Formalization of Knuth–Bendix Orders for Lambda-Free Higher-Order Terms

Heiko Becker, Jasmin Christian Blanchette, Uwe Waldmann, and Daniel Wand

August 3, 2025

Abstract

This Isabelle/HOL formalization defines Knuth–Bendix orders for higher-order terms without λ-abstraction and proves many useful properties about them. The main order fully coincides with the standard transfinite KBO with subterm coefficients on first-order terms. It appears promising as the basis of a higher-order superposition calculus.

Contents

1 Introduction

This Isabelle/HOL formalization defines Knuth–Bendix orders for higher-order terms without λ-abstraction and proves many useful properties about them. The main order fully coincides with the standard transfinite KBO with subterm coefficients on first-order terms. It appears promising as the basis of a higher-order superposition calculus.

We refer to the following conference paper for details:

> Heiko Becker, Jasmin Christian Blanchette, Uwe Waldmann, Daniel Wand:
> A Transfinite Knuth-Bendix Order for Lambda-Free Higher-Order Terms.
> CADE 2017: 432-453
> https://www21.in.tum.de/~blanchet/lambda_free_kbo_conf.pdf

2 Utilities for Knuth–Bendix Orders for Lambda-Free Higher-Order Terms

```
theory Lambda_Free_KBO_Util
imports Lambda_Free_RPOs.Lambda_Free_Term Lambda_Free_RPOs.Extension_Orders Polynomials.Polynomials
begin

locale kbo_basic_basis = gt_sym (>s)
    for gt_sym :: 's ⇒ 's ⇒ bool (infix ‹>s› 50) +
  fixes
    wt_sym :: 's ⇒ nat and
    ε :: nat and
    ground_heads_var :: 'v ⇒ 's set and
    extf :: 's ⇒ (('s, 'v) tm ⇒ ('s, 'v) tm ⇒ bool) ⇒ ('s, 'v) tm list ⇒ ('s, 'v) tm list ⇒
      bool
  assumes
    ε_gt_0: ε > 0 and
    wt_sym_ge_ε: wt_sym f ≥ ε and
    ground_heads_var_nonempty: ground_heads_var x ≠ {} and
    extf_ext_irrefl_before_trans: ext_irrefl_before_trans (extf f) and
    extf_ext_compat_list_strong: ext_compat_list_strong (extf f) and
    extf_ext_hd_or_tl: ext_hd_or_tl (extf f)
begin

lemma wt_sym_gt_0: wt_sym f > 0
  by (rule less_le_trans[OF ε_gt_0 wt_sym_ge_ε])

end

locale kbo_std_basis = ground_heads (>s) arity_sym arity_var
    for
      gt_sym :: 's ⇒ 's ⇒ bool (infix ‹>s› 50) and
      arity_sym :: 's ⇒ enat and
      arity_var :: 'v ⇒ enat +
  fixes
    wt_sym :: 's ⇒ 'n::{ord,semiring_1} and
    ε :: nat and
    δ :: nat and
    extf :: 's ⇒ (('s, 'v) tm ⇒ ('s, 'v) tm ⇒ bool) ⇒ ('s, 'v) tm list ⇒ ('s, 'v) tm list ⇒
      bool
  assumes
    ε_gt_0: ε > 0 and
    δ_le_ε: δ ≤ ε and
    arity_hd_ne_infinity_if_δ_gt_0: δ > 0 ⟹ arity_hd ζ ≠ ∞ and
    wt_sym_ge: wt_sym f ≥ of_nat (ε − the_enat (of_nat δ ∗ arity_sym f)) and
    unary_wt_sym_0_gt: arity_sym f = 1 ⟹ wt_sym f = 0 ⟹ f >s g ∨ g = f and
    unary_wt_sym_0_imp_δ_eq_ε: arity_sym f = 1 ⟹ wt_sym f = 0 ⟹ δ = ε and
    extf_ext_irrefl_before_trans: ext_irrefl_before_trans (extf f) and
```

```
  extf_ext_compat_list_strong: ext_compat_list_strong (extf f) and
  extf_ext_hd_or_tl: ext_hd_or_tl (extf f) and
  extf_ext_snoc_if_δ_eq_ε: δ = ε ⟹ ext_snoc (extf f)
begin

lemma arity_sym_ne_infinity_if_δ_gt_0: δ > 0 ⟹ arity_sym f ≠ ∞
  by (metis arity_hd.simps(2) arity_hd_ne_infinity_if_δ_gt_0)

lemma arity_var_ne_infinity_if_δ_gt_0: δ > 0 ⟹ arity_var x ≠ ∞
  by (metis arity_hd.simps(1) arity_hd_ne_infinity_if_δ_gt_0)

lemma arity_ne_infinity_if_δ_gt_0: δ > 0 ⟹ arity s ≠ ∞
  unfolding arity_def
  by (induct s rule: tm_induct_apps)
    (metis arity_hd_ne_infinity_if_δ_gt_0 enat.distinct(2) enat.exhaust idiff_enat_enat)

lemma extf_ext_irrefl: ext_irrefl (extf f)
  by (rule ext_irrefl_before_trans.axioms(1)[OF extf_ext_irrefl_before_trans])

lemma extf_ext: ext (extf f)
  by (rule ext_irrefl.axioms(1)[OF extf_ext_irrefl])

lemma
  extf_ext_compat_cons: ext_compat_cons (extf f) and
  extf_ext_compat_snoc: ext_compat_snoc (extf f) and
  extf_ext_singleton: ext_singleton (extf f)
  by (rule ext_compat_list_strong.axioms[OF extf_ext_compat_list_strong])+

lemma extf_ext_compat_list: ext_compat_list (extf f)
  using extf_ext_compat_list_strong
  by (simp add: ext_compat_list_axioms_def ext_compat_list_def ext_compat_list_strong.compat_list
    ext_compat_list_strong_def ext_singleton.axioms(1))

lemma extf_ext_wf_bounded: ext_wf_bounded (extf f)
  unfolding ext_wf_bounded_def using extf_ext_irrefl_before_trans extf_ext_hd_or_tl by simp

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_irrefl = ext_irrefl.irrefl[OF extf_ext_irrefl]
lemmas extf_trans_from_irrefl =
  ext_irrefl_before_trans.trans_from_irrefl[OF extf_ext_irrefl_before_trans]
lemmas extf_compat_cons = ext_compat_cons.compat_cons[OF extf_ext_compat_cons]
lemmas extf_compat_append_left = ext_compat_cons.compat_append_left[OF extf_ext_compat_cons]
lemmas extf_compat_append_right = ext_compat_snoc.compat_append_right[OF extf_ext_compat_snoc]
lemmas extf_compat_list = ext_compat_list.compat_list[OF extf_ext_compat_list]
lemmas extf_singleton = ext_singleton.singleton[OF extf_ext_singleton]
lemmas extf_wf_bounded = ext_wf_bounded.wf_bounded[OF extf_ext_wf_bounded]

lemmas extf_snoc_if_δ_eq_ε = ext_snoc.snoc[OF extf_ext_snoc_if_δ_eq_ε]

lemma extf_singleton_nil_if_δ_eq_ε: δ = ε ⟹ extf f gt [s] []
  by (rule extf_snoc_if_δ_eq_ε[of _ _ [], simplified])

end

sublocale kbo_basic_basis < kbo_std_basis _ _ λ_. ∞ λ_. ∞ _ _ 0
  unfolding kbo_std_basis_def kbo_std_basis_axioms_def
  by (auto simp: wt_sym_gt_0 ε_gt_0 wt_sym_ge_ε less_not_refl2 ground_heads_var_nonempty
    gt_sym_axioms ground_heads_def ground_heads_axioms_def extf_ext_irrefl_before_trans
    extf_ext_compat_list_strong extf_ext_hd_or_tl)

end
```

3 The Applicative Knuth–Bendix Order for Lambda-Free Higher-Order Terms

```
theory Lambda_Free_KBO_App
imports Lambda_Free_KBO_Util
abbrevs >t = >ₜ
  and ≥t = ≥ₜ
begin
```

This theory defines the applicative Knuth–Bendix order, a variant of KBO for λ-free higher-order terms. It corresponds to the order obtained by applying the standard first-order KBO on the applicative encoding of higher-order terms and assigning the lowest precedence to the application symbol.

```
locale kbo_app = gt_sym (>ₛ)
    for gt_sym :: 's ⇒ 's ⇒ bool (infix ‹>ₛ› 50) +
  fixes
    wt_sym :: 's ⇒ nat and
    ε :: nat and
    ext :: (('s, 'v) tm ⇒ ('s, 'v) tm ⇒ bool) ⇒ ('s, 'v) tm list ⇒ ('s, 'v) tm list ⇒ bool
  assumes
    ε_gt_0: ε > 0 and
    wt_sym_ge_ε: wt_sym f ≥ ε and
    ext_ext_irrefl_before_trans: ext_irrefl_before_trans ext and
    ext_ext_compat_list: ext_compat_list ext and
    ext_ext_hd_or_tl: ext_hd_or_tl ext
begin

lemma ext_mono[mono]: gt ≤ gt' ⟹ ext gt ≤ ext gt'
  by (simp add: ext.mono ext_ext_compat_list[unfolded ext_compat_list_def, THEN conjunct1])

fun wt :: ('s, 'v) tm ⇒ nat where
  wt (Hd (Var x)) = ε
| wt (Hd (Sym f)) = wt_sym f
| wt (App s t) = wt s + wt t

inductive gt :: ('s, 'v) tm ⇒ ('s, 'v) tm ⇒ bool (infix ‹>ₜ› 50) where
  gt_wt: vars_mset t ⊇# vars_mset s ⟹ wt t > wt s ⟹ t >ₜ s
| gt_sym_sym: wt_sym g = wt_sym f ⟹ g >ₛ f ⟹ Hd (Sym g) >ₜ Hd (Sym f)
| gt_sym_app: vars s = {} ⟹ wt t = wt s ⟹ t = Hd (Sym g) ⟹ is_App s ⟹ t >ₜ s
| gt_app_app: vars_mset t ⊇# vars_mset s ⟹ wt t = wt s ⟹ t = App t1 t2 ⟹ s = App s1 s2 ⟹
    ext (>ₜ) [t1, t2] [s1, s2] ⟹ t >ₜ s

abbreviation ge :: ('s, 'v) tm ⇒ ('s, 'v) tm ⇒ bool (infix ‹≥ₜ› 50) where
  t ≥ₜ s ≡ t >ₜ s ∨ t = s

end

end
```

4 The Graceful Standard Knuth–Bendix Order for Lambda-Free Higher-Order Terms

```
theory Lambda_Free_KBO_Std
imports Lambda_Free_KBO_Util Nested_Multisets_Ordinals.Multiset_More
abbrevs >t = >ₜ
  and ≥t = ≥ₜ
begin
```

This theory defines the standard version of the graceful Knuth–Bendix order for λ-free higher-order terms. Standard means that one symbol is allowed to have a weight of 0.

4.1 Setup

```
locale kbo_std = kbo_std_basis _ _ arity_sym arity_var wt_sym
  for
    arity_sym :: 's ⇒ enat and
    arity_var :: 'v ⇒ enat and
    wt_sym :: 's ⇒ nat
begin
```

4.2 Weights

```
primrec wt :: ('s, 'v) tm ⇒ nat where
  wt (Hd ζ) = (LEAST w. ∃f ∈ ground_heads ζ. w = wt_sym f + the_enat (δ ∗ arity_sym f))
| wt (App s t) = (wt s − δ) + wt t

lemma wt_Hd_Sym: wt (Hd (Sym f)) = wt_sym f + the_enat (δ ∗ arity_sym f)
  by simp

lemma exists_wt_sym: ∃f ∈ ground_heads ζ. wt (Hd ζ) = wt_sym f + the_enat (δ ∗ arity_sym f)
  by (auto intro: Least_in_nonempty_set_imp_ex)

lemma wt_le_wt_sym: f ∈ ground_heads ζ =⇒ wt (Hd ζ) ≤ wt_sym f + the_enat (δ ∗ arity_sym f)
  using not_le_imp_less not_less_Least by fastforce

lemma enat_the_enat_δ_times_arity_sym[simp]: enat (the_enat (δ ∗ arity_sym f)) = δ ∗ arity_sym f
  using arity_sym_ne_infinity_if_δ_gt_0 imult_is_infinity zero_enat_def by fastforce

lemma wt_arg_le: wt (arg s) ≤ wt s
  by (cases s) auto

lemma wt_ge_ε: wt s ≥ ε
  by (induct s, metis exists_wt_sym of_nat_eq_enat le_diff_conv of_nat_id wt_sym_ge,
    simp add: add_increasing)

lemma wt_ge_δ: wt s ≥ δ
  by (meson δ_le_ε order.trans enat_ord_simps(1) wt_ge_ε)

lemma wt_gt_δ_if_superunary: arity_hd (head s) > 1 =⇒ wt s > δ
proof (induct s)
  case ζ: (Hd ζ)
  obtain g where
    g_in_grs: g ∈ ground_heads ζ and
    wt_ζ: wt (Hd ζ) = wt_sym g + the_enat (δ ∗ arity_sym g)
    using exists_wt_sym by blast

  have arity_hd ζ > 1
    using ζ by auto
  hence ary_g: arity_sym g > 1
    using ground_heads_arity[OF g_in_grs] by simp

  show ?case
  proof (cases δ = 0)
    case True
    thus ?thesis
      by (metis ε_gt_0 gr0I leD wt_ge_ε)
  next
    case δ_ne_0: False
    hence ary_g_ninf: arity_sym g ≠ ∞
      using arity_sym_ne_infinity_if_δ_gt_0 by blast
    hence δ < the_enat (enat δ ∗ arity_sym g)
      using δ_ne_0 ary_g by (cases arity_sym g) (auto simp: one_enat_def)
    thus ?thesis
      unfolding wt_ζ by simp
  qed
```

next
  case (App s t)
  thus ?case
    using wt_ge_δ[of t] **by** force
qed

lemma wt_App_δ: wt (App s t) = wt t ⟹ wt s = δ
  by (simp add: order.antisym wt_ge_δ)

lemma wt_App_ge_fun: wt (App s t) ≥ wt s
  by (metis diff_le_mono2 wt_ge_δ le_diff_conv wt.simps(2))

lemma wt_hd_le: wt (Hd (head s)) ≤ wt s
  by (induct s, simp) (metis head_App leD le_less_trans not_le_imp_less wt_App_ge_fun)

lemma wt_δ_imp_δ_eq_ε: wt s = δ ⟹ δ = ε
  by (metis δ_le_ε le_antisym wt_ge_ε)

lemma wt_ge_arity_head_if_δ_gt_0:
  assumes δ_gt_0: δ > 0
  shows wt s ≥ arity_hd (head s)
proof (induct s)
  case (Hd ζ)

  obtain f **where**
    f_in_ζ: f ∈ ground_heads ζ **and**
    wt_ζ: wt (Hd ζ) = wt_sym f + the_enat (δ ∗ arity_sym f)
    using exists_wt_sym **by** blast

  have arity_sym f ≥ arity_hd ζ
    by (rule ground_heads_arity[OF f_in_ζ])
  hence the_enat (δ ∗ arity_sym f) ≥ arity_hd ζ
    using δ_gt_0
    by (metis One_nat_def Suc_ile_eq dual_order.trans enat_ord_simps(2)
      enat_the_enat_δ_times_arity_sym i0_lb mult.commute mult.right_neutral mult_left_mono
      one_enat_def)
  thus ?case
    unfolding wt_ζ **by** (metis add.left_neutral add_mono le_iff_add plus_enat_simps(1) tm.sel(1))
next
  case App
  thus ?case
    by (metis dual_order.trans enat_ord_simps(1) head_App wt_App_ge_fun)
qed

lemma wt_ge_num_args_if_δ_eq_0:
  assumes δ_eq_0: δ = 0
  shows wt s ≥ num_args s
  by (induct s, simp_all,
    metis (no_types) δ_eq_0 ε_gt_0 wt_δ_imp_δ_eq_ε add_le_same_cancel1 le_0_eq le_trans
      minus_nat.diff_0 not_gr_zero not_less_eq_eq)

lemma wt_ge_num_args: wary s ⟹ wt s ≥ num_args s
  using wt_ge_arity_head_if_δ_gt_0 wt_ge_num_args_if_δ_eq_0
  by (meson order.trans enat_ord_simps(1) neq0_conv wary_num_args_le_arity_head)

4.3 Inductive Definitions

inductive gt :: ('s, 'v) tm ⇒ ('s, 'v) tm ⇒ bool (**infix** ‹$>_t$› 50) **where**
  gt_wt: vars_mset t ⊇# vars_mset s ⟹ wt t > wt s ⟹ t $>_t$ s
| gt_unary: wt t = wt s ⟹ ¬ head t $\le\ge_{hd}$ head s ⟹ num_args t = 1 ⟹
    (∃f ∈ ground_heads (head t). arity_sym f = 1 ∧ wt_sym f = 0) ⟹ arg t $>_t$ s ∨ arg t = s ⟹
    t $>_t$ s
| gt_diff: vars_mset t ⊇# vars_mset s ⟹ wt t = wt s ⟹ head t $>_{hd}$ head s ⟹ t $>_t$ s
| gt_same: vars_mset t ⊇# vars_mset s ⟹ wt t = wt s ⟹ head t = head s ⟹

$(\forall f \in \mathit{ground\_heads}\ (\mathit{head}\ t).\ \mathit{extf}\ f\ (>_t)\ (\mathit{args}\ t)\ (\mathit{args}\ s)) \Longrightarrow t >_t s$

abbreviation *ge* :: $('s, 'v)\ \mathit{tm} \Rightarrow ('s, 'v)\ \mathit{tm} \Rightarrow \mathit{bool}$ (**infix** ‹$\geq_t$› *50*) **where**
$t \geq_t s \equiv t >_t s \lor t = s$

inductive *gt_wt* :: $('s, 'v)\ \mathit{tm} \Rightarrow ('s, 'v)\ \mathit{tm} \Rightarrow \mathit{bool}$ **where**
gt_wtI: $\mathit{vars\_mset}\ t \supseteq\#\ \mathit{vars\_mset}\ s \Longrightarrow \mathit{wt}\ t > \mathit{wt}\ s \Longrightarrow \mathit{gt\_wt}\ t\ s$

inductive *gt_diff* :: $('s, 'v)\ \mathit{tm} \Rightarrow ('s, 'v)\ \mathit{tm} \Rightarrow \mathit{bool}$ **where**
gt_diffI: $\mathit{vars\_mset}\ t \supseteq\#\ \mathit{vars\_mset}\ s \Longrightarrow \mathit{wt}\ t = \mathit{wt}\ s \Longrightarrow \mathit{head}\ t >_{hd} \mathit{head}\ s \Longrightarrow \mathit{gt\_diff}\ t\ s$

inductive *gt_unary* :: $('s, 'v)\ \mathit{tm} \Rightarrow ('s, 'v)\ \mathit{tm} \Rightarrow \mathit{bool}$ **where**
gt_unaryI: $\mathit{wt}\ t = \mathit{wt}\ s \Longrightarrow \neg\ \mathit{head}\ t \leq\geq_{hd} \mathit{head}\ s \Longrightarrow \mathit{num\_args}\ t = 1 \Longrightarrow$
$(\exists f \in \mathit{ground\_heads}\ (\mathit{head}\ t).\ \mathit{arity\_sym}\ f = 1 \land \mathit{wt\_sym}\ f = 0) \Longrightarrow \mathit{arg}\ t \geq_t s \Longrightarrow \mathit{gt\_unary}\ t\ s$

inductive *gt_same* :: $('s, 'v)\ \mathit{tm} \Rightarrow ('s, 'v)\ \mathit{tm} \Rightarrow \mathit{bool}$ **where**
gt_sameI: $\mathit{vars\_mset}\ t \supseteq\#\ \mathit{vars\_mset}\ s \Longrightarrow \mathit{wt}\ t = \mathit{wt}\ s \Longrightarrow \mathit{head}\ t = \mathit{head}\ s \Longrightarrow$
$(\forall f \in \mathit{ground\_heads}\ (\mathit{head}\ t).\ \mathit{extf}\ f\ (>_t)\ (\mathit{args}\ t)\ (\mathit{args}\ s)) \Longrightarrow \mathit{gt\_same}\ t\ s$

lemma *gt_iff_wt_unary_diff_same*: $t >_t s \longleftrightarrow \mathit{gt\_wt}\ t\ s \lor \mathit{gt\_unary}\ t\ s \lor \mathit{gt\_diff}\ t\ s \lor \mathit{gt\_same}\ t\ s$
by (*subst gt.simps*) (*auto simp*: *gt_wt.simps gt_unary.simps gt_diff.simps gt_same.simps*)

lemma *gt_imp_vars_mset*: $t >_t s \Longrightarrow \mathit{vars\_mset}\ t \supseteq\#\ \mathit{vars\_mset}\ s$
by (*induct rule*: *gt.induct*) (*auto intro*: *subset_mset.trans*)

lemma *gt_imp_vars*: $t >_t s \Longrightarrow \mathit{vars}\ t \supseteq \mathit{vars}\ s$
using *set_mset_mono*[*OF gt_imp_vars_mset*] **by** *simp*

4.4 Irreflexivity

theorem *gt_irrefl*: $\mathit{wary}\ s \Longrightarrow \neg\ s >_t s$
proof (*induct size s arbitrary*: *s rule*: *less_induct*)
case *less*
note *ih* = *this*(*1*) **and** *wary_s* = *this*(*2*)

show *?case*
proof
assume *s_gt_s*: $s >_t s$
show *False*
using *s_gt_s*
proof (*cases rule*: *gt.cases*)
case *gt_same*
then obtain *f* **where** *f*: $\mathit{extf}\ f\ (>_t)\ (\mathit{args}\ s)\ (\mathit{args}\ s)$
by *fastforce*
thus *False*
using *wary_s ih* **by** (*metis wary_args extf_irrefl size_in_args*)
qed (*auto simp*: *comp_hd_def gt_hd_irrefl*)
qed
qed

4.5 Transitivity

lemma *gt_imp_wt_ge*: $t >_t s \Longrightarrow \mathit{wt}\ t \geq \mathit{wt}\ s$
by (*induct rule*: *gt.induct*) *auto*

lemma *not_extf_gt_nil_singleton_if_δ_eq_ε*:
assumes *wary_s*: *wary s* **and** *δ_eq_ε*: $\delta = \varepsilon$
shows $\neg\ \mathit{extf}\ f\ (>_t)\ []\ [s]$
proof
assume *nil_gt_s*: $\mathit{extf}\ f\ (>_t)\ []\ [s]$
note *s_gt_nil* = *extf_singleton_nil_if_δ_eq_ε*[*OF δ_eq_ε*, *of f gt s*]
have $\neg\ \mathit{extf}\ f\ (>_t)\ []\ []$
by (*rule extf_irrefl*) *simp*
moreover have $\mathit{extf}\ f\ (>_t)\ []\ []$

```
    using extf_trans_from_irrefl[of {s}, OF _ _ _ _ _ _ _ nil_gt_s s_gt_nil] gt_irrefl[OF wary_s]
    by fastforce
  ultimately show False
    by sat
qed

lemma gt_sub_arg: wary (App s t) ⟹ App s t >_t t
proof (induct t arbitrary: s rule: measure_induct_rule[of size])
  case (less t)
  note ih = this(1) and wary_st = this(2)

  {
    assume wt_st: wt (App s t) = wt t
    hence δ_eq_ε: δ = ε
      using wt_App_δ wt_δ_imp_δ_eq_ε by metis
    hence δ_gt_0: δ > 0
      using ε_gt_0 by simp

    have wt_s: wt s = δ
      by (rule wt_App_δ[OF wt_st])

    have
      wary_t: wary t and
      nargs_lt: num_args s < arity_hd (head s)
      using wary_st wary.simps by blast+

    have ary_hd_s: arity_hd (head s) = 1
      by (metis One_nat_def arity.wary_AppE dual_order.order_iff_strict eSuc_enat enat_defs(1)
        enat_defs(2) ileI1 linorder_not_le not_iless0 wary_st wt_gt_δ_if_superunary wt_s)
    hence nargs_s: num_args s = 0
      by (metis enat_ord_simps(2) less_one nargs_lt one_enat_def)
    have s = Hd (head s)
      by (simp add: Hd_head_id nargs_s)
    then obtain f where
      f_in: f ∈ ground_heads (head s) and
      wt_f_etc: wt_sym f + the_enat (δ * arity_sym f) = δ
      using exists_wt_sym wt_s by fastforce

    have ary_f_1: arity_sym f = 1
    proof −
      have ary_f_ge_1: arity_sym f ≥ 1
        using ary_hd_s f_in ground_heads_arity by fastforce
      hence enat δ * arity_sym f = δ
        using wt_f_etc by (metis enat_ord_simps(1) enat_the_enat_δ_times_arity_sym le_add2
          le_antisym mult.right_neutral mult_left_mono zero_le)
      thus ?thesis
        using δ_gt_0 by (cases arity_sym f) (auto simp: one_enat_def)
    qed
    hence wt_f_0: wt_sym f = 0
      using wt_f_etc by simp

    {
      assume hd_s_ncmp_t: ¬ head s ≤≥_hd head t
      have ?case
        by (rule gt_unary[OF wt_st]) (auto simp: hd_s_ncmp_t nargs_s intro: f_in ary_f_1 wt_f_0)
    }
    moreover
    {
      assume hd_s_gt_t: head s >_hd head t
      have ?case
        by (rule gt_diff) (auto simp: hd_s_gt_t wt_s[folded δ_eq_ε])
    }
    moreover
```

```
    {
      assume head t >_hd head s
      hence False
        using ary_f_1 exists_wt_sym f_in gt_hd_def gt_sym_antisym unary_wt_sym_0_gt wt_f_0 by blast
    }
    moreover
    {
      assume hd_t_eq_s: head t = head s
      hence nargs_t_le: num_args t ≤ 1
        using ary_hd_s wary_num_args_le_arity_head[OF wary_t] by (simp add: one_enat_def)

      have extf: extf f (>_t) [t] (args t) for f
      proof (cases args t)
        case Nil
        thus ?thesis
          by (simp add: extf_singleton_nil_if_δ_eq_ε[OF δ_eq_ε])
      next
        case args_t: (Cons ta ts)
        hence ts: ts = []
          using ary_hd_s[folded hd_t_eq_s] wary_num_args_le_arity_head[OF wary_t]
            nargs_t_le by simp
        have ta: ta = arg t
          by (metis apps.simps(1) apps.simps(2) args_t tm.sel(6) tm_collapse_apps ts)
        hence t: t = App (fun t) ta
          by (metis args.simps(1) args_t not_Cons_self2 tm.exhaust_sel ts)
        have t >_t ta
          by (rule ih[of ta fun t, folded t, OF _ wary_t]) (metis ta size_arg_lt t tm.disc(2))
        thus ?thesis
          unfolding args_t ts by (metis extf_singleton gt_irrefl wary_t)
      qed
      have ?case
        by (rule gt_same)
          (auto simp: hd_t_eq_s wt_s[folded δ_eq_ε] length_0_conv[THEN iffD1, OF nargs_s] extf)
    }
    ultimately have ?case
      unfolding comp_hd_def by metis
  }
  thus ?case
    using gt_wt by fastforce
qed

lemma gt_arg: wary s ⟹ is_App s ⟹ s >_t arg s
  by (cases s) (auto intro: gt_sub_arg)

theorem gt_trans: wary u ⟹ wary t ⟹ wary s ⟹ u >_t t ⟹ t >_t s ⟹ u >_t s
proof (simp only: atomize_imp,
    rule measure_induct_rule[of λ(u, t, s). {#size u, size t, size s#}
      λ(u, t, s). wary u ⟶ wary t ⟶ wary s ⟶ u >_t t ⟶ t >_t s ⟶ u >_t s (u, t, s),
      simplified prod.case],
    simp only: split_paired_all prod.case atomize_imp[symmetric])
  fix u t s
  assume
    ih: ⋀ua ta sa. {#size ua, size ta, size sa#} < {#size u, size t, size s#} ⟹
      wary ua ⟹ wary ta ⟹ wary sa ⟹ ua >_t ta ⟹ ta >_t sa ⟹ ua >_t sa and
    wary_u: wary u and wary_t: wary t and wary_s: wary s and
    u_gt_t: u >_t t and t_gt_s: t >_t s

  have vars_mset u ⊇# vars_mset t and vars_mset t ⊇# vars_mset s
    using u_gt_t t_gt_s by (auto simp: gt_imp_vars_mset)
  hence vars_u_s: vars_mset u ⊇# vars_mset s
    by auto

  have wt_u_ge_t: wt u ≥ wt t and wt_t_ge_s: wt t ≥ wt s
```

```
  using gt_imp_wt_ge u_gt_t t_gt_s by auto

{
  assume wt_t_s: wt t = wt s and wt_u_t: wt u = wt t
  hence wt_u_s: wt u = wt s
    by simp

  have wary_arg_u: wary (arg u)
    by (rule wary_arg[OF wary_u])
  have wary_arg_t: wary (arg t)
    by (rule wary_arg[OF wary_t])
  have wary_arg_s: wary (arg s)
    by (rule wary_arg[OF wary_s])

  have u >t s
    using t_gt_s
  proof cases
    case gt_unary_t_s: gt_unary

    have t_app: is_App t
      by (metis args_Nil_iff_is_Hd gt_unary_t_s(3) length_greater_0_conv less_numeral_extra(1))

    have δ_eq_ε: δ = ε
      using gt_unary_t_s(4) unary_wt_sym_0_imp_δ_eq_ε by blast

    show ?thesis
      using u_gt_t
    proof cases
      case gt_unary_u_t: gt_unary
      have u_app: is_App u
        by (metis args_Nil_iff_is_Hd gt_unary_u_t(3) length_greater_0_conv less_numeral_extra(1))
      hence arg_u_gt_s: arg u >t s
        using ih[of arg u t s] gt_unary_u_t(5) t_gt_s size_arg_lt wary_arg_u wary_s wary_t
        by force
      hence arg_u_ge_s: arg u ≥t s
        by sat

      {
        assume size (arg u) < size t
        hence ?thesis
          using ih[of u arg u s] arg_u_gt_s gt_arg by (simp add: u_app wary_arg_u wary_s wary_u)
      }
      moreover
      {
        assume size (arg t) < size s
        hence u >t arg t
          using ih[of u t arg t] args_Nil_iff_is_Hd gt_arg gt_unary_t_s(3) u_gt_t wary_t wary_u
          by force
        hence ?thesis
          using ih[of u arg t s] args_Nil_iff_is_Hd gt_unary_t_s(3,5) size_arg_lt wary_arg_t
            wary_s wary_u by force
      }
      moreover
      {
        assume sz_u_gt_t: size u > size t and sz_t_gt_s: size t > size s

        have wt_fun_u: wt (fun u) = δ
          by (metis antisym gt_imp_wt_ge gt_unary_u_t(5) tm.collapse(2) u_app wt_App_δ wt_arg_le
            wt_t_s wt_u_s)

        have nargs_fun_u: num_args (fun u) = 0
          by (metis args.simps(1) gt_unary_u_t(3) list.size(3) one_arg_imp_Hd tm.collapse(2)
            u_app)
```

```
{
  assume hd_u_eq_s: head u = head s
  hence ary_hd_s: arity_hd (head s) = 1
    using ground_heads_arity gt_unary_u_t(3,4) hd_u_eq_s one_enat_def
      wary_num_args_le_arity_head wary_u by fastforce

  have extf: extf f (>_t) (args u) (args s) for f
  proof (cases args s)
    case Nil
    thus ?thesis
      by (metis Hd_head_id δ_eq_ε append_Nil args.simps(2) extf_singleton_nil_if_δ_eq_ε
        gt_unary_u_t(3) head_fun length_greater_0_conv less_irrefl_nat nargs_fun_u
        tm.exhaust_sel zero_neq_one)
  next
    case args_s: (Cons sa ss)
    hence ss: ss = []
      by (cases s, simp, metis One_nat_def antisym_conv ary_hd_s diff_Suc_1
        enat_ord_simps(1) le_add2 length_0_conv length_Cons list.size(4) one_enat_def
        wary_num_args_le_arity_head wary_s)
    have sa: sa = arg s
      by (metis apps.simps(1) apps.simps(2) args_s tm.sel(6) tm_collapse_apps ss)

    have s_app: is_App s
      using args_Nil_iff_is_Hd args_s by force
    have args_u: args u = [arg u]
      by (metis append_Nil args.simps(2) args_Nil_iff_is_Hd gt_unary_u_t(3) length_0_conv
        nargs_fun_u tm.collapse(2) zero_neq_one)

    have max_sz_u_t_s: Max {size s, size t, size u} = size u
      using sz_t_gt_s sz_u_gt_t by auto

    have max_sz_arg_u_t_arg_t: Max {size (arg t), size t, size (arg u)} < size u
      using size_arg_lt sz_u_gt_t t_app u_app by fastforce

    have {#size (arg u), size t, size (arg t)#} < {#size u, size t, size s#}
      using max_sz_arg_u_t_arg_t
      by (simp add: Max_lt_imp_lt_mset insert_commute max_sz_u_t_s)
    hence arg_u_gt_arg_t: arg u >_t arg t
      using ih[OF _ wary_arg_u wary_t wary_arg_t] args_Nil_iff_is_Hd gt_arg
        gt_unary_t_s(3) gt_unary_u_t(5) wary_t by force

    have max_sz_arg_s_s_arg_t: Max {size (arg s), size s, size (arg t)} < size u
      using s_app t_app size_arg_lt sz_t_gt_s sz_u_gt_t by force

    have {#size (arg t), size s, size (arg s)#} < {#size u, size t, size s#}
      by (meson add_mset_lt_lt_lt less_trans mset_lt_single_iff s_app size_arg_lt
        sz_t_gt_s sz_u_gt_t t_app)
    hence arg_t_gt_arg_s: arg t >_t arg s
      using ih[OF _ wary_arg_t wary_s wary_arg_s]
        gt_unary_t_s(5) gt_arg args_Nil_iff_is_Hd args_s wary_s by force

    have arg u >_t arg s
      using ih[of arg u arg t arg s] arg_u_gt_arg_t arg_t_gt_arg_s
      by (simp add: add_mset_lt_le_lt less_imp_le_nat s_app size_arg_lt t_app u_app
        wary_arg_s wary_arg_t wary_arg_u)
    thus ?thesis
      unfolding args_u args_s ss sa by (metis extf_singleton gt_irrefl wary_arg_u)
  qed

  have ?thesis
    by (rule gt_same[OF vars_u_s wt_u_s hd_u_eq_s]) (simp add: extf)
}
```

```
      moreover
      {
        assume head u >hd head s
        hence ?thesis
          by (rule gt_diff[OF vars_u_s wt_u_s])
      }
      moreover
      {
        assume head s >hd head u
        hence False
          using gt_hd_def gt_hd_irrefl gt_sym_antisym gt_unary_u_t(4) unary_wt_sym_0_gt by blast
      }
      moreover
      {
        assume ¬ head u ≤≥hd head s
        hence ?thesis
          by (rule gt_unary[OF wt_u_s _ gt_unary_u_t(3,4) arg_u_ge_s])
      }
      ultimately have ?thesis
        unfolding comp_hd_def by sat
    }
    ultimately show ?thesis
      by (metis args_Nil_iff_is_Hd dual_order.strict_trans2 gt_unary_t_s(3) gt_unary_u_t(3)
        length_0_conv not_le_imp_less size_arg_lt zero_neq_one)
  next
    case gt_diff_u_t: gt_diff
    have False
      using gt_diff_u_t(3) gt_hd_def gt_hd_irrefl gt_sym_antisym gt_unary_t_s(4)
        unary_wt_sym_0_gt by blast
    thus ?thesis
      by sat
  next
    case gt_same_u_t: gt_same

    have hd_u_ncomp_s: ¬ head u ≤≥hd head s
      by (rule gt_unary_t_s(2)[folded gt_same_u_t(3)])

    have num_args u ≤ 1
      by (metis enat_ord_simps(1) ground_heads_arity gt_same_u_t(3) gt_unary_t_s(4) one_enat_def
        order_trans wary_num_args_le_arity_head wary_u)
    hence nargs_u: num_args u = 1
      by (cases args u,
        metis Hd_head_id δ_eq_ε append_Nil args.simps(2) gt_same_u_t(3,4) gt_unary_t_s(3,4)
          head_fun list.size(3) not_extf_gt_nil_singleton_if_δ_eq_ε one_arg_imp_Hd
          tm.collapse(2)[OF t_app] wary_arg_t,
        simp)

    have arg u >t arg t
      by (metis extf_singleton[THEN iffD1] append_Nil args.simps args_Nil_iff_is_Hd
        comp_hd_def gt_hd_def gt_irrefl gt_same_u_t(3,4) gt_unary_t_s(2,3) head_fun
        length_0_conv nargs_u one_arg_imp_Hd t_app tm.collapse(2) u_gt_t wary_u)
    hence arg u >t s
      using ih[OF _ wary_arg_u wary_arg_t wary_s] gt_unary_t_s(5)
      by (metis add_mset_lt_left_lt add_mset_lt_lt_lt args_Nil_iff_is_Hd list.size(3) nargs_u
        size_arg_lt t_app zero_neq_one)
    hence arg_u_ge_s: arg u ≥t s
      by sat
    show ?thesis
      by (rule gt_unary[OF wt_u_s hd_u_ncomp_s nargs_u _ arg_u_ge_s])
        (simp add: gt_same_u_t(3) gt_unary_t_s(4))
  qed (simp add: wt_u_t)
next
  case gt_diff_t_s: gt_diff
```

```
  show ?thesis
    using u_gt_t
  proof cases
    case gt_unary_u_t: gt_unary
    have is_App u
      by (metis args_Nil_iff_is_Hd gt_unary_u_t(3) length_greater_0_conv less_numeral_extra(1))
    hence arg u >_t s
      using ih[of arg u t s] gt_unary_u_t(5) t_gt_s size_arg_lt wary_arg_u wary_s wary_t
      by force
    hence arg_u_ge_s: arg u ≥_t s
      by sat

    {
      assume head u = head s
      hence False
        using gt_diff_t_s(3) gt_unary_u_t(2) unfolding comp_hd_def by force
    }
    moreover
    {
      assume head s >_hd head u
      hence False
        using gt_hd_def gt_hd_irrefl gt_sym_antisym gt_unary_u_t(4) unary_wt_sym_0_gt by blast
    }
    moreover
    {
      assume head u >_hd head s
      hence ?thesis
        by (rule gt_diff[OF vars_u_s wt_u_s])
    }
    moreover
    {
      assume ¬ head u ≤≥_hd head s
      hence ?thesis
        by (rule gt_unary[OF wt_u_s _ gt_unary_u_t(3,4) arg_u_ge_s])
    }
    ultimately show ?thesis
      unfolding comp_hd_def by sat
  next
    case gt_diff_u_t: gt_diff
    have head u >_hd head s
      using gt_diff_u_t(3) gt_diff_t_s(3) gt_hd_trans by blast
    thus ?thesis
      by (rule gt_diff[OF vars_u_s wt_u_s])
  next
    case gt_same_u_t: gt_same
    have head u >_hd head s
      using gt_diff_t_s(3) gt_same_u_t(3) by simp
    thus ?thesis
      by (rule gt_diff[OF vars_u_s wt_u_s])
  qed (simp add: wt_u_t)
next
  case gt_same_t_s: gt_same
  show ?thesis
    using u_gt_t
  proof cases
    case gt_unary_u_t: gt_unary
    have is_App u
      by (metis args_Nil_iff_is_Hd gt_unary_u_t(3) length_greater_0_conv less_numeral_extra(1))
    hence arg u >_t s
      using ih[of arg u t s] gt_unary_u_t(5) t_gt_s size_arg_lt wary_arg_u wary_s wary_t
      by force
    hence arg_u_ge_s: arg u ≥_t s
      by sat
```

have ¬ *head u* $\leq\geq_{hd}$ *head s*
  using *gt_same_t_s(3) gt_unary_u_t(2)* **by** *simp*
thus *?thesis*
  by (*rule gt_unary[OF wt_u_s _ gt_unary_u_t(3,4) arg_u_ge_s]*)
next
case *gt_diff_u_t*: *gt_diff*
have *head u* $>_{hd}$ *head s*
  using *gt_diff_u_t(3) gt_same_t_s(3)* **by** *simp*
thus *?thesis*
  by (*rule gt_diff[OF vars_u_s wt_u_s]*)
next
case *gt_same_u_t*: *gt_same*
have *hd_u_s*: *head u = head s*
  by (*simp only*: *gt_same_t_s(3) gt_same_u_t(3)*)

let *?S = set (args u) ∪ set (args t) ∪ set (args s)*

have *gt_trans_args*: ∀ *ua* ∈ *?S*. ∀ *ta* ∈ *?S*. ∀ *sa* ∈ *?S*. *ua* $>_t$ *ta* ⟶ *ta* $>_t$ *sa* ⟶ *ua* $>_t$ *sa*
proof *clarify*
  fix *sa ta ua*
  assume
    ua_in: *ua* ∈ *?S* **and** *ta_in*: *ta* ∈ *?S* **and** *sa_in*: *sa* ∈ *?S* **and**
    ua_gt_ta: *ua* $>_t$ *ta* **and** *ta_gt_sa*: *ta* $>_t$ *sa*
  have *wary_sa*: *wary sa* **and** *wary_ta*: *wary ta* **and** *wary_ua*: *wary ua*
    using *wary_args ua_in ta_in sa_in wary_u wary_t wary_s* **by** *blast+*
  show *ua* $>_t$ *sa*
    by (*auto intro!*: *ih[OF Max_lt_imp_lt_mset wary_ua wary_ta wary_sa ua_gt_ta ta_gt_sa]*)
      (*meson ua_in ta_in sa_in Un_iff max.strict_coboundedI1 max.strict_coboundedI2*
        size_in_args)+
qed
have ∀ *f* ∈ *ground_heads* (*head u*). *extf f* ($>_t$) (*args u*) (*args s*)
  by (*clarify*, *rule extf_trans_from_irrefl[of ?S _ args t*, *OF _ _ _ _ _ _ gt_trans_args]*)
    (*auto simp*: *gt_same_u_t(3,4) gt_same_t_s(4) wary_args wary_u wary_t wary_s gt_irrefl*)
thus *?thesis*
  by (*rule gt_same[OF vars_u_s wt_u_s hd_u_s]*)
qed (*simp add*: *wt_u_t*)
qed (*simp add*: *wt_t_s*)
}
thus *u* $>_t$ *s*
  using *vars_u_s wt_t_ge_s wt_u_ge_t* **by** (*force intro*: *gt_wt*)
qed

lemma *gt_antisym*: *wary s* ⟹ *wary t* ⟹ *t* $>_t$ *s* ⟹ ¬ *s* $>_t$ *t*
  using *gt_irrefl gt_trans* **by** *blast*

4.6 Subterm Property

lemma *gt_sub_fun*: *App s t* $>_t$ *s*
proof (*cases wt (App s t) > wt s*)
  case *True*
  thus *?thesis*
    using *gt_wt* **by** *simp*
next
  case *False*
  hence *wt_st*: *wt (App s t) = wt s*
    by (*meson order.antisym not_le_imp_less wt_App_ge_fun*)
  hence *δ_eq_ε*: δ = ε
    by (*metis add_diff_cancel_left′ diff_diff_cancel wt_δ_imp_δ_eq_ε wt_ge_δ wt.simps(2)*)

  have *vars_st*: *vars_mset (App s t)* ⊇# *vars_mset s*
    by *auto*
  have *hd_st*: *head (App s t) = head s*
    by *auto*

```
  have extf: ∀f ∈ ground_heads (head (App s t)). extf f (>t) (args (App s t)) (args s)
    by (simp add: δ_eq_ε extf_snoc_if_δ_eq_ε)
  show ?thesis
    by (rule gt_same[OF vars_st wt_st hd_st extf])
qed

theorem gt_proper_sub: wary t ⟹ proper_sub s t ⟹ t >t s
  by (induct t) (auto intro: gt_sub_fun gt_sub_arg gt_trans sub.intros wary_sub)
```

4.7 Compatibility with Functions

```
theorem gt_compat_fun:
  assumes
    wary_t: wary t and
    t'_gt_t: t' >t t
  shows App s t' >t App s t
proof –
  have vars_st': vars_mset (App s t') ⊇# vars_mset (App s t)
    by (simp add: t'_gt_t gt_imp_vars_mset)

  show ?thesis
  proof (cases wt t' > wt t)
    case True
    hence wt_st': wt (App s t') > wt (App s t)
      by (simp only: wt.simps)
    show ?thesis
      by (rule gt_wt[OF vars_st' wt_st'])
  next
    case False
    hence wt t' = wt t
      using t'_gt_t gt_imp_wt_ge order.not_eq_order_implies_strict by fastforce
    hence wt_st': wt (App s t') = wt (App s t)
      by (simp only: wt.simps)

    have head_st': head (App s t') = head (App s t)
      by simp

    have extf: ⋀f. extf f (>t) (args s @ [t']) (args s @ [t])
      using t'_gt_t by (metis extf_compat_list gt_irrefl[OF wary_t])

    show ?thesis
      by (rule gt_same[OF vars_st' wt_st' head_st']) (simp add: extf)
  qed
qed
```

4.8 Compatibility with Arguments

```
theorem gt_compat_arg:
  assumes wary_s't: wary (App s' t) and s'_gt_s: s' >t s
  shows App s' t >t App s t
proof –
  have vars_s't: vars_mset (App s' t) ⊇# vars_mset (App s t)
    by (simp add: s'_gt_s gt_imp_vars_mset)
  show ?thesis
    using s'_gt_s
  proof cases
    case gt_wt_s'_s: gt_wt
    have wt (App s' t) > wt (App s t)
      by (simp add: wt_ge_δ) (metis add_diff_assoc add_less_cancel_right gt_wt_s'_s(2) wt_ge_δ)
    thus ?thesis
      by (rule gt_wt[OF vars_s't])
  next
    case gt_unary_s'_s: gt_unary
    have False
```

```
      by (metis ground_heads_arity gt_unary_s'_s(3) gt_unary_s'_s(4) leD one_enat_def wary_AppE
        wary_s't)
    thus ?thesis
      by sat
  next
    case _: gt_diff
    thus ?thesis
      by (simp add: gt_diff)
  next
    case gt_same_s'_s: gt_same
    have wt_s't: wt (App s' t) = wt (App s t)
      by (simp add: gt_same_s'_s(2))
    have hd_s't: head (App s' t) = head (App s t)
      by (simp add: gt_same_s'_s(3))
    have ∀f ∈ ground_heads (head (App s' t)). extf f (>_t) (args (App s' t)) (args (App s t))
      using gt_same_s'_s(4) by (auto intro: extf_compat_append_right)
    thus ?thesis
      by (rule gt_same[OF vars_s't wt_s't hd_s't])
  qed
qed
```

4.9 Stability under Substitution

```
definition extra_wt :: ('v ⇒ ('s, 'v) tm) ⇒ ('s, 'v) tm ⇒ nat where
  extra_wt ϱ s = (∑ x ∈# vars_mset s. wt (ϱ x) − wt (Hd (Var x)))

lemma
  extra_wt_Var[simp]: extra_wt ϱ (Hd (Var x)) = wt (ϱ x) − wt (Hd (Var x)) and
  extra_wt_Sym[simp]: extra_wt ϱ (Hd (Sym f)) = 0 and
  extra_wt_App[simp]: extra_wt ϱ (App s t) = extra_wt ϱ s + extra_wt ϱ t
  unfolding extra_wt_def by simp+

lemma extra_wt_subseteq:
  assumes vars_s: vars_mset t ⊇# vars_mset s
  shows extra_wt ϱ t ≥ extra_wt ϱ s
proof (unfold extra_wt_def)
  let ?diff = λv. wt (ϱ v) − wt (Hd (Var v))

  have vars_mset s + (vars_mset t − vars_mset s) = vars_mset t
    using vars_s by (meson subset_mset.add_diff_inverse)
  hence {#?diff v. v ∈# vars_mset t#} =
    {#?diff v. v ∈# vars_mset s#} + {#?diff v. v ∈# vars_mset t − vars_mset s#}
    by (metis image_mset_union)
  thus (∑ v ∈# vars_mset t. ?diff v) ≥ (∑ v ∈# vars_mset s. ?diff v)
    by simp
qed

lemma wt_subst:
  assumes wary_ϱ: wary_subst ϱ and wary_s: wary s
  shows wt (subst ϱ s) = wt s + extra_wt ϱ s
  using wary_s
proof (induct s rule: tm.induct)
  case ζ: (Hd ζ)
  show ?case
  proof (cases ζ)
    case x: (Var x)

    let ?ξ = head (ϱ x)

    obtain g where
      g_in_grs_ξ: g ∈ ground_heads ?ξ and
      wt_ξ: wt (Hd ?ξ) = wt_sym g + the_enat (δ ∗ arity_sym g)
      using exists_wt_sym by blast
    have g ∈ ground_heads ζ
```

```
      using x g_in_grs_ξ wary_ϱ wary_subst_def by auto
    hence wt_ϱx_ge: wt (ϱ x) ≥ wt (Hd ζ)
      by (metis (full_types) dual_order.trans wt_le_wt_sym wt_ξ wt_hd_le)
    thus ?thesis
      using x by (simp add: extra_wt_def)
  qed auto
next
  case (App s t)
  note ih_s = this(1) and ih_t = this(2) and wary_st = this(3)
  have wary s
    using wary_st by (meson wary_AppE)
  hence ⋀n. extra_wt ϱ s + (wt s − δ + n) = wt (subst ϱ s) − δ + n
    using ih_s by (metis (full_types) add_diff_assoc2 ab_semigroup_add_class.add_ac(1)
      add.left_commute wt_ge_δ)
  hence extra_wt ϱ s + (wt s + wt t − δ + extra_wt ϱ t) = wt (subst ϱ s) + wt (subst ϱ t) − δ
    using ih_t wary_st
    by (metis (no_types) add_diff_assoc2 ab_semigroup_add_class.add_ac(1) wary_AppE wt_ge_δ)
  thus ?case
    by (simp add: wt_ge_δ)
qed

theorem gt_subst:
  assumes wary_ϱ: wary_subst ϱ
  shows wary t ⟹ wary s ⟹ t >_t s ⟹ subst ϱ t >_t subst ϱ s
proof (simp only: atomize_imp,
    rule measure_induct_rule[of λ(t, s). {#size t, size s#}
        λ(t, s). wary t ⟶ wary s ⟶ t >_t s ⟶ subst ϱ t >_t subst ϱ s (t, s),
      simplified prod.case],
    simp only: split_paired_all prod.case atomize_imp[symmetric])
  fix t s
  assume
    ih: ⋀ta sa. {#size ta, size sa#} < {#size t, size s#} ⟹ wary ta ⟹ wary sa ⟹ ta >_t sa ⟹
      subst ϱ ta >_t subst ϱ sa and
    wary_t: wary t and wary_s: wary s and t_gt_s: t >_t s

  show subst ϱ t >_t subst ϱ s
  proof (cases wt (subst ϱ t) = wt (subst ϱ s))
    case wt_ϱt_ne_ϱs: False

    have vars_s: vars_mset t ⊇# vars_mset s
      by (simp add: t_gt_s gt_imp_vars_mset)
    hence vars_ϱs: vars_mset (subst ϱ t) ⊇# vars_mset (subst ϱ s)
      by (rule vars_mset_subst_subseteq)

    have wt_t_ge_s: wt t ≥ wt s
      by (simp add: gt_imp_wt_ge t_gt_s)

    have wt (subst ϱ t) > wt (subst ϱ s)
      using wt_ϱt_ne_ϱs unfolding wt_subst[OF wary_ϱ wary_s] wt_subst[OF wary_ϱ wary_t]
      by (metis add_le_cancel_left add_less_le_mono extra_wt_subseteq
        order.not_eq_order_implies_strict vars_s wt_t_ge_s)
    thus ?thesis
      by (rule gt_wt[OF vars_ϱs])
  next
    case wt_ϱt_eq_ϱs: True
    show ?thesis
      using t_gt_s
    proof cases
      case gt_wt
      hence False
        using wt_ϱt_eq_ϱs wary_s wary_t
        by (metis add_diff_cancel_right' diff_le_mono2 extra_wt_subseteq wt_subst leD wary_ϱ)
      thus ?thesis
```

```
    by sat
next
  case gt_unary

  have wary_ϱt: wary (subst ϱ t)
    by (simp add: wary_subst_wary wary_t wary_ϱ)

  show ?thesis
  proof (cases t)
    case Hd
    hence False
      using gt_unary(3) by simp
    thus ?thesis
      by sat
  next
    case t: (App t1 t2)
    hence t2: t2 = arg t
      by simp
    hence wary_t2: wary t2
      using wary_t by blast

    show ?thesis
    proof (cases t2 = s)
      case True
      moreover have subst ϱ t >_t subst ϱ t2
        using gt_sub_arg wary_ϱt unfolding t by simp
      ultimately show ?thesis
        by simp
    next
      case t2_ne_s: False
      hence t2_gt_s: t2 >_t s
        using gt_unary(5) t2 by blast

      have subst ϱ t2 >_t subst ϱ s
        by (rule ih[OF _ wary_t2 wary_s t2_gt_s]) (simp add: t)
      thus ?thesis
        by (metis gt_sub_arg gt_trans subst.simps(2) t wary_ϱ wary_ϱt wary_s wary_subst_wary
          wary_t2)
    qed
  qed
next
  case _: gt_diff
  note vars_s = this(1) and hd_t_gt_hd_s = this(3)
  have vars_ϱs: vars_mset (subst ϱ t) ⊇# vars_mset (subst ϱ s)
    by (rule vars_mset_subst_subseteq[OF vars_s])

  have head (subst ϱ t) >_hd head (subst ϱ s)
    by (meson hd_t_gt_hd_s wary_subst_ground_heads gt_hd_def rev_subsetD wary_ϱ)
  thus ?thesis
    by (rule gt_diff[OF vars_ϱs wt_ϱt_eq_ϱs])
next
  case _: gt_same
  note vars_s = this(1) and hd_s_eq_hd_t = this(3) and extf = this(4)

  have vars_ϱs: vars_mset (subst ϱ t) ⊇# vars_mset (subst ϱ s)
    by (rule vars_mset_subst_subseteq[OF vars_s])
  have hd_ϱt: head (subst ϱ t) = head (subst ϱ s)
    by (simp add: hd_s_eq_hd_t)

  {
    fix f
    assume f_in_grs: f ∈ ground_heads (head (subst ϱ t))
```

```
      let ?S = set (args t) ∪ set (args s)

      have extf_args_s_t: extf f (>t) (args t) (args s)
        using extf_f_in_grs wary_subst_ground_heads wary_ϱ by blast
      have extf f (>t) (map (subst ϱ) (args t)) (map (subst ϱ) (args s))
      proof (rule extf_map[of ?S, OF _ _ _ _ _ _ extf_args_s_t])
        show ∀x ∈ ?S. ¬ subst ϱ x >t subst ϱ x
          using gt_irrefl wary_t wary_s wary_args wary_ϱ wary_subst_wary by fastforce
      next
        show ∀z ∈ ?S. ∀y ∈ ?S. ∀x ∈ ?S. subst ϱ z >t subst ϱ y ⟶ subst ϱ y >t subst ϱ x ⟶
          subst ϱ z >t subst ϱ x
          using gt_trans wary_t wary_s wary_args wary_ϱ wary_subst_wary by (metis Un_iff)
      next
        have sz_a: ∀ta ∈ ?S. ∀sa ∈ ?S. {#size ta, size sa#} < {#size t, size s#}
          by (fastforce intro: Max_lt_imp_lt_mset dest: size_in_args)
        show ∀y ∈ ?S. ∀x ∈ ?S. y >t x ⟶ subst ϱ y >t subst ϱ x
          using ih sz_a size_in_args wary_t wary_s wary_args wary_ϱ wary_subst_wary by fastforce
      qed auto
      hence extf f (>t) (args (subst ϱ t)) (args (subst ϱ s))
        by (auto simp: hd_s_eq_hd_t intro: extf_compat_append_left)
    }
    hence ∀f ∈ ground_heads (head (subst ϱ t)).
      extf f (>t) (args (subst ϱ t)) (args (subst ϱ s))
      by blast
    thus ?thesis
      by (rule gt_same[OF vars_ϱs wt_ϱt_eq_ϱs hd_ϱt])
  qed
 qed
qed
```

4.10 Totality on Ground Terms

```
theorem gt_total_ground:
  assumes extf_total: ⋀f. ext_total (extf f)
  shows ground t ⟹ ground s ⟹ t >t s ∨ s >t t ∨ t = s
proof (simp only: atomize_imp,
    rule measure_induct_rule[of λ(t, s). {# size t, size s #}
      λ(t, s). ground t ⟶ ground s ⟶ t >t s ∨ s >t t ∨ t = s (t, s), simplified prod.case],
    simp only: split_paired_all prod.case atomize_imp[symmetric])
  fix t s :: ('s, 'v) tm
  assume
    ih: ⋀ta sa. {# size ta, size sa #} < {# size t, size s #} ⟹ ground ta ⟹ ground sa ⟹
      ta >t sa ∨ sa >t ta ∨ ta = sa and
    gr_t: ground t and gr_s: ground s

  let ?case = t >t s ∨ s >t t ∨ t = s

  have
    vars_t: vars_mset t ⊇# vars_mset s and
    vars_s: vars_mset s ⊇# vars_mset t
    by (simp only: vars_mset_empty_iff[THEN iffD2, OF gr_s]
      vars_mset_empty_iff[THEN iffD2, OF gr_t])+

  show ?case
  proof (cases wt t = wt s)
    case False
    moreover
    {
      assume wt t > wt s
      hence t >t s
        by (rule gt_wt[OF vars_t])
    }
    moreover
    {
```

```
      assume wt s > wt t
      hence s >_t t
        by (rule gt_wt[OF vars_s])
    }
    ultimately show ?thesis
      by linarith
  next
    case wt_t: True
    note wt_s = wt_t[symmetric]

    obtain g where ξ: head t = Sym g
      by (metis ground_head[OF gr_t] hd.collapse(2))
    obtain f where ζ: head s = Sym f
      by (metis ground_head[OF gr_s] hd.collapse(2))

    {
      assume g_gt_f: g >_s f
      have t >_t s
        by (rule gt_diff[OF vars_t wt_t]) (simp add: ξ ζ g_gt_f gt_hd_def)
    }
    moreover
    {
      assume f_gt_g: f >_s g
      have s >_t t
        by (rule gt_diff[OF vars_s wt_s]) (simp add: ξ ζ f_gt_g gt_hd_def)
    }
    moreover
    {
      assume g_eq_f: g = f
      hence hd_t: head t = head s
        using ξ ζ by force
      note hd_s = hd_t[symmetric]

      have gr_ts: ∀ t ∈ set (args t). ground t
        using gr_t ground_args by auto
      have gr_ss: ∀ s ∈ set (args s). ground s
        using gr_s ground_args by auto

      let ?ts = args t
      let ?ss = args s

      have ?thesis
      proof (cases ?ts = ?ss)
        case ts_eq_ss: True
        show ?thesis
          using ξ ζ g_eq_f ts_eq_ss by (simp add: tm_expand_apps)
      next
        case False
        hence extf g (>_t) (args t) ?ss ∨ extf g (>_t) ?ss ?ts
          using ih gr_ss gr_ts
            ext_total.total[OF extf_total, rule_format, of set ?ts ∪ set ?ss (>_t) ?ts ?ss g]
          by (metis Un_commute Un_iff in_lists_iff_set less_multiset_doubletons size_in_args sup_ge2)
        moreover
        {
          assume extf: extf g (>_t) ?ts ?ss
          have t >_t s
            by (rule gt_same[OF vars_t wt_t hd_t]) (simp add: extf ξ)
        }
        moreover
        {
          assume extf: extf g (>_t) ?ss ?ts
          have s >_t t
            by (rule gt_same[OF vars_s wt_s hd_s]) (simp add: extf[unfolded g_eq_f] ζ)
```

```
      }
      ultimately show ?thesis
        by sat
    qed
  }
  ultimately show ?thesis
    using gt_sym_total by blast
 qed
qed
```

4.11 Well-foundedness

```
abbreviation gtw :: ('s, 'v) tm ⇒ ('s, 'v) tm ⇒ bool (infix ‹>_tw› 50) where
 (>_tw) ≡ λt s. wary t ∧ wary s ∧ t >_t s

abbreviation gtwg :: ('s, 'v) tm ⇒ ('s, 'v) tm ⇒ bool (infix ‹>_twg› 50) where
 (>_twg) ≡ λt s. ground t ∧ t >_tw s

lemma ground_gt_unary:
  assumes gr_t: ground t
  shows ¬ gt_unary t s
proof
  assume gt_unary_t_s: gt_unary t s
  hence t >_t s
    using gt_iff_wt_unary_diff_same by blast
  hence gr_s: ground s
    using gr_t gt_imp_vars by blast

  have ngr_t_or_s: ¬ ground t ∨ ¬ ground s
    using gt_unary_t_s by cases (blast dest: ground_head not_comp_hd_imp_Var)

  show False
    using gr_t gr_s ngr_t_or_s by sat
qed

theorem gt_wf: wfP (λs t. t >_tw s)
proof –
  have ground_wfP: wfP (λs t. t >_twg s)
    unfolding wfP_iff_no_inf_chain
  proof
    assume ∃f. inf_chain (>_twg) f
    then obtain t where t_bad: bad (>_twg) t
      unfolding inf_chain_def bad_def by blast

    let ?ff = worst_chain (>_twg) (λt s. size t > size s)

    note wf_sz = wf_app[OF wellorder_class.wf, of size, simplified]

    have ffi_ground: ⋀i. ground (?ff i) and ffi_wary: ⋀i. wary (?ff i)
      using worst_chain_bad[OF wf_sz t_bad, unfolded inf_chain_def] by fast+

    have inf_chain (>_twg) ?ff
      by (rule worst_chain_bad[OF wf_sz t_bad])
    hence bad_wt_diff_same:
      inf_chain (λt s. ground t ∧ (gt_wt t s ∨ gt_diff t s ∨ gt_same t s)) ?ff
      unfolding inf_chain_def using gt_iff_wt_unary_diff_same ground_gt_unary by blast

    have wf_wt: wf {(s, t). ground t ∧ gt_wt t s}
      by (rule wf_subset[OF wf_app[of _ wt, OF wf_less]]) (auto simp: gt_wt.simps)

    have wt_O_diff_same: {(s, t). ground t ∧ gt_wt t s}
      O {(s, t). ground t ∧ (gt_diff t s ∨ gt_same t s)} ⊆ {(s, t). ground t ∧ gt_wt t s}
      unfolding gt_wt.simps gt_diff.simps gt_same.simps by auto
```

```
have wt_diff_same_as_union: {(s, t). ground t ∧ (gt_wt t s ∨ gt_diff t s ∨ gt_same t s)} =
  {(s, t). ground t ∧ gt_wt t s} ∪ {(s, t). ground t ∧ (gt_diff t s ∨ gt_same t s)}
  by auto

obtain k1 where bad_diff_same:
  inf_chain (λt s. ground t ∧ (gt_diff t s ∨ gt_same t s)) (λi. ?ff (i + k1))
  using wf_infinite_down_chain_compatible[OF wf_wt _ wt_O_diff_same, of ?ff] bad_wt_diff_same
  unfolding inf_chain_def wt_diff_same_as_union[symmetric] by auto

have wf {(s, t). ground s ∧ ground t ∧ sym (head t) >s sym (head s)}
  using gt_sym_wf unfolding wfp_def wf_iff_no_infinite_down_chain by fast
moreover have {(s, t). ground t ∧ gt_diff t s}
  ⊆ {(s, t). ground s ∧ ground t ∧ sym (head t) >s sym (head s)}
proof (clarsimp, intro conjI)
  fix s t
  assume gr_t: ground t and gt_diff_t_s: gt_diff t s
  thus gr_s: ground s
    using gt_iff_wt_unary_diff_same gt_imp_vars by fastforce
  show sym (head t) >s sym (head s)
    using gt_diff_t_s by cases (simp add: gt_hd_def gr_s gr_t ground_hd_in_ground_heads)
qed
ultimately have wf_diff: wf {(s, t). ground t ∧ gt_diff t s}
  by (rule wf_subset)

have diff_O_same: {(s, t). ground t ∧ gt_diff t s} O {(s, t). ground t ∧ gt_same t s}
  ⊆ {(s, t). ground t ∧ gt_diff t s}
  unfolding gt_diff.simps gt_same.simps by auto

have diff_same_as_union: {(s, t). ground t ∧ (gt_diff t s ∨ gt_same t s)} =
  {(s, t). ground t ∧ gt_diff t s} ∪ {(s, t). ground t ∧ gt_same t s}
  by auto

obtain k2 where bad_same: inf_chain (λt s. ground t ∧ gt_same t s) (λi. ?ff (i + k2))
  using wf_infinite_down_chain_compatible[OF wf_diff _ diff_O_same, of λi. ?ff (i + k1)]
    bad_diff_same
  unfolding inf_chain_def diff_same_as_union[symmetric] by (auto simp: add.assoc)
hence hd_sym: ⋀i. is_Sym (head (?ff (i + k2)))
  unfolding inf_chain_def by (simp add: ground_head)

define f where f = sym (head (?ff k2))

have hd_eq_f: head (?ff (i + k2)) = Sym f for i
  unfolding f_def
proof (induct i)
  case 0
  thus ?case
    by (auto simp: hd.collapse(2)[OF hd_sym, of 0, simplified])
next
  case (Suc ia)
  thus ?case
    using bad_same unfolding inf_chain_def gt_same.simps by simp
qed

define max_args where max_args = wt (?ff k2)

have wt_eq_max_args: wt (?ff (i + k2)) = max_args for i
  unfolding max_args_def
proof (induct i)
  case (Suc ia)
  thus ?case
    using bad_same unfolding inf_chain_def gt_same.simps by simp
qed auto
```

```
have nargs_le_max_args: num_args (?ff (i + k2)) ≤ max_args for i
  unfolding wt_eq_max_args[of i, symmetric] by (rule wt_ge_num_args[OF ffi_wary])

let ?U_of = λi. set (args (?ff (i + k2)))

define U where U = (⋃ i. ?U_of i)

have gr_u: ⋀u. u ∈ U ⟹ ground u
  unfolding U_def by (blast dest: ground_args[OF _ ffi_ground])
have wary_u: ⋀u. u ∈ U ⟹ wary u
  unfolding U_def by (blast dest: wary_args[OF _ ffi_wary])

have ¬ bad (>twg) u if u_in: u ∈ ?U_of i for u i
proof
  assume u_bad: bad (>twg) u
  have sz_u: size u < size (?ff (i + k2))
    by (rule size_in_args[OF u_in])

  show False
  proof (cases i + k2)
    case 0
    thus False
      using sz_u min_worst_chain_0[OF wf_sz u_bad] by simp
  next
    case Suc
    hence gt: ?ff (i + k2 − 1) >tw ?ff (i + k2)
      using worst_chain_pred[OF wf_sz t_bad] by auto
    moreover have ?ff (i + k2) >tw u
      using gt gt_proper_sub sub_args sz_u u_in wary_args by auto
    ultimately have ?ff (i + k2 − 1) >tw u
      using gt_trans by blast
    thus False
      using Suc sz_u min_worst_chain_Suc[OF wf_sz u_bad] ffi_ground by fastforce
  qed
qed
hence u_good: ⋀u. u ∈ U ⟹ ¬ bad (>twg) u
  unfolding U_def by blast

let ?gtwu = λt s. t ∈ U ∧ t >tw s

have gtwu_irrefl: ⋀x. ¬ ?gtwu x x
  using gt_irrefl by auto

have ⋀i j. ∀ t ∈ set (args (?ff (i + k2))). ∀ s ∈ set (args (?ff (j + k2))). t >t s ⟶
  t ∈ U ∧ t >tw s
  using wary_u unfolding U_def by blast
moreover have ⋀i. extf f (>t) (args (?ff (i + k2))) (args (?ff (Suc i + k2)))
  using bad_same hd_eq_f unfolding inf_chain_def gt_same.simps by auto
ultimately have ⋀i. extf f ?gtwu (args (?ff (i + k2))) (args (?ff (Suc i + k2)))
  by (rule extf_mono_strong)
hence inf_chain (extf f ?gtwu) (λi. args (?ff (i + k2)))
  unfolding inf_chain_def by blast
hence nwf_ext:
  ¬ wfP (λxs ys. length ys ≤ max_args ∧ length xs ≤ max_args ∧ extf f ?gtwu ys xs)
  unfolding inf_chain_def wfp_def wf_iff_no_infinite_down_chain using nargs_le_max_args by fast

have gtwu_le_gtwg: ?gtwu ≤ (>twg)
  by (auto intro!: gr_u)

have wfP (λs t. ?gtwu t s)
  unfolding wfP_iff_no_inf_chain
proof (intro notI, elim exE)
  fix f
```

```
      assume bad_f: inf_chain ?gtwu f
      hence bad_f0: bad ?gtwu (f 0)
        by (rule inf_chain_bad)
      hence f 0 ∈ U
        using bad_f unfolding inf_chain_def by blast
      hence ¬ bad (>_twg) (f 0)
        using u_good by blast
      hence ¬ bad ?gtwu (f 0)
        using bad_f inf_chain_bad inf_chain_subset[OF _ gtwu_le_gtwg] by blast
      thus False
        using bad_f0 by sat
    qed
    hence wf_ext: wfP (λxs ys. length ys ≤ max_args ∧ length xs ≤ max_args ∧ extf f ?gtwu ys xs)
      using extf_wf_bounded[of ?gtwu] gtwu_irrefl by blast

    show False
      using nwf_ext wf_ext by blast
  qed

  let ?subst = subst grounding_ρ

  have wfP (λs t. ?subst t >_twg ?subst s)
    by (rule wfP_app[OF ground_wfP])
  hence wfP (λs t. ?subst t >_tw ?subst s)
    by (simp add: ground_grounding_ρ)
  thus ?thesis
    by (auto intro: wfp_subset wary_subst_wary[OF wary_grounding_ρ] gt_subst[OF wary_grounding_ρ])
qed

end

end
```

5 The Graceful Basic Knuth–Bendix Order for Lambda-Free Higher-Order Terms

```
theory Lambda_Free_KBO_Basic
imports Lambda_Free_KBO_Std
begin
```

This theory defines the basic version of the graceful Knuth–Bendix order (KBO) for λ-free higher-order terms. Basic means that all symbols must have a positive weight. The results are lifted from the standard KBO.

```
locale kbo_basic = kbo_basic_basis _ _ _ _ ground_heads_var
  for ground_heads_var :: 'v ⇒ 's set
begin

sublocale kbo_std: kbo_std _ _ _ _ 0 _ λ_. ∞ λ_. ∞
  by (simp add: ε_gt_0 kbo_std_def kbo_std_basis_axioms)

fun wt :: ('s, 'v) tm ⇒ nat where
  wt (Hd ζ) = (LEAST w. ∃f ∈ ground_heads ζ. w = wt_sym f)
| wt (App s t) = wt s + wt t

inductive gt :: ('s, 'v) tm ⇒ ('s, 'v) tm ⇒ bool (infix ‹>_t› 50) where
  gt_wt: vars_mset t ⊇# vars_mset s ⟹ wt t > wt s ⟹ t >_t s
| gt_diff: vars_mset t ⊇# vars_mset s ⟹ wt t = wt s ⟹ head t >_hd head s ⟹ t >_t s
| gt_same: vars_mset t ⊇# vars_mset s ⟹ wt t = wt s ⟹ head t = head s ⟹
    (∀f ∈ ground_heads (head s). extf f (>_t) (args t) (args s)) ⟹ t >_t s

lemma arity_hd_eq_inf[simp]: arity_hd ζ = ∞
  by (cases ζ) auto
```

```
lemma waryI[intro, simp]: wary s
  by (simp add: wary_inf_ary)

lemma basic_wt_eq_wt: wt s = kbo_std.wt s
  by (induct s) auto

lemma
  basic_gt_and_gt_le_gt: (λt s. t >_t s ∧ local.kbo_std.gt t s) ≤ kbo_std.gt and
  gt_and_basic_gt_le_basic_gt: (λt s. local.kbo_std.gt t s ∧ t >_t s) ≤ (>_t)
  by auto

lemma basic_gt_iff_gt: t >_t s ⟷ kbo_std.gt t s
proof
  assume t >_t s
  thus kbo_std.gt t s
  proof induct
    case gt_wt
    thus ?case
      by (auto intro: kbo_std.gt_wt simp: basic_wt_eq_wt[symmetric])
  next
    case gt_diff
    thus ?case
      by (auto intro: kbo_std.gt_diff simp: basic_wt_eq_wt[symmetric])
  next
    case gt_same
    thus ?case
      using extf_mono[OF basic_gt_and_gt_le_gt]
      by (force simp: basic_wt_eq_wt[symmetric] intro!: kbo_std.gt_same)
  qed
next
  assume kbo_std.gt t s
  thus t >_t s
  proof induct
    case gt_wt_t_s: gt_wt
    thus ?case
      by (auto intro: gt_wt simp: basic_wt_eq_wt[symmetric])
  next
    case gt_unary_t_s: (gt_unary t s)
    have False
      using gt_unary_t_s(4) by (metis less_nat_zero_code wt_sym_gt_0)
    thus ?case
      by satx
  next
    case gt_diff_t_s: gt_diff
    thus ?case
      by (auto intro: gt_diff simp: basic_wt_eq_wt[symmetric])
  next
    case gt_same_t_s: gt_same
    thus ?case
      using extf_mono[OF gt_and_basic_gt_le_basic_gt]
      by (force intro!: gt_same simp: basic_wt_eq_wt[symmetric])
  qed
qed

theorem gt_irrefl: ¬ s >_t s
  unfolding basic_gt_iff_gt by (rule kbo_std.gt_irrefl[simplified])

theorem gt_trans: u >_t t ⟹ t >_t s ⟹ u >_t s
  unfolding basic_gt_iff_gt by (rule kbo_std.gt_trans[simplified])

theorem gt_proper_sub: proper_sub s t ⟹ t >_t s
  unfolding basic_gt_iff_gt by (rule kbo_std.gt_proper_sub[simplified])
```

theorem *gt_compat_fun*: $t' >_t t \Longrightarrow App\ s\ t' >_t App\ s\ t$
  unfolding *basic_gt_iff_gt* **by** (*rule kbo_std.gt_compat_fun*[*simplified*])

theorem *gt_compat_arg*: $s' >_t s \Longrightarrow App\ s'\ t >_t App\ s\ t$
  unfolding *basic_gt_iff_gt* **by** (*rule kbo_std.gt_compat_arg*[*simplified*])

lemma *wt_subst*: *wary_subst* $\varrho \Longrightarrow$ *wt* (*subst* ϱ *s*) = *wt s* + *kbo_std.extra_wt* ϱ *s*
  unfolding *basic_gt_iff_gt basic_wt_eq_wt* **by** (*rule kbo_std.wt_subst*[*simplified*])

theorem *gt_subst*: *wary_subst* $\varrho \Longrightarrow t >_t s \Longrightarrow$ *subst* ϱ $t >_t$ *subst* ϱ *s*
  unfolding *basic_gt_iff_gt* **by** (*rule kbo_std.gt_subst*[*simplified*])

theorem *gt_wf*: *wfP* ($\lambda s\ t.\ t >_t s$)
  unfolding *basic_gt_iff_gt*[*abs_def*] **by** (*rule kbo_std.gt_wf*[*simplified*])

end

end

6 The Graceful Transfinite Knuth–Bendix Order with Subterm Coefficients for Lambda-Free Higher-Order Terms

theory *Lambda_Free_TKBO_Coefs*
imports *Lambda_Free_KBO_Util Nested_Multisets_Ordinals.Signed_Syntactic_Ordinal*
abbrevs =*p* = $=_p$
  and >*p* = $>_p$
  and ≥*p* = $\geq_p$
  and >*t* = $>_t$
  and ≥*t* = $\geq_t$
  and !*h* = $_h$
begin

This theory defines the graceful transfinite Knuth–Bendix order (KBO) with subterm coefficients for λ-free higher-order terms. The proof was developed by copying that of the standard KBO and generalizing it along two axes: subterm coefficients and ordinals. Both features complicate the definitions and proofs substantially.

6.1 Setup

locale *tkbo_coefs* = *kbo_std_basis* _ _ *arity_sym arity_var wt_sym*
    for
      arity_sym :: *'s* ⇒ *enat* **and**
      arity_var :: *'v* ⇒ *enat* **and**
      wt_sym :: *'s* ⇒ *hmultiset* +
  fixes *coef_sym* :: *'s* ⇒ *nat* ⇒ *hmultiset*
  assumes *coef_sym_gt_0*: *coef_sym f i* > *0*
begin

abbreviation δ_h :: *hmultiset* **where**
  $\delta_h \equiv$ *of_nat* δ

abbreviation ε_h :: *hmultiset* **where**
  $\varepsilon_h \equiv$ *of_nat* ε

abbreviation $arity\_sym_h$ :: *'s* ⇒ *hmultiset* **where**
  $arity\_sym_h$ *f* ≡ *hmset_of_enat* (*arity_sym f*)

abbreviation $arity\_var_h$ :: *'v* ⇒ *hmultiset* **where**
  $arity\_var_h$ *f* ≡ *hmset_of_enat* (*arity_var f*)

abbreviation $arity\_hd_h$ :: (*'s*, *'v*) *hd* ⇒ *hmultiset* **where**
  $arity\_hd_h$ *f* ≡ *hmset_of_enat* (*arity_hd f*)

abbreviation arity$_h$ :: ('s, 'v) tm ⇒ hmultiset **where**
  arity$_h$ s ≡ hmset_of_enat (arity s)

lemma arity$_h$_conv: arity$_h$ s = arity_hd$_h$ (head s) − of_nat (num_args s)
  unfolding arity_def **by** simp

lemma arity$_h$_App[simp]: arity$_h$ (App s t) = arity$_h$ s − 1
  by (simp add: one_enat_def)

lemmas wary_App$_h$[intro] = wary_App[folded of_nat_lt_hmset_of_enat_iff]
lemmas wary_AppE$_h$ = wary_AppE[folded of_nat_lt_hmset_of_enat_iff]
lemmas wary_num_args_le_arity_head$_h$ =
  wary_num_args_le_arity_head[folded of_nat_le_hmset_of_enat_iff]
lemmas wary_apps$_h$ = wary_apps[folded of_nat_le_hmset_of_enat_iff]
lemmas wary_cases_apps$_h$[consumes 1, case_names apps] =
  wary_cases_apps[folded of_nat_le_hmset_of_enat_iff]

lemmas ground_heads_arity$_h$ = ground_heads_arity[folded hmset_of_enat_le]
lemmas some_ground_head_arity$_h$ = some_ground_head_arity[folded hmset_of_enat_le]
lemmas ε$_h$_gt_0 = ε_gt_0[folded of_nat_less_hmset, unfolded of_nat_0]
lemmas δ$_h$_le_ε$_h$ = δ_le_ε[folded of_nat_le_hmset]
lemmas arity_hd$_h$_lt_ω_if_δ$_h$_gt_0 = arity_hd_ne_infinity_if_δ_gt_0
  [folded of_nat_less_hmset, unfolded of_nat_0, folded hmset_of_enat_lt_iff_ne_infinity]

lemma wt_sym_ge$_h$: wt_sym f ≥ ε$_h$ − δ$_h$ ∗ arity_sym$_h$ f
proof −
  have of_nat (the_enat (of_nat δ ∗ arity_sym f)) = δ$_h$ ∗ arity_sym$_h$ f
  by (cases arity_sym f, simp add: of_nat_eq_enat,
    metis arity_sym_ne_infinity_if_δ_gt_0 gr_zeroI mult_eq_0_iff of_nat_0 the_enat_0)
  thus ?thesis
    using wt_sym_ge[unfolded of_nat_minus_hmset] **by** metis
qed

lemmas unary_wt_sym_0_gt$_h$ = unary_wt_sym_0_gt[folded hmset_of_enat_inject, unfolded hmset_of_enat_1]
lemmas unary_wt_sym_0_imp_δ$_h$_eq_ε$_h$ = unary_wt_sym_0_imp_δ_eq_ε
  [folded of_nat_inject_hmset, unfolded of_nat_0]
lemmas extf_ext_snoc_if_δ$_h$_eq_ε$_h$ = extf_ext_snoc_if_δ_eq_ε[folded of_nat_inject_hmset]
lemmas extf_snoc_if_δ$_h$_eq_ε$_h$ = ext_snoc.snoc[OF extf_ext_snoc_if_δ$_h$_eq_ε$_h$]
lemmas arity_sym$_h$_lt_ω_if_δ$_h$_gt_0 = arity_sym_ne_infinity_if_δ_gt_0
  [folded of_nat_less_hmset hmset_of_enat_lt_iff_ne_infinity, unfolded of_nat_0]
lemmas arity_var$_h$_lt_ω_if_δ$_h$_gt_0 = arity_var_ne_infinity_if_δ_gt_0
  [folded of_nat_less_hmset hmset_of_enat_lt_iff_ne_infinity, unfolded of_nat_0]
lemmas arity$_h$_lt_ω_if_δ$_h$_gt_0 = arity_ne_infinity_if_δ_gt_0
  [folded of_nat_less_hmset hmset_of_enat_lt_iff_ne_infinity, unfolded of_nat_0]
lemmas warywary_subst_subst$_h$_conv = wary_subst_def[folded hmset_of_enat_le]
lemmas extf_singleton_nil_if_δ$_h$_eq_ε$_h$ = extf_singleton_nil_if_δ_eq_ε[folded of_nat_inject_hmset]

lemma arity_sym$_h$_if_δ$_h$_gt_0_E:
  assumes δ_gt_0: δ$_h$ > 0
  obtains n **where** arity_sym$_h$ f = of_nat n
  using arity_sym$_h$_lt_ω_if_δ$_h$_gt_0 assms lt_ω_imp_ex_of_nat **by** blast

lemma arity_var$_h$_if_δ$_h$_gt_0_E:
  assumes δ_gt_0: δ$_h$ > 0
  obtains n **where** arity_var$_h$ f = of_nat n
  using arity_var$_h$_lt_ω_if_δ$_h$_gt_0 assms lt_ω_imp_ex_of_nat **by** blast

6.2 Weights and Subterm Coefficients

abbreviation zhmset_of_tpoly :: ('a, hmultiset) tpoly ⇒ ('a, zhmultiset) tpoly **where**
  zhmset_of_tpoly ≡ map_tpoly (λx. x) zhmset_of

abbreviation eval_ztpoly :: ('a ⇒ zhmultiset) ⇒ ('a, hmultiset) tpoly ⇒ zhmultiset **where**

eval_ztpoly A p ≡ *eval_tpoly A* (*zhmset_of_tpoly p*)

lemma *eval_tpoly_eq_eval_ztpoly*[*simp*]:
zhmset_of (*eval_tpoly A p*) = *eval_ztpoly* (λ*v*. *zhmset_of* (*A v*)) *p*
by (*induct p*, *simp_all add*: *zhmset_of_sum_list zhmset_of_prod_list o_def*,
simp_all cong: *map_cong*)

definition *min_ground_head* :: (′*s*, ′*v*) *hd* ⇒ ′*s* **where**
min_ground_head ζ =
(*SOME f*. *f* ∈ *ground_heads* ζ ∧
(∀ *g* ∈ *ground_heads* ζ. *wt_sym g* + δ_h ∗ *arity_sym*$_h$ *g* ≥ *wt_sym f* + δ_h ∗ *arity_sym*$_h$ *f*))

datatype ′*va pvar* =
PWt ′*va*
| *PCoef* ′*va nat*

primrec *min_passign* :: ′*v pvar* ⇒ *hmultiset* **where**
min_passign (*PWt x*) = *wt_sym* (*min_ground_head* (*Var x*))
| *min_passign* (*PCoef* _ _) = *1*

abbreviation *min_zpassign* :: ′*v pvar* ⇒ *zhmultiset* **where**
min_zpassign v ≡ *zhmset_of* (*min_passign v*)

lemma *min_zpassign_simps*[*simp*]:
min_zpassign (*PWt x*) = *zhmset_of* (*wt_sym* (*min_ground_head* (*Var x*)))
min_zpassign (*PCoef x i*) = *1*
by (*simp_all add*: *zhmset_of_1*)

definition *legal_passign* :: (′*v pvar* ⇒ *hmultiset*) ⇒ *bool* **where**
legal_passign A ⟷ (∀ *x*. *A x* ≥ *min_passign x*)

definition *legal_zpassign* :: (′*v pvar* ⇒ *zhmultiset*) ⇒ *bool* **where**
legal_zpassign A ⟷ (∀ *x*. *A x* ≥ *min_zpassign x*)

lemma *legal_min_passign*: *legal_passign min_passign*
unfolding *legal_passign_def* **by** *simp*

lemma *legal_min_zpassign*: *legal_zpassign min_zpassign*
unfolding *legal_zpassign_def* **by** *simp*

lemma *assign_ge_0*[*intro*]: *legal_zpassign A* ⟹ *A x* ≥ *0*
unfolding *legal_zpassign_def* **by** (*auto intro*: *dual_order.trans*)

definition
eq_tpoly :: (′*v pvar*, *hmultiset*) *tpoly* ⇒ (′*v pvar*, *hmultiset*) *tpoly* ⇒ *bool* (**infix** ‹$=_p$› *50*)
where
q $=_p$ *p* ⟷ (∀ *A*. *legal_zpassign A* ⟶ *eval_ztpoly A q* = *eval_ztpoly A p*)

definition
ge_tpoly :: (′*v pvar*, *hmultiset*) *tpoly* ⇒ (′*v pvar*, *hmultiset*) *tpoly* ⇒ *bool* (**infix** ‹$\geq_p$› *50*)
where
q $\geq_p$ *p* ⟷ (∀ *A*. *legal_zpassign A* ⟶ *eval_ztpoly A q* ≥ *eval_ztpoly A p*)

definition
gt_tpoly :: (′*v pvar*, *hmultiset*) *tpoly* ⇒ (′*v pvar*, *hmultiset*) *tpoly* ⇒ *bool* (**infix** ‹$>_p$› *50*)
where
q $>_p$ *p* ⟷ (∀ *A*. *legal_zpassign A* ⟶ *eval_ztpoly A q* > *eval_ztpoly A p*)

lemma *gt_tpoly_imp_ge*[*intro*]: *q* $>_p$ *p* ⟹ *q* $\geq_p$ *p*
unfolding *ge_tpoly_def gt_tpoly_def* **by** (*simp add*: *le_less*)

lemma *eq_tpoly_refl*[*simp*]: *p* $=_p$ *p*
unfolding *eq_tpoly_def* **by** *simp*

```
lemma ge_tpoly_refl[simp]: p ≥p p
  unfolding ge_tpoly_def by simp

lemma gt_tpoly_irrefl: ¬ p >p p
  unfolding gt_tpoly_def legal_zpassign_def by fast

lemma
  eq_eq_tpoly_trans: r =p q ⟹ q =p p ⟹ r =p p and
  eq_ge_tpoly_trans: r =p q ⟹ q ≥p p ⟹ r ≥p p and
  eq_gt_tpoly_trans: r =p q ⟹ q >p p ⟹ r >p p and
  ge_eq_tpoly_trans: r ≥p q ⟹ q =p p ⟹ r ≥p p and
  ge_ge_tpoly_trans: r ≥p q ⟹ q ≥p p ⟹ r ≥p p and
  ge_gt_tpoly_trans: r ≥p q ⟹ q >p p ⟹ r >p p and
  gt_eq_tpoly_trans: r >p q ⟹ q =p p ⟹ r >p p and
  gt_ge_tpoly_trans: r >p q ⟹ q ≥p p ⟹ r >p p and
  gt_gt_tpoly_trans: r >p q ⟹ q >p p ⟹ r >p p
  unfolding eq_tpoly_def ge_tpoly_def gt_tpoly_def
  by (auto intro: order.trans less_trans less_le_trans le_less_trans)+

primrec coef_hd :: ('s, 'v) hd ⇒ nat ⇒ ('v pvar, hmultiset) tpoly where
  coef_hd (Var x) i = PVar (PCoef x i)
| coef_hd (Sym f) i = PNum (coef_sym f i)

lemma coef_hd_gt_0:
  assumes legal: legal_zpassign A
  shows eval_ztpoly A (coef_hd ζ i) > 0
  unfolding legal_zpassign_def
proof (cases ζ)
  case (Var x1)
  thus ?thesis
    using legal[unfolded legal_zpassign_def, rule_format, of PCoef x i for x]
    by (auto simp: coef_sym_gt_0 zhmset_of_1 intro: dual_order.strict_trans1 zero_less_one)
next
  case (Sym x2)
  thus ?thesis
    using legal[unfolded legal_zpassign_def, rule_format, of PWt x for x]
    by simp (metis coef_sym_gt_0 zhmset_of_0 zhmset_of_less)
qed

primrec coef :: ('s, 'v) tm ⇒ nat ⇒ ('v pvar, hmultiset) tpoly where
  coef (Hd ζ) i = coef_hd ζ i
| coef (App s _) i = coef s (i + 1)

lemma coef_apps[simp]: coef (apps s ss) i = coef s (i + length ss)
  by (induct ss arbitrary: s i) auto

lemma coef_gt_0: legal_zpassign A ⟹ eval_ztpoly A (coef s i) > 0
   by (induct s arbitrary: i) (auto intro: coef_hd_gt_0)

lemma exists_min_ground_head:
  ∃f. f ∈ ground_heads ζ ∧
     (∀g ∈ ground_heads ζ. wt_sym g + δh ∗ arity_symh g ≥ wt_sym f + δh ∗ arity_symh f)
proof −
  let ?R = {(f, g). f ∈ ground_heads ζ ∧ g ∈ ground_heads ζ ∧
    wt_sym g + δh ∗ arity_symh g > wt_sym f + δh ∗ arity_symh f}

  have wf_R: wf ?R
    using wf_app[of {(M, N). M < N} λf. wt_sym f + δh ∗ arity_symh f, OF wf]
    by (auto intro: wf_subset)
  have ∃f. f ∈ ground_heads ζ
    by (meson ground_heads_nonempty subsetI subset_empty)
  thus ?thesis
```

```
    using wf_eq_minimal[THEN iffD1, OF wf_R] by force
qed

lemma min_ground_head_Sym[simp]: min_ground_head (Sym f) = f
  unfolding min_ground_head_def by auto

lemma min_ground_head_in_ground_heads: min_ground_head ζ ∈ ground_heads ζ
  unfolding min_ground_head_def using someI_ex[OF exists_min_ground_head] by blast

lemma min_ground_head_min:
  f ∈ ground_heads ζ ⟹
   wt_sym f + δ_h ∗ arity_sym_h f ≥ wt_sym (min_ground_head ζ) + δ_h ∗ arity_sym_h (min_ground_head ζ)
  unfolding min_ground_head_def using someI_ex[OF exists_min_ground_head] by blast

lemma min_ground_head_antimono:
  ground_heads ζ ⊆ ground_heads ξ ⟹
   wt_sym (min_ground_head ζ) + δ_h ∗ arity_sym_h (min_ground_head ζ)
   ≥ wt_sym (min_ground_head ξ) + δ_h ∗ arity_sym_h (min_ground_head ξ)
  using min_ground_head_in_ground_heads min_ground_head_min by blast

primrec wt0 :: ('s, 'v) hd ⇒ ('v pvar, hmultiset) tpoly where
  wt0 (Var x) = PVar (PWt x)
| wt0 (Sym f) = PNum (wt_sym f)

lemma wt0_ge_min_ground_head:
  legal_zpassign A ⟹ eval_ztpoly A (wt0 ζ) ≥ zhmset_of (wt_sym (min_ground_head ζ))
  by (cases ζ, simp_all, metis legal_zpassign_def min_zpassign_simps(1))

lemma eval_ztpoly_nonneg: legal_zpassign A ⟹ eval_ztpoly A p ≥ 0
  by (induct p) (auto cong: map_cong intro!: sum_list_nonneg prod_list_nonneg)

lemma in_zip_imp_size_lt_apps: (s, y) ∈ set (zip ss ys) ⟹ size s < size (apps (Hd ζ) ss)
  by (auto dest!: set_zip_leftD simp: size_in_args)

function wt :: ('s, 'v) tm ⇒ ('v pvar, hmultiset) tpoly where
  wt (apps (Hd ζ) ss) =
   PSum ([wt0 ζ, PNum (δ_h ∗ (arity_sym_h (min_ground_head ζ) − of_nat (length ss)))] @
     map (λ(s, i). PMult [coef_hd ζ i, wt s]) (zip ss [0..<length ss]))
  by (erule tm_exhaust_apps) simp
termination
  by (lexicographic_order simp: in_zip_imp_size_lt_apps)

definition
  wt_args :: nat ⇒ ('v pvar ⇒ zhmultiset) ⇒ ('s, 'v) hd ⇒ ('s, 'v) tm list ⇒ zhmultiset
where
  wt_args i A ζ ss = sum_list
     (map (eval_ztpoly A ∘ (λ(s, i). PMult [coef_hd ζ i, wt s])) (zip ss [i..<i + length ss]))

lemma wt_Hd[simp]: wt (Hd ζ) = PSum [wt0 ζ, PNum (δ_h ∗ arity_sym_h (min_ground_head ζ))]
  by (rule wt.simps[of _ [], simplified])

lemma coef_hd_cong:
  (∀x ∈ vars_hd ζ. ∀i. A (PCoef x i) = B (PCoef x i)) ⟹
   eval_ztpoly A (coef_hd ζ i) = eval_ztpoly B (coef_hd ζ i)
  by (cases ζ) auto

lemma wt0_cong:
  assumes pwt_eq: ∀x ∈ vars_hd ζ. A (PWt x) = B (PWt x)
  shows eval_ztpoly A (wt0 ζ) = eval_ztpoly B (wt0 ζ)
  using pwt_eq by (cases ζ) auto

lemma wt_cong:
  assumes
```

```
    ∀ x ∈ vars s. A (PWt x) = B (PWt x) and
    ∀ x ∈ vars s. ∀ i. A (PCoef x i) = B (PCoef x i)
  shows eval_ztpoly A (wt s) = eval_ztpoly B (wt s)
  using assms
proof (induct s rule: tm_induct_apps)
  case (apps ζ ss)
  note ih = this(1) and pwt_eq = this(2) and pcoef_eq = this(3)

  have ih': eval_ztpoly A (wt s) = eval_ztpoly B (wt s) if s_in: s ∈ set ss for s
  proof (rule ih[OF s_in])
    show ∀ x ∈ vars s. A (PWt x) = B (PWt x)
      using pwt_eq s_in by force
    show ∀ x ∈ vars s. ∀ i. A (PCoef x i) = B (PCoef x i)
      using pcoef_eq s_in by force
  qed

  have wt0_eq: eval_ztpoly A (wt0 ζ) = eval_ztpoly B (wt0 ζ)
    by (rule wt0_cong) (simp add: pwt_eq)
  have coef_ζ_eq: eval_ztpoly A (coef_hd ζ i) = eval_ztpoly B (coef_hd ζ i) for i
    by (rule coef_hd_cong) (simp add: pcoef_eq)

  show ?case
    using ih' wt0_eq coef_ζ_eq by (auto dest!: set_zip_leftD intro!: arg_cong[of _ _ sum_list])
qed

lemma ground_eval_ztpoly_wt_eq: ground s ⟹ eval_ztpoly A (wt s) = eval_ztpoly B (wt s)
  by (rule wt_cong) auto

lemma exists_wt_sym:
  assumes legal: legal_zpassign A
  shows ∃ f ∈ ground_heads ζ. eval_ztpoly A (wt (Hd ζ)) ≥ zhmset_of (wt_sym f + δ_h * arity_sym_h f)
  unfolding eq_tpoly_def
proof (cases ζ)
  case Var
  thus ?thesis
    using legal[unfolded legal_zpassign_def]
    by simp (metis add_le_cancel_right ground_heads.simps(1) min_ground_head_in_ground_heads
      min_zpassign_simps(1) zhmset_of_plus)
next
  case Sym
  thus ?thesis
    by (simp add: zhmset_of_plus)
qed

lemma wt_ge_ε_h:
  assumes legal: legal_zpassign A
  shows eval_ztpoly A (wt s) ≥ zhmset_of ε_h
proof (induct s rule: tm_induct_apps)
  case (apps ζ ss)
  note ih = this(1)

  {
    assume ss_eq_nil: ss = []

    have ε_h ≤ wt_sym (min_ground_head ζ) + δ_h * arity_sym_h (min_ground_head ζ)
      using wt_sym_ge_h[of min_ground_head ζ]
      by (metis add_diff_cancel_left' leD leI le_imp_minus_plus_hmset le_minus_plus_same_hmset
        less_le_trans)
    hence zhmset_of ε_h
      ≤ zhmset_of (wt_sym (min_ground_head ζ)) + zhmset_of (δ_h * arity_sym_h (min_ground_head ζ))
      by (metis zhmset_of_le zhmset_of_plus)
    also have ...
      ≤ eval_tpoly A (map_tpoly (λx. x) zhmset_of (wt0 ζ))
```

```
        + zhmset_of (δ_h * arity_sym_h (min_ground_head ζ))
      using wt0_ge_min_ground_head[OF legal] by simp
    finally have ?case
      using ss_eq_nil by simp
  }
  moreover
  {
    let ?arg_wt =
      eval_tpoly A ∘ (map_tpoly (λx. x) zhmset_of ∘ (λ(s, i). PMult [coef_hd ζ i, wt s]))

    assume ss_ne_nil: ss ≠ []
    hence zhmset_of ε_h
      ≤ eval_tpoly A (map_tpoly (λx. x) zhmset_of (PMult [coef_hd ζ 0, wt (hd ss)]))
      by (simp add: ih coef_hd_gt_0[OF legal] nonneg_le_mult_right_mono_zhmset)
    also have ... = hd (map ?arg_wt (zip ss [0..<length ss]))
      using ss_ne_nil by (simp add: hd_map zip_nth_conv hd_conv_nth)
    also have ... ≤ sum_list (map ?arg_wt (zip ss [0..<length ss]))
      by (rule hd_le_sum_list,
        metis (no_types, lifting) length_greater_0_conv list.collapse list.simps(3) list.simps(9)
          ss_ne_nil upt_conv_Cons zip_Cons_Cons,
        simp add: eval_ztpoly_nonneg legal)
    also have ...
      ≤ eval_tpoly A (map_tpoly (λx. x) zhmset_of (wt0 ζ)) +
        (zhmset_of (δ_h * (arity_sym_h (min_ground_head ζ) − of_nat (length ss))) +
         sum_list (map ?arg_wt (zip ss [0..<length ss])))
    proof −
      have 0 ≤ eval_tpoly A (map_tpoly (λp. p) zhmset_of (wt0 ζ))
        using legal eval_ztpoly_nonneg by blast
      then show ?thesis
        by (meson leD leI le_add_same_cancel2 less_le_trans zhmset_of_nonneg)
    qed
    finally have ?case
      by simp
  }
  ultimately show ?case
    by linarith
qed

lemma wt_args_ge_length_times_ε_h:
  assumes legal: legal_zpassign A
  shows wt_args i A ζ ss ≥ of_nat (length ss) * zhmset_of ε_h
  unfolding wt_args_def
  by (rule sum_list_ge_length_times[unfolded wt_args_def,
      of map (eval_ztpoly A ∘ (λ(s, i). PMult [coef_hd ζ i, wt s])) (zip ss [i..<i + length ss]),
      simplified],
    auto intro!: mult_le_mono_hmset[of 1, simplified] nonneg_le_mult_right_mono_zhmset coef_hd_gt_0
      simp: legal zero_less_iff_1_le_hmset[symmetric] coef_hd_gt_0 wt_ge_ε_h)

lemma wt_ge_δ_h: legal_zpassign A ⟹ eval_ztpoly A (wt s) ≥ zhmset_of δ_h
  using δ_h_le_ε_h[folded zhmset_of_le] order.trans wt_ge_ε_h zhmset_of_le by blast

lemma wt_gt_0: legal_zpassign A ⟹ eval_ztpoly A (wt s) > 0
  using ε_h_gt_0[folded zhmset_of_less, unfolded zhmset_of_0] wt_ge_ε_h by (blast intro: less_le_trans)

lemma wt_gt_δ_h_if_superunary:
  assumes
    legal: legal_zpassign A and
    superunary: arity_hd_h (head s) > 1
  shows eval_ztpoly A (wt s) > zhmset_of δ_h
proof (cases δ_h = ε_h)
  case δ_ne_ε: False
  show ?thesis
    using order.not_eq_order_implies_strict[OF δ_ne_ε δ_h_le_ε_h, folded zhmset_of_less]
```

```
      wt_ge_εh[OF legal] by (blast intro: less_le_trans)
next
  case δ_eq_ε: True
  show ?thesis
    using superunary
  proof (induct s rule: tm_induct_apps)
    case (apps ζ ss)
    have arity_hdh ζ > 1
      using apps(2) by simp
    hence min_gr_ary: arity_symh (min_ground_head ζ) > 1
      using ground_heads_arityh less_le_trans min_ground_head_in_ground_heads by blast

    have zhmset_of δh < eval_ztpoly A (wt0 ζ) + zhmset_of (δh ∗ arity_symh (min_ground_head ζ))
      unfolding δ_eq_ε
      by (rule add_strict_increasing2[OF eval_ztpoly_nonneg[OF legal]], unfold zhmset_of_less,
        rule gt_0_lt_mult_gt_1_hmset[OF εh_gt_0 min_gr_ary])
    also have ... ≤ eval_ztpoly A (wt0 ζ)
      + zhmset_of (δh ∗ (arity_symh (min_ground_head ζ) − of_nat (length ss)))
        + zhmset_of (of_nat (length ss) ∗ εh)
      by (auto simp: εh_gt_0 δ_eq_ε zmset_of_le zhmset_of_plus[symmetric] algebra_simps
            simp del: ring_distribs simp: ring_distribs[symmetric])
        (metis add.commute le_minus_plus_same_hmset)
    also have ... ≤ eval_ztpoly A (wt0 ζ)
      + zhmset_of (δh ∗ (arity_symh (min_ground_head ζ) − of_nat (length ss))) + wt_args 0 A ζ ss
      using wt_args_ge_length_times_εh[OF legal] by (simp add: zhmset_of_times of_nat_zhmset)
    finally show ?case
      by (simp add: wt_args_def add_ac(1) comp_def)
  qed
qed

lemma wt_App_plus_δh_ge:
  eval_ztpoly A (wt (App s t)) + zhmset_of δh
    ≥ eval_ztpoly A (wt s) + eval_ztpoly A (coef s 0) ∗ eval_ztpoly A (wt t)
proof (cases s rule: tm_exhaust_apps)
  case s: (apps ζ ss)
  show ?thesis
  proof (cases arity_symh (min_ground_head ζ) = ω)
    case ary_eq_ω: True
    show ?thesis
      unfolding ary_eq_ω s App_apps wt.simps
      by (auto simp: diff_diff_add_hmset[symmetric] add.assoc)
  next
    case False
    show ?thesis
      unfolding s App_apps wt.simps
      by (simp add: algebra_simps zhmset_of_plus[symmetric] zmset_of_le,
        simp del: diff_diff_add_hmset add: add.commute[of 1] le_minus_plus_same_hmset
          distrib_left[of _ 1 :: hmultiset, unfolded mult.right_neutral, symmetric]
          diff_diff_add_hmset[symmetric])
  qed
qed

lemma wt_App_fun_δh:
  assumes
    legal: legal_zpassign A and
    wt_st: eval_ztpoly A (wt (App s t)) = eval_ztpoly A (wt t)
  shows eval_ztpoly A (wt s) = zhmset_of δh
proof −
  have eval_ztpoly A (wt (App s t)) = eval_ztpoly A (wt t)
    using wt_st by simp
  hence wt_s_t_le_δ_t: eval_ztpoly A (wt s) + eval_ztpoly A (coef s 0) ∗ eval_ztpoly A (wt t)
    ≤ zhmset_of δh + eval_ztpoly A (wt t)
    using wt_App_plus_δh_ge by (metis add.commute)
```

also have ... $\leq$ *eval_ztpoly A* (*wt s*) + *eval_ztpoly A* (*wt t*)
  using *wt_ge_*δ_h[*OF legal*] **by** *simp*
finally have *eval_ztpoly A* (*coef s 0*) $*$ *eval_ztpoly A* (*wt t*) $\leq$ *eval_ztpoly A* (*wt t*)
  by *simp*
hence *eval_ztpoly A* (*coef s 0*) = *1*
  using *eval_ztpoly_nonneg*[*OF legal*]
  by (*metis* (*no_types*, *lifting*) *coef_gt_0 dual_order.order_iff_strict leD legal mult_cancel_right1*
    nonneg_le_mult_right_mono_zhmset wt_gt_0)
thus *?thesis*
  using *wt_s_t_le_*δ*_t* **by** (*simp add*: *add.commute antisym wt_ge_*δ_h[*OF legal*])
qed

lemma *wt_App_arg_*δ_h:
assumes
  legal: *legal_zpassign A* **and**
  wt_st: *eval_ztpoly A* (*wt* (*App s t*)) = *eval_ztpoly A* (*wt s*)
shows *eval_ztpoly A* (*wt t*) = *zhmset_of* δ_h
proof −
have *eval_ztpoly A* (*wt* (*App s t*)) + *zhmset_of* δ_h = *eval_ztpoly A* (*wt s*) + *zhmset_of* δ_h
  using *wt_st* **by** *simp*
hence *eval_ztpoly A* (*coef s 0*) $*$ *eval_ztpoly A* (*wt t*) $\leq$ *zhmset_of* δ_h (**is** *?k* $*$ *?w* $\leq$ _)
  by (*metis add_le_cancel_left wt_App_plus_*δ_h*_ge*)
hence *?k* $*$ *?w* = *zhmset_of* δ_h
  using *wt_ge_*δ_h[*OF legal*] *coef_gt_0*[*OF legal*, *unfolded zero_less_iff_1_le_hmset*]
  by (*simp add*: *antisym nonneg_le_mult_right_mono_zhmset*)
hence *?w* $\leq$ *zhmset_of* δ_h
  by (*metis coef_gt_0*[*OF legal*] *dual_order.order_iff_strict eval_ztpoly_nonneg*[*OF legal*]
    nonneg_le_mult_right_mono_zhmset)
thus *?thesis*
  by (*simp add*: *antisym wt_ge_*δ_h[*OF legal*])
qed

lemma *wt_App_ge_fun*: *wt* (*App s t*) $\geq_p$ *wt s*
unfolding *ge_tpoly_def*
proof *clarify*
fix *A*
assume *legal*: *legal_zpassign A*

have *zhmset_of* δ_h $\leq$ *eval_ztpoly A* (*coef s 0*) $*$ *eval_ztpoly A* (*wt t*)
  by (*simp add*: *coef_gt_0 legal nonneg_le_mult_right_mono_zhmset wt_ge_*δ_h)
hence *eval_ztpoly A* (*wt s*) + *zhmset_of* δ_h $\leq$ *eval_ztpoly A* (*wt* (*App s t*)) + *zhmset_of* δ_h
  by (*metis add_le_cancel_right add_less_le_mono not_le wt_App_plus_*δ_h*_ge*)
thus *eval_ztpoly A* (*wt s*) $\leq$ *eval_ztpoly A* (*wt* (*App s t*))
  by *simp*
qed

lemma *wt_App_ge_arg*: *wt* (*App s t*) $\geq_p$ *wt t*
unfolding *ge_tpoly_def*
by (*cases s rule*: *tm_exhaust_apps*, *simp*, *unfold App_apps wt.simps*)
  (*auto simp*: *comp_def coef_hd_gt_0 eval_ztpoly_nonneg nonneg_le_mult_right_mono_zhmset*
    intro!: *sum_list_nonneg eval_ztpoly_nonneg add_increasing*)

lemma *wt_*δ_h*_imp_*δ_h*_eq_*ε_h:
assumes
  legal: *legal_zpassign A* **and**
  *wt_s_eq_*δ: *eval_ztpoly A* (*wt s*) = *zhmset_of* δ_h
shows $\delta_h = \varepsilon_h$
using δ_h*_le_*ε_h *wt_ge_*ε_h[*OF legal*, *of s*, *unfolded wt_s_eq_*δ *zhmset_of_le*] **by** (*rule antisym*)

lemma *wt_ge_vars*: *wt t* $\geq_p$ *wt s* $\Longrightarrow$ *vars t* $\supseteq$ *vars s*
proof (*induct s*)
case *t*: (*Hd* ζ)
note *wt_ge_*ζ = *this*(*1*)

```
  show ?case
  proof (cases ζ)
    case ζ: (Var x)

    {
      assume z_ni_t: x ∉ vars t

      let ?A = min_zpassign
      let ?B = λv. if v = PWt x then eval_ztpoly ?A (wt t) + ?A v + 1 else ?A v

      have legal_B: legal_zpassign ?B
        unfolding legal_zpassign_def
        by (auto simp: legal_min_zpassign intro!: add_increasing eval_ztpoly_nonneg)

      have eval_B_eq_A: eval_ztpoly ?B (wt t) = eval_ztpoly ?A (wt t)
        by (rule wt_cong) (auto simp: z_ni_t)
      have eval_ztpoly ?B (wt (Hd (Var x))) > eval_ztpoly ?B (wt t)
        by (auto simp: eval_B_eq_A zero_less_iff_1_le_zhmset zhmset_of_plus[symmetric]
          algebra_simps)
      hence False
        using wt_ge_ζ ζ unfolding ge_tpoly_def
        by (blast dest: leD intro: legal_B legal_min_zpassign)
    }
    thus ?thesis
      by (auto simp: ζ)
  qed simp
next
  case (App s1 s2)
  note ih1 = this(1) and ih2 = this(2) and wt_t_ge_wt_s1s2 = this(3)

  have vars s1 ⊆ vars t
    using ih1 wt_t_ge_wt_s1s2 wt_App_ge_fun order_trans unfolding ge_tpoly_def by blast
  moreover have vars s2 ⊆ vars t
    using ih2 wt_t_ge_wt_s1s2 wt_App_ge_arg order_trans unfolding ge_tpoly_def by blast
  ultimately show ?case
    by simp
qed

lemma sum_coefs_ge_num_args_if_δ_h_eq_0:
  assumes
    legal: legal_passign A and
    δ_eq_0: δ_h = 0 and
    wary_s: wary s
  shows sum_coefs (eval_tpoly A (wt s)) ≥ num_args s
proof (cases s rule: tm_exhaust_apps)
  case s: (apps ζ ss)
  show ?thesis
    unfolding s
  proof (induct ss rule: rev_induct)
    case (snoc sa ss)
    note ih = this

    let ?Az = λv. zhmset_of (A v)

    have legalz: legal_zpassign ?Az
      using legal unfolding legal_passign_def legal_zpassign_def zhmset_of_le by assumption

    have eval_ztpoly ?Az (coef_hd ζ (length ss)) > 0
      using legal coef_hd_gt_0 eval_tpoly_eq_eval_ztpoly
      by (simp add: coef_hd_gt_0[OF legalz])
    hence k: eval_tpoly A (coef_hd ζ (length ss)) > 0 (is ?k > _)
      unfolding eval_tpoly_eq_eval_ztpoly[symmetric] zhmset_of_less[symmetric] zhmset_of_0
      by assumption
```

have *eval_ztpoly ?Az* (*wt sa*) > *0* (**is** *?w* > _)
by (*simp add*: *wt_gt_0*[*OF legalz*])
hence *w*: *eval_tpoly A* (*wt sa*) > *0* (**is** *?w* > _)
unfolding *eval_tpoly_eq_eval_ztpoly*[*symmetric*] *zhmset_of_less*[*symmetric*] *zhmset_of_0*
by *assumption*

have *?k* ∗ *?w* > *0*
using *k w* **by** *simp*
hence *sum_coefs* (*?k* ∗ *?w*) > *0*
by (*rule sum_coefs_gt_0*[*THEN iffD2*])
thus *?case*
using *ih* **by** (*simp del*: *apps_append add*: *s δ_eq_0*)
qed *simp*
qed

6.3 Inductive Definitions

inductive *gt* :: (′*s*, ′*v*) *tm* ⇒ (′*s*, ′*v*) *tm* ⇒ *bool* (**infix** ‹$>_t$› *50*) **where**
gt_wt: *wt t* $>_p$ *wt s* ⟹ *t* $>_t$ *s*
| *gt_unary*: *wt t* $\geq_p$ *wt s* ⟹ ¬ *head t* $\leq\geq_{hd}$ *head s* ⟹ *num_args t* = *1* ⟹
(∃ *f* ∈ *ground_heads* (*head t*). *arity_sym f* = *1* ∧ *wt_sym f* = *0*) ⟹ *arg t* $>_t$ *s* ∨ *arg t* = *s* ⟹
t $>_t$ *s*
| *gt_diff*: *wt t* $\geq_p$ *wt s* ⟹ *head t* $>_{hd}$ *head s* ⟹ *t* $>_t$ *s*
| *gt_same*: *wt t* $\geq_p$ *wt s* ⟹ *head t* = *head s* ⟹
(∀ *f* ∈ *ground_heads* (*head t*). *extf f* ($>_t$) (*args t*) (*args s*)) ⟹ *t* $>_t$ *s*

abbreviation *ge* :: (′*s*, ′*v*) *tm* ⇒ (′*s*, ′*v*) *tm* ⇒ *bool* (**infix** ‹$\geq_t$› *50*) **where**
t $\geq_t$ *s* ≡ *t* $>_t$ *s* ∨ *t* = *s*

inductive *gt_wt* :: (′*s*, ′*v*) *tm* ⇒ (′*s*, ′*v*) *tm* ⇒ *bool* **where**
gt_wtI: *wt t* $>_p$ *wt s* ⟹ *gt_wt t s*

inductive *gt_unary* :: (′*s*, ′*v*) *tm* ⇒ (′*s*, ′*v*) *tm* ⇒ *bool* **where**
gt_unaryI: *wt t* $\geq_p$ *wt s* ⟹ ¬ *head t* $\leq\geq_{hd}$ *head s* ⟹ *num_args t* = *1* ⟹
(∃ *f* ∈ *ground_heads* (*head t*). *arity_sym f* = *1* ∧ *wt_sym f* = *0*) ⟹ *arg t* $\geq_t$ *s* ⟹ *gt_unary t s*

inductive *gt_diff* :: (′*s*, ′*v*) *tm* ⇒ (′*s*, ′*v*) *tm* ⇒ *bool* **where**
gt_diffI: *wt t* $\geq_p$ *wt s* ⟹ *head t* $>_{hd}$ *head s* ⟹ *gt_diff t s*

inductive *gt_same* :: (′*s*, ′*v*) *tm* ⇒ (′*s*, ′*v*) *tm* ⇒ *bool* **where**
gt_sameI: *wt t* $\geq_p$ *wt s* ⟹ *head t* = *head s* ⟹
(∀ *f* ∈ *ground_heads* (*head t*). *extf f* ($>_t$) (*args t*) (*args s*)) ⟹ *gt_same t s*

lemma *gt_iff_wt_unary_diff_same*: *t* $>_t$ *s* ⟷ *gt_wt t s* ∨ *gt_unary t s* ∨ *gt_diff t s* ∨ *gt_same t s*
by (*subst gt.simps*) (*auto simp*: *gt_wt.simps gt_unary.simps gt_diff.simps gt_same.simps*)

lemma *gt_imp_wt*: *t* $>_t$ *s* ⟹ *wt t* $\geq_p$ *wt s*
by (*blast elim*: *gt.cases*)

lemma *gt_imp_vars*: *t* $>_t$ *s* ⟹ *vars t* ⊇ *vars s*
by (*erule wt_ge_vars*[*OF gt_imp_wt*])

6.4 Irreflexivity

theorem *gt_irrefl*: *wary s* ⟹ ¬ *s* $>_t$ *s*
proof (*induct size s arbitrary*: *s rule*: *less_induct*)
case *less*
note *ih* = *this*(*1*) **and** *wary_s* = *this*(*2*)

show *?case*
proof
assume *s_gt_s*: *s* $>_t$ *s*
show *False*

```
      using s_gt_s
    proof (cases rule: gt.cases)
      case gt_same
      then obtain f where f: extf f (>t) (args s) (args s)
        by fastforce
      thus False
        using wary_s ih by (metis wary_args extf_irrefl size_in_args)
    qed (auto simp: comp_hd_def gt_tpoly_irrefl gt_hd_irrefl)
  qed
qed
```

6.5 Transitivity

```
lemma not_extf_gt_nil_singleton_if_δh_eq_εh:
  assumes wary_s: wary s and δ_eq_ε: δh = εh
  shows ¬ extf f (>t) [] [s]
proof
  assume nil_gt_s: extf f (>t) [] [s]
  note s_gt_nil = extf_singleton_nil_if_δh_eq_εh[OF δ_eq_ε, of f gt s]
  have ¬ extf f (>t) [] []
    by (rule extf_irrefl) simp
  moreover have extf f (>t) [] []
    using extf_trans_from_irrefl[of {s}, OF _ _ _ _ _ _ nil_gt_s s_gt_nil] gt_irrefl[OF wary_s]
    by fastforce
  ultimately show False
    by sat
qed

lemma gt_sub_arg: wary (App s t) ⟹ App s t >t t
proof (induct t arbitrary: s rule: measure_induct_rule[of size])
  case (less t)
  note ih = this(1) and wary_st = this(2)

  {
    fix A
    assume
      legal: legal_zpassign A and
      wt_st: eval_ztpoly A (wt (App s t)) = eval_ztpoly A (wt t)

    have δ_eq_ε: δh = εh
      using wt_App_fun_δh[OF legal] wt_δh_imp_δh_eq_εh[OF legal] wt_st by blast
    hence δ_gt_0: δh > 0
      using εh_gt_0 by simp

    have wt_s: eval_ztpoly A (wt s) = zhmset_of δh
      by (rule wt_App_fun_δh[OF legal wt_st])

    have wary_t: wary t
      by (rule wary_AppEh[OF wary_st])
    have nargs_lt: of_nat (num_args s) < arity_hdh (head s)
      by (rule wary_AppEh[OF wary_st])

    have ary_hd_s: arity_hdh (head s) = 1
      by (metis gr_implies_not_zero_hmset legal lt_1_iff_eq_0_hmset nargs_lt neq_iff
        wt_gt_δh_if_superunary wt_s)
    hence nargs_s: num_args s = 0
      by (metis less_one nargs_lt of_nat_1 of_nat_less_hmset)
    hence s_eq_hd: s = Hd (head s)
      by (simp add: Hd_head_id)
    obtain f where
      f_in: f ∈ ground_heads (head s) and
      wt_f_etc: wt_sym f + δh ∗ arity_symh f = δh
    proof −
      assume a: ⋀f. ⟦f ∈ local.ground_heads (head s); wt_sym f + δh ∗ arity_symh f = δh⟧ ⟹ thesis
```

```
  have ⋀f. δ_h − δ_h ∗ arity_sym_h f ≤ wt_sym f
    using wt_s by (metis legal wt_δ_h_imp_δ_h_eq_ε_h wt_sym_ge_h)
  hence ⋀s. ¬ δ_h ∗ arity_sym_h s + wt_sym s < δ_h
    by (metis add_diff_cancel_left′ le_imp_minus_plus_hmset leD le_minus_plus_same_hmset
        less_le_trans)
  thus thesis
    using a wt_s s_eq_hd
    by (metis exists_wt_sym legal add.commute order.not_eq_order_implies_strict zhmset_of_le)
qed

have ary_f_1: arity_sym f = 1
  by (metis δ_gt_0 add_diff_cancel_left′ ary_hd_s diff_le_self_hmset dual_order.order_iff_strict
    f_in ground_heads_arity_h gt_0_lt_mult_gt_1_hmset hmset_of_enat_1 hmset_of_enat_inject leD
    wt_f_etc)
hence wt_f_0: wt_sym f = 0
  using wt_f_etc by simp

{
  assume hd_s_ncmp_t: ¬ head s ≤≥_hd head t
  have ?case
    by (rule gt_unary[OF wt_App_ge_arg])
      (auto simp: hd_s_ncmp_t nargs_s intro: f_in ary_f_1 wt_f_0)
}
moreover
{
  assume hd_s_gt_t: head s >_hd head t
  have ?case
    by (rule gt_diff[OF wt_App_ge_arg]) (simp add: hd_s_gt_t)
}
moreover
{
  assume head t >_hd head s
  hence False
    using ary_f_1 wt_f_0 f_in gt_hd_irrefl gt_sym_antisym unary_wt_sym_0_gt_h hmset_of_enat_1
    unfolding gt_hd_def by metis
}
moreover
{
  assume hd_t_eq_s: head t = head s
  hence nargs_t_le: num_args t ≤ 1
    using ary_hd_s wary_num_args_le_arity_head_h[OF wary_t] of_nat_le_hmset by fastforce

  have extf: extf f (>_t) [t] (args t) for f
  proof (cases args t)
    case Nil
    thus ?thesis
      by (simp add: extf_singleton_nil_if_δ_h_eq_ε_h[OF δ_eq_ε])
  next
    case args_t: (Cons ta ts)
    hence ts: ts = []
      using ary_hd_s[folded hd_t_eq_s] wary_num_args_le_arity_head_h[OF wary_t] of_nat_le_hmset
        nargs_t_le by simp
    have ta: ta = arg t
      by (metis apps.simps(1) apps.simps(2) args_t tm.sel(6) tm_collapse_apps ts)
    hence t: t = App (fun t) ta
      by (metis args.simps(1) args_t not_Cons_self2 tm.exhaust_sel ts)
    have t >_t ta
      by (rule ih[of ta fun t, folded t, OF _ wary_t]) (metis ta size_arg_lt t tm.disc(2))
    thus ?thesis
      unfolding args_t ts by (metis extf_singleton gt_irrefl wary_t)
  qed
  have ?case
    by (rule gt_same[OF wt_App_ge_arg])
```

```
        (simp_all add: hd_t_eq_s length_0_conv[THEN iffD1, OF nargs_s] extf)
    }
    ultimately have ?case
      unfolding comp_hd_def by metis
  }
  thus ?case
    using gt_wt by (metis ge_tpoly_def gt_tpoly_def wt_App_ge_arg order.not_eq_order_implies_strict)
qed

lemma gt_arg: wary s ⟹ is_App s ⟹ s >t arg s
  by (cases s) (auto intro: gt_sub_arg)

theorem gt_trans: wary u ⟹ wary t ⟹ wary s ⟹ u >t t ⟹ t >t s ⟹ u >t s
proof (simp only: atomize_imp,
    rule measure_induct_rule[of λ(u, t, s). {#size u, size t, size s#}
        λ(u, t, s). wary u ⟶ wary t ⟶ wary s ⟶ u >t t ⟶ t >t s ⟶ u >t s (u, t, s),
      simplified prod.case],
    simp only: split_paired_all prod.case atomize_imp[symmetric])
  fix u t s
  assume
    ih: ⋀ua ta sa. {#size ua, size ta, size sa#} < {#size u, size t, size s#} ⟹
      wary ua ⟹ wary ta ⟹ wary sa ⟹ ua >t ta ⟹ ta >t sa ⟹ ua >t sa and
    wary_u: wary u and wary_t: wary t and wary_s: wary s and
    u_gt_t: u >t t and t_gt_s: t >t s

  have wt_u_ge_t: wt u ≥p wt t and wt_t_ge_s: wt t ≥p wt s
    using gt_imp_wt u_gt_t t_gt_s by auto
  hence wt_u_ge_s: wt u ≥p wt s
    by (rule ge_ge_tpoly_trans)

  have wary_arg_u: wary (arg u)
    by (rule wary_arg[OF wary_u])
  have wary_arg_t: wary (arg t)
    by (rule wary_arg[OF wary_t])
  have wary_arg_s: wary (arg s)
    by (rule wary_arg[OF wary_s])

  show u >t s
    using t_gt_s
  proof cases
    case gt_wt_t_s: gt_wt
    hence wt u >p wt s
      using wt_u_ge_t ge_gt_tpoly_trans by blast
    thus ?thesis
      by (rule gt_wt)
  next
    case gt_unary_t_s: gt_unary

    have t_app: is_App t
      by (metis args_Nil_iff_is_Hd gt_unary_t_s(3) length_greater_0_conv less_numeral_extra(1))
    hence nargs_fun_t: num_args (fun t) < arity_hd (head (fun t))
      by (metis tm.collapse(2) wary_AppE wary_t)

    have δ_eq_ε: δh = εh
      using gt_unary_t_s(4) unary_wt_sym_0_imp_δh_eq_εh by blast

    show ?thesis
      using u_gt_t
    proof cases
      case gt_wt_u_t: gt_wt
      hence wt u >p wt s
        using wt_t_ge_s gt_ge_tpoly_trans by blast
      thus ?thesis
```

```
    by (rule gt_wt)
next
  case gt_unary_u_t: gt_unary
  have u_app: is_App u
    by (metis args_Nil_iff_is_Hd gt_unary_u_t(3) length_greater_0_conv less_numeral_extra(1))
  hence nargs_fun_u: num_args (fun u) = 0
    by (metis args.simps(1) gt_unary_u_t(3) list.size(3) one_arg_imp_Hd tm.collapse(2))

  have arg_u_gt_s: arg u >_t s
    using ih[of arg u t s] u_app gt_unary_u_t(5) t_gt_s size_arg_lt wary_arg_u wary_s wary_t
    by force
  hence arg_u_ge_s: arg u ≥_t s
    by sat

  {
    assume size (arg u) < size t
    hence {#size u, size (arg u), size s#} < {#size u, size t, size s#}
      by simp
    hence ?thesis
      using ih[of u arg u s] arg_u_gt_s gt_arg u_app wary_s wary_u by blast
  }
  moreover
  {
    assume size (arg t) < size s
    hence u >_t arg t
      using ih[of u t arg t] args_Nil_iff_is_Hd gt_arg gt_unary_t_s(3) u_gt_t wary_t wary_u
      by force
    hence ?thesis
      using ih[of u arg t s] args_Nil_iff_is_Hd gt_unary_t_s(3,5) size_arg_lt wary_arg_t
        wary_s wary_u by force
  }
  moreover
  {
    assume sz_u_gt_t: size u > size t and sz_t_gt_s: size t > size s

    {
      assume hd_u_eq_s: head u = head s
      hence ary_hd_s: arity_hd (head s) = 1
        using ground_heads_arity gt_unary_u_t(3,4) hd_u_eq_s one_enat_def
          wary_num_args_le_arity_head wary_u by fastforce

      have extf: extf f (>_t) (args u) (args s) for f
      proof (cases args s)
        case Nil
        thus ?thesis
          by (metis δ_eq_ε args.elims args_Nil_iff_is_Hd extf_snoc_if_δ_h_eq_ε_h length_0_conv
            nargs_fun_u tm.sel(4) u_app)
      next
        case args_s: (Cons sa ss)
        hence ss: ss = []
          by (cases s, simp, metis One_nat_def antisym_conv ary_hd_s diff_Suc_1
            enat_ord_simps(1) le_add2 length_0_conv length_Cons list.size(4) one_enat_def
            wary_num_args_le_arity_head wary_s)
        have sa: sa = arg s
          by (metis apps.simps(1) apps.simps(2) args_s tm.sel(6) tm_collapse_apps ss)

        have s_app: is_App s
          using args_Nil_iff_is_Hd args_s by force
        have args_u: args u = [arg u]
          by (metis append_Nil args.simps(2) args_Nil_iff_is_Hd gt_unary_u_t(3) length_0_conv
            nargs_fun_u tm.collapse(2) zero_neq_one)

        have max_sz_arg_u_t_arg_t: Max {size (arg t), size t, size (arg u)} < size u
```

```
          using size_arg_lt sz_u_gt_t t_app u_app by fastforce

        have {#size (arg u), size t, size (arg t)#} < {#size u, size t, size s#}
          using max_sz_arg_u_t_arg_t by (auto intro!: Max_lt_imp_lt_mset)
        hence arg_u_gt_arg_t: arg u >_t arg t
          using ih[OF _ wary_arg_u wary_t wary_arg_t] args_Nil_iff_is_Hd gt_arg
            gt_unary_t_s(3) gt_unary_u_t(5) wary_t by force

        have max_sz_arg_s_s_arg_t: Max {size (arg s), size s, size (arg t)} < size u
          using s_app t_app size_arg_lt sz_t_gt_s sz_u_gt_t by force

        have {#size (arg t), size s, size (arg s)#} < {#size u, size t, size s#}
          using max_sz_arg_s_s_arg_t by (auto intro!: Max_lt_imp_lt_mset)
        hence arg_t_gt_arg_s: arg t >_t arg s
          using ih[OF _ wary_arg_t wary_s wary_arg_s]
            gt_unary_t_s(5) gt_arg args_Nil_iff_is_Hd args_s wary_s by force

        have {#size (arg u), size (arg t), size (arg s)#} < {#size u, size t, size s#}
          by (auto intro!: add_mset_lt_lt_lt simp: size_arg_lt u_app t_app s_app)
        hence arg u >_t arg s
          using ih[of arg u arg t arg s] arg_u_gt_arg_t arg_t_gt_arg_s wary_arg_s
            wary_arg_t wary_arg_u by blast
        thus ?thesis
          unfolding args_u args_s ss sa by (metis extf_singleton gt_irrefl wary_arg_u)
      qed

      have ?thesis
        by (rule gt_same[OF wt_u_ge_s hd_u_eq_s]) (simp add: extf)
    }
    moreover
    {
      assume head u >_hd head s
      hence ?thesis
        by (rule gt_diff[OF wt_u_ge_s])
    }
    moreover
    {
      assume head s >_hd head u
      hence False
        using gt_hd_def gt_hd_irrefl gt_sym_antisym gt_unary_u_t(4) unary_wt_sym_0_gt by blast
    }
    moreover
    {
      assume ¬ head u ≤≥_hd head s
      hence ?thesis
        by (rule gt_unary[OF wt_u_ge_s _ gt_unary_u_t(3,4) arg_u_ge_s])
    }
    ultimately have ?thesis
      unfolding comp_hd_def by sat
  }
  ultimately show ?thesis
    by (meson less_le_trans linorder_not_le size_arg_lt t_app u_app)
next
  case gt_diff_u_t: gt_diff
  have False
    using gt_diff_u_t(2) gt_hd_def gt_hd_irrefl gt_sym_antisym gt_unary_t_s(4) unary_wt_sym_0_gt
    by blast
  thus ?thesis
    by sat
next
  case gt_same_u_t: gt_same

  have hd_u_ncomp_s: ¬ head u ≤≥_hd head s
```

by (*rule gt_unary_t_s*(*2*)[*folded gt_same_u_t*(*2*)])

have $\exists f \in$ *ground_heads* (*head u*). *arity_sym f* = *1* $\wedge$ *wt_sym f* = *0*
by (*rule gt_unary_t_s*(*4*)[*folded gt_same_u_t*(*2*)])
hence *arity_hd* (*head u*) = *1*
by (*metis dual_order.order_iff_strict gr_implies_not_zero_hmset ground_heads_arity gt_same_u_t*(*2*) *head_fun hmset_of_enat_1 hmset_of_enat_less lt_1_iff_eq_0_hmset nargs_fun_t*)
hence *num_args u* $\leq$ *1*
using *of_nat_le_hmset wary_num_args_le_arity_head*$_h$ *wary_u* **by** *fastforce*
hence *nargs_u*: *num_args u* = *1*
by (*cases args u*,
metis Hd_head_id δ*_eq_*ε *append_Nil args.simps*(*2*)
ex_in_conv[*THEN iffD2*, *OF ground_heads_nonempty*] *gt_same_u_t*(*2*,*3*) *gt_unary_t_s*(*3*)
head_fun list.size(*3*) *not_extf_gt_nil_singleton_if_*δ_h*_eq_*ε_h *one_arg_imp_Hd*
tm.collapse(*2*)[*OF t_app*] *wary_arg_t*,
simp)
hence *u_app*: *is_App u*
by (*cases u*) *auto*

have *arg u* $>_t$ *arg t*
by (*metis extf_singleton*[*THEN iffD1*] *append_Nil args.simps args_Nil_iff_is_Hd comp_hd_def gt_hd_def gt_irrefl gt_same_u_t*(*2*,*3*) *gt_unary_t_s*(*2*,*3*) *head_fun length_0_conv nargs_u one_arg_imp_Hd t_app tm.collapse*(*2*) *u_gt_t wary_u*)
moreover have {#*size* (*arg u*), *size* (*arg t*), *size s*#} < {#*size u*, *size t*, *size s*#}
by (*auto intro*!: *add_mset_lt_lt_lt simp*: *size_arg_lt u_app t_app*)
ultimately have *arg u* $>_t$ *s*
using *ih*[*OF* _ *wary_arg_u wary_arg_t wary_s*] *gt_unary_t_s*(*5*) **by** *blast*
hence *arg_u_ge_s*: *arg u* $\geq_t$ *s*
by *sat*
show *?thesis*
by (*rule gt_unary*[*OF wt_u_ge_s hd_u_ncomp_s nargs_u* _ *arg_u_ge_s*])
(*simp add*: *gt_same_u_t*(*2*) *gt_unary_t_s*(*4*))
qed
next
case *gt_diff_t_s*: *gt_diff*
show *?thesis*
using *u_gt_t*
proof *cases*
case *gt_wt_u_t*: *gt_wt*
hence *wt u* $>_p$ *wt s*
using *wt_t_ge_s gt_ge_tpoly_trans* **by** *blast*
thus *?thesis*
by (*rule gt_wt*)
next
case *gt_unary_u_t*: *gt_unary*
have *u_app*: *is_App u*
by (*metis args_Nil_iff_is_Hd gt_unary_u_t*(*3*) *length_greater_0_conv less_numeral_extra*(*1*))
hence *arg u* $>_t$ *s*
using *ih*[*of arg u t s*] *gt_unary_u_t*(*5*) *t_gt_s size_arg_lt wary_arg_u wary_s wary_t*
by *force*
hence *arg_u_ge_s*: *arg u* $\geq_t$ *s*
by *sat*

{
assume *head u* = *head s*
hence *False*
using *gt_diff_t_s*(*2*) *gt_unary_u_t*(*2*) **unfolding** *comp_hd_def* **by** *force*
}
moreover
{
assume *head s* $>_{hd}$ *head u*
hence *False*

```
        using gt_hd_def gt_hd_irrefl gt_sym_antisym gt_unary_u_t(4) unary_wt_sym_0_gt by blast
      }
      moreover
      {
        assume head u >_hd head s
        hence ?thesis
          by (rule gt_diff[OF wt_u_ge_s])
      }
      moreover
      {
        assume ¬ head u ≤≥_hd head s
        hence ?thesis
          by (rule gt_unary[OF wt_u_ge_s _ gt_unary_u_t(3,4) arg_u_ge_s])
      }
      ultimately show ?thesis
        unfolding comp_hd_def by sat
    next
      case gt_diff_u_t: gt_diff
      have head u >_hd head s
        using gt_diff_u_t(2) gt_diff_t_s(2) gt_hd_trans by blast
      thus ?thesis
        by (rule gt_diff[OF wt_u_ge_s])
    next
      case gt_same_u_t: gt_same
      have head u >_hd head s
        using gt_diff_t_s(2) gt_same_u_t(2) by simp
      thus ?thesis
        by (rule gt_diff[OF wt_u_ge_s])
    qed
  next
    case gt_same_t_s: gt_same
    show ?thesis
      using u_gt_t
    proof cases
      case gt_wt_u_t: gt_wt
      hence wt u >_p wt s
        using wt_t_ge_s gt_ge_tpoly_trans by blast
      thus ?thesis
        by (rule gt_wt)
    next
      case gt_unary_u_t: gt_unary
      have is_App u
        by (metis args_Nil_iff_is_Hd gt_unary_u_t(3) length_greater_0_conv less_numeral_extra(1))
      hence arg u >_t s
        using ih[of arg u t s] gt_unary_u_t(5) t_gt_s size_arg_lt wary_arg_u wary_s wary_t
        by force
      hence arg_u_ge_s: arg u ≥_t s
        by sat

      have ¬ head u ≤≥_hd head s
        using gt_same_t_s(2) gt_unary_u_t(2) by simp
      thus ?thesis
        by (rule gt_unary[OF wt_u_ge_s _ gt_unary_u_t(3,4) arg_u_ge_s])
    next
      case gt_diff_u_t: gt_diff
      have head u >_hd head s
        using gt_diff_u_t(2) gt_same_t_s(2) by simp
      thus ?thesis
        by (rule gt_diff[OF wt_u_ge_s])
    next
      case gt_same_u_t: gt_same
      have hd_u_s: head u = head s
        by (simp only: gt_same_t_s(2) gt_same_u_t(2))
```

```
      let ?S = set (args u) ∪ set (args t) ∪ set (args s)

      have gt_trans_args: ∀ua ∈ ?S. ∀ta ∈ ?S. ∀sa ∈ ?S. ua >_t ta ⟶ ta >_t sa ⟶ ua >_t sa
      proof clarify
        fix sa ta ua
        assume
          ua_in: ua ∈ ?S and ta_in: ta ∈ ?S and sa_in: sa ∈ ?S and
          ua_gt_ta: ua >_t ta and ta_gt_sa: ta >_t sa
        have wary_sa: wary sa and wary_ta: wary ta and wary_ua: wary ua
          using wary_args ua_in ta_in sa_in wary_u wary_t wary_s by blast+
        show ua >_t sa
          by (auto intro!: ih[OF Max_lt_imp_lt_mset wary_ua wary_ta wary_sa ua_gt_ta ta_gt_sa])
            (meson ua_in ta_in sa_in Un_iff max.strict_coboundedI1 max.strict_coboundedI2
              size_in_args)+
      qed
      have ∀f ∈ ground_heads (head u). extf f (>_t) (args u) (args s)
        by (clarify, rule extf_trans_from_irrefl[of ?S _ args t, OF _ _ _ _ _ _ gt_trans_args])
          (auto simp: gt_same_u_t(2,3) gt_same_t_s(3) wary_args wary_u wary_t wary_s gt_irrefl)
      thus ?thesis
        by (rule gt_same[OF wt_u_ge_s hd_u_s])
    qed
  qed
qed

lemma gt_antisym: wary s ⟹ wary t ⟹ t >_t s ⟹ ¬ s >_t t
  using gt_irrefl gt_trans by blast
```

6.6 Subterm Property

```
lemma gt_sub_fun: App s t >_t s
proof (cases wt (App s t) >_p wt s)
  case True
  thus ?thesis
    using gt_wt by simp
next
  case False
  hence δ_eq_ε: δ_h = ε_h
    using wt_App_ge_fun dual_order.order_iff_strict wt_App_arg_δ_h wt_δ_h_imp_δ_h_eq_ε_h
    unfolding gt_tpoly_def ge_tpoly_def by fast

  have hd_st: head (App s t) = head s
    by auto
  have extf: ∀f ∈ ground_heads (head (App s t)). extf f (>_t) (args (App s t)) (args s)
    by (simp add: δ_eq_ε extf_snoc_if_δ_h_eq_ε_h)
  show ?thesis
    by (rule gt_same[OF wt_App_ge_fun hd_st extf])
qed

theorem gt_proper_sub: wary t ⟹ proper_sub s t ⟹ t >_t s
  by (induct t) (auto intro: gt_sub_fun gt_sub_arg gt_trans sub.intros wary_sub)
```

6.7 Compatibility with Functions

```
lemma gt_compat_fun:
  assumes
    wary_t: wary t and
    t'_gt_t: t' >_t t
  shows App s t' >_t App s t
proof (rule gt_same; clarify?)
  show wt (App s t') ≥_p wt (App s t)
    using gt_imp_wt[OF t'_gt_t, unfolded ge_tpoly_def]
    by (cases s rule: tm_exhaust_apps,
      auto simp del: apps_append simp: ge_tpoly_def App_apps eval_ztpoly_nonneg
```

intro: ordered_comm_semiring_class.comm_mult_left_mono)
next
fix f
have extf f ($>_t$) (args s @ [t′]) (args s @ [t])
using t′_gt_t **by** (metis extf_compat_list gt_irrefl[OF wary_t])
thus extf f ($>_t$) (args (App s t′)) (args (App s t))
by simp
qed simp

theorem gt_compat_fun_strong:
assumes
wary_t: wary t **and**
t′_gt_t: t′ $>_t$ t
shows apps s (t′ # us) $>_t$ apps s (t # us)
proof (induct us rule: rev_induct)
case Nil
show ?case
using t′_gt_t **by** (auto intro!: gt_compat_fun[OF wary_t])
next
case (snoc u us)
note ih = snoc

let ?v′ = apps s (t′ # us @ [u])
let ?v = apps s (t # us @ [u])

have wt ?v′ $\geq_p$ wt ?v
using gt_imp_wt[OF ih]
by (cases s rule: tm_exhaust_apps,
simp del: apps_append add: App_apps apps_append[symmetric] ge_tpoly_def,
subst (1 2) zip_eq_butlast_last, simp+)
moreover have head ?v′ = head ?v
by simp
moreover have ∀f ∈ ground_heads (head ?v′). extf f ($>_t$) (args ?v′) (args ?v)
by (metis args_apps extf_compat_list gt_irrefl[OF wary_t] t′_gt_t)
ultimately show ?case
by (rule gt_same)
qed

6.8 Compatibility with Arguments

theorem gt_compat_arg_weak:
assumes
wary_st: wary (App s t) **and**
wary_s′t: wary (App s′ t) **and**
coef_s′_0_ge_s: coef s′ 0 $\geq_p$ coef s 0 **and**
s′_gt_s: s′ $>_t$ s
shows App s′ t $>_t$ App s t
proof −
obtain ζ ss **where** s: s = apps (Hd ζ) ss
by (metis tm_exhaust_apps)
obtain ζ′ ss′ **where** s′: s′ = apps (Hd ζ′) ss′
by (metis tm_exhaust_apps)

have len_ss_lt: of_nat (length ss) < $arity\_sym_h$ (min_ground_head ζ)
using wary_st[unfolded s] $ground\_heads\_arity_h$ less_le_trans min_ground_head_in_ground_heads
by (metis (no_types) tm_collapse_apps tm_inject_apps $wary\_AppE_h$)

have δ_etc:
$\delta_h + \delta_h * (arity\_sym_h$ (min_ground_head ζ) − of_nat (length ss) − 1) =
$\delta_h * (arity\_sym_h$ (min_ground_head ζ) − of_nat (length ss))
if wary: wary (App (apps (Hd ζ) ss) t) **for** ζ ss
proof (cases $\delta_h > 0$)
case True
then obtain n **where** n: of_nat n = $arity\_sym_h$ (min_ground_head ζ)

```
    by (metis arity_sym_h_if_δ_h_gt_0_E)

  have of_nat (length ss) < arity_sym_h (min_ground_head ζ)
    using wary
    by (metis (no_types) wary_AppE_h ground_heads_arity_h le_less_trans
      min_ground_head_in_ground_heads not_le tm_collapse_apps tm_inject_apps)
  thus ?thesis
    by (fold n, subst of_nat_1[symmetric], fold of_nat_minus_hmset, simp,
      metis Suc_diff_Suc mult_Suc_right of_nat_add of_nat_mult)
qed simp

have coef_ζ'_ge_ζ: coef_hd ζ' (length ss') ≥_p coef_hd ζ (length ss)
  by (rule coef_s'_0_ge_s[unfolded s s', simplified])

have wt_s'_ge_s: wt s' ≥_p wt s
  by (rule gt_imp_wt[OF s'_gt_s])

have ζ_tms_len_ss_tms_wt_t_le:
  eval_ztpoly A (coef_hd ζ (length ss)) ∗ eval_ztpoly A (wt t)
   ≤ eval_ztpoly A (coef_hd ζ' (length ss')) ∗ eval_ztpoly A (wt t)
  if legal: legal_zpassign A for A
  using legal coef_ζ'_ge_ζ[unfolded ge_tpoly_def]
  by (simp add: eval_ztpoly_nonneg mult_right_mono)

have wt_s't_ge_st: wt (App s' t) ≥_p wt (App s t)
  unfolding s s'
  by (clarsimp simp del: apps_append simp: App_apps ge_tpoly_def add_ac(1)[symmetric]
       intro!: add_mono[OF _ ζ_tms_len_ss_tms_wt_t_le],
    rule add_le_imp_le_left[of zhmset_of δ_h],
    unfold add_ac(1)[symmetric] add.commute[of 1] diff_diff_add[symmetric],
    subst (1 3) ac_simps(3)[unfolded add_ac(1)[symmetric]], subst (1 3) add_ac(1),
    simp only: zhmset_of_plus[symmetric] δ_etc[OF wary_st[unfolded s]]
      δ_etc[OF wary_s't[unfolded s']] add_ac(1)
      wt_s'_ge_s[unfolded s s', unfolded ge_tpoly_def add_ac(1)[symmetric], simplified])
show ?thesis
  using s'_gt_s
proof cases
  case gt_wt_s'_s: gt_wt

  have wt (App s' t) >_p wt (App s t)
    unfolding s s'
    by (clarsimp simp del: apps_append simp: App_apps gt_tpoly_def add_ac(1)[symmetric]
         intro!: add_less_le_mono[OF _ ζ_tms_len_ss_tms_wt_t_le],
      rule add_less_imp_less_left[of zhmset_of δ_h],
      unfold add_ac(1)[symmetric] add.commute[of 1] diff_diff_add[symmetric],
      subst (1 3) ac_simps(3)[unfolded add_ac(1)[symmetric]],
      subst (1 3) add_ac(1),
      simp only: zhmset_of_plus[symmetric] δ_etc[OF wary_st[unfolded s]]
        δ_etc[OF wary_s't[unfolded s']] add_ac(1)
        gt_wt_s'_s[unfolded s s', unfolded gt_tpoly_def add_ac(1)[symmetric], simplified])
  thus ?thesis
    by (rule gt_wt)
next
  case gt_unary_s'_s: gt_unary
  have False
    by (metis ground_heads_arity_h gt_unary_s'_s(3) gt_unary_s'_s(4) hmset_of_enat_1 leD of_nat_1
      wary_AppE_h wary_s't)
  thus ?thesis
    by sat
next
  case gt_diff_s'_s: gt_diff
  show ?thesis
    by (rule gt_diff[OF wt_s't_ge_st]) (simp add: gt_diff_s'_s(2))
```

```
  next
    case gt_same_s′_s: gt_same
    have hd_s′t: head (App s′ t) = head (App s t)
      by (simp add: gt_same_s′_s(2))
    have ∀f ∈ ground_heads (head (App s′ t)). extf f (>_t) (args (App s′ t)) (args (App s t))
      using gt_same_s′_s(3) by (auto intro: extf_compat_append_right)
    thus ?thesis
      by (rule gt_same[OF wt_s′t_ge_st hd_s′t])
  qed
qed
```

6.9 Stability under Substitution

```
primrec
  subst_zpassign :: ('v ⇒ ('s, 'v) tm) ⇒ ('v pvar ⇒ zhmultiset) ⇒ 'v pvar ⇒ zhmultiset
where
  subst_zpassign ϱ A (PWt x) =
   eval_ztpoly A (wt (ϱ x)) − zhmset_of (δ_h ∗ arity_sym_h (min_ground_head (Var x)))
| subst_zpassign ϱ A (PCoef x i) = eval_ztpoly A (coef (ϱ x) i)

lemma legal_subst_zpassign:
  assumes
    legal: legal_zpassign A and
    wary_ϱ: wary_subst ϱ
  shows legal_zpassign (subst_zpassign ϱ A)
  unfolding legal_zpassign_def
proof
  fix v
  show subst_zpassign ϱ A v ≥ min_zpassign v
  proof (cases v)
    case v: (PWt x)
    obtain ζ ss where ϱx: ϱ x = apps (Hd ζ) ss
      by (rule tm_exhaust_apps)

    have ghd_ζ: ground_heads ζ ⊆ ground_heads_var x
      using wary_ϱ[unfolded wary_subst_def, rule_format, of x, unfolded ϱx] by simp

    have zhmset_of (wt_sym (min_ground_head (Var x)) + δ_h ∗ arity_sym_h (min_ground_head (Var x)))
      ≤ eval_ztpoly A (wt0 ζ) + zhmset_of (δ_h ∗ arity_sym_h (min_ground_head ζ))
    proof −
      have mgh_x_min:
        zhmset_of (wt_sym (min_ground_head (Var x)) + δ_h ∗ arity_sym_h (min_ground_head (Var x)))
         ≤ zhmset_of (wt_sym (min_ground_head ζ) + δ_h ∗ arity_sym_h (min_ground_head ζ))
        by (simp add: zmset_of_le zhmset_of_le ghd_ζ min_ground_head_antimono)
      have wt_mgh_le_wt0: zhmset_of (wt_sym (min_ground_head ζ)) ≤ eval_ztpoly A (wt0 ζ)
        using wt0_ge_min_ground_head[OF legal] by blast
      show ?thesis
        by (rule order_trans[OF mgh_x_min]) (simp add: zhmset_of_plus wt_mgh_le_wt0)
    qed
    also have ... ≤ eval_ztpoly A (wt0 ζ)
      + zhmset_of ((δ_h ∗ (arity_sym_h (min_ground_head ζ) − of_nat (length ss)))
        + of_nat (length ss) ∗ δ_h)
    proof −
      have zhmset_of (δ_h ∗ arity_sym_h (min_ground_head ζ))
        ≤ zhmset_of (δ_h ∗ (of_nat (length ss)
          + (arity_sym_h (min_ground_head ζ) − of_nat (length ss))))
        by (metis add.commute le_minus_plus_same_hmset mult_le_mono2_hmset zhmset_of_le)
      thus ?thesis
        by (simp add: add.commute add.left_commute distrib_left mult.commute)
    qed
    also have ... ≤ eval_ztpoly A (wt0 ζ)
      + zhmset_of ((δ_h ∗ (arity_sym_h (min_ground_head ζ) − of_nat (length ss)))
        + of_nat (length ss) ∗ ε_h)
      using δ_h_le_ε_h zhmset_of_le by auto
```

```
    also have ... ≤ eval_ztpoly A (wt0 ζ)
      + zhmset_of (δ_h * (arity_sym_h (min_ground_head ζ) − of_nat (length ss))) + wt_args 0 A ζ ss
      using wt_args_ge_length_times_ε_h[OF legal]
      by (simp add: algebra_simps zhmset_of_plus zhmset_of_times of_nat_zhmset)
    finally have wt_x_le_ζssts:
      zhmset_of (wt_sym (min_ground_head (Var x)) + δ_h * arity_sym_h (min_ground_head (Var x)))
       ≤ eval_ztpoly A (wt0 ζ)
         + zhmset_of (δ_h * (arity_sym_h (min_ground_head ζ) − of_nat (length ss)))
         + wt_args 0 A ζ ss
      by assumption

    show ?thesis
      using wt_x_le_ζssts[unfolded wt_args_def]
      by (simp add: v ϱx comp_def le_diff_eq add.assoc[symmetric] ZHMSet_plus[symmetric]
        zmset_of_plus[symmetric] hmsetmset_plus[symmetric] zmset_of_le)
  next
    case (PCoef x i)
    thus ?thesis
      using coef_gt_0[OF legal, unfolded zero_less_iff_1_le_hmset]
      by (simp add: zhmset_of_1 zero_less_iff_1_le_zhmset)
  qed
qed

lemma wt_subst:
  assumes
    legal: legal_zpassign A and
    wary_ϱ: wary_subst ϱ
  shows wary s ⟹ eval_ztpoly A (wt (subst ϱ s)) = eval_ztpoly (subst_zpassign ϱ A) (wt s)
proof (induct s rule: tm_induct_apps)
  case (apps ζ ss)
  note ih = this(1) and wary_ζss = this(2)

  have wary_nth_ss: ⋀i. i < length ss ⟹ wary (ss ! i)
    using wary_args[OF _ wary_ζss] by force

  show ?case
  proof (cases ζ)
    case ζ: (Var x)
    show ?thesis
    proof (cases ϱ x rule: tm_exhaust_apps)
      case ϱx: (apps ξ ts)

      have wary_ϱx: wary (ϱ x)
        using wary_ϱ wary_subst_def by blast

      have coef_subst: ⋀i. eval_tpoly A (zhmset_of_tpoly (coef_hd ξ (i + length ts))) =
        eval_tpoly (subst_zpassign ϱ A) (zhmset_of_tpoly (coef_hd (Var x) i))
        by (simp add: ϱx)

      have tedious_ary_arith:
        arity_sym_h (min_ground_head (Var x))
         + (arity_sym_h (min_ground_head ξ) − (of_nat (length ss) + of_nat (length ts))) =
         arity_sym_h (min_ground_head ξ) − of_nat (length ts)
         + (arity_sym_h (min_ground_head (Var x)) − of_nat (length ss))
        if δ_gt_0: δ_h > 0
      proof −
        obtain m where m: of_nat m = arity_sym_h (min_ground_head (Var x))
          by (metis arity_sym_h_if_δ_h_gt_0_E[OF δ_gt_0])
        obtain n where n: of_nat n = arity_sym_h (min_ground_head ξ)
          by (metis arity_sym_h_if_δ_h_gt_0_E[OF δ_gt_0])

        have m ≥ length ss
          unfolding of_nat_le_hmset[symmetric] m using wary_ζss[unfolded ζ]
```

```
      by (cases rule: wary_cases_apps_h, clarsimp,
        metis arity_hd.simps(1) enat_ile enat_ord_simps(1) ground_heads_arity
          hmset_of_enat_inject hmset_of_enat_of_nat le_trans m min_ground_head_in_ground_heads
          of_nat_eq_enat of_nat_le_hmset_of_enat_iff)
    moreover have n_ge_len_ss_ts: n ≥ length ss + length ts
    proof −
      have of_nat (length ss) + of_nat (length ts) ≤ arity_hd_h ζ + of_nat (length ts)
        using wary_ζss wary_cases_apps_h by fastforce
      also have ... = arity_var_h x + of_nat (length ts)
        by (simp add: ζ)
      also have ... ≤ arity_h (ϱ x) + of_nat (length ts)
        using wary_ϱ wary_subst_def by auto
      also have ... = arity_h (apps (Hd ξ) ts) + of_nat (length ts)
        by (simp add: ϱx)
      also have ... = arity_hd_h ξ
        using wary_ϱx[unfolded ϱx]
        by (cases rule: wary_cases_apps_h, cases arity_hd ξ,
          simp add: of_nat_add[symmetric] of_nat_minus_hmset[symmetric],
          metis δ_gt_0 arity_hd_ne_infinity_if_δ_gt_0 of_nat_0 of_nat_less_hmset)
      also have ... ≤ arity_sym_h (min_ground_head ξ)
        using ground_heads_arity_h min_ground_head_in_ground_heads by blast
      finally show ?thesis
        unfolding of_nat_le_hmset[symmetric] n by simp
    qed
    moreover have n ≥ length ts
      using n_ge_len_ss_ts by simp
    ultimately show ?thesis
      by (fold m n of_nat_add of_nat_minus_hmset, unfold of_nat_inject_hmset, fastforce)
  qed

  have eval_tpoly A (zhmset_of_tpoly (wt (subst ϱ (apps (Hd (Var x)) ss)))) =
    eval_tpoly A (zhmset_of_tpoly (wt0 ξ))
    + zhmset_of (δ_h * (arity_sym_h (min_ground_head ξ)
      − (of_nat (length ts) + of_nat (length ss))))
    + wt_args 0 A ξ (ts @ map (subst ϱ) ss)
    by (simp del: apps_append add: apps_append[symmetric] ϱx wt_args_def comp_def)
  also have ... = eval_tpoly A (zhmset_of_tpoly (wt0 ξ))
    + zhmset_of (δ_h * (arity_sym_h (min_ground_head ξ)
      − (of_nat (length ts) + of_nat (length ss))))
    + wt_args 0 A ξ ts + wt_args (length ts) A ξ (map (subst ϱ) ss)
    by (simp add: wt_args_def zip_append_0_upt[of ts map (subst ϱ) ss, simplified])
  also have ... = eval_tpoly A (zhmset_of_tpoly (wt0 ξ))
    + zhmset_of (δ_h * (arity_sym_h (min_ground_head ξ)
      − (of_nat (length ts) + of_nat (length ss))))
    + wt_args 0 A ξ ts + wt_args 0 (subst_zpassign ϱ A) (Var x) ss
    by (auto intro!: arg_cong[of _ _ sum_list] nth_map_conv
      simp: wt_args_def coef_subst add.commute zhmset_of_times ih[OF nth_mem wary_nth_ss])
  also have ... = eval_tpoly (subst_zpassign ϱ A) (zhmset_of_tpoly (wt0 (Var x)))
    + zhmset_of (δ_h * (arity_sym_h (min_ground_head (Var x)) − of_nat (length ss)))
    + wt_args 0 (subst_zpassign ϱ A) (Var x) ss
    by (simp add: ϱx wt_args_def comp_def algebra_simps ring_distribs(1)[symmetric]
          zhmset_of_times zhmset_of_plus[symmetric] zhmset_of_0[symmetric])
      (use tedious_ary_arith in fastforce)
  also have ... = eval_tpoly (subst_zpassign ϱ A) (zhmset_of_tpoly (wt (apps (Hd (Var x)) ss)))
    by (simp add: wt_args_def comp_def)
  finally show ?thesis
    unfolding ζ by assumption
 qed
next
 case ζ: (Sym f)

 have eval_tpoly A (zhmset_of_tpoly (wt (subst ϱ (apps (Hd (Sym f)) ss)))) =
   zhmset_of (wt_sym f) + zhmset_of (δ_h * (arity_sym_h f − of_nat (length ss)))
```

```
      + wt_args 0 A (Sym f) (map (subst ϱ) ss)
      by (simp add: wt_args_def comp_def)
    also have ... = zhmset_of (wt_sym f) + zhmset_of (δ_h ∗ (arity_sym_h f − of_nat (length ss)))
      + wt_args 0 (subst_zpassign ϱ A) (Sym f) ss
      by (auto simp: wt_args_def ih[OF _ wary_nth_ss] intro!: arg_cong[of _ _ sum_list]
        nth_map_conv)
    also have ... = eval_tpoly (subst_zpassign ϱ A) (zhmset_of_tpoly (wt (apps (Hd (Sym f)) ss)))
      by (simp add: wt_args_def comp_def)
    finally show ?thesis
      unfolding ζ by assumption
  qed
qed

theorem gt_subst:
  assumes wary_ϱ: wary_subst ϱ
  shows wary t ⟹ wary s ⟹ t >_t s ⟹ subst ϱ t >_t subst ϱ s
proof (simp only: atomize_imp,
    rule measure_induct_rule[of λ(t, s). {#size t, size s#}
        λ(t, s). wary t ⟶ wary s ⟶ t >_t s ⟶ subst ϱ t >_t subst ϱ s (t, s),
      simplified prod.case],
    simp only: split_paired_all prod.case atomize_imp[symmetric])
  fix t s
  assume
    ih: ⋀ta sa. {#size ta, size sa#} < {#size t, size s#} ⟹ wary ta ⟹ wary sa ⟹ ta >_t sa ⟹
      subst ϱ ta >_t subst ϱ sa and
    wary_t: wary t and wary_s: wary s and t_gt_s: t >_t s

  show subst ϱ t >_t subst ϱ s
    using t_gt_s
  proof cases
    case gt_wt_t_s: gt_wt

    have wt (subst ϱ t) >_p wt (subst ϱ s)
      by (auto simp: gt_tpoly_def wary_s wary_t wt_subst[OF _ wary_ϱ]
        intro: gt_wt_t_s[unfolded gt_tpoly_def, rule_format]
        elim: legal_subst_zpassign[OF _ wary_ϱ])
    thus ?thesis
      by (rule gt_wt)
  next
    assume wt_t_ge_s: wt t ≥_p wt s

    have wt_ϱt_ge_ϱs: wt (subst ϱ t) ≥_p wt (subst ϱ s)
      by (auto simp: ge_tpoly_def wary_s wary_t wt_subst[OF _ wary_ϱ]
        intro: wt_t_ge_s[unfolded ge_tpoly_def, rule_format]
        elim: legal_subst_zpassign[OF _ wary_ϱ])

    {
      case gt_unary

      have wary_ϱt: wary (subst ϱ t)
        by (simp add: wary_subst_wary wary_t wary_ϱ)

      show ?thesis
      proof (cases t)
        case Hd
        hence False
          using gt_unary(3) by simp
        thus ?thesis
          by sat
      next
        case t: (App t1 t2)
        hence t2: t2 = arg t
          by simp
```

```
    hence wary_t2: wary t2
      using wary_t by blast

    show ?thesis
    proof (cases t2 = s)
      case True
      moreover have subst ϱ t >_t subst ϱ t2
        using gt_sub_arg wary_ϱt unfolding t by simp
      ultimately show ?thesis
        by simp
    next
      case t2_ne_s: False
      hence t2_gt_s: t2 >_t s
        using gt_unary(5) t2 by blast

      have subst ϱ t2 >_t subst ϱ s
        by (rule ih[OF _ wary_t2 wary_s t2_gt_s]) (simp add: t)
      thus ?thesis
        by (metis gt_sub_arg gt_trans subst.simps(2) t wary_ϱ wary_ϱt wary_s wary_subst_wary
          wary_t2)
    qed
  qed
}
{
  case _: gt_diff
  note hd_t_gt_hd_s = this(2)

  have head (subst ϱ t) >_hd head (subst ϱ s)
    by (meson hd_t_gt_hd_s wary_subst_ground_heads gt_hd_def rev_subsetD wary_ϱ)
  thus ?thesis
    by (rule gt_diff[OF wt_ϱt_ge_ϱs])
}
{
  case _: gt_same
  note hd_s_eq_hd_t = this(2) and extf = this(3)

  have hd_ϱt: head (subst ϱ t) = head (subst ϱ s)
    by (simp add: hd_s_eq_hd_t)

  {
    fix f
    assume f_in_grs: f ∈ ground_heads (head (subst ϱ t))

    let ?S = set (args t) ∪ set (args s)

    have extf_args_s_t: extf f (>_t) (args t) (args s)
      using extf f_in_grs wary_subst_ground_heads wary_ϱ by blast
    have extf f (>_t) (map (subst ϱ) (args t)) (map (subst ϱ) (args s))
    proof (rule extf_map[of ?S, OF _ _ _ _ _ _ extf_args_s_t])
      show ∀ x ∈ ?S. ¬ subst ϱ x >_t subst ϱ x
        using gt_irrefl wary_t wary_s wary_args wary_ϱ wary_subst_wary by fastforce
    next
      show ∀ z ∈ ?S. ∀ y ∈ ?S. ∀ x ∈ ?S. subst ϱ z >_t subst ϱ y ⟶ subst ϱ y >_t subst ϱ x ⟶
        subst ϱ z >_t subst ϱ x
        using gt_trans wary_t wary_s wary_args wary_ϱ wary_subst_wary by (metis Un_iff)
    next
      have sz_a: ∀ ta ∈ ?S. ∀ sa ∈ ?S. {#size ta, size sa#} < {#size t, size s#}
        by (fastforce intro: Max_lt_imp_lt_mset dest: size_in_args)
      show ∀ y ∈ ?S. ∀ x ∈ ?S. y >_t x ⟶ subst ϱ y >_t subst ϱ x
        using ih sz_a size_in_args wary_t wary_s wary_args wary_ϱ wary_subst_wary by fastforce
    qed auto
    hence extf f (>_t) (args (subst ϱ t)) (args (subst ϱ s))
      by (auto simp: hd_s_eq_hd_t intro: extf_compat_append_left)
```

```
      }
      hence ∀f ∈ ground_heads (head (subst ϱ t)).
        extf f (>t) (args (subst ϱ t)) (args (subst ϱ s))
        by blast
      thus ?thesis
        by (rule gt_same[OF wt_ϱt_ge_ϱs hd_ϱt])
    }
  qed
qed
```

6.10 Totality on Ground Terms

```
lemma wt_total_ground:
  assumes
    gr_t: ground t and
    gr_s: ground s
  shows wt t >p wt s ∨ wt s >p wt t ∨ wt t =p wt s
  unfolding gt_tpoly_def eq_tpoly_def
  by (subst (1 2 3) ground_eval_ztpoly_wt_eq[OF gr_t, of _ undefined],
    subst (1 2 3) ground_eval_ztpoly_wt_eq[OF gr_s, of _ undefined], auto)


theorem gt_total_ground:
  assumes extf_total: ⋀f. ext_total (extf f)
  shows ground t ⟹ ground s ⟹ t >t s ∨ s >t t ∨ t = s
proof (simp only: atomize_imp,
    rule measure_induct_rule[of λ(t, s). {# size t, size s #}
      λ(t, s). ground t ⟶ ground s ⟶ t >t s ∨ s >t t ∨ t = s (t, s), simplified prod.case],
    simp only: split_paired_all prod.case atomize_imp[symmetric])
  fix t s :: ('s, 'v) tm
  assume
    ih: ⋀ta sa. {# size ta, size sa #} < {# size t, size s #} ⟹ ground ta ⟹ ground sa ⟹
      ta >t sa ∨ sa >t ta ∨ ta = sa and
    gr_t: ground t and gr_s: ground s

  let ?case = t >t s ∨ s >t t ∨ t = s

  {
    assume wt t >p wt s
    hence t >t s
      by (rule gt_wt)
  }
  moreover
  {
    assume wt s >p wt t
    hence s >t t
      by (rule gt_wt)
  }
  moreover
  {
    assume wt t =p wt s
    hence wt_t_ge_s: wt t ≥p wt s and wt_s_ge_t: wt s ≥p wt t
      by (simp add: eq_tpoly_def ge_tpoly_def)+

    obtain g where ξ: head t = Sym g
      by (metis ground_head[OF gr_t] hd.collapse(2))
    obtain f where ζ: head s = Sym f
      by (metis ground_head[OF gr_s] hd.collapse(2))

    {
      assume g_gt_f: g >s f
      have t >t s
        by (rule gt_diff[OF wt_t_ge_s]) (simp add: ξ ζ g_gt_f gt_hd_def)
    }
```

```
    moreover
    {
      assume f_gt_g: f >_s g
      have s >_t t
        by (rule gt_diff[OF wt_s_ge_t]) (simp add: ξ ζ f_gt_g gt_hd_def)
    }
    moreover
    {
      assume g_eq_f: g = f
      hence hd_t: head t = head s
        using ξ ζ by force
      note hd_s = hd_t[symmetric]

      let ?ts = args t
      let ?ss = args s

      have gr_ts: ∀ t ∈ set ?ts. ground t
        using gr_t ground_args by auto
      have gr_ss: ∀ s ∈ set ?ss. ground s
        using gr_s ground_args by auto

      have ?case
      proof (cases ?ts = ?ss)
        case ts_eq_ss: True
        show ?thesis
          using ξ ζ g_eq_f ts_eq_ss by (simp add: tm_expand_apps)
      next
        case False
        hence extf g (>_t) ?ts ?ss ∨ extf g (>_t) ?ss ?ts
          using ih gr_ss gr_ts less_multiset_doubletons
            ext_total.total[OF extf_total, rule_format, of set ?ts ∪ set ?ss (>_t) ?ts ?ss g]
          by (metis Un_commute Un_iff in_lists_iff_set size_in_args sup_ge2)
        moreover
        {
          assume extf: extf g (>_t) ?ts ?ss
          have t >_t s
            by (rule gt_same[OF wt_t_ge_s hd_t]) (simp add: extf ξ)
        }
        moreover
        {
          assume extf: extf g (>_t) ?ss ?ts
          have s >_t t
            by (rule gt_same[OF wt_s_ge_t hd_s]) (simp add: extf[unfolded g_eq_f] ζ)
        }
        ultimately show ?thesis
          by sat
      qed
    }
    ultimately have ?case
      using gt_sym_total by blast
  }
  ultimately show ?case
    using wt_total_ground[OF gr_t gr_s] by fast
qed
```

6.11 Well-foundedness

```
abbreviation gtw :: ('s, 'v) tm ⇒ ('s, 'v) tm ⇒ bool (infix ‹>_tw› 50) where
  (>_tw) ≡ λt s. wary t ∧ wary s ∧ t >_t s

abbreviation gtwg :: ('s, 'v) tm ⇒ ('s, 'v) tm ⇒ bool (infix ‹>_twg› 50) where
  (>_twg) ≡ λt s. ground t ∧ t >_tw s

lemma ground_gt_unary:
```

```
  assumes gr_t: ground t
  shows ¬ gt_unary t s
proof
  assume gt_unary_t_s: gt_unary t s
  hence t >_t s
    using gt_iff_wt_unary_diff_same by blast
  hence gr_s: ground s
    using gr_t gt_imp_vars by blast

  have ngr_t_or_s: ¬ ground t ∨ ¬ ground s
    using gt_unary_t_s by cases (blast dest: ground_head not_comp_hd_imp_Var)

  show False
    using gr_t gr_s ngr_t_or_s by sat
qed

theorem gt_wf: wfP (λs t. t >_tw s)
proof −
  have ground_wfP: wfP (λs t. t >_twg s)
    unfolding wfP_iff_no_inf_chain
  proof
    assume ∃f. inf_chain (>_twg) f
    then obtain t where t_bad: bad (>_twg) t
      unfolding inf_chain_def bad_def by blast

    let ?ff = worst_chain (>_twg) (λt s. size t > size s)
    let ?A = min_passign

    note wf_sz = wf_app[OF wellorder_class.wf, of size, simplified]

    have ffi_ground: ⋀i. ground (?ff i) and ffi_wary: ⋀i. wary (?ff i)
      using worst_chain_bad[OF wf_sz t_bad, unfolded inf_chain_def] by fast+

    have inf_chain (>_twg) ?ff
      by (rule worst_chain_bad[OF wf_sz t_bad])
    hence bad_wt_diff_same:
      inf_chain (λt s. ground t ∧ (gt_wt t s ∨ gt_diff t s ∨ gt_same t s)) ?ff
      unfolding inf_chain_def using gt_iff_wt_unary_diff_same ground_gt_unary by blast

    have wf_wt: wf {(s, t). ground t ∧ gt_wt t s}
      by (rule wf_subset[OF wf_app[of _ eval_tpoly ?A ∘ wt, OF wf_less_hmultiset]],
        simp add: gt_wt.simps gt_tpoly_def, fold zhmset_of_less,
        auto simp: legal_min_zpassign gt_wt.simps gt_tpoly_def)

    have wt_O_diff_same: {(s, t). ground t ∧ gt_wt t s}
        O {(s, t). ground t ∧ wt t =_p wt s ∧ (gt_diff t s ∨ gt_same t s)}
      ⊆ {(s, t). ground t ∧ gt_wt t s}
      unfolding gt_wt.simps gt_diff.simps gt_same.simps by (auto intro: ge_gt_tpoly_trans)

    have wt_diff_same_as_union:
      {(s, t). ground t ∧ (gt_wt t s ∨ gt_diff t s ∨ gt_same t s)} =
       {(s, t). ground t ∧ gt_wt t s}
       ∪ {(s, t). ground t ∧ wt t =_p wt s ∧ (gt_diff t s ∨ gt_same t s)}
      using gt_ge_tpoly_trans gt_tpoly_irrefl wt_ge_vars wt_total_ground
      by (fastforce simp: gt_wt.simps gt_diff.simps gt_same.simps)

    obtain k1 where bad_diff_same:
      inf_chain (λt s. ground t ∧ wt t =_p wt s ∧ (gt_diff t s ∨ gt_same t s)) (λi. ?ff (i + k1))
      using wf_infinite_down_chain_compatible[OF wf_wt _ wt_O_diff_same, of ?ff] bad_wt_diff_same
      unfolding inf_chain_def wt_diff_same_as_union[symmetric] by auto

    have wf {(s, t). ground s ∧ ground t ∧ wt t =_p wt s ∧ sym (head t) >_s sym (head s)}
      using gt_sym_wf unfolding wfp_def wf_iff_no_infinite_down_chain by fast
```

```
moreover have {(s, t). ground t ∧ wt t =p wt s ∧ gt_diff t s}
  ⊆ {(s, t). ground s ∧ ground t ∧ wt t =p wt s ∧ sym (head t) >s sym (head s)}
proof (clarsimp, intro conjI)
  fix s t
  assume gr_t: ground t and gt_diff_t_s: gt_diff t s
  thus gr_s: ground s
    using gt_iff_wt_unary_diff_same gt_imp_vars by fastforce
  show sym (head t) >s sym (head s)
    using gt_diff_t_s by cases (simp add: gt_hd_def gr_s gr_t ground_hd_in_ground_heads)
qed
ultimately have wf_diff: wf {(s, t). ground t ∧ wt t =p wt s ∧ gt_diff t s}
  by (rule wf_subset)

have diff_O_same:
  {(s, t). ground t ∧ wt t =p wt s ∧ gt_diff t s}
     O {(s, t). ground t ∧ wt t =p wt s ∧ gt_same t s}
   ⊆ {(s, t). ground t ∧ wt t =p wt s ∧ gt_diff t s}
  unfolding gt_diff.simps gt_same.simps by (auto intro: ge_ge_tpoly_trans simp: eq_tpoly_def)

have diff_same_as_union:
  {(s, t). ground t ∧ wt t =p wt s ∧ (gt_diff t s ∨ gt_same t s)} =
   {(s, t). ground t ∧ wt t =p wt s ∧ gt_diff t s}
   ∪ {(s, t). ground t ∧ wt t =p wt s ∧ gt_same t s}
  by auto

obtain k2 where
  bad_same: inf_chain (λt s. ground t ∧ wt t =p wt s ∧ gt_same t s) (λi. ?ff (i + k2))
  using wf_infinite_down_chain_compatible[OF wf_diff _ diff_O_same, of λi. ?ff (i + k1)]
    bad_diff_same
  unfolding inf_chain_def diff_same_as_union[symmetric] by (auto simp: add.assoc)
hence hd_sym: ⋀i. is_Sym (head (?ff (i + k2)))
  unfolding inf_chain_def by (simp add: ground_head)

define f where f = sym (head (?ff k2))
define w where w = eval_tpoly ?A (wt (?ff k2))

have head (?ff (i + k2)) = Sym f ∧ eval_tpoly ?A (wt (?ff (i + k2))) = w for i
proof (induct i)
  case 0
  thus ?case
    by (auto simp: f_def w_def hd.collapse(2)[OF hd_sym, of 0, simplified])
next
  case (Suc ia)
  thus ?case
    using bad_same unfolding inf_chain_def gt_same.simps zhmset_of_inject[symmetric]
    by (simp add: eq_tpoly_def legal_min_zpassign)
qed
note hd_eq_f = this[THEN conjunct1] and wt_eq_w = this[THEN conjunct2]

define max_args where
  max_args = (if δh = 0 then sum_coefs w else the_enat (arity_sym f))

have nargs_le_max_args: num_args (?ff (i + k2)) ≤ max_args for i
proof (cases δh = 0)
  case δ_ne_0: False
  hence ary_f_ne_inf: arity_sym f ≠ ∞
    using arity_sym_ne_infinity_if_δ_gt_0 of_nat_0 by blast
  have enat (num_args (worst_chain (λt s. ground t ∧ t >tw s) (λt s. size s < size t) (i + k2))) ≤ arity_sym f
    using wary_num_args_le_arity_head[OF ffi_wary[of i + k2]] by (simp add: hd_eq_f)
  with δ_ne_0 show ?thesis
    by (simp del: enat_ord_simps add: max_args_def  enat_ord_simps(1)[symmetric] enat_the_enat_iden[OF
ary_f_ne_inf])
next
```

case *δ_eq_0*: *True*
show *?thesis*
using *sum_coefs_ge_num_args_if_*δ_h*_eq_0*[*OF legal_min_passign δ_eq_0 ffi_wary*[*of i* + *k2*]]
by (*simp add*: *max_args_def δ_eq_0 wt_eq_w*)
qed

let *?U_of* = *λi. set* (*args* (*?ff* (*i* + *k2*)))

define *U* **where** *U* = (⋃ *i. ?U_of i*)

have *gr_u*: ⋀*u. u* ∈ *U* ⟹ *ground u*
unfolding *U_def* **by** (*blast dest*: *ground_args*[*OF _ ffi_ground*])
have *wary_u*: ⋀*u. u* ∈ *U* ⟹ *wary u*
unfolding *U_def* **by** (*blast dest*: *wary_args*[*OF _ ffi_wary*])

have ¬ *bad* ($>_{twg}$) *u* **if** *u_in*: *u* ∈ *?U_of i* **for** *u i*
proof
assume *u_bad*: *bad* ($>_{twg}$) *u*
have *sz_u*: *size u* < *size* (*?ff* (*i* + *k2*))
by (*rule size_in_args*[*OF u_in*])

show *False*
proof (*cases i* + *k2*)
case *0*
thus *False*
using *sz_u min_worst_chain_0*[*OF wf_sz u_bad*] **by** *simp*
next
case *Suc*
hence *gt*: *?ff* (*i* + *k2* − *1*) $>_{tw}$ *?ff* (*i* + *k2*)
using *worst_chain_pred*[*OF wf_sz t_bad*] **by** *auto*
moreover have *?ff* (*i* + *k2*) $>_{tw}$ *u*
using *gt gt_proper_sub sub_args sz_u u_in wary_args* **by** *auto*
ultimately have *?ff* (*i* + *k2* − *1*) $>_{tw}$ *u*
using *gt_trans* **by** *blast*
thus *False*
using *Suc sz_u min_worst_chain_Suc*[*OF wf_sz u_bad*] *ffi_ground* **by** *fastforce*
qed
qed
hence *u_good*: ⋀*u. u* ∈ *U* ⟹ ¬ *bad* ($>_{twg}$) *u*
unfolding *U_def* **by** *blast*

let *?gtwu* = *λt s. t* ∈ *U* ∧ *t* $>_{tw}$ *s*

have *gtwu_irrefl*: ⋀*x.* ¬ *?gtwu x x*
using *gt_irrefl* **by** *auto*

have ⋀*i j.* ∀ *t* ∈ *set* (*args* (*?ff* (*i* + *k2*))). ∀ *s* ∈ *set* (*args* (*?ff* (*j* + *k2*))). *t* $>_t$ *s* ⟶
t ∈ *U* ∧ *t* $>_{tw}$ *s*
using *wary_u* **unfolding** *U_def* **by** *blast*
moreover have ⋀*i. extf f* ($>_t$) (*args* (*?ff* (*i* + *k2*))) (*args* (*?ff* (*Suc i* + *k2*)))
using *bad_same hd_eq_f* **unfolding** *inf_chain_def gt_same.simps* **by** *auto*
ultimately have ⋀*i. extf f ?gtwu* (*args* (*?ff* (*i* + *k2*))) (*args* (*?ff* (*Suc i* + *k2*)))
by (*rule extf_mono_strong*)
hence *inf_chain* (*extf f ?gtwu*) (*λi. args* (*?ff* (*i* + *k2*)))
unfolding *inf_chain_def* **by** *blast*
hence *nwf_ext*:
¬ *wfP* (*λxs ys. length ys* ≤ *max_args* ∧ *length xs* ≤ *max_args* ∧ *extf f ?gtwu ys xs*)
unfolding *inf_chain_def wfp_def wf_iff_no_infinite_down_chain* **using** *nargs_le_max_args* **by** *fast*

have *gtwu_le_gtwg*: *?gtwu* ≤ ($>_{twg}$)
by (*auto intro*!: *gr_u*)

have *wfP* (*λs t. ?gtwu t s*)

```
      unfolding wfP_iff_no_inf_chain
    proof (intro notI, elim exE)
      fix f
      assume bad_f: inf_chain ?gtwu f
      hence bad_f0: bad ?gtwu (f 0)
        by (rule inf_chain_bad)
      hence f 0 ∈ U
        using bad_f unfolding inf_chain_def by blast
      hence ¬ bad (>_twg) (f 0)
        using u_good by blast
      hence ¬ bad ?gtwu (f 0)
        using bad_f inf_chain_bad inf_chain_subset[OF _ gtwu_le_gtwg] by blast
      thus False
        using bad_f0 by sat
    qed
    hence wf_ext: wfP (λxs ys. length ys ≤ max_args ∧ length xs ≤ max_args ∧ extf f ?gtwu ys xs)
      using extf_wf_bounded[of ?gtwu] gtwu_irrefl by blast

    show False
      using nwf_ext wf_ext by blast
  qed

  let ?subst = subst grounding_ϱ

  have wfP (λs t. ?subst t >_twg ?subst s)
    by (rule wfP_app[OF ground_wfP])
  hence wfP (λs t. ?subst t >_tw ?subst s)
    by (simp add: ground_grounding_ϱ)
  thus ?thesis
    by (auto intro: wfp_subset wary_subst_wary[OF wary_grounding_ϱ] gt_subst[OF wary_grounding_ϱ])
qed

end

end
```

7 Properties of Lambda-Free KBO on the Lambda Encoding

```
theory Lambda_Encoding_KBO
imports Lambda_Free_RPOs.Lambda_Encoding Lambda_Free_KBO_Basic
begin
```

This theory explores the properties of the λ-free KBO on the proposed encoding of λ-expressions.

```
locale kbo_lambda_encoding = kbo_basic _ _ _ _ λ_ :: 'v. UNIV :: 's set + lambda_encoding lam
  for lam :: 's +
  assumes
    gt_db_db: j > i ⟹ db j >_s db i and
    wt_db_db: wt_sym (db j) = wt_sym (db i)
begin

notation gt (infix ‹>_t› 50)
notation gt_hd (infix ‹>_hd› 50)

abbreviation ge :: ('s, 'v) tm ⇒ ('s, 'v) tm ⇒ bool (infix ‹≥_t› 50) where
  t ≥_t s ≡ t >_t s ∨ t = s

lemma wary_raw_db_subst: wary_subst (raw_db_subst i x)
  unfolding wary_subst_def by (simp add: arity_def)

lemma wt_subst_db: wt (subst_db i x s) = wt (subst (raw_db_subst j x) s)
  by (induct s arbitrary: i j)
    (clarsimp simp: raw_db_subst_def wt_db_db split: hd.splits,
      metis lambda_encoding.subst_db.simps(2) subst.simps(2) wt.simps(2))
```

```
lemma subst_db_Suc_ge: subst_db (Suc i) x s ≥t subst_db i x s
proof (induct s arbitrary: i)
  case (Hd x)
  then show ?case
    by (auto intro: gt_diff simp: wt_db_db gt_db_db gt_sym_imp_hd)
next
  case (App s1 s2)
  show ?case
    by (simp, safe)
      (metis (full_types) App.hyps(1) App.hyps(2) gt_compat_arg gt_compat_fun gt_trans)+
qed

lemma gt_subst_db: t >t s ⟹ subst_db i x t >t subst_db i x s
proof (simp only: atomize_imp,
    rule measure_induct_rule[of λ(t, s, i). {#size t, size s#}
        λ(t, s, i). t >t s ⟶ subst_db i x t >t subst_db i x s (t, s, i),
      simplified prod.case],
    simp only: split_paired_all prod.case atomize_imp[symmetric] atomize_all[symmetric])
  fix t s :: ('s, 'v) tm and i :: nat
  assume
    ih: ⋀ta sa i. {#size ta, size sa#} < {#size t, size s#} ⟹ ta >t sa ⟹
      subst_db i x ta >t subst_db i x sa and
    t_gt_s: t >t s

  let ?ϱ = subst_db i x

  show ?ϱ t >t ?ϱ s
  proof (cases wt (?ϱ t) = wt (?ϱ s))
    case wt_ϱt_ne_ϱs: False

    have vars_s: vars_mset t ⊇# vars_mset s
      using gt.cases t_gt_s by blast
    hence vars_ϱs: vars_mset (?ϱ t) ⊇# vars_mset (?ϱ s)
      by (simp add: var_mset_subst_db_subseteq)

    have wt_t_ge_s: wt t ≥ wt s
      by (metis dual_order.strict_implies_order eq_imp_le gt.cases t_gt_s)

    have wt (?ϱ t) > wt (?ϱ s)
      using wt_ϱt_ne_ϱs unfolding wt_subst_db
      by (metis (full_types) gt.simps gt_subst t_gt_s wary_raw_db_subst wt_subst_db)
    thus ?thesis
      by (rule gt_wt[OF vars_ϱs])
  next
    case wt_ϱt_eq_ϱs: True
    show ?thesis
      using t_gt_s
    proof cases
      case gt_wt
      hence False
        using wt_ϱt_eq_ϱs
        by (metis add_less_le_mono kbo_std.extra_wt_subseteq nat_less_le wary_raw_db_subst wt_subst
            wt_subst_db)
      thus ?thesis
        by sat
    next
      case _: gt_diff
      note vars_s = this(1) and hd_t_gt_hd_s = this(3)
      have vars_ϱs: vars_mset (?ϱ t) ⊇# vars_mset (?ϱ s)
        by (simp add: var_mset_subst_db_subseteq vars_s)
      term gt_hd
      have head (?ϱ t) >hd head (?ϱ s)
```

```
        by (smt Set.set_insert gt_hd_def hd_t_gt_hd_s head_subst_db insert_subset wary_raw_db_subst
            wary_subst_ground_heads)
      thus ?thesis
        by (rule gt_diff[OF vars_ϱs wt_ϱt_eq_ϱs])
    next
      case _: gt_same
      note vars_s = this(1) and hd_s_eq_hd_t = this(3) and extf = this(4)

      have vars_ϱs: vars_mset (?ϱ t) ⊇# vars_mset (?ϱ s)
        by (simp add: var_mset_subst_db_subseteq vars_s)
      have hd_ϱt: head (?ϱ t) = head (?ϱ s)
        by (simp add: hd_s_eq_hd_t head_subst_db)

      {
        fix f
        assume f_in_grs: f ∈ ground_heads (head (?ϱ s))

        let ?ϱa = subst_db (if head s = Sym lam then i + 1 else i) x
        let ?S = set (args t) ∪ set (args s)

        have extf_args_s_t: extf f (>_t) (args t) (args s)
          using extf f_in_grs hd_s_eq_hd_t head_subst_db wary_raw_db_subst wary_subst_ground_heads
          by (metis (no_types, lifting) insert_subset mk_disjoint_insert)
        have extf f (>_t) (map ?ϱa (args t)) (map ?ϱa (args s))
        proof (rule extf_map[of ?S, OF _ _ _ _ _ _ _ extf_args_s_t])
          show ∀ x ∈ ?S. ¬ ?ϱa x >_t ?ϱa x
            using gt_irrefl by blast
        next
          show ∀ z ∈ ?S. ∀ y ∈ ?S. ∀ x ∈ ?S. ?ϱa z >_t ?ϱa y ⟶ ?ϱa y >_t ?ϱa x ⟶ ?ϱa z >_t ?ϱa x
            using gt_trans by blast
        next
          have sz_a: ∀ ta ∈ ?S. ∀ sa ∈ ?S. {#size ta, size sa#} < {#size t, size s#}
            by (fastforce intro: Max_lt_imp_lt_mset dest: size_in_args)
          show ∀ y ∈ ?S. ∀ x ∈ ?S. y >_t x ⟶ ?ϱa y >_t ?ϱa x
            using ih sz_a by blast
        qed auto
        hence extf f (>_t) (args (?ϱ t)) (args (?ϱ s))
          by (simp add: args_subst_db hd_s_eq_hd_t)
      }
      hence ∀ f ∈ ground_heads (head (?ϱ s)). extf f (>_t) (args (?ϱ t)) (args (?ϱ s))
        by blast
      thus ?thesis
        by (rule gt_same[OF vars_ϱs wt_ϱt_eq_ϱs hd_ϱt])
    qed
  qed
qed

end

end
```

8 Knuth–Bendix Orders for Lambda-Free Higher-Order Terms

```
theory Lambda_Free_KBOs
imports Lambda_Free_KBO_App Lambda_Free_KBO_Basic Lambda_Free_TKBO_Coefs Lambda_Encoding_KBO
begin

locale simple_kbo_instances
begin

definition arity_sym :: nat ⇒ enat where
  arity_sym n = ∞
```

definition *arity_var* :: *nat* ⇒ *enat* **where**
arity_var n = ∞

definition *ground_head_var* :: *nat* ⇒ *nat set* **where**
ground_head_var x = *UNIV*

definition *gt_sym* :: *nat* ⇒ *nat* ⇒ *bool* **where**
gt_sym g f ⟷ *g* > *f*

definition ε :: *nat* **where**
ε = *1*

definition δ :: *nat* **where**
δ = *0*

definition *wt_sym* :: *nat* ⇒ *nat* **where**
wt_sym n = *1*

definition wt_sym_h :: *nat* ⇒ *hmultiset* **where**
wt_sym_h *n* = *1*

definition $coef\_sym_h$ :: *nat* ⇒ *nat* ⇒ *hmultiset* **where**
$coef\_sym_h$ *n i* = *1*

sublocale *kbo_app*: *kbo_app gt_sym wt_sym* ε *len_lexext*
by *unfold_locales* (*auto simp*: *gt_sym_def* ε*_def wt_sym_def intro*: *wf_less*[*folded wfp_def*])

sublocale *kbo_basic*: *kbo_basic gt_sym wt_sym* ε λf. *len_lexext ground_head_var*
by *unfold_locales* (*auto simp*: *ground_head_var_def gt_sym_def* ε*_def wt_sym_def*)

sublocale *kbo_std*: *kbo_std ground_head_var gt_sym* ε δ λf. *len_lexext arity_sym arity_var wt_sym*
by *unfold_locales*
(*auto simp*: *arity_sym_def arity_var_def ground_head_var_def* ε*_def* δ*_def wt_sym_def*)

sublocale *tkbo_coefs*: *tkbo_coefs ground_head_var gt_sym* ε δ λf. *len_lexext arity_sym arity_var*
wt_sym_h $coef\_sym_h$
by *unfold_locales* (*auto simp*: ε*_def* δ*_def* wt_sym_h*_def* $coef\_sym_h$*_def*)

end

end