ⟹ − *L* = *L'* ·*l* *σ* ⟹ *D* · *σ* ⊆# *C* ⟹
(∀ *j*. (*C* + {#*L*#}, *j*) ∈# *P* ⟶ *j* ≤ *i*) ⟹
(*N*, *P* + {#(*C* + {#*L*#}, *i*)#}, *Q*, *n*) $\leadsto_w$ (*N*, *P* + {#(*C*, *i*)#}, *Q*, *n*)
| *backward_reduction_Q*: *D* + {#*L'*#} ∈# *image_mset fst N* ⟹ − *L* = *L'* ·*l* *σ* ⟹ *D* · *σ* ⊆# *C* ⟹
(*N*, *P*, *Q* + {#(*C* + {#*L*#}, *i*)#}, *n*) $\leadsto_w$ (*N*, *P* + {#(*C*, *i*)#}, *Q*, *n*)
| *clause_processing*: (*N* + {#(*C*, *i*)#}, *P*, *Q*, *n*) $\leadsto_w$ (*N*, *P* + {#(*C*, *i*)#}, *Q*, *n*)
| *inference_computation*: (∀ (*D*, *j*) ∈# *P*. *weight* (*C*, *i*) ≤ *weight* (*D*, *j*)) ⟹
N = *mset_set* ((λ*D*. (*D*, *n*)) ‘ *concls_of*
(*inference_system.inferences_between* (*ord_FO_*Γ *S*) (*set_mset* (*image_mset fst Q*)) *C*)) ⟹
({#}, *P* + {#(*C*, *i*)#}, *Q*, *n*) $\leadsto_w$ (*N*, {#(*D*, *j*) ∈# *P*. *D* ≠ *C*#}, *Q* + {#(*C*, *i*)#}, *Suc n*)

lemma *weighted_RP_imp_RP*: *St* $\leadsto_w$ *St'* ⟹ *state_of_wstate St* ⇝ *state_of_wstate St'*
⟨*proof*⟩

lemma *final_weighted_RP*: ¬ ({#}, {#}, *Q*, *n*) $\leadsto_w$ *St*
⟨*proof*⟩

context
fixes
Sts :: *'a wstate llist*
assumes
full_deriv: *full_chain* ($\leadsto_w$) *Sts* **and**
empty_P0: *P_of_wstate* (*lhd Sts*) = {} **and**
empty_Q0: *Q_of_wstate* (*lhd Sts*) = {}
begin

lemma *finite_Sts0*: *finite* (*clss_of_wstate* (*lhd Sts*))
⟨*proof*⟩

lemmas *deriv* = *full_chain_imp_chain*[*OF full_deriv*]
lemmas *lhd_lmap_Sts* = *llist.map_sel*(*1*)[*OF chain_not_lnull*[*OF deriv*]]

lemma *deriv_RP*: *chain* (⇝) (*lmap state_of_wstate Sts*)
⟨*proof*⟩

lemma *finite_Sts0_RP*: *finite* (*clss_of_state* (*lhd* (*lmap state_of_wstate Sts*)))
⟨*proof*⟩

lemma *empty_P0_RP*: *P_of_state* (*lhd* (*lmap state_of_wstate Sts*)) = {}
⟨*proof*⟩

lemma *empty_Q0_RP*: *Q_of_state* (*lhd* (*lmap state_of_wstate Sts*)) = {}
⟨*proof*⟩

lemmas *Sts_thms = deriv_RP finite_Sts0_RP empty_P0_RP empty_Q0_RP*

theorem *weighted_RP_model*:
  $St \leadsto_w St' \Longrightarrow I \models s$ *grounding_of_wstate* $St' \longleftrightarrow I \models s$ *grounding_of_wstate St*
  ⟨*proof*⟩

abbreviation *S_gQ* :: *'a clause* ⇒ *'a clause* **where**
  S_gQ ≡ *S_Q* (*lmap state_of_wstate Sts*)

interpretation *sq*: *selection S_gQ*
  ⟨*proof*⟩

interpretation *gd*: *ground_resolution_with_selection S_gQ*
  ⟨*proof*⟩

interpretation *src*: *standard_redundancy_criterion_reductive gd.ord_Γ*
  ⟨*proof*⟩

interpretation *src*: *standard_redundancy_criterion_counterex_reducing gd.ord_Γ*
  ground_resolution_with_selection.INTERP S_gQ
  ⟨*proof*⟩

lemmas *ord_Γ_saturated_upto_def* = *src.saturated_upto_def*
lemmas *ord_Γ_saturated_upto_complete* = *src.saturated_upto_complete*
lemmas *ord_Γ_contradiction_Rf* = *src.contradiction_Rf*

theorem *weighted_RP_sound*:
  assumes {#} ∈ *clss_of_state* (*Liminf_wstate Sts*)
  shows ¬ *satisfiable* (*grounding_of_wstate* (*lhd Sts*))
  ⟨*proof*⟩

abbreviation *RP_filtered_measure* :: (*'a wclause* ⇒ *bool*) ⇒ *'a wstate* ⇒ *nat* × *nat* × *nat* **where**
  RP_filtered_measure ≡ λ*p* (*N*, *P*, *Q*, *n*).
    (*sum_mset* (*image_mset* (λ(*C*, *i*). *Suc* (*size C*)) {#*Di* ∈# *N* + *P* + *Q*. *p Di*#}),
     size {#*Di* ∈# *N*. *p Di*#}, *size* {#*Di* ∈# *P*. *p Di*#})

abbreviation *RP_combined_measure* :: *nat* ⇒ *'a wstate* ⇒ *nat* × (*nat* × *nat* × *nat*) × (*nat* × *nat* × *nat*) **where**
  RP_combined_measure ≡ λ*w St*.
    (*w* + *1* − *n_of_wstate St*, *RP_filtered_measure* (λ(*C*, *i*). *i* ≤ *w*) *St*,
     RP_filtered_measure (λ*Ci*. *True*) *St*)

abbreviation (*input*) *RP_filtered_relation* :: ((*nat* × *nat* × *nat*) × (*nat* × *nat* × *nat*)) *set* **where**
  RP_filtered_relation ≡ *natLess* <*lex*> *natLess* <*lex*> *natLess*

abbreviation (*input*) *RP_combined_relation* :: ((*nat* × ((*nat* × *nat* × *nat*) × (*nat* × *nat* × *nat*))) ×
    (*nat* × ((*nat* × *nat* × *nat*) × (*nat* × *nat* × *nat*)))) *set* **where**
  RP_combined_relation ≡ *natLess* <*lex*> *RP_filtered_relation* <*lex*> *RP_filtered_relation*

abbreviation (*fst3* :: *'b* ∗ *'c* ∗ *'d* ⇒ *'b*) ≡ *fst*
abbreviation (*snd3* :: *'b* ∗ *'c* ∗ *'d* ⇒ *'c*) ≡ λ*x*. *fst* (*snd x*)
abbreviation (*trd3* :: *'b* ∗ *'c* ∗ *'d* ⇒ *'d*) ≡ λ*x*. *snd* (*snd x*)

lemma
  wf_RP_filtered_relation: *wf RP_filtered_relation* **and**
  wf_RP_combined_relation: *wf RP_combined_relation*
  ⟨*proof*⟩

lemma *multiset_sum_of_Suc_f_monotone*: $N \subset\# M \Longrightarrow (\sum x \in\# N.\ Suc\ (f\ x)) < (\sum x \in\# M.\ Suc\ (f\ x))$
⟨*proof*⟩

lemma *multiset_sum_monotone_f'*:
  assumes $CC \subset\# DD$
  shows $(\sum (C, i) \in\# CC.\ Suc\ (f\ C)) < (\sum (C, i) \in\# DD.\ Suc\ (f\ C))$

⟨*proof*⟩

lemma *filter_mset_strict_subset*:
assumes $x \in\# M$ **and** $\neg\ p\ x$
shows $\{\#y \in\# M.\ p\ y\#\} \subset\# M$
⟨*proof*⟩

lemma *weighted_RP_measure_decreasing_N*:
assumes $St \leadsto_w St'$ **and** $(C, l) \in\#$ *wN_of_wstate St*
shows (*RP_filtered_measure* ($\lambda Ci.$ *True*) St', *RP_filtered_measure* ($\lambda Ci.$ *True*) *St*)
$\in$ *RP_filtered_relation*
⟨*proof*⟩

lemma *weighted_RP_measure_decreasing_P*:
assumes $St \leadsto_w St'$ **and** $(C, i) \in\#$ *wP_of_wstate St*
shows (*RP_combined_measure* (*weight* (C, i)) St', *RP_combined_measure* (*weight* (C, i)) *St*)
$\in$ *RP_combined_relation*
⟨*proof*⟩

lemma *preserve_min_or_delete_completely*:
assumes $St \leadsto_w St'$ $(C, i) \in\#$ *wP_of_wstate St*
$\forall k.\ (C, k) \in\#$ *wP_of_wstate St* $\longrightarrow i \le k$
shows $(C, i) \in\#$ *wP_of_wstate* St' $\vee$ ($\forall j.\ (C, j) \notin\#$ *wP_of_wstate* St')
⟨*proof*⟩

lemma *preserve_min_P*:
assumes
$St \leadsto_w St'$ $(C, j) \in\#$ *wP_of_wstate* St' **and**
$(C, i) \in\#$ *wP_of_wstate St* **and**
$\forall k.\ (C, k) \in\#$ *wP_of_wstate St* $\longrightarrow i \le k$
shows $(C, i) \in\#$ *wP_of_wstate* St'
⟨*proof*⟩

lemma *preserve_min_P_Sts*:
assumes
enat (*Suc k*) $<$ *llength Sts* **and**
$(C, i) \in\#$ *wP_of_wstate* (*lnth Sts k*) **and**
$(C, j) \in\#$ *wP_of_wstate* (*lnth Sts* (*Suc k*)) **and**
$\forall j.\ (C, j) \in\#$ *wP_of_wstate* (*lnth Sts k*) $\longrightarrow i \le j$
shows $(C, i) \in\#$ *wP_of_wstate* (*lnth Sts* (*Suc k*))
⟨*proof*⟩

lemma *in_lnth_in_Supremum_ldrop*:
assumes $i <$ *llength xs* **and** $x \in\#$ (*lnth xs i*)
shows $x \in$ *Sup_llist* (*lmap set_mset* (*ldrop* (*enat i*) *xs*))
⟨*proof*⟩

lemma *persistent_wclause_in_P_if_persistent_clause_in_P*:
assumes $C \in$ *Liminf_llist* (*lmap P_of_state* (*lmap state_of_wstate Sts*))
shows $\exists i.\ (C, i) \in$ *Liminf_llist* (*lmap* (*set_mset* $\circ$ *wP_of_wstate*) *Sts*)
⟨*proof*⟩

lemma *lfinite_not_LNil_nth_llast*:
assumes *lfinite Sts* **and** *Sts* $\neq$ *LNil*
shows $\exists i <$ *llength Sts*. *lnth Sts i* $=$ *llast Sts* $\wedge$ ($\forall j <$ *llength Sts*. $j \le i$)
⟨*proof*⟩

lemma *fair_if_finite*:
assumes *fin*: *lfinite Sts*
shows *fair_state_seq* (*lmap state_of_wstate Sts*)
⟨*proof*⟩

lemma *N_of_state_state_of_wstate_wN_of_wstate*:

assumes $C \in N\_of\_state\ (state\_of\_wstate\ St)$
shows $\exists\, i.\ (C,\ i) \in\#\ wN\_of\_wstate\ St$
⟨*proof*⟩

lemma *in_wN_of_wstate_in_N_of_wstate*: $(C,\ i) \in\#\ wN\_of\_wstate\ St \Longrightarrow C \in N\_of\_wstate\ St$
⟨*proof*⟩

lemma *in_wP_of_wstate_in_P_of_wstate*: $(C,\ i) \in\#\ wP\_of\_wstate\ St \Longrightarrow C \in P\_of\_wstate\ St$
⟨*proof*⟩

lemma *in_wQ_of_wstate_in_Q_of_wstate*: $(C,\ i) \in\#\ wQ\_of\_wstate\ St \Longrightarrow C \in Q\_of\_wstate\ St$
⟨*proof*⟩

lemma *n_of_wstate_weighted_RP_increasing*: $St \leadsto_w St' \Longrightarrow n\_of\_wstate\ St \leq n\_of\_wstate\ St'$
⟨*proof*⟩

lemma *nth_of_wstate_monotonic*:
assumes $j < llength\ Sts$ **and** $i \leq j$
shows $n\_of\_wstate\ (lnth\ Sts\ i) \leq n\_of\_wstate\ (lnth\ Sts\ j)$
⟨*proof*⟩

lemma *infinite_chain_relation_measure*:
assumes
measure_decreasing: $\bigwedge St\ St'.\ P\ St \Longrightarrow R\ St\ St' \Longrightarrow (m\ St',\ m\ St) \in mR$ **and**
non_infer_chain: $chain\ R\ (ldrop\ (enat\ k)\ Sts)$ **and**
inf: $llength\ Sts = \infty$ **and**
P: $\bigwedge i.\ P\ (lnth\ (ldrop\ (enat\ k)\ Sts)\ i)$
shows $chain\ (\lambda x\ y.\ (x,\ y) \in mR)^{-1-1}\ (lmap\ m\ (ldrop\ (enat\ k)\ Sts))$
⟨*proof*⟩

theorem *weighted_RP_fair*: $fair\_state\_seq\ (lmap\ state\_of\_wstate\ Sts)$
⟨*proof*⟩

corollary *weighted_RP_saturated*: $src.saturated\_upto\ (Liminf\_llist\ (lmap\ grounding\_of\_wstate\ Sts))$
⟨*proof*⟩

corollary *weighted_RP_complete*:
$\neg\ satisfiable\ (grounding\_of\_wstate\ (lhd\ Sts)) \Longrightarrow \{\#\} \in Q\_of\_state\ (Liminf\_wstate\ Sts)$
⟨*proof*⟩

end

end

locale *weighted_FO_resolution_prover_with_size_timestamp_factors* =
FO_resolution_prover S subst_atm id_subst comp_subst renamings_apart atm_of_atms mgu less_atm
for
$S :: ('a :: wellorder)\ clause \Rightarrow 'a\ clause$ **and**
$subst\_atm :: 'a \Rightarrow 's \Rightarrow 'a$ **and**
$id\_subst :: 's$ **and**
$comp\_subst :: 's \Rightarrow 's \Rightarrow 's$ **and**
$renamings\_apart :: 'a\ literal\ multiset\ list \Rightarrow 's\ list$ **and**
$atm\_of\_atms :: 'a\ list \Rightarrow 'a$ **and**
$mgu :: 'a\ set\ set \Rightarrow 's\ option$ **and**
$less\_atm :: 'a \Rightarrow 'a \Rightarrow bool$ +
fixes
$size\_atm :: 'a \Rightarrow nat$ **and**
$size\_factor :: nat$ **and**
$timestamp\_factor :: nat$
assumes
timestamp_factor_pos: $timestamp\_factor > 0$
begin

fun *weight* :: *'a wclause* ⇒ *nat* **where**
weight $(C,\ i)$ = *size_factor* ∗ *size_multiset* (*size_literal size_atm*) C + *timestamp_factor* ∗ i

lemma *weight_mono*: $i < j \Longrightarrow$ *weight* $(C,\ i) <$ *weight* $(C,\ j)$
⟨*proof*⟩

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

sublocale *wrp*: *weighted_FO_resolution_prover* _ _ _ _ _ _ _ _ _ _ *weight*
⟨*proof*⟩

notation *wrp.weighted_RP* (**infix** ‹$\leadsto_w$› *50*)

end

end

3 A Deterministic Ordered Resolution Prover for First-Order Clauses

The *deterministic_RP* prover introduced below is a deterministic program that works on finite lists, committing to a strategy for assigning priorities to clauses. However, it is not fully executable: It abstracts over operations on atoms and employs logical specifications instead of executable functions for auxiliary notions.

theory *Deterministic_FO_Ordered_Resolution_Prover*
imports
Polynomial_Factorization.Missing_List
Weighted_FO_Ordered_Resolution_Prover
Lambda_Free_RPOs.Lambda_Free_Util
begin

3.1 Library

lemma *apfst_fst_snd*: *apfst f x* = (*f* (*fst x*), *snd x*)
⟨*proof*⟩

lemma *apfst_comp_rpair_const*: *apfst f* ∘ (λ*x*. (*x*, *y*)) = (λ*x*. (*x*, *y*)) ∘ *f*
⟨*proof*⟩

lemma *length_remove1_less*[*termination_simp*]: *x* ∈ *set xs* ⟹ *length* (*remove1 x xs*) < *length xs*
⟨*proof*⟩

lemma *map_filter_neq_eq_filter_map*:
map f (*filter* (λ*y*. *f x* ≠ *f y*) *xs*) = *filter* (λ*z*. *f x* ≠ *z*) (*map f xs*)
⟨*proof*⟩

lemma *mset_map_remdups_gen*:
mset (*map f* (*remdups_gen f xs*)) = *mset* (*remdups_gen* (λ*x*. *x*) (*map f xs*))
⟨*proof*⟩

lemma *mset_remdups_gen_ident*: *mset* (*remdups_gen* (λ*x*. *x*) *xs*) = *mset_set* (*set xs*)
⟨*proof*⟩

lemma *funpow_fixpoint*: *f x* = *x* ⟹ (*f* ^^ *n*) *x* = *x*
⟨*proof*⟩

lemma *rtranclp_imp_eq_image*: (∀ *x y*. *R x y* ⟶ *f x* = *f y*) ⟹ R^{**} *x y* ⟹ *f x* = *f y*
⟨*proof*⟩

lemma *tranclp_imp_eq_image*: (∀ *x y*. *R x y* ⟶ *f x* = *f y*) ⟹ R^{++} *x y* ⟹ *f x* = *f y*
⟨*proof*⟩

3.2 Prover

type-synonym *'a lclause* = *'a literal list*
type-synonym *'a dclause* = *'a lclause* × *nat*
type-synonym *'a dstate* = *'a dclause list* × *'a dclause list* × *'a dclause list* × *nat*

locale *deterministic_FO_resolution_prover* =
weighted_FO_resolution_prover_with_size_timestamp_factors S subst_atm id_subst comp_subst renamings_apart atm_of_atms mgu less_atm size_atm timestamp_factor size_factor
for
S :: (*'a* :: *wellorder*) *clause* ⇒ *'a clause* **and**
subst_atm :: *'a* ⇒ *'s* ⇒ *'a* **and**
id_subst :: *'s* **and**
comp_subst :: *'s* ⇒ *'s* ⇒ *'s* **and**
renamings_apart :: *'a literal multiset list* ⇒ *'s list* **and**
atm_of_atms :: *'a list* ⇒ *'a* **and**
mgu :: *'a set set* ⇒ *'s option* **and**
less_atm :: *'a* ⇒ *'a* ⇒ *bool* **and**
size_atm :: *'a* ⇒ *nat* **and**
timestamp_factor :: *nat* **and**
size_factor :: *nat* +
assumes
S_empty: *S C* = {#}
begin

lemma *less_atm_irrefl*: ¬ *less_atm A A*
⟨*proof*⟩

fun *wstate_of_dstate* :: *'a dstate* ⇒ *'a wstate* **where**
wstate_of_dstate (*N*, *P*, *Q*, *n*) =
(*mset* (*map* (*apfst mset*) *N*), *mset* (*map* (*apfst mset*) *P*), *mset* (*map* (*apfst mset*) *Q*), *n*)

fun *state_of_dstate* :: *'a dstate* ⇒ *'a state* **where**
state_of_dstate (*N*, *P*, *Q*, _) =
(*set* (*map* (*mset* ∘ *fst*) *N*), *set* (*map* (*mset* ∘ *fst*) *P*), *set* (*map* (*mset* ∘ *fst*) *Q*))

abbreviation *clss_of_dstate* :: *'a dstate* ⇒ *'a clause set* **where**
clss_of_dstate St ≡ *clss_of_state* (*state_of_dstate St*)

fun *is_final_dstate* :: *'a dstate* ⇒ *bool* **where**
is_final_dstate (*N*, *P*, *Q*, *n*) ⟷ *N* = [] ∧ *P* = []

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

abbreviation *rtrancl_weighted_RP* (**infix** ‹$\leadsto_w{}^*$› *50*) **where**
($\leadsto_w{}^*$) ≡ ($\leadsto_w$)**

abbreviation *trancl_weighted_RP* (**infix** ‹$\leadsto_w{}^+$› *50*) **where**
($\leadsto_w{}^+$) ≡ ($\leadsto_w$)$^{++}$

definition *is_tautology* :: *'a lclause* ⇒ *bool* **where**
is_tautology C ⟷ (∃ *A* ∈ *set* (*map atm_of C*). *Pos A* ∈ *set C* ∧ *Neg A* ∈ *set C*)

definition *subsume* :: *'a lclause list* ⇒ *'a lclause* ⇒ *bool* **where**
subsume Ds C ⟷ (∃ *D* ∈ *set Ds*. *subsumes* (*mset D*) (*mset C*))

definition *strictly_subsume* :: *'a lclause list* ⇒ *'a lclause* ⇒ *bool* **where**
strictly_subsume Ds C ⟷ (∃ *D* ∈ *set Ds*. *strictly_subsumes* (*mset D*) (*mset C*))

definition *is_reducible_on* :: *'a literal* ⇒ *'a lclause* ⇒ *'a literal* ⇒ *'a lclause* ⇒ *bool* **where**
is_reducible_on M D L C ⟷ *subsumes* (*mset D* + {#− *M*#}) (*mset C* + {#*L*#})

definition *is_reducible_lit* :: *'a lclause list* ⇒ *'a lclause* ⇒ *'a literal* ⇒ *bool* **where**
is_reducible_lit Ds C L ⟷

$(\exists\, D \in set\ Ds.\ \exists\, L' \in set\ D.\ \exists\, \sigma.\ -\ L = L' \cdot l\ \sigma \wedge mset\ (remove1\ L'\ D) \cdot \sigma \subseteq\#\ mset\ C)$

primrec *reduce* :: *'a lclause list* ⇒ *'a lclause* ⇒ *'a lclause* ⇒ *'a lclause* **where**
reduce _ _ [] = []
| *reduce Ds C* (*L* # *C'*) =
(*if is_reducible_lit Ds* (*C* @ *C'*) *L then reduce Ds C C' else L* # *reduce Ds* (*L* # *C*) *C'*)

abbreviation *is_irreducible* :: *'a lclause list* ⇒ *'a lclause* ⇒ *bool* **where**
is_irreducible Ds C ≡ *reduce Ds* [] *C* = *C*

abbreviation *is_reducible* :: *'a lclause list* ⇒ *'a lclause* ⇒ *bool* **where**
is_reducible Ds C ≡ *reduce Ds* [] *C* ≠ *C*

definition *reduce_all* :: *'a lclause* ⇒ *'a dclause list* ⇒ *'a dclause list* **where**
reduce_all D = *map* (*apfst* (*reduce* [*D*] []))

fun *reduce_all2* :: *'a lclause* ⇒ *'a dclause list* ⇒ *'a dclause list* × *'a dclause list* **where**
reduce_all2 _ [] = ([], [])
| *reduce_all2 D* (*Ci* # *Cs*) =
(*let*
(*C*, *i*) = *Ci*;
C' = *reduce* [*D*] [] *C*
in
(*if C'* = *C then apsnd else apfst*) (*Cons* (*C'*, *i*)) (*reduce_all2 D Cs*))

fun *remove_all* :: *'b list* ⇒ *'b list* ⇒ *'b list* **where**
remove_all xs [] = *xs*
| *remove_all xs* (*y* # *ys*) = (*if y* ∈ *set xs then remove_all* (*remove1 y xs*) *ys else remove_all xs ys*)

lemma *remove_all_mset_minus*: *mset ys* ⊆# *mset xs* ⟹ *mset* (*remove_all xs ys*) = *mset xs* − *mset ys*
⟨*proof*⟩

definition *resolvent* :: *'a lclause* ⇒ *'a* ⇒*'a lclause* ⇒ *'a lclause* ⇒ *'a lclause* **where**
resolvent D A CA Ls =
map (λ*M*. *M* ·*l* (*the* (*mgu* {*insert A* (*atms_of* (*mset Ls*))}))) (*remove_all CA Ls* @ *D*)

definition *resolvable* :: *'a* ⇒ *'a lclause* ⇒ *'a lclause* ⇒ *'a lclause* ⇒ *bool* **where**
resolvable A D CA Ls ⟷
(*let* σ = (*mgu* {*insert A* (*atms_of* (*mset Ls*))}) *in*
σ ≠ *None*
∧ *Ls* ≠ []
∧ *maximal_wrt* (*A* ·*a the* σ) ((*add_mset* (*Neg A*) (*mset D*)) · *the* σ)
∧ *strictly_maximal_wrt* (*A* ·*a the* σ) ((*mset CA* − *mset Ls*) · *the* σ)
∧ (∀ *L* ∈ *set Ls*. *is_pos L*))

definition *resolve_on* :: *'a* ⇒ *'a lclause* ⇒ *'a lclause* ⇒ *'a lclause list* **where**
resolve_on A D CA = *map* (*resolvent D A CA*) (*filter* (*resolvable A D CA*) (*subseqs CA*))

definition *resolve* :: *'a lclause* ⇒ *'a lclause* ⇒ *'a lclause list* **where**
resolve C D =
concat (*map* (λ*L*.
(*case L of*
Pos A ⇒ []
| *Neg A* ⇒
if maximal_wrt A (*mset D*) *then*
resolve_on A (*remove1 L D*) *C*
else
[])) *D*)

definition *resolve_rename* :: *'a lclause* ⇒ *'a lclause* ⇒ *'a lclause list* **where**
resolve_rename C D =
(*let* σ*s* = *renamings_apart* [*mset D*, *mset C*] *in*
resolve (*map* (λ*L*. *L* ·*l last* σ*s*) *C*) (*map* (λ*L*. *L* ·*l hd* σ*s*) *D*))

```
definition resolve_rename_either_way :: 'a lclause ⇒ 'a lclause ⇒ 'a lclause list where
  resolve_rename_either_way C D = resolve_rename C D @ resolve_rename D C

fun select_min_weight_clause :: 'a dclause ⇒ 'a dclause list ⇒ 'a dclause where
  select_min_weight_clause Ci [] = Ci
| select_min_weight_clause Ci (Dj # Djs) =
   select_min_weight_clause
     (if weight (apfst mset Dj) < weight (apfst mset Ci) then Dj else Ci) Djs

lemma select_min_weight_clause_in: select_min_weight_clause P0 P ∈ set (P0 # P)
  ⟨proof⟩

function remdups_clss :: 'a dclause list ⇒ 'a dclause list where
  remdups_clss [] = []
| remdups_clss (Ci # Cis) =
   (let
      Ci' = select_min_weight_clause Ci Cis
    in
      Ci' # remdups_clss (filter (λ(D, _). mset D ≠ mset (fst Ci')) (Ci # Cis)))
  ⟨proof⟩
  termination
    ⟨proof⟩

declare remdups_clss.simps(2) [simp del]

fun deterministic_RP_step :: 'a dstate ⇒ 'a dstate where
  deterministic_RP_step (N, P, Q, n) =
   (if ∃ Ci ∈ set (P @ Q). fst Ci = [] then
      ([], [], remdups_clss P @ Q, n + length (remdups_clss P))
    else
      (case N of
        [] ⇒
        (case P of
           [] ⇒ (N, P, Q, n)
         | P0 # P' ⇒
           let
             (C, i) = select_min_weight_clause P0 P';
             N = map (λD. (D, n)) (remdups_gen mset (resolve_rename C C
               @ concat (map (resolve_rename_either_way C ∘ fst) Q)));
             P = filter (λ(D, j). mset D ≠ mset C) P;
             Q = (C, i) # Q;
             n = Suc n
           in
             (N, P, Q, n))
      | (C, i) # N ⇒
        let
          C = reduce (map fst (P @ Q)) [] C
        in
          if C = [] then
            ([], [], [([], i)], Suc n)
          else if is_tautology C ∨ subsume (map fst (P @ Q)) C then
            (N, P, Q, n)
          else
            let
              P = reduce_all C P;
              (back_to_P, Q) = reduce_all2 C Q;
              P = back_to_P @ P;
              Q = filter (Not ∘ strictly_subsume [C] ∘ fst) Q;
              P = filter (Not ∘ strictly_subsume [C] ∘ fst) P;
              P = (C, i) # P
            in
              (N, P, Q, n)))
```

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

partial-function (*option*) *deterministic_RP* :: *'a dstate* ⇒ *'a lclause list option* **where**
deterministic_RP St =
(*if is_final_dstate St then*
let (_, _, *Q*, _) = *St in Some* (*map fst Q*)
else
deterministic_RP (*deterministic_RP_step St*))

lemma *is_final_dstate_imp_not_weighted_RP*: *is_final_dstate St* ⟹ ¬ *wstate_of_dstate St* $\leadsto_w$ *St'*
⟨*proof*⟩

lemma *is_final_dstate_funpow_imp_deterministic_RP_neq_None*:
is_final_dstate ((*deterministic_RP_step* ^^ *k*) *St*) ⟹ *deterministic_RP St* ≠ *None*
⟨*proof*⟩

lemma *is_reducible_lit_mono_cls*:
mset C ⊆# *mset C'* ⟹ *is_reducible_lit Ds C L* ⟹ *is_reducible_lit Ds C' L*
⟨*proof*⟩

lemma *is_reducible_lit_mset_iff*:
mset C = *mset C'* ⟹ *is_reducible_lit Ds C' L* ⟷ *is_reducible_lit Ds C L*
⟨*proof*⟩

lemma *is_reducible_lit_remove1_Cons_iff*:
assumes *L* ∈ *set C'*
shows *is_reducible_lit Ds* (*C* @ *remove1 L* (*M* # *C'*)) *L* ⟷
is_reducible_lit Ds (*M* # *C* @ *remove1 L C'*) *L*
⟨*proof*⟩

lemma *reduce_mset_eq*: *mset C* = *mset C'* ⟹ *reduce Ds C E* = *reduce Ds C' E*
⟨*proof*⟩

lemma *reduce_rotate*[*simp*]: *reduce Ds* (*C* @ [*L*]) *E* = *reduce Ds* (*L* # *C*) *E*
⟨*proof*⟩

lemma *mset_reduce_subset*: *mset* (*reduce Ds C E*) ⊆# *mset E*
⟨*proof*⟩

lemma *reduce_idem*: *reduce Ds C* (*reduce Ds C E*) = *reduce Ds C E*
⟨*proof*⟩

lemma *is_reducible_lit_imp_is_reducible*:
L ∈ *set C'* ⟹ *is_reducible_lit Ds* (*C* @ *remove1 L C'*) *L* ⟹ *reduce Ds C C'* ≠ *C'*
⟨*proof*⟩

lemma *is_reducible_imp_is_reducible_lit*:
reduce Ds C C' ≠ *C'* ⟹ ∃ *L* ∈ *set C'*. *is_reducible_lit Ds* (*C* @ *remove1 L C'*) *L*
⟨*proof*⟩

lemma *is_irreducible_iff_nexists_is_reducible_lit*:
reduce Ds C C' = *C'* ⟷ ¬ (∃ *L* ∈ *set C'*. *is_reducible_lit Ds* (*C* @ *remove1 L C'*) *L*)
⟨*proof*⟩

lemma *is_irreducible_mset_iff*: *mset E* = *mset E'* ⟹ *reduce Ds C E* = *E* ⟷ *reduce Ds C E'* = *E'*
⟨*proof*⟩

lemma *select_min_weight_clause_min_weight*:
assumes *Ci* = *select_min_weight_clause P0 P*
shows *weight* (*apfst mset Ci*) = *Min* ((*weight* ∘ *apfst mset*) ‘ *set* (*P0* # *P*))
⟨*proof*⟩

lemma *remdups_clss_Nil_iff*: *remdups_clss Cs* = [] ⟷ *Cs* = []
⟨*proof*⟩

lemma *empty_N_if_Nil_in_P_or_Q*:
assumes *nil_in*: [] ∈ *fst* ‘ *set* (*P* @ *Q*)
shows *wstate_of_dstate* (*N*, *P*, *Q*, *n*) $\leadsto_w{}^*$ *wstate_of_dstate* ([], *P*, *Q*, *n*)
⟨*proof*⟩

lemma *remove_strictly_subsumed_clauses_in_P*:
assumes
c_in: *C* ∈ *fst* ‘ *set N* **and**
p_nsubs: ∀ *D* ∈ *fst* ‘ *set P*. ¬ *strictly_subsume* [*C*] *D*
shows *wstate_of_dstate* (*N*, *P* @ *P'*, *Q*, *n*)
$\leadsto_w{}^*$ *wstate_of_dstate* (*N*, *P* @ *filter* (*Not* ∘ *strictly_subsume* [*C*] ∘ *fst*) *P'*, *Q*, *n*)
⟨*proof*⟩

lemma *remove_strictly_subsumed_clauses_in_Q*:
assumes *c_in*: *C* ∈ *fst* ‘ *set N*
shows *wstate_of_dstate* (*N*, *P*, *Q* @ *Q'*, *n*)
$\leadsto_w{}^*$ *wstate_of_dstate* (*N*, *P*, *Q* @ *filter* (*Not* ∘ *strictly_subsume* [*C*] ∘ *fst*) *Q'*, *n*)
⟨*proof*⟩

lemma *reduce_clause_in_P*:
assumes
c_in: *C* ∈ *fst* ‘ *set N* **and**
p_irred: ∀ (*E*, *k*) ∈ *set* (*P* @ *P'*). *k* > *j* ⟶ *is_irreducible* [*C*] *E*
shows *wstate_of_dstate* (*N*, *P* @ (*D* @ *D'*, *j*) # *P'*, *Q*, *n*)
$\leadsto_w{}^*$ *wstate_of_dstate* (*N*, *P* @ (*D* @ *reduce* [*C*] *D D'*, *j*) # *P'*, *Q*, *n*)
⟨*proof*⟩

lemma *reduce_clause_in_Q*:
assumes
c_in: *C* ∈ *fst* ‘ *set N* **and**
p_irred: ∀ (*E*, *k*) ∈ *set P*. *k* > *j* ⟶ *is_irreducible* [*C*] *E* **and**
d'_red: *reduce* [*C*] *D D'* ≠ *D'*
shows *wstate_of_dstate* (*N*, *P*, *Q* @ (*D* @ *D'*, *j*) # *Q'*, *n*)
$\leadsto_w{}^*$ *wstate_of_dstate* (*N*, (*D* @ *reduce* [*C*] *D D'*, *j*) # *P*, *Q* @ *Q'*, *n*)
⟨*proof*⟩

lemma *reduce_clauses_in_P*:
assumes
c_in: *C* ∈ *fst* ‘ *set N* **and**
p_irred: ∀ (*E*, *k*) ∈ *set P*. *is_irreducible* [*C*] *E*
shows *wstate_of_dstate* (*N*, *P* @ *P'*, *Q*, *n*) $\leadsto_w{}^*$ *wstate_of_dstate* (*N*, *P* @ *reduce_all C P'*, *Q*, *n*)
⟨*proof*⟩

lemma *reduce_clauses_in_Q*:
assumes
c_in: *C* ∈ *fst* ‘ *set N* **and**
p_irred: ∀ (*E*, *k*) ∈ *set P*. *is_irreducible* [*C*] *E*
shows *wstate_of_dstate* (*N*, *P*, *Q* @ *Q'*, *n*)
$\leadsto_w{}^*$ *wstate_of_dstate* (*N*, *fst* (*reduce_all2 C Q'*) @ *P*, *Q* @ *snd* (*reduce_all2 C Q'*), *n*)
⟨*proof*⟩

lemma *eligible_iff*:
eligible S σ *As DA* ⟷ *As* = [] ∨ *length As* = *1* ∧ *maximal_wrt* (*hd As* ·*a* σ) (*DA* · σ)
⟨*proof*⟩

lemma *ord_resolve_one_side_prem*:
ord_resolve S CAs DA AAs As σ *E* ⟹ *length CAs* = *1* ∧ *length AAs* = *1* ∧ *length As* = *1*
⟨*proof*⟩

lemma *ord_resolve_rename_one_side_prem*:

ord_resolve_rename S CAs DA AAs As σ E ⟹ length CAs = 1 ∧ length AAs = 1 ∧ length As = 1
⟨*proof*⟩

abbreviation *Bin_ord_resolve* :: *'a clause ⇒ 'a clause ⇒ 'a clause set* **where**
Bin_ord_resolve C D ≡ {E. ∃ AA A σ. ord_resolve S [C] D [AA] [A] σ E}

abbreviation *Bin_ord_resolve_rename* :: *'a clause ⇒ 'a clause ⇒ 'a clause set* **where**
Bin_ord_resolve_rename C D ≡ {E. ∃ AA A σ. ord_resolve_rename S [C] D [AA] [A] σ E}

lemma *resolve_on_eq_UNION_Bin_ord_resolve*:
mset ‘ set (resolve_on A D CA) =
{E. ∃ AA σ. ord_resolve S [mset CA] ({#Neg A#} + mset D) [AA] [A] σ E}
⟨*proof*⟩

lemma *set_resolve_eq_UNION_set_resolve_on*:
set (resolve C D) =
(⋃ L ∈ set D.
(case L of
Pos _ ⇒ {}
| Neg A ⇒ if maximal_wrt A (mset D) then set (resolve_on A (remove1 L D) C) else {}))
⟨*proof*⟩

lemma *resolve_eq_Bin_ord_resolve*: *mset ‘ set (resolve C D) = Bin_ord_resolve (mset C) (mset D)*
⟨*proof*⟩

lemma *poss_in_map_clauseD*:
poss AA ⊆# map_clause f C ⟹ ∃ AA0. poss AA0 ⊆# C ∧ AA = {#f A. A ∈# AA0#}
⟨*proof*⟩

lemma *poss_subset_filterD*:
poss AA ⊆# {#L ·l ϱ. L ∈# mset C#} ⟹ ∃ AA0. poss AA0 ⊆# mset C ∧ AA = AA0 ·am ϱ
⟨*proof*⟩

lemma *neg_in_map_literalD*: *Neg A ∈ map_literal f ‘ D ⟹ ∃ A0. Neg A0 ∈ D ∧ A = f A0*
⟨*proof*⟩

lemma *neg_in_filterD*: *Neg A ∈# {#L ·l ϱ'. L ∈# mset D#} ⟹ ∃ A0. Neg A0 ∈# mset D ∧ A = A0 ·a ϱ'*
⟨*proof*⟩

lemma *resolve_rename_eq_Bin_ord_resolve_rename*:
mset ‘ set (resolve_rename C D) = Bin_ord_resolve_rename (mset C) (mset D)
⟨*proof*⟩

lemma *bin_ord_FO_Γ_def*:
ord_FO_Γ S = {Infer {#CA#} DA E | CA DA AA A σ E. ord_resolve_rename S [CA] DA [AA] [A] σ E}
⟨*proof*⟩

lemma *ord_FO_Γ_side_prem*: *γ ∈ ord_FO_Γ S ⟹ side_prems_of γ = {#THE D. D ∈# side_prems_of γ#}*
⟨*proof*⟩

lemma *ord_FO_Γ_infer_from_Collect_eq*:
{γ ∈ ord_FO_Γ S. infer_from (DD ∪ {C}) γ ∧ C ∈# prems_of γ} =
{γ ∈ ord_FO_Γ S. ∃ D ∈ DD ∪ {C}. prems_of γ = {#C, D#}}
⟨*proof*⟩

lemma *inferences_between_eq_UNION*: *inference_system.inferences_between (ord_FO_Γ S) Q C =*
inference_system.inferences_between (ord_FO_Γ S) {C} C
∪ (⋃ D ∈ Q. inference_system.inferences_between (ord_FO_Γ S) {D} C)
⟨*proof*⟩

lemma *concls_of_inferences_between_singleton_eq_Bin_ord_resolve_rename*:
concls_of (inference_system.inferences_between (ord_FO_Γ S) {D} C) =
Bin_ord_resolve_rename C C ∪ Bin_ord_resolve_rename C D ∪ Bin_ord_resolve_rename D C

⟨proof⟩

lemma concls_of_inferences_between_eq_Bin_ord_resolve_rename:
concls_of (inference_system.inferences_between (ord_FO_Γ S) Q C) =
Bin_ord_resolve_rename C C ∪ (⋃ D ∈ Q. Bin_ord_resolve_rename C D ∪ Bin_ord_resolve_rename D C)
⟨proof⟩

lemma resolve_rename_either_way_eq_congls_of_inferences_between:
mset ‘ set (resolve_rename C C) ∪ (⋃ D ∈ Q. mset ‘ set (resolve_rename_either_way C D)) =
concls_of (inference_system.inferences_between (ord_FO_Γ S) (mset ‘ Q) (mset C))
⟨proof⟩

lemma compute_inferences:
assumes
ci_in: (C, i) ∈ set P **and**
ci_min: ∀ (D, j) ∈# mset (map (apfst mset) P). weight (mset C, i) ≤ weight (D, j)
shows
wstate_of_dstate ([], P, Q, n) ⇝$_w$
wstate_of_dstate (map (λD. (D, n)) (remdups_gen mset (resolve_rename C C @
concat (map (resolve_rename_either_way C ∘ fst) Q))),
filter (λ(D, j). mset D ≠ mset C) P, (C, i) # Q, Suc n)
(**is** _ ⇝$_w$ wstate_of_dstate (?N, _))
⟨proof⟩

lemma nonfinal_deterministic_RP_step:
assumes
nonfinal: ¬ is_final_dstate St **and**
step: St′ = deterministic_RP_step St
shows wstate_of_dstate St ⇝$_w{}^+$ wstate_of_dstate St′
⟨proof⟩

lemma final_deterministic_RP_step: is_final_dstate St ⟹ deterministic_RP_step St = St
⟨proof⟩

lemma deterministic_RP_SomeD:
assumes deterministic_RP (N, P, Q, n) = Some R
shows ∃ N′ P′ Q′ n′. (∃ k. (deterministic_RP_step ^^ k) (N, P, Q, n) = (N′, P′, Q′, n′))
∧ is_final_dstate (N′, P′, Q′, n′) ∧ R = map fst Q′
⟨proof⟩

context
fixes
N0 :: ′a dclause list **and**
n0 :: nat **and**
R :: ′a lclause list
begin

abbreviation St0 :: ′a dstate **where**
St0 ≡ (N0, [], [], n0)

abbreviation grounded_N0 **where**
grounded_N0 ≡ grounding_of_clss (set (map (mset ∘ fst) N0))

abbreviation grounded_R :: ′a clause set **where**
grounded_R ≡ grounding_of_clss (set (map mset R))

primcorec derivation_from :: ′a dstate ⇒ ′a dstate llist **where**
derivation_from St =
LCons St (if is_final_dstate St then LNil else derivation_from (deterministic_RP_step St))

abbreviation Sts :: ′a dstate llist **where**
Sts ≡ derivation_from St0

abbreviation *wSts* :: *'a wstate llist* **where**
wSts ≡ *lmap wstate_of_dstate Sts*

lemma *full_deriv_wSts_trancl_weighted_RP*: *full_chain* ($\leadsto_w{}^+$) *wSts*
⟨*proof*⟩

lemmas *deriv_wSts_trancl_weighted_RP* = *full_chain_imp_chain*[*OF full_deriv_wSts_trancl_weighted_RP*]

definition *sswSts* :: *'a wstate llist* **where**
sswSts = (*SOME wSts'*.
full_chain ($\leadsto_w$) *wSts'* ∧ *emb wSts wSts'* ∧ *lhd wSts'* = *lhd wSts* ∧ *llast wSts'* = *llast wSts*)

lemma *sswSts*:
full_chain ($\leadsto_w$) *sswSts* ∧ *emb wSts sswSts* ∧ *lhd sswSts* = *lhd wSts* ∧ *llast sswSts* = *llast wSts*
⟨*proof*⟩

lemmas *full_deriv_sswSts_weighted_RP* = *sswSts*[*THEN conjunct1*]
lemmas *emb_sswSts* = *sswSts*[*THEN conjunct2*, *THEN conjunct1*]
lemmas *lfinite_sswSts_iff* = *emb_lfinite*[*OF emb_sswSts*]
lemmas *lhd_sswSts* = *sswSts*[*THEN conjunct2*, *THEN conjunct2*, *THEN conjunct1*]
lemmas *llast_sswSts* = *sswSts*[*THEN conjunct2*, *THEN conjunct2*, *THEN conjunct2*]

lemmas *deriv_sswSts_weighted_RP* = *full_chain_imp_chain*[*OF full_deriv_sswSts_weighted_RP*]

lemma *not_lnull_sswSts*: ¬ *lnull sswSts*
⟨*proof*⟩

lemma *empty_ssgP0*: *wrp.P_of_wstate* (*lhd sswSts*) = {}
⟨*proof*⟩

lemma *empty_ssgQ0*: *wrp.Q_of_wstate* (*lhd sswSts*) = {}
⟨*proof*⟩

lemmas *sswSts_thms* = *full_deriv_sswSts_weighted_RP empty_ssgP0 empty_ssgQ0*

abbreviation *S_ssgQ* :: *'a clause* ⇒ *'a clause* **where**
S_ssgQ ≡ *wrp.S_gQ sswSts*

abbreviation *ord_Γ* :: *'a inference set* **where**
ord_Γ ≡ *ground_resolution_with_selection.ord_Γ S_ssgQ*

abbreviation *Rf* :: *'a clause set* ⇒ *'a clause set* **where**
Rf ≡ *standard_redundancy_criterion.Rf*

abbreviation *Ri* :: *'a clause set* ⇒ *'a inference set* **where**
Ri ≡ *standard_redundancy_criterion.Ri ord_Γ*

abbreviation *saturated_upto* :: *'a clause set* ⇒ *bool* **where**
saturated_upto ≡ *redundancy_criterion.saturated_upto ord_Γ Rf Ri*

context
assumes *drp_some*: *deterministic_RP St0* = *Some R*
begin

lemma *lfinite_Sts*: *lfinite Sts*
⟨*proof*⟩

lemma *lfinite_wSts*: *lfinite wSts*
⟨*proof*⟩

lemmas *lfinite_sswSts* = *lfinite_sswSts_iff*[*THEN iffD2*, *OF lfinite_wSts*]

theorem

deterministic_RP_saturated: *saturated_upto grounded_R* (**is** *?saturated*) **and**
deterministic_RP_model: *I* ⊨*s grounded_N0* ⟷ *I* ⊨*s grounded_R* (**is** *?model*)
⟨*proof*⟩

corollary *deterministic_RP_refutation*:
¬ *satisfiable grounded_N0* ⟷ {#} ∈ *grounded_R* (**is** *?lhs* ⟷ *?rhs*)
⟨*proof*⟩

end

context
assumes *drp_none*: *deterministic_RP St0* = *None*
begin

theorem *deterministic_RP_complete*: *satisfiable grounded_N0*
⟨*proof*⟩

end

end

end

end

4 Integration of IsaFoR Terms and the Knuth–Bendix Order

This theory implements the abstract interface for atoms and substitutions using the IsaFoR library.

theory *IsaFoR_Term*
imports
Deriving.Derive
Ordered_Resolution_Prover.Abstract_Substitution
First_Order_Terms.Unification
First_Order_Terms.Subsumption
HOL−Cardinals.Wellorder_Extension
Open_Induction.Restricted_Predicates
Knuth_Bendix_Order.KBO
begin

hide-const (**open**) *mgu*

abbreviation *subst_apply_literal* ::
(′*f*, ′*v*) *term literal* ⇒ (′*f*, ′*v*, ′*w*) *gsubst* ⇒ (′*f*, ′*w*) *term literal* (**infixl** ‹·*lit*› *60*) **where**
L ·*lit* σ ≡ *map_literal* (λ*A*. *A* · σ) *L*

definition *subst_apply_clause* ::
(′*f*, ′*v*) *term clause* ⇒ (′*f*, ′*v*, ′*w*) *gsubst* ⇒ (′*f*, ′*w*) *term clause* (**infixl** ‹·*cls*› *60*) **where**
C ·*cls* σ = *image_mset* (λ*L*. *L* ·*lit* σ) *C*

abbreviation *vars_lit* :: (′*f*, ′*v*) *term literal* ⇒ ′*v set* **where**
vars_lit L ≡ *vars_term* (*atm_of L*)

definition *vars_clause* :: (′*f*, ′*v*) *term clause* ⇒ ′*v set* **where**
vars_clause C = *Union* (*set_mset* (*image_mset vars_lit C*))

definition *vars_clause_list* :: (′*f*, ′*v*) *term clause list* ⇒ ′*v set* **where**
vars_clause_list Cs = *Union* (*vars_clause* ‘ *set Cs*)

definition *vars_partitioned* :: (′*f*,′*v*) *term clause list* ⇒ *bool* **where**
vars_partitioned Cs ⟷
(∀ *i* < *length Cs*. ∀ *j* < *length Cs*. *i* ≠ *j* ⟶ (*vars_clause* (*Cs* ! *i*) ∩ *vars_clause* (*Cs* ! *j*)) = {})

lemma *vars_clause_mono*: $S \subseteq\# C \Longrightarrow vars\_clause\ S \subseteq vars\_clause\ C$
⟨*proof*⟩

interpretation *substitution_ops* $(\cdot)$ *Var* $(\circ_s)$ ⟨*proof*⟩

lemma *is_ground_atm_is_ground_on_var*:
assumes *is_ground_atm* $(A \cdot \sigma)$ **and** $v \in vars\_term\ A$
shows *is_ground_atm* $(\sigma\ v)$
⟨*proof*⟩

lemma *is_ground_lit_is_ground_on_var*:
assumes *ground_lit*: *is_ground_lit* (*subst_lit* $L\ \sigma$) **and** *v_in_L*: $v \in vars\_lit\ L$
shows *is_ground_atm* $(\sigma\ v)$
⟨*proof*⟩

lemma *is_ground_cls_is_ground_on_var*:
assumes
ground_clause: *is_ground_cls* (*subst_cls* $C\ \sigma$) **and**
v_in_C: $v \in vars\_clause\ C$
shows *is_ground_atm* $(\sigma\ v)$
⟨*proof*⟩

lemma *is_ground_cls_list_is_ground_on_var*:
assumes *ground_list*: *is_ground_cls_list* (*subst_cls_list* $Cs\ \sigma$)
and *v_in_Cs*: $v \in vars\_clause\_list\ Cs$
shows *is_ground_atm* $(\sigma\ v)$
⟨*proof*⟩

lemma *same_on_vars_lit*:
assumes $\forall v \in vars\_lit\ L.\ \sigma\ v = \tau\ v$
shows *subst_lit* $L\ \sigma$ = *subst_lit* $L\ \tau$
⟨*proof*⟩

lemma *in_list_of_mset_in_S*:
assumes $i <$ *length* (*list_of_mset* S)
shows *list_of_mset* $S\ !\ i \in\# S$
⟨*proof*⟩

lemma *same_on_vars_clause*:
assumes $\forall v \in vars\_clause\ S.\ \sigma\ v = \tau\ v$
shows *subst_cls* $S\ \sigma$ = *subst_cls* $S\ \tau$
⟨*proof*⟩

interpretation *substitution* $(\cdot)$ *Var* :: _ ⇒ ('*f*, *nat*) *term* $(\circ_s)$
⟨*proof*⟩

lemma *vars_partitioned_var_disjoint*:
assumes *vars_partitioned Cs*
shows *var_disjoint Cs*
⟨*proof*⟩

lemma *vars_in_instance_in_range_term*:
vars_term (*subst_atm_abbrev* $A\ \sigma$) ⊆ *Union* (*image vars_term* (*range* σ))
⟨*proof*⟩

lemma *vars_in_instance_in_range_lit*: *vars_lit* (*subst_lit* $L\ \sigma$) ⊆ *Union* (*image vars_term* (*range* σ))
⟨*proof*⟩

lemma *vars_in_instance_in_range_cls*:
vars_clause (*subst_cls* $C\ \sigma$) ⊆ *Union* (*image vars_term* (*range* σ))
⟨*proof*⟩

primrec *renamings_apart* :: ('*f*, *nat*) *term clause list* ⇒ (('*f*, *nat*) *subst*) *list* **where**

renamings_apart [] = []
| *renamings_apart* (*C* # *Cs*) =
(*let* *σs* = *renamings_apart Cs in*
(*λv. Var* (*v* + *Max* (*vars_clause_list* (*subst_cls_lists Cs σs*) ∪ {*0*}) + *1*)) # *σs*)

definition *var_map_of_subst* :: (*'f*, *nat*) *subst* ⇒ *nat* ⇒ *nat* **where**
var_map_of_subst σ v = *the_Var* (*σ v*)

lemma *len_renamings_apart*: *length* (*renamings_apart Cs*) = *length Cs*
⟨*proof*⟩

lemma *renamings_apart_is_Var*: ∀ *σ* ∈ *set* (*renamings_apart Cs*). ∀ *x*. *is_Var* (*σ x*)
⟨*proof*⟩

lemma *renamings_apart_inj*: ∀ *σ* ∈ *set* (*renamings_apart Cs*). *inj σ*
⟨*proof*⟩

lemma *finite_vars_clause*[*simp*]: *finite* (*vars_clause x*)
⟨*proof*⟩

lemma *finite_vars_clause_list*[*simp*]: *finite* (*vars_clause_list Cs*)
⟨*proof*⟩

lemma *Suc_Max_notin_set*: *finite X* ⟹ *Suc* (*v* + *Max* (*insert 0 X*)) ∉ *X*
⟨*proof*⟩

lemma *vars_partitioned_Nil*[*simp*]: *vars_partitioned* []
⟨*proof*⟩

lemma *subst_cls_lists_Nil*[*simp*]: *subst_cls_lists Cs* [] = []
⟨*proof*⟩

lemma *vars_clause_hd_partitioned_from_tl*:
assumes *Cs* ≠[]
shows *vars_clause* (*hd* (*subst_cls_lists Cs* (*renamings_apart Cs*)))
∩ *vars_clause_list* (*tl* (*subst_cls_lists Cs* (*renamings_apart Cs*))) = {}
⟨*proof*⟩

lemma *vars_partitioned_renamings_apart*: *vars_partitioned* (*subst_cls_lists Cs* (*renamings_apart Cs*))
⟨*proof*⟩

interpretation *substitution_renamings* (·) *Var* :: _ ⇒ (*'f*, *nat*) *term* ($\circ_s$) *renamings_apart Fun undefined*
⟨*proof*⟩

fun *pairs* :: *'a list* ⇒ (*'a* × *'a*) *list* **where**
pairs (*x* # *y* # *xs*) = (*x*, *y*) # *pairs* (*y* # *xs*) |
pairs _ = []

derive *compare term*
derive *compare literal*

lemma *class_linorder_compare*: *class.linorder* (*le_of_comp compare*) (*lt_of_comp compare*)
⟨*proof*⟩

context begin
interpretation *compare_linorder*: *linorder*
le_of_comp compare
lt_of_comp compare
⟨*proof*⟩

definition *Pairs* **where**
Pairs AAA = *concat* (*compare_linorder.sorted_list_of_set*
((*pairs* ∘ *compare_linorder.sorted_list_of_set*) ‘ *AAA*))

lemma *unifies_all_pairs_iff*:
  $(\forall p \in set\ (pairs\ xs).\ fst\ p \cdot \sigma = snd\ p \cdot \sigma) \longleftrightarrow (\forall a \in set\ xs.\ \forall b \in set\ xs.\ a \cdot \sigma = b \cdot \sigma)$
⟨*proof*⟩

lemma *in_pair_in_set*:
  assumes $(A,B) \in set\ ((pairs\ As))$
  shows $A \in set\ As \wedge B \in set\ As$
  ⟨*proof*⟩

lemma *in_pairs_sorted_list_of_set_in_set*:
  assumes
    finite AAA
    $\forall AA \in AAA.$ *finite AA*
    $AB\_pairs \in (pairs \circ compare\_linorder.sorted\_list\_of\_set)$ ‘ *AAA* **and**
    $(A :: \_ :: compare,\ B) \in set\ AB\_pairs$
  shows $\exists AA.\ AA \in AAA \wedge A \in AA \wedge B \in AA$
⟨*proof*⟩

lemma *unifiers_Pairs*:
  assumes
    finite AAA **and**
    $\forall AA \in AAA.$ *finite AA*
  shows $unifiers\ (set\ (Pairs\ AAA)) = \{\sigma.\ is\_unifiers\ \sigma\ AAA\}$
⟨*proof*⟩

end

definition *mgu_sets AAA = map_option subst_of (unify (Pairs AAA) [])*

lemma *mgu_sets_is_imgu*:
  fixes $AAA :: ('a :: compare,\ nat)\ term\ set\ set$ **and** $\sigma :: ('a,\ nat)\ subst$
  assumes *fin*: *finite AAA* $\forall AA \in AAA.$ *finite AA* **and** *mgu_sets AAA = Some* σ
  shows *is_imgu* σ *AAA*
⟨*proof*⟩

interpretation *mgu* $(\cdot)$ *Var* $:: \_ \Rightarrow ('f :: compare,\ nat)\ term\ (\circ_s)$ *renamings_apart*
  Fun undefined mgu_sets
⟨*proof*⟩

interpretation *imgu* $(\cdot)$ *Var* $:: \_ \Rightarrow ('f :: compare,\ nat)\ term\ (\circ_s)$ *renamings_apart*
  Fun undefined mgu_sets
⟨*proof*⟩

derive *linorder prod*
derive *linorder list*

This part extends and integrates and the Knuth–Bendix order defined in IsaFoR.

record *'f weights* =
  $w :: 'f \times nat \Rightarrow nat$
  $w0 :: nat$
  $pr\_strict :: 'f \times nat \Rightarrow 'f \times nat \Rightarrow bool$
  $least :: 'f \Rightarrow bool$
  $scf :: 'f \times nat \Rightarrow nat \Rightarrow nat$

class *weighted* =
  fixes *weights* :: *'a weights*
  assumes *weights_adm*:
    admissible_kbo
      $(w\ weights)\ (w0\ weights)\ (pr\_strict\ weights)\ ((pr\_strict\ weights)^{==})\ (least\ weights)\ (scf\ weights)$
  and *pr_strict_total*: $fi = gj \vee pr\_strict\ weights\ fi\ gj \vee pr\_strict\ weights\ gj\ fi$
  and *pr_strict_asymp*: *asymp (pr_strict weights)*
  and *scf_ok*: $i < n \Longrightarrow scf\ weights\ (f,\ n)\ i \leq 1$

instantiation *unit* :: *weighted* **begin**

definition *weights_unit* :: *unit weights* **where** *weights_unit* =
  ⦇*w* = *Suc* ∘ *snd*, *w0* = *1*, *pr_strict* = λ(_, *n*) (_, *m*). *n* > *m*, *least* = λ_. *True*, *scf* = λ_ _. *1*⦈

instance
  ⟨*proof*⟩
end

global-interpretation *KBO*:
  admissible_kbo
    w (*weights* :: ′*f* :: *weighted weights*) *w0* (*weights* :: ′*f* :: *weighted weights*)
    pr_strict weights ((*pr_strict weights*)$^{==}$) *least weights scf weights*
    defines *weight* = *KBO.weight*
    and *kbo* = *KBO.kbo*
  ⟨*proof*⟩

lemma *kbo_code*[*code*]: *kbo s t* =
  (*let wt* = *weight t*; *ws* = *weight s in*
  if vars_term_ms (*KBO.SCF t*) ⊆# *vars_term_ms* (*KBO.SCF s*) ∧ *wt* ≤ *ws*
  then
    (*if wt* < *ws then* (*True*, *True*)
    else
      (*case s of*
        Var y ⇒ (*False*, *case t of Var x* ⇒ *True* | *Fun g ts* ⇒ *ts* = [] ∧ *least weights g*)
      | *Fun f ss* ⇒
          (*case t of*
            Var x ⇒ (*True*, *True*)
          | *Fun g ts* ⇒
              if pr_strict weights (*f*, *length ss*) (*g*, *length ts*) *then* (*True*, *True*)
              else if (*f*, *length ss*) = (*g*, *length ts*) *then lex_ext_unbounded kbo ss ts*
              else (*False*, *False*))))
  else (*False*, *False*))
  ⟨*proof*⟩

definition *less_kbo s t* = *fst* (*kbo t s*)

lemma *less_kbo_gtotal*: *ground s* ⟹ *ground t* ⟹ *s* = *t* ∨ *less_kbo s t* ∨ *less_kbo t s*
  ⟨*proof*⟩

lemma *less_kbo_subst*:
  fixes σ :: (′*f* :: *weighted*, ′*v*) *subst*
  shows *less_kbo s t* ⟹ *less_kbo* (*s* · σ) (*t* · σ)
  ⟨*proof*⟩

lemma *wfP_less_kbo*: *wfP less_kbo*
⟨*proof*⟩

instantiation *term* :: (*weighted*, *type*) *linorder* **begin**

definition *leq_term* = (*SOME leq*. {(*s*,*t*). *less_kbo s t*} ⊆ *leq* ∧ *Well_order leq* ∧ *Field leq* = *UNIV*)

lemma *less_trm_extension*: {(*s*,*t*). *less_kbo s t*} ⊆ *leq_term*
  ⟨*proof*⟩

lemma *less_trm_well_order*: *well_order leq_term*
  ⟨*proof*⟩

definition *less_eq_term* :: (′*a* :: *weighted*, ′*b*) *term* ⇒ _ ⇒ *bool* **where**
  less_eq_term = *in_rel leq_term*
definition *less_term* :: (′*a* :: *weighted*, ′*b*) *term* ⇒ _ ⇒ *bool* **where**
  less_term s t = *strict* (≤) *s t*

lemma *leq_term_minus_Id*: *leq_term* − *Id* = {(*x*,*y*). *x* < *y*}
⟨*proof*⟩

lemma *less_term_alt*: (<) = *in_rel* (*leq_term* − *Id*)
⟨*proof*⟩

instance
⟨*proof*⟩

end

instantiation *term* :: (*weighted*, *type*) *wellorder* **begin**
instance
⟨*proof*⟩
end

lemma *ground_less_less_kbo*: *ground s* ⟹ *ground t* ⟹ *s* < *t* ⟹ *less_kbo s t*
⟨*proof*⟩

lemma *less_kbo_less*: *less_kbo s t* ⟹ *s* < *t*
⟨*proof*⟩

lemma *is_ground_atm_ground*: *is_ground_atm t* ⟷ *ground t*
⟨*proof*⟩

end

5 An Executable Algorithm for Clause Subsumption

This theory provides an executable functional implementation of clause subsumption, building on the IsaFoR library.

theory *Executable_Subsumption*
imports
IsaFoR_Term
First_Order_Terms.Matching
begin

5.1 Naive Implementation of Clause Subsumption

fun *subsumes_list* **where**
subsumes_list [] *Ks* σ = *True*
| *subsumes_list* (*L* # *Ls*) *Ks* σ =
(∃ *K* ∈ *set Ks*. *is_pos K* = *is_pos L* ∧
(*case match_term_list* [(*atm_of L*, *atm_of K*)] σ *of*
None ⇒ *False*
| *Some* ϱ ⇒ *subsumes_list Ls* (*remove1 K Ks*) ϱ))

lemma *atm_of_map_literal*[*simp*]: *atm_of* (*map_literal f l*) = *f* (*atm_of l*)
⟨*proof*⟩

definition *extends_subst* σ τ = (∀ *x* ∈ *dom* σ. σ *x* = τ *x*)

lemma *extends_subst_refl*[*simp*]: *extends_subst* σ σ
⟨*proof*⟩

lemma *extends_subst_trans*: *extends_subst* σ τ ⟹ *extends_subst* τ ϱ ⟹ *extends_subst* σ ϱ
⟨*proof*⟩

lemma *extends_subst_dom*: *extends_subst* σ τ ⟹ *dom* σ ⊆ *dom* τ
⟨*proof*⟩

lemma *extends_subst_extends*: *extends_subst* σ τ $\Longrightarrow$ $x \in$ *dom* σ $\Longrightarrow$ τ $x = \sigma$ x
⟨*proof*⟩

lemma *extends_subst_fun_upd_new*:
σ x = *None* $\Longrightarrow$ *extends_subst* $(\sigma(x \mapsto t))$ τ $\longleftrightarrow$ *extends_subst* σ τ $\wedge$ τ x = *Some* t
⟨*proof*⟩

lemma *extends_subst_fun_upd_matching*:
σ x = *Some* t $\Longrightarrow$ *extends_subst* $(\sigma(x \mapsto t))$ τ $\longleftrightarrow$ *extends_subst* σ τ
⟨*proof*⟩

lemma *extends_subst_empty*[*simp*]: *extends_subst Map.empty* τ
⟨*proof*⟩

lemma *extends_subst_cong_term*:
extends_subst σ τ $\Longrightarrow$ *vars_term* $t \subseteq$ *dom* σ $\Longrightarrow$ $t \cdot$ *subst_of_map Var* σ = $t \cdot$ *subst_of_map Var* τ
⟨*proof*⟩

lemma *extends_subst_cong_lit*:
extends_subst σ τ $\Longrightarrow$ *vars_lit* $L \subseteq$ *dom* σ $\Longrightarrow$ L $\cdot$*lit subst_of_map Var* σ = L $\cdot$*lit subst_of_map Var* τ
⟨*proof*⟩

definition *subsumes_modulo* C D σ =
($\exists\,\tau$. *dom* τ = *vars_clause* C $\cup$ *dom* σ $\wedge$ *extends_subst* σ τ $\wedge$ *subst_cls* C (*subst_of_map Var* τ) $\subseteq\#$ D)

abbreviation *subsumes_list_modulo* **where**
subsumes_list_modulo Ls Ks σ $\equiv$ *subsumes_modulo* (*mset Ls*) (*mset Ks*) σ

lemma *vars_clause_add_mset*[*simp*]: *vars_clause* (*add_mset* L C) = *vars_lit* L $\cup$ *vars_clause* C
⟨*proof*⟩

lemma *subsumes_list_modulo_Cons*: *subsumes_list_modulo* (L # *Ls*) *Ks* σ $\longleftrightarrow$
($\exists\,K \in$ *set Ks*. $\exists\,\tau$. *extends_subst* σ τ $\wedge$ *dom* τ = *vars_lit* L $\cup$ *dom* σ $\wedge$ L $\cdot$*lit* (*subst_of_map Var* τ) = K
$\wedge$ *subsumes_list_modulo Ls* (*remove1* K *Ks*) τ)
⟨*proof*⟩

lemma *decompose_Some_var_terms*: *decompose* (*Fun f ss*) (*Fun g ts*) = *Some eqs* $\Longrightarrow$
$f = g$ $\wedge$ *length ss* = *length ts* $\wedge$ *eqs* = *zip ss ts* $\wedge$
($\bigcup$(t, u)$\in$*set* ((*Fun f ss*, *Fun g ts*) # P). *vars_term* t) =
($\bigcup$(t, u)$\in$*set* (*eqs* @ P). *vars_term* t)
⟨*proof*⟩

lemma *match_term_list_sound*: *match_term_list tus* σ = *Some* τ $\Longrightarrow$
extends_subst σ τ $\wedge$ *dom* τ = ($\bigcup$(t, u)$\in$*set tus*. *vars_term* t) $\cup$ *dom* σ $\wedge$
($\forall$(t,u)$\in$*set tus*. $t \cdot$ *subst_of_map Var* τ = u)
⟨*proof*⟩

lemma *match_term_list_complete*: *match_term_list tus* σ = *None* $\Longrightarrow$
extends_subst σ τ $\Longrightarrow$ *dom* τ = ($\bigcup$(t, u)$\in$*set tus*. *vars_term* t) $\cup$ *dom* σ $\Longrightarrow$
($\exists$(t,u)$\in$*set tus*. $t \cdot$ *subst_of_map Var* $\tau \neq u$)
⟨*proof*⟩

lemma *unique_extends_subst*:
assumes *extends*: *extends_subst* σ τ *extends_subst* σ ϱ **and**
dom: *dom* τ = *vars_term* t $\cup$ *dom* σ *dom* ϱ = *vars_term* t $\cup$ *dom* σ **and**
eq: $t \cdot$ *subst_of_map Var* ϱ = $t \cdot$ *subst_of_map Var* τ
shows $\varrho = \tau$
⟨*proof*⟩

lemma *subsumes_list_alt*:
subsumes_list Ls Ks σ $\longleftrightarrow$ *subsumes_list_modulo Ls Ks* σ
⟨*proof*⟩

lemma *subsumes_subsumes_list*[*code_unfold*]:
subsumes (*mset Ls*) (*mset Ks*) = *subsumes_list Ls Ks Map.empty*
⟨*proof*⟩

lemma *strictly_subsumes_subsumes_list*[*code_unfold*]:
strictly_subsumes (*mset Ls*) (*mset Ks*) =
(*subsumes_list Ls Ks Map.empty* ∧ ¬ *subsumes_list Ks Ls Map.empty*)
⟨*proof*⟩

lemma *subsumes_list_filterD*: *subsumes_list Ls* (*filter P Ks*) σ ⟹ *subsumes_list Ls Ks* σ
⟨*proof*⟩

lemma *subsumes_list_filterI*:
assumes *match*: (⋀*L K* σ τ. *L* ∈ *set Ls* ⟹
match_term_list [(*atm_of L*, *atm_of K*)] σ = *Some* τ ⟹ *is_pos L* = *is_pos K* ⟹ *P K*)
shows *subsumes_list Ls Ks* σ ⟹ *subsumes_list Ls* (*filter P Ks*) σ
⟨*proof*⟩

lemma *subsumes_list_Cons_filter_iff*:
assumes *sorted_wrt*: *sorted_wrt leq* (*L* # *Ls*) **and** *trans*: *transp leq*
and *match*: (⋀*L K* σ τ.
match_term_list [(*atm_of L*, *atm_of K*)] σ = *Some* τ ⟹ *is_pos L* = *is_pos K* ⟹ *leq L K*)
shows *subsumes_list* (*L* # *Ls*) (*filter* (*leq L*) *Ks*) σ ⟷ *subsumes_list* (*L* # *Ls*) *Ks* σ
⟨*proof*⟩

definition *leq_head* :: (′*f*::*linorder*, ′*v*) *term* ⇒ (′*f*, ′*v*) *term* ⇒ *bool* **where**
leq_head t u = (*case* (*root t*, *root u*) *of*
(*None*, _) ⇒ *True*
| (_, *None*) ⇒ *False*
| (*Some f*, *Some g*) ⇒ *f* ≤ *g*)
definition *leq_lit L K* = (*case* (*K*, *L*) *of*
(*Neg* _, *Pos* _) ⇒ *True*
| (*Pos* _, *Neg* _) ⇒ *False*
| _ ⇒ *leq_head* (*atm_of L*) (*atm_of K*))

lemma *transp_leq_lit*[*simp*]: *transp leq_lit*
⟨*proof*⟩

lemma *reflp_leq_lit*[*simp*]: *reflp_on A leq_lit*
⟨*proof*⟩

lemma *total_leq_lit*[*simp*]: *totalp_on A leq_lit*
⟨*proof*⟩

lemma *leq_head_subst*[*simp*]: *leq_head t* (*t* · σ)
⟨*proof*⟩

lemma *leq_lit_match*:
fixes *L K* :: (′*f* :: *linorder*, ′*v*) *term literal*
shows *match_term_list* [(*atm_of L*, *atm_of K*)] σ = *Some* τ ⟹ *is_pos L* = *is_pos K* ⟹ *leq_lit L K*
⟨*proof*⟩

5.2 Optimized Implementation of Clause Subsumption

fun *subsumes_list_filter* **where**
subsumes_list_filter [] *Ks* σ = *True*
| *subsumes_list_filter* (*L* # *Ls*) *Ks* σ =
(*let Ks* = *filter* (*leq_lit L*) *Ks in*
(∃ *K* ∈ *set Ks*. *is_pos K* = *is_pos L* ∧
(*case match_term_list* [(*atm_of L*, *atm_of K*)] σ *of*
None ⇒ *False*
| *Some* ϱ ⇒ *subsumes_list_filter Ls* (*remove1 K Ks*) ϱ)))

lemma *sorted_wrt_subsumes_list_subsumes_list_filter*:

sorted_wrt leq_lit Ls $\Longrightarrow$ *subsumes_list Ls Ks* σ = *subsumes_list_filter Ls Ks* σ
⟨*proof*⟩

5.3 Definition of Deterministic QuickSort

This is the functional description of the standard variant of deterministic QuickSort that always chooses the first list element as the pivot as given by Hoare in 1962. For a list that is already sorted, this leads to $n(n-1)$ comparisons, but as is well known, the average case is much better.

The code below is adapted from Manuel Eberl's *Quick_Sort_Cost* AFP entry, but without invoking probability theory and using a predicate instead of a set.

fun *quicksort* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *'a list* ⇒ *'a list* **where**
  quicksort _ [] = []
| *quicksort R* (*x* # *xs*) =
    quicksort R (*filter* (λ*y*. *R y x*) *xs*) @ [*x*] @ *quicksort R* (*filter* (λ*y*. ¬ *R y x*) *xs*)

We can easily show that this QuickSort is correct:

theorem *mset_quicksort* [*simp*]: *mset* (*quicksort R xs*) = *mset xs*
  ⟨*proof*⟩

corollary *set_quicksort* [*simp*]: *set* (*quicksort R xs*) = *set xs*
  ⟨*proof*⟩

theorem *sorted_wrt_quicksort*:
  assumes *transp R* **and** *totalp_on* (*set xs*) *R* **and** *reflp_on* (*set xs*) *R*
  shows *sorted_wrt R* (*quicksort R xs*)
⟨*proof*⟩

End of the material adapted from Eberl's *Quick_Sort_Cost*.

lemma *subsumes_list_subsumes_list_filter*[*abs_def*, *code_unfold*]:
  subsumes_list Ls Ks σ = *subsumes_list_filter* (*quicksort leq_lit Ls*) *Ks* σ
  ⟨*proof*⟩

end

6 An Executable Simple Ordered Resolution Prover for First-Order Clauses

This theory provides an executable functional implementation of the *deterministic_RP* prover, building on the IsaFoR library for the notion of terms and on the Knuth–Bendix order.

theory *Executable_FO_Ordered_Resolution_Prover*
  imports
    Deterministic_FO_Ordered_Resolution_Prover
    Executable_Subsumption
    HOL−Library.Code_Target_Nat
    Show.Show_Instances
    IsaFoR_Term
begin

global-interpretation *RP*: *deterministic_FO_resolution_prover* **where**
  S = λ_. {#} **and**
  subst_atm = (·) **and**
  id_subst = *Var* :: _ ⇒ (*'f* :: {*weighted*, *compare_order*}, *nat*) *term* **and**
  comp_subst = ($\circ_s$) **and**
  renamings_apart = *renamings_apart* **and**
  atm_of_atms = *Fun undefined* **and**
  mgu = *mgu_sets* **and**
  less_atm = *less_kbo* **and**
  size_atm = *size* **and**
  timestamp_factor = *1* **and**
  size_factor = *1*

defines *deterministic_RP* = *RP.deterministic_RP*
and *deterministic_RP_step* = *RP.deterministic_RP_step*
and *is_final_dstate* = *RP.is_final_dstate*
and *is_reducible_lit* = *RP.is_reducible_lit*
and *is_tautology* = *RP.is_tautology*
and *maximal_wrt* = *RP.maximal_wrt*
and *reduce* = *RP.reduce*
and *reduce_all* = *RP.reduce_all*
and *reduce_all2* = *RP.reduce_all2*
and *remdups_clss* = *RP.remdups_clss*
and *resolve* = *RP.resolve*
and *resolve_on* = *RP.resolve_on*
and *resolvable* = *RP.resolvable*
and *resolvent* = *RP.resolvent*
and *resolve_rename* = *RP.resolve_rename*
and *resolve_rename_either_way* = *RP.resolve_rename_either_way*
and *select_min_weight_clause* = *RP.select_min_weight_clause*
and *strictly_maximal_wrt* = *RP.strictly_maximal_wrt*
and *strictly_subsume* = *RP.strictly_subsume*
and *subsume* = *RP.subsume*
and *weight* = *RP.weight*
and *St0* = *RP.St0*
and *sorted_list_of_set* = *linorder.sorted_list_of_set* (*le_of_comp compare*)
and *sort_key* = *linorder.sort_key* (*le_of_comp compare*)
and *insort_key* = *linorder.insort_key* (*le_of_comp compare*)
⟨*proof*⟩

declare
RP.deterministic_RP.simps[*code*]
RP.deterministic_RP_step.simps[*code*]
RP.is_final_dstate.simps[*code*]
RP.is_tautology_def[*code*]
RP.reduce.simps[*code*]
RP.reduce_all_def[*code*]
RP.reduce_all2.simps[*code*]
RP.resolve_rename_def[*code*]
RP.resolve_rename_either_way_def[*code*]
RP.select_min_weight_clause.simps[*code*]
RP.weight.simps[*code*]
St0_def[*code*]
substitution_ops.strictly_subsumes_def[*code*]
substitution_ops.subst_cls_lists_def[*code*]
substitution_ops.subst_lit_def[*code*]
substitution_ops.subst_cls_def[*code*]

lemma *remove1_mset_subset_eq*: *remove1_mset a A* ⊆# *B* ⟷ *A* ⊆# *add_mset a B*
⟨*proof*⟩

lemma *Bex_cong*: (⋀*b*. *b* ∈ *B* ⟹ *P b* = *Q b*) ⟹ *Bex B P* = *Bex B Q*
⟨*proof*⟩

lemma *is_reducible_lit_code*[*code*]: *RP.is_reducible_lit Ds C L* =
(∃ *D* ∈ *set Ds*. (∃ *L′* ∈ *set D*.
if is_pos L′ = *is_neg L then*
(*case match_term_list* [(*atm_of L′*, *atm_of L*)] *Map.empty of*
None ⇒ *False*
| *Some σ* ⇒ *subsumes_list* (*remove1 L′ D*) *C σ*)
else False))
⟨*proof*⟩

declare
Pairs_def[*folded sorted_list_of_set_def*, *code*]
linorder.sorted_list_of_set_sort_remdups[*OF class_linorder_compare*,

folded sorted_list_of_set_def sort_key_def, code]
linorder.sort_key_def[*OF class_linorder_compare, folded sort_key_def insort_key_def, code*]
linorder.insort_key.simps[*OF class_linorder_compare, folded insort_key_def, code*]

export-code *St0* **in** *SML*
export-code *deterministic_RP* **in** *SML* **module-name** *RP*

instantiation *nat* :: *weighted* **begin**
definition *weights_nat* :: *nat weights* **where** *weights_nat* =
(|*w* = *Suc* ∘ *prod_encode*, *w0* = *1*, *pr_strict* = λ(*f*, *n*) (*g*, *m*). *f* > *g* ∨ *f* = *g* ∧ *n* > *m*, *least* = λ*n*. *n* = *0*, *scf* = λ_ _. *1*|)

instance
⟨*proof*⟩
end

definition *prover* :: ((*nat*, *nat*) *Term.term literal list* × *nat*) *list* ⇒ *bool* **where**
prover N = (*case deterministic_RP* (*St0 N 0*) *of*
None ⇒ *True*
| *Some R* ⇒ [] ∉ *set R*)

theorem *prover_complete_refutation*: *prover N* ⟷ *satisfiable* (*RP.grounded_N0 N*)
⟨*proof*⟩

definition *string_literal_of_nat* :: *nat* ⇒ *String.literal* **where**
string_literal_of_nat n = *String.implode* (*show n*)

export-code *prover Fun Var Pos Neg string_literal_of_nat 0::nat Suc* **in** *SML* **module-name** *RPx*

abbreviation 𝔭 ≡ *Fun 42*
abbreviation 𝔞 ≡ *Fun 0* []
abbreviation 𝔟 ≡ *Fun 1* []
abbreviation 𝔠 ≡ *Fun 2* []
abbreviation *X* ≡ *Var 0*
abbreviation *Y* ≡ *Var 1*
abbreviation *Z* ≡ *Var 2*

value *prover*
([([*Neg* (𝔭[*X*,*Y*,*Z*]), *Pos* (𝔭[*Y*,*Z*,*X*])], *1*),
([*Pos* (𝔭[𝔠,𝔞,𝔟])], *1*),
([*Neg* (𝔭[𝔟,𝔠,𝔞])], *1*)]
:: ((*nat*, *nat*) *Term.term literal list* × *nat*) *list*)

value *prover*
([([*Pos* (𝔭[*X*,*Y*])], *1*), ([*Neg* (𝔭[*X*,*X*])], *1*)]
:: ((*nat*, *nat*) *Term.term literal list* × *nat*) *list*)

value *prover* ([([*Neg* (𝔭[*X*,*Y*,*Z*]), *Pos* (𝔭[*Y*,*Z*,*X*])], *1*)]
:: ((*nat*, *nat*) *Term.term literal list* × *nat*) *list*)

definition *mk_MSC015_1* :: *nat* ⇒ ((*nat*, *nat*) *Term.term literal list* × *nat*) *list* **where**
mk_MSC015_1 n =
(*let*
init = ([*Pos* (𝔭 (*replicate n* 𝔞))], *1*);
rules = *map* (λ*i*. ([*Neg* (𝔭 (*map Var* [*0* ..< *n* − *i* − *1*] @ 𝔞 # *replicate i* 𝔟)),
Pos (𝔭 (*map Var* [*0* ..< *n* − *i* − *1*] @ 𝔟 # *replicate i* 𝔞))], *1*)) [*0* ..< *n*];
goal = ([*Neg* (𝔭 (*replicate n* 𝔟))], *1*)
in init # *rules* @ [*goal*])

value *prover* (*mk_MSC015_1 1*)
value *prover* (*mk_MSC015_1 2*)
value *prover* (*mk_MSC015_1 3*)

```
value prover (mk_MSC015_1 4)
value prover (mk_MSC015_1 5)
value prover (mk_MSC015_1 10)

lemma
  assumes
    p a a a a a a a a a a a a a a
    (∀x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13.
      ¬ p x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 a ∨
      p x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 b)
    (∀x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12.
      ¬ p x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 a b ∨
      p x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 b a)
    (∀x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11.
      ¬ p x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 a b b ∨
      p x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 b a a)
    (∀x1 x2 x3 x4 x5 x6 x7 x8 x9 x10.
      ¬ p x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 a b b b ∨
      p x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 b a a a)
    (∀x1 x2 x3 x4 x5 x6 x7 x8 x9.
      ¬ p x1 x2 x3 x4 x5 x6 x7 x8 x9 a b b b b ∨
      p x1 x2 x3 x4 x5 x6 x7 x8 x9 b a a a a)
    (∀x1 x2 x3 x4 x5 x6 x7 x8.
      ¬ p x1 x2 x3 x4 x5 x6 x7 x8 a b b b b b ∨
      p x1 x2 x3 x4 x5 x6 x7 x8 b a a a a a)
    (∀x1 x2 x3 x4 x5 x6 x7.
      ¬ p x1 x2 x3 x4 x5 x6 x7 a b b b b b b ∨
      p x1 x2 x3 x4 x5 x6 x7 b a a a a a a)
    (∀x1 x2 x3 x4 x5 x6.
      ¬ p x1 x2 x3 x4 x5 x6 a b b b b b b b ∨
      p x1 x2 x3 x4 x5 x6 b a a a a a a a)
    (∀x1 x2 x3 x4 x5.
      ¬ p x1 x2 x3 x4 x5 a b b b b b b b b ∨
      p x1 x2 x3 x4 x5 b a a a a a a a a)
    (∀x1 x2 x3 x4.
      ¬ p x1 x2 x3 x4 a b b b b b b b b b ∨
      p x1 x2 x3 x4 b a a a a a a a a a)
    (∀x1 x2 x3.
      ¬ p x1 x2 x3 a b b b b b b b b b b ∨
      p x1 x2 x3 b a a a a a a a a a a)
    (∀x1 x2.
      ¬ p x1 x2 a b b b b b b b b b b b ∨
      p x1 x2 b a a a a a a a a a a a)
    (∀x1.
      ¬ p x1 a b b b b b b b b b b b b ∨
      p x1 b a a a a a a a a a a a a)
    (¬ p a b b b b b b b b b b b b b ∨
      p b a a a a a a a a a a a a a)
    ¬ p b b b b b b b b b b b b b b
  shows False
  ⟨proof⟩

end
```