

Formalization of Recursive Path Orders for Lambda-Free Higher-Order Terms

Jasmin Christian Blanchette, Uwe Waldmann, and Daniel Wand

April 18, 2024

Abstract

This Isabelle/HOL formalization defines recursive path orders (RPOs) for higher-order terms without λ -abstraction and proves many useful properties about them. The main order fully coincides with the standard RPO on first-order terms also in the presence of currying, distinguishing it from previous work. An optimized variant is formalized as well. It appears promising as the basis of a higher-order superposition calculus.

Contents

1	Introduction	1
2	Utilities for Lambda-Free Orders	1
2.1	Function Power	1
2.2	Least Operator	2
2.3	Antisymmetric Relations	2
2.4	Acyclic Relations	2
2.5	Reflexive, Transitive Closure	2
2.6	Well-Founded Relations	2
2.7	Wellorders	3
2.8	Lists	3
2.9	Extended Natural Numbers	4
2.10	Multisets	4
3	Lambda-Free Higher-Order Terms	4
3.1	Precedence on Symbols	5
3.2	Heads	5
3.3	Terms	5
3.4	hsize	8
3.5	Substitutions	8
3.6	Subterms	8
3.7	Maximum Arities	9
3.8	Potential Heads of Ground Instances of Variables	10
4	Infinite (Non-Well-Founded) Chains	12
5	Extension Orders	13
5.1	Locales	13
5.2	Lexicographic Extension	15
5.3	Reverse (Right-to-Left) Lexicographic Extension	17
5.4	Generic Length Extension	18
5.5	Length-Lexicographic Extension	19
5.6	Reverse (Right-to-Left) Length-Lexicographic Extension	20
5.7	Dershowitz–Manna Multiset Extension	21
5.8	Huet–Oppen Multiset Extension	23
5.9	Componentwise Extension	25

6	The Applicative Recursive Path Order for Lambda-Free Higher-Order Terms	26
7	The Graceful Recursive Path Order for Lambda-Free Higher-Order Terms	27
7.1	Setup	27
7.2	Inductive Definitions	28
7.3	Transitivity	28
7.4	Irreflexivity	28
7.5	Subterm Property	28
7.6	Compatibility with Functions	29
7.7	Compatibility with Arguments	29
7.8	Stability under Substitution	29
7.9	Totality on Ground Terms	29
7.10	Well-foundedness	29
8	The Optimized Graceful Recursive Path Order for Lambda-Free Higher-Order Terms	30
8.1	Setup	30
8.2	Definition of the Order	30
8.3	Transitivity	30
8.4	Conditional Equivalence with Unoptimized Version	30
9	An Encoding of Lambdas in Lambda-Free Higher-Order Logic	31
10	Recursive Path Orders for Lambda-Free Higher-Order Terms	32

1 Introduction

This Isabelle/HOL formalization defines recursive path orders (RPOs) for higher-order terms without λ -abstraction and proves many useful properties about them. The main order fully coincides with the standard RPO on first-order terms also in the presence of currying, distinguishing it from previous work. An optimized variant is formalized as well. It appears promising as the basis of a higher-order superposition calculus.

We refer to the following conference paper for details:

Jasmin Christian Blanchette, Uwe Waldmann, Daniel Wand:
 A Lambda-Free Higher-Order Recursive Path Order.
 FoSSaCS 2017: 461-479
https://www.cs.vu.nl/~jbe248/lambda_free_rpo_conf.pdf

2 Utilities for Lambda-Free Orders

```
theory Lambda_Free_Util
imports HOL-Library.Extended_Nat HOL-Library.Multiset_Order
begin
```

This theory gathers various lemmas that likely belong elsewhere in Isabelle or the *Archive of Formal Proofs*. Most (but certainly not all) of them are used to formalize orders on λ -free higher-order terms.

2.1 Function Power

```
lemma funpow_lesseq_iter:
  fixes f :: ('a::order)  $\Rightarrow$  'a
  assumes mono:  $\bigwedge k. k \leq f k$  and m_le_n:  $m \leq n$ 
  shows  $(f \hat{\sim} m) k \leq (f \hat{\sim} n) k$ 
  <proof>
```

```
lemma funpow_less_iter:
  fixes f :: ('a::order)  $\Rightarrow$  'a
  assumes mono:  $\bigwedge k. k < f k$  and m_lt_n:  $m < n$ 
  shows  $(f \hat{\sim} m) k < (f \hat{\sim} n) k$ 
  <proof>
```

2.2 Least Operator

lemma *Least_eq[simp]*: $(LEAST\ y.\ y = x) = x$ and $(LEAST\ y.\ x = y) = x$ for $x :: 'a::order$
<proof>

lemma *Least_in_nonempty_set_imp_ex*:
fixes $f :: 'b \Rightarrow ('a::wellorder)$
assumes
 $A_nemp: A \neq \{\}$ and
 $P_least: P (LEAST\ y.\ \exists x \in A.\ y = f\ x)$
shows $\exists x \in A.\ P (f\ x)$
<proof>

lemma *Least_eq_0_enat*: $P\ 0 \implies (LEAST\ x :: enat.\ P\ x) = 0$
<proof>

2.3 Antisymmetric Relations

lemma *irrefl_trans_imp_antisym*: $irrefl\ r \implies trans\ r \implies antisym\ r$
<proof>

lemma *irreflp_transp_imp_antisymP*: $irreflp\ p \implies transp\ p \implies antisymp\ p$
<proof>

2.4 Acyclic Relations

lemma *finite_nonempty_ex_succ_imp_cyclic*:
assumes
 $fin: finite\ A$ and
 $nemp: A \neq \{\}$ and
 $ex_y: \forall x \in A.\ \exists y \in A.\ (y, x) \in r$
shows $\neg acyclic\ r$
<proof>

2.5 Reflexive, Transitive Closure

lemma *relcomp_subset_left_imp_relcomp_trancl_subset_left*:
assumes $sub: R\ O\ S \subseteq R$
shows $R\ O\ S^* \subseteq R$
<proof>

lemma *f_chain_in_rtrancl*:
assumes $m_le_n: m \leq n$ and $f_chain: \forall i \in \{m..<n\}.\ (f\ i, f\ (Suc\ i)) \in R$
shows $(f\ m, f\ n) \in R^*$
<proof>

lemma *f_rev_chain_in_rtrancl*:
assumes $m_le_n: m \leq n$ and $f_chain: \forall i \in \{m..<n\}.\ (f\ (Suc\ i), f\ i) \in R$
shows $(f\ n, f\ m) \in R^*$
<proof>

2.6 Well-Founded Relations

lemma *wf_app*: $wf\ r \implies wf\ \{(x, y).\ (f\ x, f\ y) \in r\}$
<proof>

lemma *wfP_app*: $wfP\ p \implies wfP\ (\lambda x\ y.\ p\ (f\ x)\ (f\ y))$
<proof>

lemma *wf_exists_minimal*:
assumes $wf: wf\ r$ and $Q: Q\ x$
shows $\exists x.\ Q\ x \wedge (\forall y.\ (f\ y, f\ x) \in r \longrightarrow \neg Q\ y)$
<proof>

lemma *wfP_exists_minimal*:
assumes *wf*: $wfP\ p$ **and** *Q*: $Q\ x$
shows $\exists x. Q\ x \wedge (\forall y. p\ (f\ y)\ (f\ x) \longrightarrow \neg Q\ y)$
<proof>

lemma *finite_irrefl_trans_imp_wf*: $finite\ r \Longrightarrow irrefl\ r \Longrightarrow trans\ r \Longrightarrow wf\ r$
<proof>

lemma *finite_irreflp_transp_imp_wfp*:
 $finite\ \{(x, y). p\ x\ y\} \Longrightarrow irreflp\ p \Longrightarrow transp\ p \Longrightarrow wfP\ p$
<proof>

lemma *wf_infinite_down_chain_compatible*:
assumes
wf_R: $wf\ R$ **and**
inf_chain_RS: $\forall i. (f\ (Suc\ i), f\ i) \in R \cup S$ **and**
O_subset: $R\ O\ S \subseteq R$
shows $\exists k. \forall i. (f\ (Suc\ (i + k)), f\ (i + k)) \in S$
<proof>

2.7 Wellorders

lemma (*in_wellorder*) *exists_minimal*:
fixes $x :: 'a$
assumes $P\ x$
shows $\exists x. P\ x \wedge (\forall y. P\ y \longrightarrow y \geq x)$
<proof>

2.8 Lists

lemma *rev_induct2*[*consumes 1, case_names Nil snoc*]:
 $length\ xs = length\ ys \Longrightarrow P\ []\ [] \Longrightarrow$
 $(\bigwedge x\ xs\ y\ ys. length\ xs = length\ ys \Longrightarrow P\ xs\ ys \Longrightarrow P\ (xs\ @\ [x])\ (ys\ @\ [y])) \Longrightarrow P\ xs\ ys$
<proof>

lemma *hd_in_set*: $length\ xs \neq 0 \Longrightarrow hd\ xs \in set\ xs$
<proof>

lemma *in_lists_iff_set*: $xs \in lists\ A \longleftrightarrow set\ xs \subseteq A$
<proof>

lemma *butlast_append_Cons*[*simp*]: $butlast\ (xs\ @\ y\ \# \ ys) = xs\ @\ butlast\ (y\ \# \ ys)$
<proof>

lemma *rev_in_lists*[*simp*]: $rev\ xs \in lists\ A \longleftrightarrow xs \in lists\ A$
<proof>

lemma *hd_le_sum_list*:
fixes $xs :: 'a::ordered_ab_semigroup_monoid_add_imp_le\ list$
assumes $xs \neq []$ **and** $\forall i < length\ xs. xs\ !\ i \geq 0$
shows $hd\ xs \leq sum_list\ xs$
<proof>

lemma *sum_list_ge_length_times*:
fixes $a :: 'a::\{ordered_ab_semigroup_add,semiring_1\}$
assumes $\forall i < length\ xs. xs\ !\ i \geq a$
shows $sum_list\ xs \geq of_nat\ (length\ xs) * a$
<proof>

lemma *prod_list_nonneg*:
fixes $xs :: ('a :: \{ordered_semiring_0,linordered_nonzero_semiring\})\ list$
assumes $\bigwedge x. x \in set\ xs \Longrightarrow x \geq 0$
shows $prod_list\ xs \geq 0$
<proof>

lemma *zip_append_0_upt*:
 $zip\ (xs\ @\ ys)\ [0..<length\ xs + length\ ys] =$
 $zip\ xs\ [0..<length\ xs]\ @\ zip\ ys\ [length\ xs..<length\ xs + length\ ys]$
 ⟨proof⟩

lemma *zip_eq_butlast_last*:
assumes *len_gt0*: $length\ xs > 0$ **and** *len_eq*: $length\ xs = length\ ys$
shows $zip\ xs\ ys = zip\ (butlast\ xs)\ (butlast\ ys)\ @\ [(last\ xs,\ last\ ys)]$
 ⟨proof⟩

2.9 Extended Natural Numbers

lemma *the_enat_0[simp]*: $the_enat\ 0 = 0$
 ⟨proof⟩

lemma *the_enat_1[simp]*: $the_enat\ 1 = 1$
 ⟨proof⟩

lemma *enat_le_minus_1_imp_lt*: $m \leq n - 1 \implies n \neq \infty \implies n \neq 0 \implies m < n$ **for** $m\ n :: enat$
 ⟨proof⟩

lemma *enat_diff_diff_eq*: $k - m - n = k - (m + n)$ **for** $k\ m\ n :: enat$
 ⟨proof⟩

lemma *enat_sub_add_same[intro]*: $n \leq m \implies m = m - n + n$ **for** $m\ n :: enat$
 ⟨proof⟩

lemma *enat_the_enat_iden[simp]*: $n \neq \infty \implies enat\ (the_enat\ n) = n$
 ⟨proof⟩

lemma *the_enat_minus_nat*: $m \neq \infty \implies the_enat\ (m - enat\ n) = the_enat\ m - n$
 ⟨proof⟩

lemma *enat_the_enat_le*: $enat\ (the_enat\ x) \leq x$
 ⟨proof⟩

lemma *enat_the_enat_minus_le*: $enat\ (the_enat\ (x - y)) \leq x$
 ⟨proof⟩

lemma *enat_le_imp_minus_le*: $k \leq m \implies k - n \leq m$ **for** $k\ m\ n :: enat$
 ⟨proof⟩

lemma *add_diff_assoc2_enat*: $m \geq n \implies m - n + p = m + p - n$ **for** $m\ n\ p :: enat$
 ⟨proof⟩

lemma *enat_mult_minus_distrib*: $enat\ x * (y - z) = enat\ x * y - enat\ x * z$
 ⟨proof⟩

2.10 Multisets

declare
filter_eq_replicate_mset [simp]
image_mset_subseteq_mono [intro]
count_gt_imp_in_mset [intro]

end

3 Lambda-Free Higher-Order Terms

theory *Lambda_Free_Term*
imports *Lambda_Free_Util*
abbrevs $>s = >_s$

```

and >h = >hd
and <=>h = <=>hd
begin

```

This theory defines λ -free higher-order terms and related locales.

3.1 Precedence on Symbols

```

locale gt_sym =
  fixes
    gt_sym :: 's  $\Rightarrow$  's  $\Rightarrow$  bool (infix >s 50)
  assumes
    gt_sym_irrefl:  $\neg f >_s f$  and
    gt_sym_trans:  $h >_s g \Longrightarrow g >_s f \Longrightarrow h >_s f$  and
    gt_sym_total:  $f >_s g \vee g >_s f \vee g = f$  and
    gt_sym_wf: wfP ( $\lambda f g. g >_s f$ )
begin

```

```

lemma gt_sym_antisym:  $f >_s g \Longrightarrow \neg g >_s f$ 
  <proof>

```

```

end

```

3.2 Heads

```

datatype (plugins del: size) (syms_hd: 's, vars_hd: 'v) hd =
  is_Var: Var (var: 'v)
| Sym (sym: 's)

```

```

abbreviation is_Sym :: ('s, 'v) hd  $\Rightarrow$  bool where
  is_Sym  $\zeta \equiv \neg$  is_Var  $\zeta$ 

```

```

lemma finite_vars_hd[simp]: finite (vars_hd  $\zeta$ )
  <proof>

```

```

lemma finite_syms_hd[simp]: finite (syms_hd  $\zeta$ )
  <proof>

```

3.3 Terms

```

consts head0 :: 'a

```

```

datatype (syms: 's, vars: 'v) tm =
  is_Hd: Hd (head: ('s, 'v) hd)
| App (fun: ('s, 'v) tm) (arg: ('s, 'v) tm)
where
  head (App s _) = head0 s
| fun (Hd  $\zeta$ ) = Hd  $\zeta$ 
| arg (Hd  $\zeta$ ) = Hd  $\zeta$ 

```

```

overloading head0  $\equiv$  head0 :: ('s, 'v) tm  $\Rightarrow$  ('s, 'v) hd
begin

```

```

primrec head0 :: ('s, 'v) tm  $\Rightarrow$  ('s, 'v) hd where
  head0 (Hd  $\zeta$ ) =  $\zeta$ 
| head0 (App s _) = head0 s

```

```

end

```

```

lemma head_App[simp]: head (App s t) = head s
  <proof>

```

```

declare tm.sel(2)[simp del]

```

lemma *head_fun[simp]*: $\text{head} (\text{fun } s) = \text{head } s$
<proof>

abbreviation *ground* :: $('s, 'v) \text{ tm} \Rightarrow \text{bool}$ **where**
ground t $\equiv \text{vars } t = \{\}$

abbreviation *is_App* :: $('s, 'v) \text{ tm} \Rightarrow \text{bool}$ **where**
is_App s $\equiv \neg \text{is_Hd } s$

lemma
size_fun_lt: $\text{is_App } s \Longrightarrow \text{size} (\text{fun } s) < \text{size } s$ **and**
size_arg_lt: $\text{is_App } s \Longrightarrow \text{size} (\text{arg } s) < \text{size } s$
<proof>

lemma
finite_vars[simp]: $\text{finite} (\text{vars } s)$ **and**
finite_syms[simp]: $\text{finite} (\text{syms } s)$
<proof>

lemma
vars_head_subseteq: $\text{vars_hd} (\text{head } s) \subseteq \text{vars } s$ **and**
syms_head_subseteq: $\text{syms_hd} (\text{head } s) \subseteq \text{syms } s$
<proof>

fun *args* :: $('s, 'v) \text{ tm} \Rightarrow ('s, 'v) \text{ tm list}$ **where**
args (Hd _) = []
| *args (App s t)* = *args s* @ [*t*]

lemma *set_args_fun*: $\text{set} (\text{args} (\text{fun } s)) \subseteq \text{set} (\text{args } s)$
<proof>

lemma *arg_in_args*: $\text{is_App } s \Longrightarrow \text{arg } s \in \text{set} (\text{args } s)$
<proof>

lemma
vars_args_subseteq: $si \in \text{set} (\text{args } s) \Longrightarrow \text{vars } si \subseteq \text{vars } s$ **and**
syms_args_subseteq: $si \in \text{set} (\text{args } s) \Longrightarrow \text{syms } si \subseteq \text{syms } s$
<proof>

lemma *args_Nil_iff_is_Hd*: $\text{args } s = [] \longleftrightarrow \text{is_Hd } s$
<proof>

abbreviation *num_args* :: $('s, 'v) \text{ tm} \Rightarrow \text{nat}$ **where**
num_args s $\equiv \text{length} (\text{args } s)$

lemma *size_ge_num_args*: $\text{size } s \geq \text{num_args } s$
<proof>

lemma *Hd_head_id*: $\text{num_args } s = 0 \Longrightarrow \text{Hd} (\text{head } s) = s$
<proof>

lemma *one_arg_imp_Hd*: $\text{num_args } s = 1 \Longrightarrow s = \text{App } t \ u \Longrightarrow t = \text{Hd} (\text{head } t)$
<proof>

lemma *size_in_args*: $s \in \text{set} (\text{args } t) \Longrightarrow \text{size } s < \text{size } t$
<proof>

primrec *apps* :: $('s, 'v) \text{ tm} \Rightarrow ('s, 'v) \text{ tm list} \Rightarrow ('s, 'v) \text{ tm}$ **where**
apps s [] = *s*
| *apps s (t # ts)* = *apps (App s t) ts*

lemma
vars_apps[simp]: $\text{vars} (\text{apps } s \ ss) = \text{vars } s \cup (\bigcup s \in \text{set } ss. \text{vars } s)$ **and**

syms_apps[simp]: $\text{syms } (\text{apps } s \text{ } ss) = \text{syms } s \cup (\bigcup s \in \text{set } ss. \text{syms } s)$ **and**
head_apps[simp]: $\text{head } (\text{apps } s \text{ } ss) = \text{head } s$ **and**
args_apps[simp]: $\text{args } (\text{apps } s \text{ } ss) = \text{args } s @ ss$ **and**
is_App_apps[simp]: $\text{is_App } (\text{apps } s \text{ } ss) \longleftrightarrow \text{args } (\text{apps } s \text{ } ss) \neq []$ **and**
fun_apps_Nil[simp]: $\text{fun } (\text{apps } s \text{ } []) = \text{fun } s$ **and**
fun_apps_Cons[simp]: $\text{fun } (\text{apps } (\text{App } s \text{ } sa) \text{ } ss) = \text{apps } s (\text{butlast } (sa \# ss))$ **and**
arg_apps_Nil[simp]: $\text{arg } (\text{apps } s \text{ } []) = \text{arg } s$ **and**
arg_apps_Cons[simp]: $\text{arg } (\text{apps } (\text{App } s \text{ } sa) \text{ } ss) = \text{last } (sa \# ss)$
 ⟨proof⟩

lemma *apps_append[simp]*: $\text{apps } s (ss @ ts) = \text{apps } (\text{apps } s \text{ } ss) \text{ } ts$
 ⟨proof⟩

lemma *App_apps*: $\text{App } (\text{apps } s \text{ } ts) \text{ } t = \text{apps } s (ts @ [t])$
 ⟨proof⟩

lemma *tm_inject_apps[iff, induct_simp]*: $\text{apps } (\text{Hd } \zeta) \text{ } ss = \text{apps } (\text{Hd } \xi) \text{ } ts \longleftrightarrow \zeta = \xi \wedge ss = ts$
 ⟨proof⟩

lemma *tm_collapse_apps[simp]*: $\text{apps } (\text{Hd } (\text{head } s)) (\text{args } s) = s$
 ⟨proof⟩

lemma *tm_expand_apps*: $\text{head } s = \text{head } t \implies \text{args } s = \text{args } t \implies s = t$
 ⟨proof⟩

lemma *tm_exhaust_apps_sel[case_names apps]*: $(s = \text{apps } (\text{Hd } (\text{head } s)) (\text{args } s) \implies P) \implies P$
 ⟨proof⟩

lemma *tm_exhaust_apps[case_names apps]*: $(\bigwedge \zeta \text{ } ss. s = \text{apps } (\text{Hd } \zeta) \text{ } ss \implies P) \implies P$
 ⟨proof⟩

lemma *tm_induct_apps[case_names apps]*:
assumes $\bigwedge \zeta \text{ } ss. (\bigwedge s. s \in \text{set } ss \implies P \text{ } s) \implies P (\text{apps } (\text{Hd } \zeta) \text{ } ss)$
shows $P \text{ } s$
 ⟨proof⟩

lemma
ground_fun: $\text{ground } s \implies \text{ground } (\text{fun } s)$ **and**
ground_arg: $\text{ground } s \implies \text{ground } (\text{arg } s)$
 ⟨proof⟩

lemma *ground_head*: $\text{ground } s \implies \text{is_Sym } (\text{head } s)$
 ⟨proof⟩

lemma *ground_args*: $t \in \text{set } (\text{args } s) \implies \text{ground } s \implies \text{ground } t$
 ⟨proof⟩

primrec *vars_mset* :: $('s, 'v) \text{tm} \Rightarrow 'v \text{multiset}$ **where**
vars_mset $(\text{Hd } \zeta) = \text{mset_set } (\text{vars_hd } \zeta)$
 | *vars_mset* $(\text{App } s \text{ } t) = \text{vars_mset } s + \text{vars_mset } t$

lemma *set_vars_mset[simp]*: $\text{set_mset } (\text{vars_mset } t) = \text{vars } t$
 ⟨proof⟩

lemma *vars_mset_empty_iff[iff]*: $\text{vars_mset } s = \{\#\} \longleftrightarrow \text{ground } s$
 ⟨proof⟩

lemma *vars_mset_fun[intro]*: $\text{vars_mset } (\text{fun } t) \subseteq\# \text{vars_mset } t$
 ⟨proof⟩

lemma *vars_mset_arg[intro]*: $\text{vars_mset } (\text{arg } t) \subseteq\# \text{vars_mset } t$
 ⟨proof⟩

3.4 hsize

The hsize of a term is the number of heads (Syms or Vars) in the term.

primrec $hsize :: ('s, 'v) tm \Rightarrow nat$ **where**
 $hsize (Hd \zeta) = 1$
 $| hsize (App s t) = hsize s + hsize t$

lemma $hsize_size: hsize t * 2 = size t + 1$
 $\langle proof \rangle$

lemma $hsize_pos[simp]: hsize t > 0$
 $\langle proof \rangle$

lemma $hsize_fun_lt: is_App s \Longrightarrow hsize (fun s) < hsize s$
 $\langle proof \rangle$

lemma $hsize_arg_lt: is_App s \Longrightarrow hsize (arg s) < hsize s$
 $\langle proof \rangle$

lemma $hsize_ge_num_args: hsize s \geq hsize s$
 $\langle proof \rangle$

lemma $hsize_in_args: s \in set (args t) \Longrightarrow hsize s < hsize t$
 $\langle proof \rangle$

lemma $hsize_apps: hsize (apps t ts) = hsize t + sum_list (map hsize ts)$
 $\langle proof \rangle$

lemma $hsize_args: 1 + sum_list (map hsize (args t)) = hsize t$
 $\langle proof \rangle$

3.5 Substitutions

primrec $subst :: ('v \Rightarrow ('s, 'v) tm) \Rightarrow ('s, 'v) tm \Rightarrow ('s, 'v) tm$ **where**
 $subst \varrho (Hd \zeta) = (case \zeta of Var x \Rightarrow \varrho x | Sym f \Rightarrow Hd (Sym f))$
 $| subst \varrho (App s t) = App (subst \varrho s) (subst \varrho t)$

lemma $subst_apps[simp]: subst \varrho (apps s ts) = apps (subst \varrho s) (map (subst \varrho) ts)$
 $\langle proof \rangle$

lemma $head_subst[simp]: head (subst \varrho s) = head (subst \varrho (Hd (head s)))$
 $\langle proof \rangle$

lemma $args_subst[simp]:$
 $args (subst \varrho s) = (case head s of Var x \Rightarrow args (\varrho x) | Sym f \Rightarrow []) @ map (subst \varrho) (args s)$
 $\langle proof \rangle$

lemma $ground_imp_subst_iden: ground s \Longrightarrow subst \varrho s = s$
 $\langle proof \rangle$

lemma $vars_mset_subst[simp]: vars_mset (subst \varrho s) = (\sum \# \{ \#vars_mset (\varrho x). x \in \# vars_mset s \# \})$
 $\langle proof \rangle$

lemma $vars_mset_subst_subsetq:$
 $vars_mset t \supseteq \# vars_mset s \Longrightarrow vars_mset (subst \varrho t) \supseteq \# vars_mset (subst \varrho s)$
 $\langle proof \rangle$

lemma $vars_subst_subsetq: vars t \supseteq vars s \Longrightarrow vars (subst \varrho t) \supseteq vars (subst \varrho s)$
 $\langle proof \rangle$

3.6 Subterms

inductive $sub :: ('s, 'v) tm \Rightarrow ('s, 'v) tm \Rightarrow bool$ **where**

$sub_refl: sub\ s\ s$
 $|\ sub_fun: sub\ s\ t \implies sub\ s\ (App\ u\ t)$
 $|\ sub_arg: sub\ s\ t \implies sub\ s\ (App\ t\ u)$

inductive-cases $sub_HdE[simplified, elim]: sub\ s\ (Hd\ \xi)$
inductive-cases $sub_AppE[simplified, elim]: sub\ s\ (App\ t\ u)$
inductive-cases $sub_Hd_HdE[simplified, elim]: sub\ (Hd\ \zeta)\ (Hd\ \xi)$
inductive-cases $sub_Hd_AppE[simplified, elim]: sub\ (Hd\ \zeta)\ (App\ t\ u)$

lemma $in_vars_imp_sub: x \in vars\ s \iff sub\ (Hd\ (Var\ x))\ s$
 $\langle proof \rangle$

lemma $sub_args: s \in set\ (args\ t) \implies sub\ s\ t$
 $\langle proof \rangle$

lemma $sub_size: sub\ s\ t \implies size\ s \leq size\ t$
 $\langle proof \rangle$

lemma $sub_subst: sub\ s\ t \implies sub\ (subst\ \rho\ s)\ (subst\ \rho\ t)$
 $\langle proof \rangle$

abbreviation $proper_sub :: ('s, 'v)\ tm \Rightarrow ('s, 'v)\ tm \Rightarrow bool$ **where**
 $proper_sub\ s\ t \equiv sub\ s\ t \wedge s \neq t$

lemma $proper_sub_Hd[simp]: \neg\ proper_sub\ s\ (Hd\ \zeta)$
 $\langle proof \rangle$

lemma $proper_sub_subst:$
assumes $psub: proper_sub\ s\ t$
shows $proper_sub\ (subst\ \rho\ s)\ (subst\ \rho\ t)$
 $\langle proof \rangle$

3.7 Maximum Arities

locale $arity =$
fixes

$arity_sym :: 's \Rightarrow enat$ **and**
 $arity_var :: 'v \Rightarrow enat$

begin

primrec $arity_hd :: ('s, 'v)\ hd \Rightarrow enat$ **where**
 $arity_hd\ (Var\ x) = arity_var\ x$
 $|\ arity_hd\ (Sym\ f) = arity_sym\ f$

definition $arity :: ('s, 'v)\ tm \Rightarrow enat$ **where**
 $arity\ s = arity_hd\ (head\ s) - num_args\ s$

lemma $arity_simps[simp]:$
 $arity\ (Hd\ \zeta) = arity_hd\ \zeta$
 $arity\ (App\ s\ t) = arity\ s - 1$
 $\langle proof \rangle$

lemma $arity_apps[simp]: arity\ (apps\ s\ ts) = arity\ s - length\ ts$
 $\langle proof \rangle$

inductive $wary :: ('s, 'v)\ tm \Rightarrow bool$ **where**
 $wary_Hd[intro]: wary\ (Hd\ \zeta)$
 $|\ wary_App[intro]: wary\ s \implies wary\ t \implies num_args\ s < arity_hd\ (head\ s) \implies wary\ (App\ s\ t)$

inductive-cases $wary_HdE: wary\ (Hd\ \zeta)$
inductive-cases $wary_AppE: wary\ (App\ s\ t)$
inductive-cases $wary_binaryE[simplified]: wary\ (App\ (App\ s\ t)\ u)$

lemma $wary_fun[intro]: wary\ t \implies wary\ (fun\ t)$

<proof>

lemma *wary_arg*[intro]: $wary\ t \implies wary\ (arg\ t)$
<proof>

lemma *wary_args*: $s \in set\ (args\ t) \implies wary\ t \implies wary\ s$
<proof>

lemma *wary_sub*: $sub\ s\ t \implies wary\ t \implies wary\ s$
<proof>

lemma *wary_inf_ary*: $(\bigwedge \zeta. arity_hd\ \zeta = \infty) \implies wary\ s$
<proof>

lemma *wary_num_args_le_arity_head*: $wary\ s \implies num_args\ s \leq arity_hd\ (head\ s)$
<proof>

lemma *wary_apps*:
 $wary\ s \implies (\bigwedge sa. sa \in set\ ss \implies wary\ sa) \implies length\ ss \leq arity\ s \implies wary\ (apps\ s\ ss)$
<proof>

lemma *wary_cases_apps*[consumes 1, case_names apps]:
assumes
 wary_t: $wary\ t$ **and**
 apps: $\bigwedge \zeta. t = apps\ (Hd\ \zeta)\ ss \implies (\bigwedge sa. sa \in set\ ss \implies wary\ sa) \implies length\ ss \leq arity_hd\ \zeta \implies P$
shows *P*
<proof>

lemma *arity_hd_head*: $wary\ s \implies arity_hd\ (head\ s) = arity\ s + num_args\ s$
<proof>

lemma *arity_head_ge*: $arity_hd\ (head\ s) \geq arity\ s$
<proof>

inductive *wary_fo* :: $('s, 'v)\ tm \Rightarrow bool$ **where**
wary_foI[intro]: $is_Hd\ s \vee is_Sym\ (head\ s) \implies length\ (args\ s) = arity_hd\ (head\ s) \implies$
 $(\forall t \in set\ (args\ s). wary_fo\ t) \implies wary_fo\ s$

lemma *wary_fo_args*: $s \in set\ (args\ t) \implies wary_fo\ t \implies wary_fo\ s$
<proof>

lemma *wary_fo_arg*: $wary_fo\ (App\ s\ t) \implies wary_fo\ t$
<proof>

end

3.8 Potential Heads of Ground Instances of Variables

locale *ground_heads* = *gt_sym* ($>_s$) + *arity_arity_sym* *arity_var*
for
 gt_sym :: $'s \Rightarrow 's \Rightarrow bool$ (**infix** $>_s\ 50$) **and**
 arity_sym :: $'s \Rightarrow enat$ **and**
 arity_var :: $'v \Rightarrow enat$ +
fixes
 ground_heads_var :: $'v \Rightarrow 's\ set$
assumes
 ground_heads_var_arity: $f \in ground_heads_var\ x \implies arity_sym\ f \geq arity_var\ x$ **and**
 ground_heads_var_nonempty: $ground_heads_var\ x \neq \{\}$
begin

primrec *ground_heads* :: $('s, 'v)\ hd \Rightarrow 's\ set$ **where**
 ground_heads (Var *x*) = *ground_heads_var* *x*
| *ground_heads* (Sym *f*) = {*f*}

lemma *ground_heads_arity*: $f \in \text{ground_heads } \zeta \implies \text{arity_sym } f \geq \text{arity_hd } \zeta$
 ⟨proof⟩

lemma *ground_heads_nonempty[simp]*: $\text{ground_heads } \zeta \neq \{\}$
 ⟨proof⟩

lemma *sym_in_ground_heads*: $\text{is_Sym } \zeta \implies \text{sym } \zeta \in \text{ground_heads } \zeta$
 ⟨proof⟩

lemma *ground_hd_in_ground_heads*: $\text{ground } s \implies \text{sym } (\text{head } s) \in \text{ground_heads } (\text{head } s)$
 ⟨proof⟩

lemma *some_ground_head_arity*: $\text{arity_sym } (\text{SOME } f. f \in \text{ground_heads } (\text{Var } x)) \geq \text{arity_var } x$
 ⟨proof⟩

definition *wary_subst* :: $('v \Rightarrow ('s, 'v) \text{tm}) \Rightarrow \text{bool}$ **where**
 $\text{wary_subst } \varrho \longleftrightarrow$
 $(\forall x. \text{wary } (\varrho x) \wedge \text{arity } (\varrho x) \geq \text{arity_var } x \wedge \text{ground_heads } (\text{head } (\varrho x)) \subseteq \text{ground_heads_var } x)$

definition *strict_wary_subst* :: $('v \Rightarrow ('s, 'v) \text{tm}) \Rightarrow \text{bool}$ **where**
 $\text{strict_wary_subst } \varrho \longleftrightarrow$
 $(\forall x. \text{wary } (\varrho x) \wedge \text{arity } (\varrho x) \in \{\text{arity_var } x, \infty\}$
 $\wedge \text{ground_heads } (\text{head } (\varrho x)) \subseteq \text{ground_heads_var } x)$

lemma *strict_imp_wary_subst*: $\text{strict_wary_subst } \varrho \implies \text{wary_subst } \varrho$
 ⟨proof⟩

lemma *wary_subst_wary*:
assumes $\text{wary_}\varrho$: $\text{wary_subst } \varrho$ **and** $\text{wary_}s$: $\text{wary } s$
shows $\text{wary } (\text{subst } \varrho s)$
 ⟨proof⟩

lemmas $\text{strict_wary_subst_wary} = \text{wary_subst_wary}[\text{OF } \text{strict_imp_wary_subst}]$

lemma *wary_subst_ground_heads*:
assumes $\text{wary_}\varrho$: $\text{wary_subst } \varrho$
shows $\text{ground_heads } (\text{head } (\text{subst } \varrho s)) \subseteq \text{ground_heads } (\text{head } s)$
 ⟨proof⟩

lemmas $\text{strict_wary_subst_ground_heads} = \text{wary_subst_ground_heads}[\text{OF } \text{strict_imp_wary_subst}]$

definition *grounding_* ϱ :: $'v \Rightarrow ('s, 'v) \text{tm}$ **where**
 $\text{grounding_}\varrho x = (\text{let } s = \text{Hd } (\text{Sym } (\text{SOME } f. f \in \text{ground_heads_var } x)) \text{ in}$
 $\text{apps } s (\text{replicate } (\text{the_enat } (\text{arity } s - \text{arity_var } x)) s))$

lemma *ground_grounding_* ϱ : $\text{ground } (\text{subst } \text{grounding_}\varrho s)$
 ⟨proof⟩

lemma *strict_wary_grounding_* ϱ : $\text{strict_wary_subst } \text{grounding_}\varrho$
 ⟨proof⟩

lemmas $\text{wary_grounding_}\varrho = \text{strict_wary_grounding_}\varrho[\text{THEN } \text{strict_imp_wary_subst}]$

definition *gt_hd* :: $('s, 'v) \text{hd} \Rightarrow ('s, 'v) \text{hd} \Rightarrow \text{bool}$ (**infix** $>_{hd}$ 50) **where**
 $\xi >_{hd} \zeta \longleftrightarrow (\forall g \in \text{ground_heads } \xi. \forall f \in \text{ground_heads } \zeta. g >_s f)$

definition *comp_hd* :: $('s, 'v) \text{hd} \Rightarrow ('s, 'v) \text{hd} \Rightarrow \text{bool}$ (**infix** \leq_{hd} 50) **where**
 $\xi \leq_{hd} \zeta \longleftrightarrow \xi = \zeta \vee \xi >_{hd} \zeta \vee \zeta >_{hd} \xi$

lemma *gt_hd_irrefl*: $\neg \zeta >_{hd} \zeta$
 ⟨proof⟩

lemma *gt_hd_trans*: $\chi >_{hd} \xi \implies \xi >_{hd} \zeta \implies \chi >_{hd} \zeta$

<proof>

lemma *gt_sym_imp_hd*: $g >_s f \implies \text{Sym } g >_{hd} \text{Sym } f$
<proof>

lemma *not_comp_hd_imp_Var*: $\neg \xi \leq_{hd} \zeta \implies \text{is_Var } \zeta \vee \text{is_Var } \xi$
<proof>

end

end

4 Infinite (Non-Well-Founded) Chains

theory *Infinite_Chain*

imports *Lambda_Free_Util*

begin

This theory defines the concept of a minimal bad (or non-well-founded) infinite chain, as found in the term rewriting literature to prove the well-foundedness of syntactic term orders.

context

fixes $p :: 'a \Rightarrow 'a \Rightarrow \text{bool}$

begin

definition *inf_chain* :: $(\text{nat} \Rightarrow 'a) \Rightarrow \text{bool}$ **where**
 $\text{inf_chain } f \longleftrightarrow (\forall i. p (f i) (f (\text{Suc } i)))$

lemma *wfP_iff_no_inf_chain*: $\text{wfP } (\lambda x y. p y x) \longleftrightarrow (\nexists f. \text{inf_chain } f)$
<proof>

lemma *inf_chain_offset*: $\text{inf_chain } f \implies \text{inf_chain } (\lambda j. f (j + i))$
<proof>

definition *bad* :: $'a \Rightarrow \text{bool}$ **where**
 $\text{bad } x \longleftrightarrow (\exists f. \text{inf_chain } f \wedge f 0 = x)$

lemma *inf_chain_bad*:
assumes $\text{bad}_f: \text{inf_chain } f$
shows $\text{bad } (f i)$
<proof>

context

fixes $gt :: 'a \Rightarrow 'a \Rightarrow \text{bool}$

assumes $\text{wf}: \text{wf } \{(x, y). gt y x\}$

begin

primrec *worst_chain* :: $\text{nat} \Rightarrow 'a$ **where**
 $\text{worst_chain } 0 = (\text{SOME } x. \text{bad } x \wedge (\forall y. \text{bad } y \longrightarrow \neg gt x y))$
 $|\ \text{worst_chain } (\text{Suc } i) = (\text{SOME } x. \text{bad } x \wedge p (\text{worst_chain } i) x \wedge$
 $(\forall y. \text{bad } y \wedge p (\text{worst_chain } i) y \longrightarrow \neg gt x y))$

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

context

fixes $x :: 'a$

assumes $x_bad: \text{bad } x$

begin

lemma

$\text{bad_worst_chain}_0: \text{bad } (\text{worst_chain } 0)$ **and**
 $\text{min_worst_chain}_0: \neg gt (\text{worst_chain } 0) x$
<proof>

lemma
bad_worst_chain_Suc: *bad* (*worst_chain* (*Suc* *i*)) **and**
worst_chain_pred: *p* (*worst_chain* *i*) (*worst_chain* (*Suc* *i*)) **and**
min_worst_chain_Suc: *p* (*worst_chain* *i*) *x* $\implies \neg$ *gt* (*worst_chain* (*Suc* *i*)) *x*
⟨*proof*⟩

lemma *bad_worst_chain*: *bad* (*worst_chain* *i*)
⟨*proof*⟩

lemma *worst_chain_bad*: *inf_chain* *worst_chain*
⟨*proof*⟩

end

context
fixes *x* :: '*a*
assumes
x_bad: *bad* *x* **and**
p_trans: $\bigwedge z y x. p z y \implies p y x \implies p z x$
begin

lemma *worst_chain_not_gt*: \neg *gt* (*worst_chain* *i*) (*worst_chain* (*Suc* *i*)) **for** *i*
⟨*proof*⟩

end

end

end

lemma *inf_chain_subset*: *inf_chain* *p* *f* $\implies p \leq q \implies$ *inf_chain* *q* *f*
⟨*proof*⟩

hide-fact (**open**) *bad_worst_chain_0* *bad_worst_chain_Suc*

end

5 Extension Orders

theory *Extension_Orders*
imports *Lambda_Free_Util* *Infinite_Chain* *HOL-Cardinals*.*Wellorder_Extension*
begin

This theory defines locales for categorizing extension orders used for orders on λ -free higher-order terms and defines variants of the lexicographic and multiset orders.

5.1 Locales

locale *ext* =
fixes *ext* :: ('*a* \Rightarrow '*a* \Rightarrow *bool*) \Rightarrow '*a* *list* \Rightarrow '*a* *list* \Rightarrow *bool*
assumes
mono_strong: $(\forall y \in \text{set } ys. \forall x \in \text{set } xs. \text{gt } y x \longrightarrow \text{gt}' y x) \implies \text{ext } \text{gt } ys \ xs \implies \text{ext } \text{gt}' \ ys \ xs$ **and**
map: *finite* *A* $\implies ys \in \text{lists } A \implies xs \in \text{lists } A \implies (\forall x \in A. \neg \text{gt } (f x) (f x)) \implies$
 $(\forall z \in A. \forall y \in A. \forall x \in A. \text{gt } (f z) (f y) \longrightarrow \text{gt } (f y) (f x) \longrightarrow \text{gt } (f z) (f x)) \implies$
 $(\forall y \in A. \forall x \in A. \text{gt } y x \longrightarrow \text{gt } (f y) (f x)) \implies \text{ext } \text{gt } ys \ xs \implies \text{ext } \text{gt } (\text{map } f \ ys) (\text{map } f \ xs)$
begin

lemma *mono[mono]*: *gt* \leq *gt'* $\implies \text{ext } \text{gt} \leq \text{ext } \text{gt}'$
⟨*proof*⟩

end

locale *ext_irrefl* = *ext* +

assumes *irrefl*: $(\forall x \in \text{set } xs. \neg \text{gt } x x) \implies \neg \text{ext } \text{gt } xs \text{ } xs$

locale *ext_trans* = *ext* +
assumes *trans*: $zs \in \text{lists } A \implies ys \in \text{lists } A \implies xs \in \text{lists } A \implies$
 $(\forall z \in A. \forall y \in A. \forall x \in A. \text{gt } z y \longrightarrow \text{gt } y x \longrightarrow \text{gt } z x) \implies \text{ext } \text{gt } zs \text{ } ys \implies \text{ext } \text{gt } ys \text{ } xs \implies$
 $\text{ext } \text{gt } zs \text{ } xs$

locale *ext_irrefl_before_trans* = *ext_irrefl* +
assumes *trans_from_irrefl*: $\text{finite } A \implies zs \in \text{lists } A \implies ys \in \text{lists } A \implies xs \in \text{lists } A \implies$
 $(\forall x \in A. \neg \text{gt } x x) \implies (\forall z \in A. \forall y \in A. \forall x \in A. \text{gt } z y \longrightarrow \text{gt } y x \longrightarrow \text{gt } z x) \implies \text{ext } \text{gt } zs \text{ } ys \implies$
 $\text{ext } \text{gt } ys \text{ } xs \implies \text{ext } \text{gt } zs \text{ } xs$

locale *ext_trans_before_irrefl* = *ext_trans* +
assumes *irrefl_from_trans*: $(\forall z \in \text{set } xs. \forall y \in \text{set } xs. \forall x \in \text{set } xs. \text{gt } z y \longrightarrow \text{gt } y x \longrightarrow \text{gt } z x) \implies$
 $(\forall x \in \text{set } xs. \neg \text{gt } x x) \implies \neg \text{ext } \text{gt } xs \text{ } xs$

locale *ext_irrefl_trans_strong* = *ext_irrefl* +
assumes *trans_strong*: $(\forall z \in \text{set } zs. \forall y \in \text{set } ys. \forall x \in \text{set } xs. \text{gt } z y \longrightarrow \text{gt } y x \longrightarrow \text{gt } z x) \implies$
 $\text{ext } \text{gt } zs \text{ } ys \implies \text{ext } \text{gt } ys \text{ } xs \implies \text{ext } \text{gt } zs \text{ } xs$

sublocale *ext_irrefl_trans_strong* < *ext_irrefl_before_trans*
<proof>

sublocale *ext_irrefl_trans_strong* < *ext_trans*
<proof>

sublocale *ext_irrefl_trans_strong* < *ext_trans_before_irrefl*
<proof>

locale *ext_snoc* = *ext* +
assumes *snoc*: $\text{ext } \text{gt } (xs @ [x]) \text{ } xs$

locale *ext_compat_cons* = *ext* +
assumes *compat_cons*: $\text{ext } \text{gt } ys \text{ } xs \implies \text{ext } \text{gt } (x \# ys) (x \# xs)$
begin

lemma *compat_append_left*: $\text{ext } \text{gt } ys \text{ } xs \implies \text{ext } \text{gt } (zs @ ys) (zs @ xs)$
<proof>

end

locale *ext_compat_snoc* = *ext* +
assumes *compat_snoc*: $\text{ext } \text{gt } ys \text{ } xs \implies \text{ext } \text{gt } (ys @ [x]) (xs @ [x])$
begin

lemma *compat_append_right*: $\text{ext } \text{gt } ys \text{ } xs \implies \text{ext } \text{gt } (ys @ zs) (xs @ zs)$
<proof>

end

locale *ext_compat_list* = *ext* +
assumes *compat_list*: $y \neq x \implies \text{gt } y x \implies \text{ext } \text{gt } (xs @ y \# xs') (xs @ x \# xs')$

locale *ext_singleton* = *ext* +
assumes *singleton*: $y \neq x \implies \text{ext } \text{gt } [y] [x] \longleftrightarrow \text{gt } y x$

locale *ext_compat_list_strong* = *ext_compat_cons* + *ext_compat_snoc* + *ext_singleton*
begin

lemma *compat_list*: $y \neq x \implies \text{gt } y x \implies \text{ext } \text{gt } (xs @ y \# xs') (xs @ x \# xs')$
<proof>

end

sublocale *ext_compat_list_strong* < *ext_compat_list*
 ⟨*proof*⟩

locale *ext_total* = *ext* +
assumes *total*: $(\forall y \in A. \forall x \in A. gt\ y\ x \vee gt\ x\ y \vee y = x) \implies ys \in lists\ A \implies xs \in lists\ A \implies$
 $ext\ gt\ ys\ xs \vee ext\ gt\ xs\ ys \vee ys = xs$

locale *ext_wf* = *ext* +
assumes *wf*: $wfP\ (\lambda x\ y. gt\ y\ x) \implies wfP\ (\lambda xs\ ys. ext\ gt\ ys\ xs)$

locale *ext_hd_or_tl* = *ext* +
assumes *hd_or_tl*: $(\forall z\ y\ x. gt\ z\ y \longrightarrow gt\ y\ x \longrightarrow gt\ z\ x) \implies (\forall y\ x. gt\ y\ x \vee gt\ x\ y \vee y = x) \implies$
 $length\ ys = length\ xs \implies ext\ gt\ (y\ \#\ ys)\ (x\ \#\ xs) \implies gt\ y\ x \vee ext\ gt\ ys\ xs$

locale *ext_wf_bounded* = *ext_irrefl_before_trans* + *ext_hd_or_tl*
begin

context

fixes *gt* :: 'a \Rightarrow 'a \Rightarrow bool

assumes

gt_irrefl: $\bigwedge z. \neg gt\ z\ z$ **and**

gt_trans: $\bigwedge z\ y\ x. gt\ z\ y \implies gt\ y\ x \implies gt\ z\ x$ **and**

gt_total: $\bigwedge y\ x. gt\ y\ x \vee gt\ x\ y \vee y = x$ **and**

gt_wf: $wfP\ (\lambda x\ y. gt\ y\ x)$

begin

lemma *irrefl_gt*: $\neg ext\ gt\ xs\ xs$
 ⟨*proof*⟩

lemma *trans_gt*: $ext\ gt\ zs\ ys \implies ext\ gt\ ys\ xs \implies ext\ gt\ zs\ xs$
 ⟨*proof*⟩

lemma *hd_or_tl_gt*: $length\ ys = length\ xs \implies ext\ gt\ (y\ \#\ ys)\ (x\ \#\ xs) \implies gt\ y\ x \vee ext\ gt\ ys\ xs$
 ⟨*proof*⟩

lemma *wf_same_length_if_total*: $wfP\ (\lambda xs\ ys. length\ ys = n \wedge length\ xs = n \wedge ext\ gt\ ys\ xs)$
 ⟨*proof*⟩

lemma *wf_bounded_if_total*: $wfP\ (\lambda xs\ ys. length\ ys \leq n \wedge length\ xs \leq n \wedge ext\ gt\ ys\ xs)$
 ⟨*proof*⟩

end

context

fixes *gt* :: 'a \Rightarrow 'a \Rightarrow bool

assumes

gt_irrefl: $\bigwedge z. \neg gt\ z\ z$ **and**

gt_wf: $wfP\ (\lambda x\ y. gt\ y\ x)$

begin

lemma *wf_bounded*: $wfP\ (\lambda xs\ ys. length\ ys \leq n \wedge length\ xs \leq n \wedge ext\ gt\ ys\ xs)$
 ⟨*proof*⟩

end

end

5.2 Lexicographic Extension

inductive *lexext* :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a list \Rightarrow 'a list \Rightarrow bool **for** *gt* **where**

lexext_Nil: *lexext* *gt* (*y* # *ys*) []

| *lexext_Cons*: $gt\ y\ x \implies lexext\ gt\ (y\ \#\ ys)\ (x\ \#\ xs)$

| *lexext_Cons_eq*: $lexext\ gt\ ys\ xs \implies lexext\ gt\ (x\ \#\ ys)\ (x\ \#\ xs)$

lemma *lexext_simps[simp]*:
 $lexext\ gt\ ys\ [] \longleftrightarrow ys \neq []$
 $\neg\ lexext\ gt\ []\ xs$
 $lexext\ gt\ (y\ \#\ ys)\ (x\ \#\ xs) \longleftrightarrow gt\ y\ x \vee x = y \wedge lexext\ gt\ ys\ xs$
<proof>

lemma *lexext_mono_strong*:
assumes
 $\forall y \in set\ ys. \forall x \in set\ xs. gt\ y\ x \longrightarrow gt'\ y\ x$ **and**
 $lexext\ gt\ ys\ xs$
shows $lexext\ gt'\ ys\ xs$
<proof>

lemma *lexext_map_strong*:
 $(\forall y \in set\ ys. \forall x \in set\ xs. gt\ y\ x \longrightarrow gt\ (f\ y)\ (f\ x)) \implies lexext\ gt\ ys\ xs \implies$
 $lexext\ gt\ (map\ f\ ys)\ (map\ f\ xs)$
<proof>

lemma *lexext_irrefl*:
assumes $\forall x \in set\ xs. \neg\ gt\ x\ x$
shows $\neg\ lexext\ gt\ xs\ xs$
<proof>

lemma *lexext_trans_strong*:
assumes
 $\forall z \in set\ zs. \forall y \in set\ ys. \forall x \in set\ xs. gt\ z\ y \longrightarrow gt\ y\ x \longrightarrow gt\ z\ x$ **and**
 $lexext\ gt\ zs\ ys$ **and** $lexext\ gt\ ys\ xs$
shows $lexext\ gt\ zs\ xs$
<proof>

lemma *lexext_snoc*: $lexext\ gt\ (xs\ @\ [x])\ xs$
<proof>

lemmas *lexext_compat_cons = lexext_Cons_eq*

lemma *lexext_compat_snoc_if_same_length*:
assumes $length\ ys = length\ xs$ **and** $lexext\ gt\ ys\ xs$
shows $lexext\ gt\ (ys\ @\ [x])\ (xs\ @\ [x])$
<proof>

lemma *lexext_compat_list*: $gt\ y\ x \implies lexext\ gt\ (xs\ @\ y\ \#\ xs')\ (xs\ @\ x\ \#\ xs')$
<proof>

lemma *lexext_singleton*: $lexext\ gt\ [y]\ [x] \longleftrightarrow gt\ y\ x$
<proof>

lemma *lexext_total*: $(\forall y \in B. \forall x \in A. gt\ y\ x \vee gt\ x\ y \vee y = x) \implies ys \in lists\ B \implies xs \in lists\ A \implies$
 $lexext\ gt\ ys\ xs \vee lexext\ gt\ xs\ ys \vee ys = xs$
<proof>

lemma *lexext_hd_or_tl*: $lexext\ gt\ (y\ \#\ ys)\ (x\ \#\ xs) \implies gt\ y\ x \vee lexext\ gt\ ys\ xs$
<proof>

interpretation *lexext*: *ext lexext*
<proof>

interpretation *lexext*: *ext_irrefl_trans_strong lexext*
<proof>

interpretation *lexext*: *ext_snoc lexext*
<proof>

interpretation *lexext*: *ext_compat_cons* *lexext*
<proof>

interpretation *lexext*: *ext_compat_list* *lexext*
<proof>

interpretation *lexext*: *ext_singleton* *lexext*
<proof>

interpretation *lexext*: *ext_total* *lexext*
<proof>

interpretation *lexext*: *ext_hd_or_tl* *lexext*
<proof>

interpretation *lexext*: *ext_wf_bounded* *lexext*
<proof>

5.3 Reverse (Right-to-Left) Lexicographic Extension

abbreviation *lexext_rev* :: ('a ⇒ 'a ⇒ bool) ⇒ 'a list ⇒ 'a list ⇒ bool **where**
lexext_rev *gt* *ys* *xs* ≡ *lexext* *gt* (rev *ys*) (rev *xs*)

lemma *lexext_rev_simps*[*simp*]:
lexext_rev *gt* *ys* [] ↔ *ys* ≠ []
¬ *lexext_rev* *gt* [] *xs*
lexext_rev *gt* (*ys* @ [*y*]) (*xs* @ [*x*]) ↔ *gt* *y* *x* ∨ *x* = *y* ∧ *lexext_rev* *gt* *ys* *xs*
<proof>

lemma *lexext_rev_cons_cons*:
assumes *length* *ys* = *length* *xs*
shows *lexext_rev* *gt* (*y* # *ys*) (*x* # *xs*) ↔ *lexext_rev* *gt* *ys* *xs* ∨ *ys* = *xs* ∧ *gt* *y* *x*
<proof>

lemma *lexext_rev_mono_strong*:
assumes
 ∀ *y* ∈ *set* *ys*. ∀ *x* ∈ *set* *xs*. *gt* *y* *x* → *gt'* *y* *x* **and**
 lexext_rev *gt* *ys* *xs*
shows *lexext_rev* *gt'* *ys* *xs*
<proof>

lemma *lexext_rev_map_strong*:
(∀ *y* ∈ *set* *ys*. ∀ *x* ∈ *set* *xs*. *gt* *y* *x* → *gt* (*f* *y*) (*f* *x*)) ⇒ *lexext_rev* *gt* *ys* *xs* ⇒
lexext_rev *gt* (*map* *f* *ys*) (*map* *f* *xs*)
<proof>

lemma *lexext_rev_irrefl*:
assumes ∀ *x* ∈ *set* *xs*. ¬ *gt* *x* *x*
shows ¬ *lexext_rev* *gt* *xs* *xs*
<proof>

lemma *lexext_rev_trans_strong*:
assumes
 ∀ *z* ∈ *set* *zs*. ∀ *y* ∈ *set* *ys*. ∀ *x* ∈ *set* *xs*. *gt* *z* *y* → *gt* *y* *x* → *gt* *z* *x* **and**
 lexext_rev *gt* *zs* *ys* **and** *lexext_rev* *gt* *ys* *xs*
shows *lexext_rev* *gt* *zs* *xs*
<proof>

lemma *lexext_rev_compat_cons_if_same_length*:
assumes *length* *ys* = *length* *xs* **and** *lexext_rev* *gt* *ys* *xs*
shows *lexext_rev* *gt* (*x* # *ys*) (*x* # *xs*)
<proof>

lemma *lexext_rev_compat_snoc*: *lexext_rev* *gt* *ys* *xs* ⇒ *lexext_rev* *gt* (*ys* @ [*x*]) (*xs* @ [*x*])

<proof>

lemma *lexext_rev_compat_list*: $gt\ y\ x \implies lexext_rev\ gt\ (xs\ @\ y\ \# \ xs')\ (xs\ @\ x\ \# \ xs')$
<proof>

lemma *lexext_rev_singleton*: $lexext_rev\ gt\ [y]\ [x] \longleftrightarrow gt\ y\ x$
<proof>

lemma *lexext_rev_total*:
 $(\forall y \in B. \forall x \in A. gt\ y\ x \vee gt\ x\ y \vee y = x) \implies ys \in lists\ B \implies xs \in lists\ A \implies$
 $lexext_rev\ gt\ ys\ xs \vee lexext_rev\ gt\ xs\ ys \vee ys = xs$
<proof>

lemma *lexext_rev_hd_or_tl*:
assumes
 $length\ ys = length\ xs$ **and**
 $lexext_rev\ gt\ (y\ \# \ ys)\ (x\ \# \ xs)$
shows $gt\ y\ x \vee lexext_rev\ gt\ ys\ xs$
<proof>

interpretation *lexext_rev*: *ext lexext_rev*
<proof>

interpretation *lexext_rev*: *ext_irrefl_trans_strong lexext_rev*
<proof>

interpretation *lexext_rev*: *ext_compat_snoc lexext_rev*
<proof>

interpretation *lexext_rev*: *ext_compat_list lexext_rev*
<proof>

interpretation *lexext_rev*: *ext_singleton lexext_rev*
<proof>

interpretation *lexext_rev*: *ext_total lexext_rev*
<proof>

interpretation *lexext_rev*: *ext_hd_or_tl lexext_rev*
<proof>

interpretation *lexext_rev*: *ext_wf_bounded lexext_rev*
<proof>

5.4 Generic Length Extension

definition *lenext* :: $('a\ list \Rightarrow 'a\ list \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow bool$ **where**
 $lenext\ gts\ ys\ xs \longleftrightarrow length\ ys > length\ xs \vee length\ ys = length\ xs \wedge gts\ ys\ xs$

lemma
lenext_mono_strong: $(gts\ ys\ xs \implies gts'\ ys\ xs) \implies lenext\ gts\ ys\ xs \implies lenext\ gts'\ ys\ xs$ **and**
lenext_map_strong: $(length\ ys = length\ xs \implies gts\ ys\ xs \implies gts\ (map\ f\ ys)\ (map\ f\ xs)) \implies$
 $lenext\ gts\ ys\ xs \implies lenext\ gts\ (map\ f\ ys)\ (map\ f\ xs)$ **and**
lenext_irrefl: $\neg gts\ xs\ xs \implies \neg lenext\ gts\ xs\ xs$ **and**
lenext_trans: $(gts\ zs\ ys \implies gts\ ys\ xs \implies gts\ zs\ xs) \implies lenext\ gts\ zs\ ys \implies lenext\ gts\ ys\ xs \implies$
 $lenext\ gts\ zs\ xs$ **and**
lenext_snoc: $lenext\ gts\ (xs\ @\ [x])\ xs$ **and**
lenext_compat_cons: $(length\ ys = length\ xs \implies gts\ ys\ xs \implies gts\ (x\ \# \ ys)\ (x\ \# \ xs)) \implies$
 $lenext\ gts\ ys\ xs \implies lenext\ gts\ (x\ \# \ ys)\ (x\ \# \ xs)$ **and**
lenext_compat_snoc: $(length\ ys = length\ xs \implies gts\ ys\ xs \implies gts\ (ys\ @\ [x])\ (xs\ @\ [x])) \implies$
 $lenext\ gts\ ys\ xs \implies lenext\ gts\ (ys\ @\ [x])\ (xs\ @\ [x])$ **and**
lenext_compat_list: $gts\ (xs\ @\ y\ \# \ xs')\ (xs\ @\ x\ \# \ xs') \implies$
 $lenext\ gts\ (xs\ @\ y\ \# \ xs')\ (xs\ @\ x\ \# \ xs')$ **and**
lenext_singleton: $lenext\ gts\ [y]\ [x] \longleftrightarrow gts\ [y]\ [x]$ **and**

$lenext_total: (gts\ ys\ xs \vee gts\ xs\ ys \vee ys = xs) \implies$
 $lenext\ gts\ ys\ xs \vee lenext\ gts\ xs\ ys \vee ys = xs$ **and**
 $lenext_hd_or_tl: (length\ ys = length\ xs \implies gts\ (y\ \# \ ys)\ (x\ \# \ xs) \implies gt\ y\ x \vee gts\ ys\ xs) \implies$
 $lenext\ gts\ (y\ \# \ ys)\ (x\ \# \ xs) \implies gt\ y\ x \vee lenext\ gts\ ys\ xs$
 <proof>

5.5 Length-Lexicographic Extension

abbreviation $len_lexext :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow bool$ **where**
 $len_lexext\ gt \equiv lenext\ (lexext\ gt)$

lemma $len_lexext_mono_strong:$

$(\forall y \in set\ ys. \forall x \in set\ xs. gt\ y\ x \longrightarrow gt'\ y\ x) \implies len_lexext\ gt\ ys\ xs \implies len_lexext\ gt'\ ys\ xs$
 <proof>

lemma $len_lexext_map_strong:$

$(\forall y \in set\ ys. \forall x \in set\ xs. gt\ y\ x \longrightarrow gt\ (f\ y)\ (f\ x)) \implies len_lexext\ gt\ ys\ xs \implies$
 $len_lexext\ gt\ (map\ f\ ys)\ (map\ f\ xs)$
 <proof>

lemma $len_lexext_irrefl: (\forall x \in set\ xs. \neg gt\ x\ x) \implies \neg len_lexext\ gt\ xs\ xs$

<proof>

lemma $len_lexext_trans_strong:$

$(\forall z \in set\ zs. \forall y \in set\ ys. \forall x \in set\ xs. gt\ z\ y \longrightarrow gt\ y\ x \longrightarrow gt\ z\ x) \implies len_lexext\ gt\ zs\ ys \implies$
 $len_lexext\ gt\ ys\ xs \implies len_lexext\ gt\ zs\ xs$
 <proof>

lemma $len_lexext_snoc: len_lexext\ gt\ (xs\ @\ [x])\ xs$

<proof>

lemma $len_lexext_compat_cons: len_lexext\ gt\ ys\ xs \implies len_lexext\ gt\ (x\ \# \ ys)\ (x\ \# \ xs)$

<proof>

lemma $len_lexext_compat_snoc: len_lexext\ gt\ ys\ xs \implies len_lexext\ gt\ (ys\ @\ [x])\ (xs\ @\ [x])$

<proof>

lemma $len_lexext_compat_list: gt\ y\ x \implies len_lexext\ gt\ (xs\ @\ y\ \# \ xs')\ (xs\ @\ x\ \# \ xs')$

<proof>

lemma $len_lexext_singleton[simp]: len_lexext\ gt\ [y]\ [x] \longleftrightarrow gt\ y\ x$

<proof>

lemma $len_lexext_total: (\forall y \in B. \forall x \in A. gt\ y\ x \vee gt\ x\ y \vee y = x) \implies ys \in lists\ B \implies xs \in lists\ A \implies$

$len_lexext\ gt\ ys\ xs \vee len_lexext\ gt\ xs\ ys \vee ys = xs$
 <proof>

lemma $len_lexext_iff_lenlex: len_lexext\ gt\ ys\ xs \longleftrightarrow (xs, ys) \in lenlex\ \{(x, y). gt\ y\ x\}$

<proof>

lemma $len_lexext_wf: wfP\ (\lambda x\ y. gt\ y\ x) \implies wfP\ (\lambda xs\ ys. len_lexext\ gt\ ys\ xs)$

<proof>

lemma $len_lexext_hd_or_tl: len_lexext\ gt\ (y\ \# \ ys)\ (x\ \# \ xs) \implies gt\ y\ x \vee len_lexext\ gt\ ys\ xs$

<proof>

interpretation $len_lexext: ext\ len_lexext$

<proof>

interpretation $len_lexext: ext_irrefl_trans_strong\ len_lexext$

<proof>

interpretation $len_lexext: ext_snoc\ len_lexext$

<proof>

interpretation $len_lexext: ext_compat_cons\ len_lexext$
 $\langle proof \rangle$

interpretation $len_lexext: ext_compat_snoc\ len_lexext$
 $\langle proof \rangle$

interpretation $len_lexext: ext_compat_list\ len_lexext$
 $\langle proof \rangle$

interpretation $len_lexext: ext_singleton\ len_lexext$
 $\langle proof \rangle$

interpretation $len_lexext: ext_total\ len_lexext$
 $\langle proof \rangle$

interpretation $len_lexext: ext_wf\ len_lexext$
 $\langle proof \rangle$

interpretation $len_lexext: ext_hd_or_tl\ len_lexext$
 $\langle proof \rangle$

interpretation $len_lexext: ext_wf_bounded\ len_lexext$
 $\langle proof \rangle$

5.6 Reverse (Right-to-Left) Length-Lexicographic Extension

abbreviation $len_lexext_rev :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow bool$ **where**
 $len_lexext_rev\ gt \equiv lenext\ (lexext_rev\ gt)$

lemma $len_lexext_rev_mono_strong:$
 $(\forall y \in set\ ys. \forall x \in set\ xs. gt\ y\ x \longrightarrow gt'\ y\ x) \Longrightarrow len_lexext_rev\ gt\ ys\ xs \Longrightarrow len_lexext_rev\ gt'\ ys\ xs$
 $\langle proof \rangle$

lemma $len_lexext_rev_map_strong:$
 $(\forall y \in set\ ys. \forall x \in set\ xs. gt\ y\ x \longrightarrow gt\ (f\ y)\ (f\ x)) \Longrightarrow len_lexext_rev\ gt\ ys\ xs \Longrightarrow$
 $len_lexext_rev\ gt\ (map\ f\ ys)\ (map\ f\ xs)$
 $\langle proof \rangle$

lemma $len_lexext_rev_irrefl: (\forall x \in set\ xs. \neg gt\ x\ x) \Longrightarrow \neg len_lexext_rev\ gt\ xs\ xs$
 $\langle proof \rangle$

lemma $len_lexext_rev_trans_strong:$
 $(\forall z \in set\ zs. \forall y \in set\ ys. \forall x \in set\ xs. gt\ z\ y \longrightarrow gt\ y\ x \longrightarrow gt\ z\ x) \Longrightarrow len_lexext_rev\ gt\ zs\ ys \Longrightarrow$
 $len_lexext_rev\ gt\ ys\ xs \Longrightarrow len_lexext_rev\ gt\ zs\ xs$
 $\langle proof \rangle$

lemma $len_lexext_rev_snoc: len_lexext_rev\ gt\ (xs\ @\ [x])\ xs$
 $\langle proof \rangle$

lemma $len_lexext_rev_compat_cons: len_lexext_rev\ gt\ ys\ xs \Longrightarrow len_lexext_rev\ gt\ (x\ \#\ ys)\ (x\ \#\ xs)$
 $\langle proof \rangle$

lemma $len_lexext_rev_compat_snoc: len_lexext_rev\ gt\ ys\ xs \Longrightarrow len_lexext_rev\ gt\ (ys\ @\ [x])\ (xs\ @\ [x])$
 $\langle proof \rangle$

lemma $len_lexext_rev_compat_list: gt\ y\ x \Longrightarrow len_lexext_rev\ gt\ (xs\ @\ y\ \#\ xs')\ (xs\ @\ x\ \#\ xs')$
 $\langle proof \rangle$

lemma $len_lexext_rev_singleton[simp]: len_lexext_rev\ gt\ [y]\ [x] \longleftrightarrow gt\ y\ x$
 $\langle proof \rangle$

lemma $len_lexext_rev_total: (\forall y \in B. \forall x \in A. gt\ y\ x \vee gt\ x\ y \vee y = x) \Longrightarrow ys \in lists\ B \Longrightarrow$
 $xs \in lists\ A \Longrightarrow len_lexext_rev\ gt\ ys\ xs \vee len_lexext_rev\ gt\ xs\ ys \vee ys = xs$

<proof>

lemma *len_lexext_rev_iff_len_lexext*: $\text{len_lexext_rev } gt \text{ } ys \text{ } xs \longleftrightarrow \text{len_lexext } gt \text{ } (\text{rev } ys) \text{ } (\text{rev } xs)$
<proof>

lemma *len_lexext_rev_wf*: $\text{wfP } (\lambda x y. gt \text{ } y \text{ } x) \implies \text{wfP } (\lambda xs ys. \text{len_lexext_rev } gt \text{ } ys \text{ } xs)$
<proof>

lemma *len_lexext_rev_hd_or_tl*:
 $\text{len_lexext_rev } gt \text{ } (y \# ys) \text{ } (x \# xs) \implies gt \text{ } y \text{ } x \vee \text{len_lexext_rev } gt \text{ } ys \text{ } xs$
<proof>

interpretation *len_lexext_rev*: *ext len_lexext_rev*
<proof>

interpretation *len_lexext_rev*: *ext_irrefl_trans_strong len_lexext_rev*
<proof>

interpretation *len_lexext_rev*: *ext_snoc len_lexext_rev*
<proof>

interpretation *len_lexext_rev*: *ext_compat_cons len_lexext_rev*
<proof>

interpretation *len_lexext_rev*: *ext_compat_snoc len_lexext_rev*
<proof>

interpretation *len_lexext_rev*: *ext_compat_list len_lexext_rev*
<proof>

interpretation *len_lexext_rev*: *ext_singleton len_lexext_rev*
<proof>

interpretation *len_lexext_rev*: *ext_total len_lexext_rev*
<proof>

interpretation *len_lexext_rev*: *ext_wf len_lexext_rev*
<proof>

interpretation *len_lexext_rev*: *ext_hd_or_tl len_lexext_rev*
<proof>

interpretation *len_lexext_rev*: *ext_wf_bounded len_lexext_rev*
<proof>

5.7 Dershowitz–Manna Multiset Extension

definition *msetext_dersh* **where**

$\text{msetext_dersh } gt \text{ } ys \text{ } xs = (\text{let } N = \text{mset } ys; M = \text{mset } xs \text{ in}$
 $(\exists Y X. Y \neq \{\#\} \wedge Y \subseteq \# N \wedge M = (N - Y) + X \wedge (\forall x. x \in \# X \longrightarrow (\exists y. y \in \# Y \wedge gt \text{ } y \text{ } x))))$

The following proof is based on that of *less_multiset_{DM}_imp_mult*.

lemma *msetext_dersh_imp_mult_rel*:

assumes

ys_a: $ys \in \text{lists } A$ **and** *xs_a*: $xs \in \text{lists } A$ **and**

ys_gt_xs: $\text{msetext_dersh } gt \text{ } ys \text{ } xs$

shows $(\text{mset } xs, \text{mset } ys) \in \text{mult } \{(x, y). x \in A \wedge y \in A \wedge gt \text{ } y \text{ } x\}$
<proof>

lemma *msetext_dersh_imp_mult*: $\text{msetext_dersh } gt \text{ } ys \text{ } xs \implies (\text{mset } xs, \text{mset } ys) \in \text{mult } \{(x, y). gt \text{ } y \text{ } x\}$
<proof>

lemma *mult_imp_msetext_dersh_rel*:
assumes

$ys_a: \text{set_mset } (\text{mset } ys) \subseteq A$ **and** $xs_a: \text{set_mset } (\text{mset } xs) \subseteq A$ **and**
 $\text{in_mult}: (\text{mset } xs, \text{mset } ys) \in \text{mult } \{(x, y). x \in A \wedge y \in A \wedge \text{gt } y \ x\}$ **and**
 $\text{trans}: \forall z \in A. \forall y \in A. \forall x \in A. \text{gt } z \ y \longrightarrow \text{gt } y \ x \longrightarrow \text{gt } z \ x$
shows $\text{msetext_dersh } \text{gt } \text{ys } \text{xs}$
 $\langle \text{proof} \rangle$

lemma $\text{msetext_dersh_mono_strong}$:
 $(\forall y \in \text{set } \text{ys}. \forall x \in \text{set } \text{xs}. \text{gt } y \ x \longrightarrow \text{gt}' \ y \ x) \Longrightarrow \text{msetext_dersh } \text{gt } \text{ys } \text{xs} \Longrightarrow$
 $\text{msetext_dersh } \text{gt}' \ \text{ys } \text{xs}$
 $\langle \text{proof} \rangle$

lemma $\text{msetext_dersh_map_strong}$:
assumes
 $\text{compat_f}: \forall y \in \text{set } \text{ys}. \forall x \in \text{set } \text{xs}. \text{gt } y \ x \longrightarrow \text{gt } (f \ y) \ (f \ x)$ **and**
 $\text{ys_gt_xs}: \text{msetext_dersh } \text{gt } \text{ys } \text{xs}$
shows $\text{msetext_dersh } \text{gt } (\text{map } f \ \text{ys}) \ (\text{map } f \ \text{xs})$
 $\langle \text{proof} \rangle$

lemma $\text{msetext_dersh_trans}$:
assumes
 $\text{zs_a}: \text{zs} \in \text{lists } A$ **and**
 $\text{ys_a}: \text{ys} \in \text{lists } A$ **and**
 $\text{xs_a}: \text{xs} \in \text{lists } A$ **and**
 $\text{trans}: \forall z \in A. \forall y \in A. \forall x \in A. \text{gt } z \ y \longrightarrow \text{gt } y \ x \longrightarrow \text{gt } z \ x$ **and**
 $\text{zs_gt_ys}: \text{msetext_dersh } \text{gt } \text{zs } \text{ys}$ **and**
 $\text{ys_gt_xs}: \text{msetext_dersh } \text{gt } \text{ys } \text{xs}$
shows $\text{msetext_dersh } \text{gt } \text{zs } \text{xs}$
 $\langle \text{proof} \rangle$

lemma $\text{msetext_dersh_irrefl_from_trans}$:
assumes
 $\text{trans}: \forall z \in \text{set } \text{xs}. \forall y \in \text{set } \text{xs}. \forall x \in \text{set } \text{xs}. \text{gt } z \ y \longrightarrow \text{gt } y \ x \longrightarrow \text{gt } z \ x$ **and**
 $\text{irrefl}: \forall x \in \text{set } \text{xs}. \neg \text{gt } x \ x$
shows $\neg \text{msetext_dersh } \text{gt } \text{xs } \text{xs}$
 $\langle \text{proof} \rangle$

lemma $\text{msetext_dersh_snoc}$: $\text{msetext_dersh } \text{gt } (\text{xs} @ [x]) \ \text{xs}$
 $\langle \text{proof} \rangle$

lemma $\text{msetext_dersh_compat_cons}$:
assumes $\text{ys_gt_xs}: \text{msetext_dersh } \text{gt } \text{ys } \text{xs}$
shows $\text{msetext_dersh } \text{gt } (x \# \text{ys}) \ (x \# \text{xs})$
 $\langle \text{proof} \rangle$

lemma $\text{msetext_dersh_compat_snoc}$: $\text{msetext_dersh } \text{gt } \text{ys } \text{xs} \Longrightarrow \text{msetext_dersh } \text{gt } (\text{ys} @ [x]) \ (\text{xs} @ [x])$
 $\langle \text{proof} \rangle$

lemma $\text{msetext_dersh_compat_list}$:
assumes $\text{y_gt_x}: \text{gt } y \ x$
shows $\text{msetext_dersh } \text{gt } (\text{xs} @ y \# \text{xs}') \ (\text{xs} @ x \# \text{xs}')$
 $\langle \text{proof} \rangle$

lemma $\text{msetext_dersh_singleton}$: $\text{msetext_dersh } \text{gt } [y] \ [x] \longleftrightarrow \text{gt } y \ x$
 $\langle \text{proof} \rangle$

lemma msetext_dersh_wf :
assumes $\text{wf_gt}: \text{wfP } (\lambda x \ y. \text{gt } y \ x)$
shows $\text{wfP } (\lambda \text{xs } \text{ys}. \text{msetext_dersh } \text{gt } \text{ys } \text{xs})$
 $\langle \text{proof} \rangle$

interpretation msetext_dersh : $\text{ext } \text{msetext_dersh}$
 $\langle \text{proof} \rangle$

interpretation *msetext_dersh*: *ext_trans_before_irrefl msetext_dersh*
(*proof*)

interpretation *msetext_dersh*: *ext_snoc msetext_dersh*
(*proof*)

interpretation *msetext_dersh*: *ext_compat_cons msetext_dersh*
(*proof*)

interpretation *msetext_dersh*: *ext_compat_snoc msetext_dersh*
(*proof*)

interpretation *msetext_dersh*: *ext_compat_list msetext_dersh*
(*proof*)

interpretation *msetext_dersh*: *ext_singleton msetext_dersh*
(*proof*)

interpretation *msetext_dersh*: *ext_wf msetext_dersh*
(*proof*)

5.8 Huet–Oppen Multiset Extension

definition *msetext_huet* **where**

msetext_huet *gt* *ys* *xs* = (let *N* = *mset* *ys*; *M* = *mset* *xs* in
 $M \neq N \wedge (\forall x. \text{count } M \ x > \text{count } N \ x \longrightarrow (\exists y. \text{gt } y \ x \wedge \text{count } N \ y > \text{count } M \ y)))$)

lemma *msetext_huet_imp_count_gt*:

assumes *ys_gt_xs*: *msetext_huet* *gt* *ys* *xs*

shows $\exists x. \text{count } (mset \ ys) \ x > \text{count } (mset \ xs) \ x$

(*proof*)

lemma *msetext_huet_imp_dersh*:

assumes *huet*: *msetext_huet* *gt* *ys* *xs*

shows *msetext_dersh* *gt* *ys* *xs*

(*proof*)

The following proof is based on that of *mult_imp_less_multiset_{HO}*.

lemma *mult_imp_msetext_huet*:

assumes

irrefl: *irreflp* *gt* **and** *trans*: *transp* *gt* **and**

in_mult: $(mset \ xs, mset \ ys) \in mult \ \{(x, y). \text{gt } y \ x\}$

shows *msetext_huet* *gt* *ys* *xs*

(*proof*)

theorem *msetext_huet_eq_dersh*: *irreflp* *gt* \implies *transp* *gt* \implies *msetext_dersh* *gt* = *msetext_huet* *gt*

(*proof*)

lemma *msetext_huet_mono_strong*:

$(\forall y \in set \ ys. \forall x \in set \ xs. \text{gt } y \ x \longrightarrow \text{gt}' \ y \ x) \implies msetext_huet \ gt \ ys \ xs \implies msetext_huet \ \text{gt}' \ ys \ xs$

(*proof*)

lemma *msetext_huet_map*:

assumes

fin: *finite* *A* **and**

ys_a: *ys* \in *lists* *A* **and** *xs_a*: *xs* \in *lists* *A* **and**

irrefl_f: $\forall x \in A. \neg \text{gt} \ (f \ x) \ (f \ x)$ **and**

trans_f: $\forall z \in A. \forall y \in A. \forall x \in A. \text{gt} \ (f \ z) \ (f \ y) \longrightarrow \text{gt} \ (f \ y) \ (f \ x) \longrightarrow \text{gt} \ (f \ z) \ (f \ x)$ **and**

compat_f: $\forall y \in A. \forall x \in A. \text{gt} \ y \ x \longrightarrow \text{gt} \ (f \ y) \ (f \ x)$ **and**

ys_gt_xs: *msetext_huet* *gt* *ys* *xs*

shows *msetext_huet* *gt* (*map* *f* *ys*) (*map* *f* *xs*) (**is** *msetext_huet* $_ \ ?fys \ ?fxs$)

(*proof*)

lemma *msetext_huet_irrefl*: $(\forall x \in set \ xs. \neg \text{gt} \ x \ x) \implies \neg msetext_huet \ gt \ xs \ xs$

<proof>

lemma *msetext_huet_trans_from_irrefl*:

assumes

fin: *finite A* **and**

zs_a: *zs* \in *lists A* **and** *ys_a*: *ys* \in *lists A* **and** *xs_a*: *xs* \in *lists A* **and**

irrefl: $\forall x \in A. \neg \text{gt } x \ x$ **and**

trans: $\forall z \in A. \forall y \in A. \forall x \in A. \text{gt } z \ y \longrightarrow \text{gt } y \ x \longrightarrow \text{gt } z \ x$ **and**

zs_gt_ys: *msetext_huet gt zs ys* **and**

ys_gt_xs: *msetext_huet gt ys xs*

shows *msetext_huet gt zs xs*

<proof>

lemma *msetext_huet_snoc*: *msetext_huet gt (xs @ [x]) xs*

<proof>

lemma *msetext_huet_compat_cons*: *msetext_huet gt ys xs \implies msetext_huet gt (x # ys) (x # xs)*

<proof>

lemma *msetext_huet_compat_snoc*: *msetext_huet gt ys xs \implies msetext_huet gt (ys @ [x]) (xs @ [x])*

<proof>

lemma *msetext_huet_compat_list*: *y \neq x \implies gt y x \implies msetext_huet gt (xs @ y # xs') (xs @ x # xs')*

<proof>

lemma *msetext_huet_singleton*: *y \neq x \implies msetext_huet gt [y] [x] \longleftrightarrow gt y x*

<proof>

lemma *msetext_huet_wf*: *wfP ($\lambda x y. \text{gt } y \ x$) \implies wfP ($\lambda xs ys. \text{msetext_huet } \text{gt } ys \ xs$)*

<proof>

lemma *msetext_huet_hd_or_tl*:

assumes

trans: $\forall z y x. \text{gt } z \ y \longrightarrow \text{gt } y \ x \longrightarrow \text{gt } z \ x$ **and**

total: $\forall y x. \text{gt } y \ x \vee \text{gt } x \ y \vee y = x$ **and**

len_eq: *length ys = length xs* **and**

yys_gt_xxs: *msetext_huet gt (y # ys) (x # xs)*

shows *gt y x \vee msetext_huet gt ys xs*

<proof>

interpretation *msetext_huet*: *ext msetext_huet*

<proof>

interpretation *msetext_huet*: *ext_irrefl_before_trans msetext_huet*

<proof>

interpretation *msetext_huet*: *ext_snoc msetext_huet*

<proof>

interpretation *msetext_huet*: *ext_compat_cons msetext_huet*

<proof>

interpretation *msetext_huet*: *ext_compat_snoc msetext_huet*

<proof>

interpretation *msetext_huet*: *ext_compat_list msetext_huet*

<proof>

interpretation *msetext_huet*: *ext_singleton msetext_huet*

<proof>

interpretation *msetext_huet*: *ext_wf msetext_huet*

<proof>

interpretation *msetext_huet*: *ext_hd_or_tl msetext_huet*
<proof>

interpretation *msetext_huet*: *ext_wf_bounded msetext_huet*
<proof>

5.9 Componentwise Extension

definition *wiseext* :: ('a ⇒ 'a ⇒ bool) ⇒ 'a list ⇒ 'a list ⇒ bool **where**
wiseext gt ys xs ↔ *length ys = length xs*
∧ (∀ i < *length ys*. *gt (ys ! i) (xs ! i)* ∨ *ys ! i = xs ! i*)
∧ (∃ i < *length ys*. *gt (ys ! i) (xs ! i)*)

lemma *wiseext_imp_len_lexext*:
assumes *cw*: *wiseext gt ys xs*
shows *len_lexext gt ys xs*
<proof>

lemma *wiseext_mono_strong*:
(∀ y ∈ *set ys*. ∀ x ∈ *set xs*. *gt y x* → *gt' y x*) ⇒ *wiseext gt ys xs* ⇒ *wiseext gt' ys xs*
<proof>

lemma *wiseext_map_strong*:
(∀ y ∈ *set ys*. ∀ x ∈ *set xs*. *gt y x* → *gt (f y) (f x)*) ⇒ *wiseext gt ys xs* ⇒
wiseext gt (map f ys) (map f xs)
<proof>

lemma *wiseext_irrefl*: (∀ x ∈ *set xs*. ¬ *gt x x*) ⇒ ¬ *wiseext gt xs xs*
<proof>

lemma *wiseext_trans_strong*:
assumes
∀ z ∈ *set zs*. ∀ y ∈ *set ys*. ∀ x ∈ *set xs*. *gt z y* → *gt y x* → *gt z x* **and**
wiseext gt zs ys **and** *wiseext gt ys xs*
shows *wiseext gt zs xs*
<proof>

lemma *wiseext_compat_cons*: *wiseext gt ys xs* ⇒ *wiseext gt (x # ys) (x # xs)*
<proof>

lemma *wiseext_compat_snoc*: *wiseext gt ys xs* ⇒ *wiseext gt (ys @ [x]) (xs @ [x])*
<proof>

lemma *wiseext_compat_list*:
assumes *y_gt_x*: *gt y x*
shows *wiseext gt (xs @ y # xs')* (*xs @ x # xs'*)
<proof>

lemma *wiseext_singleton*: *wiseext gt [y] [x]* ↔ *gt y x*
<proof>

lemma *wiseext_wf*: *wfP (λx y. gt y x)* ⇒ *wfP (λxs ys. wiseext gt ys xs)*
<proof>

lemma *wiseext_hd_or_tl*: *wiseext gt (y # ys) (x # xs)* ⇒ *gt y x* ∨ *wiseext gt ys xs*
<proof>

locale *ext_wiseext* = *ext_compat_list* + *ext_compat_cons*
begin

context
fixes *gt* :: 'a ⇒ 'a ⇒ bool
assumes

```

  gt_irrefl:  $\neg$  gt x x and
  trans_gt: ext gt zs ys  $\implies$  ext gt ys xs  $\implies$  ext gt zs xs
begin

lemma
  assumes ys_gtcw_xs: wiseext gt ys xs
  shows ext gt ys xs
  <proof>

end

end

interpretation wiseext: ext wiseext
  <proof>

interpretation wiseext: ext_irrefl_trans_strong wiseext
  <proof>

interpretation wiseext: ext_compat_cons wiseext
  <proof>

interpretation wiseext: ext_compat_snoc wiseext
  <proof>

interpretation wiseext: ext_compat_list wiseext
  <proof>

interpretation wiseext: ext_singleton wiseext
  <proof>

interpretation wiseext: ext_wf wiseext
  <proof>

interpretation wiseext: ext_hd_or_tl wiseext
  <proof>

interpretation wiseext: ext_wf_bounded wiseext
  <proof>

end

```

6 The Applicative Recursive Path Order for Lambda-Free Higher-Order Terms

```

theory Lambda_Free_RPO_App
imports Lambda_Free_Term_Extension_Orders
abbrevs >t = >t
  and  $\geq$ t =  $\geq$ t
begin

```

This theory defines the applicative recursive path order (RPO), a variant of RPO for λ -free higher-order terms. It corresponds to the order obtained by applying the standard first-order RPO on the applicative encoding of higher-order terms and assigning the lowest precedence to the application symbol.

```

locale rpo_app = gt_sym (>s)
  for gt_sym :: 's  $\Rightarrow$  's  $\Rightarrow$  bool (infix >s 50) +
  fixes ext :: (('s, 'v) tm  $\Rightarrow$  ('s, 'v) tm  $\Rightarrow$  bool)  $\Rightarrow$  ('s, 'v) tm list  $\Rightarrow$  ('s, 'v) tm list  $\Rightarrow$  bool
  assumes
    ext_ext_trans_before_irrefl: ext_trans_before_irrefl ext and
    ext_ext_compat_list: ext_compat_list ext
begin

```

lemma *ext_mono*[*mono*]: $gt \leq gt' \implies ext\ gt \leq ext\ gt'$
 ⟨*proof*⟩

inductive *gt* :: ('s, 'v) tm \Rightarrow ('s, 'v) tm \Rightarrow bool (**infix** >_t 50) **where**
 | *gt_sub*: $is_App\ t \implies (fun\ t\ >_t\ s \vee fun\ t = s) \vee (arg\ t\ >_t\ s \vee arg\ t = s) \implies t >_t\ s$
 | *gt_sym_sym*: $g >_s\ f \implies Hd\ (Sym\ g) >_t\ Hd\ (Sym\ f)$
 | *gt_sym_app*: $Hd\ (Sym\ g) >_t\ s1 \implies Hd\ (Sym\ g) >_t\ s2 \implies Hd\ (Sym\ g) >_t\ App\ s1\ s2$
 | *gt_app_app*: $ext\ (>_t)\ [t1,\ t2]\ [s1,\ s2] \implies App\ t1\ t2 >_t\ s1 \implies App\ t1\ t2 >_t\ s2 \implies$
 $App\ t1\ t2 >_t\ App\ s1\ s2$

abbreviation *ge* :: ('s, 'v) tm \Rightarrow ('s, 'v) tm \Rightarrow bool (**infix** \geq_t 50) **where**
 $t \geq_t\ s \equiv t >_t\ s \vee t = s$

end

end

7 The Graceful Recursive Path Order for Lambda-Free Higher-Order Terms

theory *Lambda_Free_RPO_Std*
imports *Lambda_Free_Term_Extension_Orders_Nested_Multisets_Ordinals.Multiset_More*
abbrevs >_t = >_t
 and $\geq_t = \geq_t$
begin

This theory defines the graceful recursive path order (RPO) for λ -free higher-order terms.

7.1 Setup

locale *rpo_basis* = *ground_heads* (>_s) *arity_sym* *arity_var*
for
 | *gt_sym* :: 's \Rightarrow 's \Rightarrow bool (**infix** >_s 50) **and**
 | *arity_sym* :: 's \Rightarrow enat **and**
 | *arity_var* :: 'v \Rightarrow enat +
fixes
 | *extf* :: 's \Rightarrow (('s, 'v) tm \Rightarrow ('s, 'v) tm \Rightarrow bool) \Rightarrow ('s, 'v) tm list \Rightarrow ('s, 'v) tm list \Rightarrow bool
assumes
 | *extf_ext_trans_before_irrefl*: *ext_trans_before_irrefl* (*extf* *f*) **and**
 | *extf_ext_compat_cons*: *ext_compat_cons* (*extf* *f*) **and**
 | *extf_ext_compat_list*: *ext_compat_list* (*extf* *f*)
begin

lemma *extf_ext_trans*: *ext_trans* (*extf* *f*)
 ⟨*proof*⟩

lemma *extf_ext*: *ext* (*extf* *f*)
 ⟨*proof*⟩

lemmas *extf_mono_strong* = *ext_mono_strong*[*OF* *extf_ext*]

lemmas *extf_mono* = *ext_mono*[*OF* *extf_ext*, *mono*]

lemmas *extf_map* = *ext_map*[*OF* *extf_ext*]

lemmas *extf_trans* = *ext_trans.trans*[*OF* *extf_ext_trans*]

lemmas *extf_irrefl_from_trans* =
ext_trans_before_irrefl.irrefl_from_trans[*OF* *extf_ext_trans_before_irrefl*]

lemmas *extf_compat_append_left* = *ext_compat_cons.compat_append_left*[*OF* *extf_ext_compat_cons*]

lemmas *extf_compat_list* = *ext_compat_list.compat_list*[*OF* *extf_ext_compat_list*]

definition *chkvar* :: ('s, 'v) tm \Rightarrow ('s, 'v) tm \Rightarrow bool **where**
 [*simp*]: *chkvar* *t* *s* \longleftrightarrow *vars_hd* (*head* *s*) \subseteq *vars* *t*

end

```

locale rpo = rpo_basis __ arity_sym arity_var
for
  arity_sym :: 's ⇒ enat and
  arity_var :: 'v ⇒ enat
begin

```

7.2 Inductive Definitions

definition

```

chksubs :: (('s, 'v) tm ⇒ ('s, 'v) tm ⇒ bool) ⇒ ('s, 'v) tm ⇒ ('s, 'v) tm ⇒ bool
where
[simp]: chksubs gt t s ⇔ (case s of App s1 s2 ⇒ gt t s1 ∧ gt t s2 | _ ⇒ True)

```

lemma *chksubs_mono*[*mono*]: $gt \leq gt' \implies \text{chksubs } gt \leq \text{chksubs } gt'$
<proof>

inductive *gt* :: ('s, 'v) tm ⇒ ('s, 'v) tm ⇒ bool (**infix** >_t 50) **where**
gt_sub: $\text{is_App } t \implies (\text{fun } t >_t s \vee \text{fun } t = s) \vee (\text{arg } t >_t s \vee \text{arg } t = s) \implies t >_t s$
| *gt_diff*: $\text{head } t >_{hd} \text{head } s \implies \text{chkvar } t s \implies \text{chksubs } (>_t) t s \implies t >_t s$
| *gt_same*: $\text{head } t = \text{head } s \implies \text{chksubs } (>_t) t s \implies$
 $(\forall f \in \text{ground_heads } (\text{head } t). \text{extf } f (>_t) (\text{args } t) (\text{args } s)) \implies t >_t s$

abbreviation *ge* :: ('s, 'v) tm ⇒ ('s, 'v) tm ⇒ bool (**infix** ≥_t 50) **where**
 $t \geq_t s \equiv t >_t s \vee t = s$

inductive *gt_sub* :: ('s, 'v) tm ⇒ ('s, 'v) tm ⇒ bool **where**
gt_subI: $\text{is_App } t \implies \text{fun } t \geq_t s \vee \text{arg } t \geq_t s \implies \text{gt_sub } t s$

inductive *gt_diff* :: ('s, 'v) tm ⇒ ('s, 'v) tm ⇒ bool **where**
gt_diffI: $\text{head } t >_{hd} \text{head } s \implies \text{chkvar } t s \implies \text{chksubs } (>_t) t s \implies \text{gt_diff } t s$

inductive *gt_same* :: ('s, 'v) tm ⇒ ('s, 'v) tm ⇒ bool **where**
gt_sameI: $\text{head } t = \text{head } s \implies \text{chksubs } (>_t) t s \implies$
 $(\forall f \in \text{ground_heads } (\text{head } t). \text{extf } f (>_t) (\text{args } t) (\text{args } s)) \implies \text{gt_same } t s$

lemma *gt_iff_sub_diff_same*: $t >_t s \iff \text{gt_sub } t s \vee \text{gt_diff } t s \vee \text{gt_same } t s$
<proof>

7.3 Transitivity

lemma *gt_fun_imp*: $\text{fun } t >_t s \implies t >_t s$
<proof>

lemma *gt_arg_imp*: $\text{arg } t >_t s \implies t >_t s$
<proof>

lemma *gt_imp_vars*: $t >_t s \implies \text{vars } t \supseteq \text{vars } s$
<proof>

theorem *gt_trans*: $u >_t t \implies t >_t s \implies u >_t s$
<proof>

7.4 Irreflexivity

theorem *gt_irrefl*: $\neg s >_t s$
<proof>

lemma *gt_antisym*: $t >_t s \implies \neg s >_t t$
<proof>

7.5 Subterm Property

lemma

gt_sub_fun: $App\ s\ t\ >_t\ s$ and
gt_sub_arg: $App\ s\ t\ >_t\ t$
 ⟨proof⟩

theorem *gt_proper_sub*: $proper_sub\ s\ t \implies t >_t s$
 ⟨proof⟩

7.6 Compatibility with Functions

lemma *gt_compat_fun*:
 assumes t'_gt_t : $t' >_t t$
 shows $App\ s\ t' >_t App\ s\ t$
 ⟨proof⟩

theorem *gt_compat_fun_strong*:
 assumes t'_gt_t : $t' >_t t$
 shows $apps\ s\ (t' \# us) >_t apps\ s\ (t \# us)$
 ⟨proof⟩

7.7 Compatibility with Arguments

theorem *gt_diff_same_compat_arg*:
 assumes
 extf_compat_snoc: $\bigwedge f. ext_compat_snoc\ (extf\ f)$ and
 diff_same: $gt_diff\ s'\ s \vee gt_same\ s'\ s$
 shows $App\ s'\ t >_t App\ s\ t$
 ⟨proof⟩

7.8 Stability under Substitution

lemma *gt_imp_chksubs_gt*:
 assumes t_gt_s : $t >_t s$
 shows $chksubs\ (>_t)\ t\ s$
 ⟨proof⟩

theorem *gt_subst*:
 assumes *wary_ρ*: $wary_subst\ \rho$
 shows $t >_t s \implies subst\ \rho\ t >_t subst\ \rho\ s$
 ⟨proof⟩

7.9 Totality on Ground Terms

theorem *gt_total_ground*:
 assumes *extf_total*: $\bigwedge f. ext_total\ (extf\ f)$
 shows $ground\ t \implies ground\ s \implies t >_t s \vee s >_t t \vee t = s$
 ⟨proof⟩

7.10 Well-foundedness

abbreviation *gtg* :: $(s, v)\ tm \Rightarrow (s, v)\ tm \Rightarrow bool$ (**infix** $>_{tg}$ 50) **where**
 $(>_{tg}) \equiv \lambda t\ s. ground\ t \wedge t >_t s$

theorem *gt_wf*:
 assumes *extf_wf*: $\bigwedge f. ext_wf\ (extf\ f)$
 shows $wfP\ (\lambda s\ t. t >_t s)$
 ⟨proof⟩

end

end

8 The Optimized Graceful Recursive Path Order for Lambda-Free Higher-Order Terms

```
theory Lambda_Free_RPO_Optim
imports Lambda_Free_RPO_Std
begin
```

This theory defines the optimized variant of the graceful recursive path order (RPO) for λ -free higher-order terms.

8.1 Setup

```
locale rpo_optim = rpo_basis __ arity_sym arity_var
  for
    arity_sym :: 's  $\Rightarrow$  enat and
    arity_var :: 'v  $\Rightarrow$  enat +
  assumes extf_ext_snoc: ext_snoc (extf f)
begin
```

```
lemmas extf_snoc = ext_snoc.snoc[OF extf_ext_snoc]
```

8.2 Definition of the Order

definition

```
chkargs :: (('s, 'v) tm  $\Rightarrow$  ('s, 'v) tm  $\Rightarrow$  bool)  $\Rightarrow$  ('s, 'v) tm  $\Rightarrow$  ('s, 'v) tm  $\Rightarrow$  bool
```

where

```
[simp]: chkargs gt t s  $\iff$  ( $\forall s' \in \text{set}(\text{args } s).$  gt t s')
```

```
lemma chkargs_mono[mono]: gt  $\leq$  gt'  $\implies$  chkargs gt  $\leq$  chkargs gt'
  <proof>
```

inductive gt :: ('s, 'v) tm \Rightarrow ('s, 'v) tm \Rightarrow bool (**infix** $>_t$ 50) **where**

```
gt_arg: ti  $\in$  set (args t)  $\implies$  ti  $>_t$  s  $\vee$  ti = s  $\implies$  t  $>_t$  s
| gt_diff: head t  $>_{hd}$  head s  $\implies$  chkvar t s  $\implies$  chkargs ( $>_t$ ) t s  $\implies$  t  $>_t$  s
| gt_same: head t = head s  $\implies$  chkargs ( $>_t$ ) t s  $\implies$ 
  ( $\forall f \in \text{ground\_heads}(\text{head } t).$  extf f ( $>_t$ ) (args t) (args s))  $\implies$  t  $>_t$  s
```

abbreviation ge :: ('s, 'v) tm \Rightarrow ('s, 'v) tm \Rightarrow bool (**infix** \geq_t 50) **where**
 $t \geq_t s \equiv t >_t s \vee t = s$

8.3 Transitivity

```
lemma gt_in_args_imp: ti  $\in$  set (args t)  $\implies$  ti  $>_t$  s  $\implies$  t  $>_t$  s
  <proof>
```

```
lemma gt_imp_vars: t  $>_t$  s  $\implies$  vars t  $\supseteq$  vars s
  <proof>
```

```
lemma gt_trans: u  $>_t$  t  $\implies$  t  $>_t$  s  $\implies$  u  $>_t$  s
  <proof>
```

```
lemma gt_sub_fun: App s t  $>_t$  s
  <proof>
```

end

8.4 Conditional Equivalence with Unoptimized Version

```
context rpo
begin
```

context

```
assumes extf_ext_snoc:  $\bigwedge f.$  ext_snoc (extf f)
```

begin

lemma *rpo_optim*: *rpo_optim ground_heads_var* ($>_s$) *extf arity_sym arity_var*
<proof>

abbreviation

chkargs :: $((s, v) tm \Rightarrow (s, v) tm \Rightarrow bool) \Rightarrow (s, v) tm \Rightarrow (s, v) tm \Rightarrow bool$

where

chkargs \equiv *rpo_optim.chkargs*

abbreviation *gt_optim* :: $(s, v) tm \Rightarrow (s, v) tm \Rightarrow bool$ (**infix** $>_{to}$ 50) **where**
 $>_{to} \equiv$ *rpo_optim.gt ground_heads_var* ($>_s$) *extf*

abbreviation *ge_optim* :: $(s, v) tm \Rightarrow (s, v) tm \Rightarrow bool$ (**infix** \geq_{to} 50) **where**
 $\geq_{to} \equiv$ *rpo_optim.ge ground_heads_var* ($>_s$) *extf*

theorem *gt_iff_optim*: $t >_t s \iff t >_{to} s$
<proof>

end

end

end

9 An Encoding of Lambdas in Lambda-Free Higher-Order Logic

theory *Lambda_Encoding*

imports *Lambda_Free_Term*

begin

This theory defines an encoding of λ -expressions as λ -free higher-order terms.

locale *lambda_encoding* =

fixes

lam :: s **and**

db :: $nat \Rightarrow s$

begin

definition *is_db* :: $s \Rightarrow bool$ **where**
is_db $f \iff (\exists i. f = db\ i)$

fun *subst_db* :: $nat \Rightarrow v \Rightarrow (s, v) tm \Rightarrow (s, v) tm$ **where**
subst_db $i\ x\ (Hd\ \zeta) = Hd\ (if\ \zeta = Var\ x\ then\ Sym\ (db\ i)\ else\ \zeta)$
 $|$ *subst_db* $i\ x\ (App\ s\ t) =$
App (*subst_db* $i\ x\ s$) (*subst_db* (*if* *head* $s = Sym\ lam$ *then* $i + 1$ *else* i) $x\ t$)

definition *raw_db_subst* :: $nat \Rightarrow v \Rightarrow v \Rightarrow (s, v) tm$ **where**
raw_db_subst $i\ x = (\lambda y. Hd\ (if\ y = x\ then\ Sym\ (db\ i)\ else\ Var\ y))$

lemma *vars_mset_subst_db*: $vars_mset\ (subst_db\ i\ x\ s) = \{\#y \in \# vars_mset\ s. y \neq x\# \}$
<proof>

lemma *head_subst_db*: $head\ (subst_db\ i\ x\ s) = head\ (subst\ (raw_db_subst\ i\ x)\ s)$
<proof>

lemma *args_subst_db*:

args (*subst_db* $i\ x\ s$) = *map* (*subst_db* (*if* *head* $s = Sym\ lam$ *then* $i + 1$ *else* i) x) (*args* s)
<proof>

lemma *var_mset_subst_db_subseteq*:

$vars_mset\ s \subseteq \# vars_mset\ t \implies vars_mset\ (subst_db\ i\ x\ s) \subseteq \# vars_mset\ (subst_db\ i\ x\ t)$
<proof>

end

end

10 Recursive Path Orders for Lambda-Free Higher-Order Terms

theory *Lambda_Free_RPOs*

imports *Lambda_Free_RPO_App Lambda_Free_RPO_Optim Lambda_Encoding*

begin

locale *simple_rpo_instances*

begin

definition *arity_sym* :: *nat* \Rightarrow *enat* **where**

arity_sym *n* = ∞

definition *arity_var* :: *nat* \Rightarrow *enat* **where**

arity_var *n* = ∞

definition *ground_head_var* :: *nat* \Rightarrow *nat set* **where**

ground_head_var *x* = *UNIV*

definition *gt_sym* :: *nat* \Rightarrow *nat* \Rightarrow *bool* **where**

gt_sym *g* *f* \longleftrightarrow *g* > *f*

sublocale *app*: *rpo_app* *gt_sym* *len_lexext*

<proof>

sublocale *std*: *rpo* *ground_head_var* *gt_sym* $\lambda f.$ *len_lexext* *arity_sym* *arity_var*

<proof>

sublocale *optim*: *rpo_optim* *ground_head_var* *gt_sym* $\lambda f.$ *len_lexext* *arity_sym* *arity_var*

<proof>

end

end