

# Formalization of the Embedding Path Order for Lambda-Free Higher-Order Terms

Alexander Bentkamp

May 26, 2024

## Abstract

This Isabelle/HOL formalization defines the Embedding Path Order (EPO) for higher-order terms without  $\lambda$ -abstraction and proves many useful properties about it. In contrast to the lambda-free recursive path orders, it does not fully coincide with RPO on first-order terms, but it is compatible with arbitrary higher-order contexts.

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>The Embedding Relation for Lambda-Free Higher-Order Terms</b>	<b>1</b>
2.1	Positions of terms	1
2.2	Embedding step	2
2.3	Embedding relation	2
2.4	How are positions preserved under embedding steps?	4
2.5	Swapping embedding steps	4
2.6	Performing embedding steps in order of a given priority	4
2.7	Embedding steps under arguments	5
2.8	Rearranging embedding steps: first above, then below arguments	6
<b>3</b>	<b>The Chop Operation on Lambda-Free Higher-Order Terms</b>	<b>6</b>
3.1	Basic properties	7
3.2	Chop and the Embedding Relation	7
3.3	Chop and Substitutions	8
<b>4</b>	<b>The Embedding Path Order for Lambda-Free Higher-Order Terms</b>	<b>8</b>
4.1	Setup	8
4.2	Inductive Definitions	9
4.3	Transitivity	10
4.4	Irreflexivity	10
4.5	Subterm Property	10
4.6	Compatibility with Contexts	10
4.7	Compatibility with Embedding Relation	10
4.8	Stability under Substitutions	11
4.9	Totality on Ground Terms	11
4.10	Well-foundedness	11

## 1 Introduction

This Isabelle/HOL formalization defines the Embedding Path Order (EPO) for higher-order terms without  $\lambda$ -abstraction and proves many useful properties about it. In contrast to the lambda-free recursive path orders, it does not fully coincide with RPO on first-order terms, but it is compatible with arbitrary higher-order contexts.

## 2 The Embedding Relation for Lambda-Free Higher-Order Terms

**theory** *Embeddings*

**imports** *Lambda\_Free\_RPOs.Lambda\_Free\_Term* *Lambda\_Free\_RPOs.Extension\_Orders*  
**begin**

### 2.1 Positions of terms

**datatype** *dir* = *Left* | *Right*

**fun** *position\_of* :: ('s, 'v) *tm*  $\Rightarrow$  *dir list*  $\Rightarrow$  *bool* **where**  
*position\_of\_Nil*: *position\_of* [] = *True* |  
*position\_of\_Hd*: *position\_of* (*Hd* \_) (*\_#\_*) = *False* |  
*position\_of\_left*: *position\_of* (*App* *t s*) (*Left* # *ds*) = *position\_of* *t ds* |  
*position\_of\_right*: *position\_of* (*App* *t s*) (*Right* # *ds*) = *position\_of* *s ds*

**definition** *opp* :: *dir*  $\Rightarrow$  *dir* **where**  
*opp* *d* = (if *d* = *Right* then *Left* else *Right*)

**lemma** *opp\_simps*[*simp*]:  
*opp* *Right* = *Left*  
*opp* *Left* = *Right*  
 ⟨*proof*⟩

**lemma** *shallower\_pos*: *position\_of* *t* (*p* @ *q* @ [*dq*])  $\Longrightarrow$  *position\_of* *t* (*p* @ [*dp*])  
 ⟨*proof*⟩

**lemma** *no\_position\_replicate\_num\_args*:  $\neg$  *position\_of* *t* (*replicate* (*num\_args* *t*) *Left* @ [*d*])  
 ⟨*proof*⟩

**lemma** *shorten\_position*: *position\_of* *t* (*p* @ *q*)  $\Longrightarrow$  *position\_of* *t* *p*  
 ⟨*proof*⟩

### 2.2 Embedding step

Embedding step at a given position. If the position is not present, default to identity.

**fun** *emb\_step\_at* :: *dir list*  $\Rightarrow$  *dir*  $\Rightarrow$  ('s, 'v) *tm*  $\Rightarrow$  ('s, 'v) *tm* **where**  
*emb\_step\_at\_left*: *emb\_step\_at* [] *Left* (*App* *t s*) = *t*  
| *emb\_step\_at\_right*: *emb\_step\_at* [] *Right* (*App* *t s*) = *s*  
| *emb\_step\_at\_left\_context*: *emb\_step\_at* (*Left* # *p*) *dir* (*App* *t s*) = *App* (*emb\_step\_at* *p dir* *t*) *s*  
| *emb\_step\_at\_right\_context*: *emb\_step\_at* (*Right* # *p*) *dir* (*App* *t s*) = *App* *t* (*emb\_step\_at* *p dir* *s*)  
| *emb\_step\_at\_head*: *emb\_step\_at* \_ \_ (*Hd* *h*) = *Hd* *h*

**abbreviation** *emb\_step\_at'* *p t* == *emb\_step\_at* (*butlast* *p*) (*last* *p*) *t*

**lemmas** *emb\_step\_at\_induct* = *emb\_step\_at.induct*[*case\_names* *left* *right* *left\_context* *right\_context* *head*]

**lemma** *emb\_step\_at\_is\_App*: *emb\_step\_at* *p d u*  $\neq$  *u*  $\Longrightarrow$  *is\_App* *u*  
 ⟨*proof*⟩

Definition of an embedding step without using positions.

**inductive** *emb\_step* (**infix**  $\rightarrow_{emb}$  50) **where**  
*left*: (*App* *t1 t2*)  $\rightarrow_{emb}$  *t1* |  
*right*: (*App* *t1 t2*)  $\rightarrow_{emb}$  *t2* |  
*context\_left*: *t*  $\rightarrow_{emb}$  *s*  $\Longrightarrow$  (*App* *t u*)  $\rightarrow_{emb}$  (*App* *s u*) |  
*context\_right*: *t*  $\rightarrow_{emb}$  *s*  $\Longrightarrow$  (*App* *u t*)  $\rightarrow_{emb}$  (*App* *u s*)

The two definitions of an embedding step are equivalent:

**lemma** *emb\_step\_equiv*: *emb\_step* *t s*  $\longleftrightarrow$  ( $\exists$  *p d*. *emb\_step\_at* *p d t* = *s*)  $\wedge$  *t*  $\neq$  *s*  
 ⟨*proof*⟩

**lemma** *emb\_step\_fun*: *is\_App* *t*  $\Longrightarrow$  *t*  $\rightarrow_{emb}$  (*fun* *t*)  
 ⟨*proof*⟩

**lemma** *emb\_step\_arg*:  $is\_App\ t \implies t \rightarrow_{emb} (arg\ t)$   
 ⟨proof⟩

**lemma** *emb\_step\_hsize*:  $t \rightarrow_{emb} s \implies hsize\ t > hsize\ s$   
 ⟨proof⟩

**lemma** *emb\_step\_vars*:  $t \rightarrow_{emb} s \implies vars\ s \subseteq vars\ t$   
 ⟨proof⟩

**lemma** *emb\_step\_equiv'*:  $emb\_step\ t\ s \iff (\exists p. p \neq [] \wedge emb\_step\_at'\ p\ t = s) \wedge t \neq s$   
 ⟨proof⟩

**lemma** *position\_if\_emb\_step\_at*:  $emb\_step\_at\ p\ d\ t = u \implies t \neq u \implies position\_of\ t\ (p\ @\ [d])$   
 ⟨proof⟩

**lemma** *emb\_step\_at\_if\_position*:  
**assumes**  
 $position\_of\ t\ (p\ @\ [d])$   
**shows**  $t \rightarrow_{emb} emb\_step\_at\ p\ d\ t$   
 ⟨proof⟩

## 2.3 Embedding relation

Definition of an embedding as a sequence of embedding steps at given positions:

**fun** *emb\_at* ::  $(dir\ list \times dir)\ list \Rightarrow ('s, 'v)\ tm \Rightarrow ('s, 'v)\ tm$  **where**  
 $emb\_at\_Nil: emb\_at\ []\ t = t$  |  
 $emb\_at\_Cons: emb\_at\ ((p,d) \# ps)\ t = emb\_step\_at\ p\ d\ (emb\_at\ ps\ t)$

Definition of an embedding without using positions:

**inductive** *emb* (**infix**  $\triangleright_{emb}$  50) **where**  
 $refl: t \triangleright_{emb} t$  |  
 $step: t \triangleright_{emb} u \implies u \rightarrow_{emb} s \implies t \triangleright_{emb} s$

**abbreviation** *emb\_neq* (**infix**  $\triangleright_{emb}$  50) **where**  $emb\_neq\ t\ s \equiv t \triangleright_{emb} s \wedge t \neq s$

The two definitions coincide:

**lemma** *emb\_equiv*:  $(t \triangleright_{emb} s) = (\exists ps. emb\_at\ ps\ t = s)$   
 ⟨proof⟩

**lemma** *emb\_at\_trans*:  $emb\_at\ ps\ t = u \implies emb\_at\ qs\ u = s \implies emb\_at\ (qs\ @\ ps)\ t = s$   
 ⟨proof⟩

**lemma** *emb\_trans*:  $t \triangleright_{emb} u \implies u \triangleright_{emb} s \implies t \triangleright_{emb} s$   
 ⟨proof⟩

**lemma** *emb\_step\_is\_emb*:  $t \rightarrow_{emb} s \implies t \triangleright_{emb} s$   
 ⟨proof⟩

**lemma** *emb\_hsize*:  $t \triangleright_{emb} s \implies hsize\ t \geq hsize\ s$   
 ⟨proof⟩

**lemma** *emb\_prepend\_step*:  $t \rightarrow_{emb} u \implies u \triangleright_{emb} s \implies t \triangleright_{emb} s$   
 ⟨proof⟩

**lemma** *sub\_emb*:  $sub\ s\ t \implies t \triangleright_{emb} s$   
 ⟨proof⟩

**lemma** *sequence\_emb\_steps*:  $t \triangleright_{emb} s \iff (\exists us. us \neq [] \wedge hd\ us = t \wedge last\ us = s \wedge (\forall i. Suc\ i < length\ us \longrightarrow us\ !\ i \rightarrow_{emb} us\ !\ Suc\ i))$   
 ⟨proof⟩

**lemma** *emb\_induct\_reverse* [*consumes 1, case\_names refl step*]:

**assumes**  
*emb*:  $t \triangleright_{emb} s$  **and**  
*refl*:  $\bigwedge t. P t t$  **and**  
*step*:  $\bigwedge t u s. t \rightarrow_{emb} u \implies u \triangleright_{emb} s \implies P u s \implies P t s$   
**shows**  
 $P t s$   
*<proof>*

**lemma** *emb\_cases\_reverse* [*consumes 1, case\_names refl step*]:

$t \triangleright_{emb} s \implies (\bigwedge t'. t = t' \implies s = t' \implies P) \implies (\bigwedge t' u s'. t = t' \implies s = s' \implies t' \rightarrow_{emb} u \implies u \triangleright_{emb} s' \implies P) \implies P$   
*<proof>*

**lemma** *emb\_vars*:  $t \triangleright_{emb} s \implies vars s \subseteq vars t$

*<proof>*

**lemma** *ground\_emb*:  $t \triangleright_{emb} s \implies ground t \implies ground s$

*<proof>*

**lemma** *arg\_emb*:  $s \in set (args t) \implies t \triangleright_{emb} s$

*<proof>*

**lemma** *emb\_step\_at\_subst*:

**assumes**

*position\_of*  $t (p @ [d])$

**shows**

$emb\_step\_at\ p\ d\ (subst\ \varrho\ t) = subst\ \varrho\ (emb\_step\_at\ p\ d\ t)$

*<proof>*

**lemma** *emb\_step\_subst*:  $t \rightarrow_{emb} s \implies subst\ \varrho\ t \rightarrow_{emb} subst\ \varrho\ s$

*<proof>*

**lemma** *emb\_subst*:  $t \triangleright_{emb} s \implies subst\ \varrho\ t \triangleright_{emb} subst\ \varrho\ s$

*<proof>*

**lemma** *emb\_hsize\_neq*:

**assumes**

$t \triangleright_{emb} s$   $t \neq s$

**shows**

$hsize\ t > hsize\ s$

*<proof>*

## 2.4 How are positions preserved under embedding steps?

Disjunct positions are preserved: For example,  $[L,R]$  is a position of  $f\ a\ (g\ b)$ . When performing an embedding step at  $[R,R]$  to obtain  $f\ a\ b$ , the position  $[L,R]$  still exists. (More precisely, it even contains the same subterm, namely  $a$ .)

**lemma** *pos\_emb\_step\_at\_disjunct*:

**assumes**

*take* (*length*  $q$ )  $p \neq q$

*take* (*length*  $p$ )  $q \neq p$

**shows**

$position\_of\ t\ (p @ [d1]) \longleftrightarrow position\_of\ (emb\_step\_at\ q\ d2\ t)\ (p @ [d1])$

*<proof>*

Even if only the last element of a position differs from the position of an embedding step, that position is preserved. For example,  $[L]$  is a position of  $f\ (g\ b)$ . After performing an embedding step at  $[R,R]$  to obtain  $f\ b$ , the position  $[L]$  still exists. (More precisely, it even contains the same subterm, namely  $f$ .)

**lemma** *pos\_emb\_step\_at\_opp*:

$position\_of\ t\ (p@[d1]) \longleftrightarrow position\_of\ (emb\_step\_at\ (p @ [opp\ d1]) @ q)\ d2\ t)\ (p@[d1])$

*<proof>*

Positions are preserved under embedding steps below them:

**lemma** *pos\_emb\_step\_at\_nested*:

**shows**  $position\_of (emb\_step\_at (p @ [d1] @ q) d2 t) (p @ [d1]) \longleftrightarrow position\_of t (p @ [d1])$   
 ⟨proof⟩

## 2.5 Swapping embedding steps

The order of embedding steps at disjunct position can be changed freely:

**lemma** *swap\_disjunct\_emb\_step\_at*:

**assumes**

$length\ p \leq length\ q \implies take\ (length\ p)\ q \neq p$   $length\ q \leq length\ p \implies take\ (length\ q)\ p \neq q$

**shows**

$emb\_step\_at\ q\ d2\ (emb\_step\_at\ p\ d1\ t) = emb\_step\_at\ p\ d1\ (emb\_step\_at\ q\ d2\ t)$

⟨proof⟩

An embedding step inside the branch that is removed in a second embedding step is useless. For example, the embedding  $f (g\ b) \rightarrow emb\ f\ b \rightarrow emb\ f$  can be achieved using a single step  $f (g\ b) \rightarrow emb\ f$ .

**lemma** *merge\_emb\_step\_at*:

$emb\_step\_at\ p\ d1\ (emb\_step\_at\ (p @ [opp\ d1] @ q) d2 t) = emb\_step\_at\ p\ d1\ t$

⟨proof⟩

When swapping two embedding steps of a position below another, one of the positions has to be slightly changed:

**lemma** *swap\_nested\_emb\_step\_at*:

$emb\_step\_at\ (p @ q)\ d2\ (emb\_step\_at\ p\ d1\ t) = emb\_step\_at\ p\ d1\ (emb\_step\_at\ (p @ [d1] @ q) d2 t)$

⟨proof⟩

## 2.6 Performing embedding steps in order of a given priority

We want to perform all embedding steps first that modify the head or the number of arguments of a term. To this end we define the function *prio\_emb\_step* that performs the embedding step with the highest priority possible. The priority is given by a function "prio" from positions to nats, where the lowest number has the highest priority.

**definition** *prio\_emb\_pos* ::  $(dir\ list \Rightarrow nat) \Rightarrow ('s, 'v)\ tm \Rightarrow ('s, 'v)\ tm \Rightarrow dir\ list$  **where**

$prio\_emb\_pos\ prio\ t\ s = (ARG\_MIN\ prio\ p.\ p \neq [] \wedge position\_of\ t\ p \wedge emb\_step\_at'\ p\ t \triangleright_{emb}\ s)$

**definition** *prio\_emb\_step* ::  $(dir\ list \Rightarrow nat) \Rightarrow ('s, 'v)\ tm \Rightarrow ('s, 'v)\ tm \Rightarrow ('s, 'v)\ tm$  **where**

$prio\_emb\_step\ prio\ t\ s = emb\_step\_at'\ (prio\_emb\_pos\ prio\ t\ s)\ t$

**lemma** *prio\_emb\_posI*:

$t \triangleright_{emb}\ s \implies t \neq s \implies prio\_emb\_pos\ prio\ t\ s \neq [] \wedge position\_of\ t\ (prio\_emb\_pos\ prio\ t\ s) \wedge emb\_step\_at'\ (prio\_emb\_pos\ prio\ t\ s)\ t \triangleright_{emb}\ s$

⟨proof⟩

**lemma** *prio\_emb\_pos\_le*:

**assumes**  $p \neq []$   $position\_of\ t\ p$   $emb\_step\_at'\ p\ t \triangleright_{emb}\ s$

**shows**  $prio\ (prio\_emb\_pos\ prio\ t\ s) \leq prio\ p$

⟨proof⟩

We want an embedding step sequence in which the priority numbers monotonely increase. We can get such a sequence if the priority function assigns greater values to deeper positions.

**lemma** *prio\_emb\_pos\_increase*:

**assumes**

$t \triangleright_{emb}\ s$   $t \neq s$   $prio\_emb\_step\ prio\ t\ s \neq s$  **and**

*valid\_prio*:  $\bigwedge p\ q\ dp\ dq.\ prio\ (p @ [dp]) > prio\ (q @ [dq]) \implies take\ (length\ p)\ q \neq p$

**shows**

$prio\ (prio\_emb\_pos\ prio\ t\ s) \leq prio\ (prio\_emb\_pos\ prio\ (prio\_emb\_step\ prio\ t\ s)\ s)$

(**is**  $prio\ ?p1 \leq prio\ ?p2$ )

⟨proof⟩

**lemma** *sequence\_prio\_emb\_steps*:

**assumes**

$t \succeq_{emb} s$

**shows**

$\exists us. us \neq [] \wedge hd\ us = t \wedge last\ us = s \wedge$

$(\forall i. Suc\ i < length\ us \longrightarrow (prio\_emb\_step\ prio\ (us\ !\ i)\ s = us\ !\ Suc\ i \wedge us\ !\ i \rightarrow_{emb}\ us\ !\ Suc\ i))$

*<proof>*

## 2.7 Embedding steps under arguments

We want to perform positions that modify the head and the number of arguments first. Formally these positions can be characterized as "list\_all (op = Left) p". We show here that embeddings at other positions do not modify the head, the number of arguments. Moreover, for each argument, the argument after the step is an embedding of the argument before the step.

**lemma** *emb\_step\_under\_args\_head*:

**assumes**

$\neg list\_all\ (\lambda x. x = Left)\ p$

**shows**

$head\ (emb\_step\_at\ p\ d\ t) = head\ t$

*<proof>*

**lemma** *emb\_step\_under\_args\_num\_args*:

**assumes**

$\neg list\_all\ (\lambda x. x = Left)\ p$

**shows**

$num\_args\ (emb\_step\_at\ p\ d\ t) = num\_args\ t$

*<proof>*

**lemma** *emb\_step\_under\_args\_emb\_step*:

**assumes**

$\neg list\_all\ (\lambda x. x = Left)\ p$

$position\_of\ t\ (p\ @\ [d])$

**obtains** *i* **where**

$i < num\_args\ t$

$args\ t\ !\ i \rightarrow_{emb}\ args\ (emb\_step\_at\ p\ d\ t)\ !\ i$  **and**

$\bigwedge j. j < num\_args\ t \implies i \neq j \implies args\ t\ !\ j = args\ (emb\_step\_at\ p\ d\ t)\ !\ j$

*<proof>*

**lemma** *emb\_step\_under\_args\_emb*:

**assumes**  $\neg list\_all\ (\lambda x. x = Left)\ p$

$position\_of\ t\ (p\ @\ [d])$

**shows**

$\forall i. i < num\_args\ t \longrightarrow args\ t\ !\ i \succeq_{emb}\ args\ (emb\_step\_at\ p\ d\ t)\ !\ i$

*<proof>*

**lemma** *position\_Left\_only\_subst*:

**assumes**  $list\_all\ (\lambda x. x = Left)\ p$

**and**  $position\_of\ (subst\ \varrho\ w)\ (p\ @\ [d])$

**and**  $num\_args\ (subst\ \varrho\ w) = num\_args\ w$

**shows**  $position\_of\ w\ (p\ @\ [d])$

*<proof>*

## 2.8 Rearranging embedding steps: first above, then below arguments

**lemma** *perform\_emb\_above\_vars0*:

**assumes**

$subst\ \varrho\ s \succeq_{emb}\ u$

**obtains** *w* **where**

$s \succeq_{emb}\ w$

$subst\ \varrho\ w \succeq_{emb}\ u$

$\forall w'. w \rightarrow_{emb}\ w' \longrightarrow \neg subst\ \varrho\ w' \succeq_{emb}\ u$

*<proof>*

**lemma** *emb\_only\_below\_vars*:

**assumes**

$subst\ \varrho\ s \succeq_{emb}\ u$

$s \succeq_{emb}\ w$

$is\_Sym\ (head\ w)$

$subst\ \varrho\ w \succeq_{emb}\ u$

$\forall w'. w \rightarrow_{emb}\ w' \longrightarrow \neg\ subst\ \varrho\ w' \succeq_{emb}\ u$

**obtains** *ws* **where**

$ws \neq []$

$hd\ ws = subst\ \varrho\ w$

$last\ ws = u$

$\forall i. Suc\ i < length\ ws \longrightarrow$

$(\exists p\ d. emb\_step\_at\ p\ d\ (ws\ !\ i) = ws\ !\ Suc\ i \wedge \neg\ list\_all\ (\lambda x. x = Left)\ p)$

$\forall i. i < length\ ws \longrightarrow head\ (ws\ !\ i) = head\ w \wedge num\_args\ (ws\ !\ i) = num\_args\ w$

$\forall i. i < length\ ws \longrightarrow (\forall k. k < num\_args\ w \longrightarrow args\ (subst\ \varrho\ w)\ !\ k \succeq_{emb}\ args\ (ws\ !\ i)\ !\ k)$

*<proof>*

**lemma** *perform\_emb\_above\_vars*:

**assumes**

$subst\ \varrho\ s \succeq_{emb}\ u$

**obtains** *w* **where**

$s \succeq_{emb}\ w$

$subst\ \varrho\ w \succeq_{emb}\ u$

$is\_Sym\ (head\ w) \implies head\ w = head\ u \wedge num\_args\ w = num\_args\ u \wedge (\forall k. k < num\_args\ w \longrightarrow args\ (subst\ \varrho\ w)\ !\ k \succeq_{emb}\ args\ u\ !\ k)$

*<proof>*

**end**

## 3 The Chop Operation on Lambda-Free Higher-Order Terms

**theory** *Chop*

**imports** *Embeddings*

**begin**

**definition** *chop* ::  $(s, v)\ tm \Rightarrow (s, v)\ tm$  **where**

$chop\ t = apps\ (hd\ (args\ t))\ (tl\ (args\ t))$

### 3.1 Basic properties

**lemma** *chop\_App\_Hd*:  $is\_Hd\ s \implies chop\ (App\ s\ t) = t$

*<proof>*

**lemma** *chop\_apps*:  $is\_App\ t \implies chop\ (apps\ t\ ts) = apps\ (chop\ t)\ ts$

*<proof>*

**lemma** *vars\_chop*:  $is\_App\ t \implies vars\ (chop\ t) \cup vars\_hd\ (head\ t) = vars\ t$

*<proof>*

**lemma** *ground\_chop*:  $is\_App\ t \implies ground\ t \implies ground\ (chop\ t)$

*<proof>*

**lemma** *hsize\_chop*:  $is\_App\ t \implies (Suc\ (hsize\ (chop\ t))) = hsize\ t$

*<proof>*

**lemma** *hsize\_chop\_lt*:  $is\_App\ t \implies hsize\ (chop\ t) < hsize\ t$

*<proof>*

**lemma** *chop\_fun*:

**assumes**  $is\_App\ t\ is\_App\ (fun\ t)$

**shows**  $App\ (chop\ (fun\ t))\ (arg\ t) = chop\ t$

*<proof>*

## 3.2 Chop and the Embedding Relation

**lemma** *emb\_step\_chop*:  $is\_App\ t \implies t \rightarrow_{emb}\ chop\ t$   
 ⟨proof⟩

**lemma** *chop\_emb\_step\_at*:  
 assumes *is\_App* *t*  
 shows  $chop\ t = emb\_step\_at\ (replicate\ (num\_args\ (fun\ t))\ Left)\ Right\ t$   
 ⟨proof⟩

**lemma** *emb\_step\_at\_chop*:  
 assumes *emb\_step\_at*:  $emb\_step\_at\ p\ Right\ t = s$   
 and *pos\_position\_of* *t* ( $p\ @\ [Right]$ )  
 and *all\_Left*:  $list\_all\ (\lambda x. x = Left)\ p$   
 shows  $chop\ t = s \vee chop\ t \rightarrow_{emb}\ s$   
 ⟨proof⟩

**lemma** *emb\_step\_at\_remove\_arg*:  
 assumes *emb\_step\_at*:  $emb\_step\_at\ p\ Left\ t = s$   
 and *pos\_position\_of* *t* ( $p\ @\ [Left]$ )  
 and *all\_Left*:  $list\_all\ (\lambda x. x = Left)\ p$   
 shows let  $i = num\_args\ t - Suc\ (length\ p)$  in  
 $head\ t = head\ s \wedge i < num\_args\ t \wedge args\ s = take\ i\ (args\ t) @ drop\ (Suc\ i)\ (args\ t)$   
 ⟨proof⟩

**lemma** *emb\_step\_cases* [*consumes* 1, *case\_names* *chop* *extended\_chop* *remove\_arg* *under\_arg*]:  
 assumes *emb*:  $t \rightarrow_{emb}\ s$   
 and *chop*:  $chop\ t = s \implies P$   
 and *extended\_chop*:  $chop\ t \rightarrow_{emb}\ s \implies P$   
 and *remove\_arg*:  $\bigwedge i. head\ t = head\ s \implies i < num\_args\ t \implies args\ s = take\ i\ (args\ t) @ drop\ (Suc\ i)\ (args\ t) \implies P$   
 and *under\_arg*:  $\bigwedge i. head\ t = head\ s \implies num\_args\ t = num\_args\ s \implies args\ t\ !\ i \rightarrow_{emb}\ args\ s\ !\ i \implies (\bigwedge j. j < num\_args\ t \implies i \neq j \implies args\ t\ !\ j = args\ s\ !\ j) \implies P$   
 shows *P*  
 ⟨proof⟩

**lemma** *chop\_position\_of*:  
 assumes *is\_App* *s*  
 shows  $position\_of\ s\ (replicate\ (num\_args\ (fun\ s))\ dir.Left\ @\ [Right])$   
 ⟨proof⟩

## 3.3 Chop and Substitutions

**lemma** *Suc\_num\_args*:  $is\_App\ t \implies Suc\ (num\_args\ (fun\ t)) = num\_args\ t$   
 ⟨proof⟩

**lemma** *fun\_subst*:  $is\_App\ s \implies subst\ \varrho\ (fun\ s) = fun\ (subst\ \varrho\ s)$   
 ⟨proof⟩

**lemma** *args\_subst\_Hd*:  
 assumes *is\_Hd* ( $subst\ \varrho\ (Hd\ (head\ s))$ )  
 shows  $args\ (subst\ \varrho\ s) = map\ (subst\ \varrho)\ (args\ s)$   
 ⟨proof⟩

**lemma** *chop\_subst\_emb0*:  
 assumes *is\_App* *s*  
 assumes *chop* ( $subst\ \varrho\ s \neq subst\ \varrho\ (chop\ s)$ )  
 shows  $emb\_step\_at\ (replicate\ (num\_args\ (fun\ s))\ Left)\ Right\ (chop\ (subst\ \varrho\ s)) = subst\ \varrho\ (chop\ s)$   
 ⟨proof⟩

**lemma** *chop\_subst\_emb*:  
 assumes *is\_App* *s*



**shows**  $\text{chop } (\text{subst } \varrho \ s) \triangleright_{emb} \text{subst } \varrho \ (\text{chop } s)$   
 ⟨proof⟩

**lemma** *chop\_subst\_Hd*:  
**assumes** *is\_App s*  
**assumes** *is\_Hd (subst  $\varrho$  (Hd (head s)))*  
**shows**  $\text{chop } (\text{subst } \varrho \ s) = \text{subst } \varrho \ (\text{chop } s)$   
 ⟨proof⟩

**lemma** *chop\_subst\_Sym*:  
**assumes** *is\_App s*  
**assumes** *is\_Sym (head s)*  
**shows**  $\text{chop } (\text{subst } \varrho \ s) = \text{subst } \varrho \ (\text{chop } s)$   
 ⟨proof⟩

end

## 4 The Embedding Path Order for Lambda-Free Higher-Order Terms

**theory** *Lambda\_Free\_EPO*  
**imports** *Chop Nested\_Multisets\_Ordinals.Multiset\_More*  
**abbrevs**  $>t = >_t$   
**and**  $\geq t = \geq_t$   
**begin**

This theory defines the embedding path order for  $\lambda$ -free higher-order terms.

### 4.1 Setup

**locale** *epo* = *ground\_heads* ( $>_s$ ) *arity\_sym* *arity\_var*  
**for**  
 $\text{gt\_sym} :: 's \Rightarrow 's \Rightarrow \text{bool}$  (**infix**  $>_s$  50) **and**  
 $\text{arity\_sym} :: 's \Rightarrow \text{enat}$  **and**  
 $\text{arity\_var} :: 'v \Rightarrow \text{enat} +$   
**fixes**  
 $\text{extf} :: 's \Rightarrow (('s, 'v) \text{tm} \Rightarrow ('s, 'v) \text{tm} \Rightarrow \text{bool}) \Rightarrow ('s, 'v) \text{tm list} \Rightarrow ('s, 'v) \text{tm list} \Rightarrow \text{bool}$   
**assumes**  
 $\text{extf\_ext\_trans\_before\_irrefl}: \text{ext\_trans\_before\_irrefl } (\text{extf } f)$  **and**  
 $\text{extf\_ext\_compat\_list}: \text{ext\_compat\_list } (\text{extf } f)$   
**assumes**  $\text{extf\_ext\_compat\_snoc}: \text{ext\_compat\_snoc } (\text{extf } f)$   
**assumes**  $\text{extf\_ext\_compat\_cons}: \text{ext\_compat\_cons } (\text{extf } f)$   
**assumes**  $\text{extf\_min\_empty}: \text{extf } f \text{ gt } [a] []$   
**begin**

**lemma** *extf\_ext\_trans*:  $\text{ext\_trans } (\text{extf } f)$   
 ⟨proof⟩

**lemma** *extf\_ext*:  $\text{ext } (\text{extf } f)$   
 ⟨proof⟩

**lemmas**  $\text{extf\_mono\_strong} = \text{ext.mono\_strong}[OF \ \text{extf\_ext}]$   
**lemmas**  $\text{extf\_mono} = \text{ext.mono}[OF \ \text{extf\_ext}, \ \text{mono}]$   
**lemmas**  $\text{extf\_map} = \text{ext.map}[OF \ \text{extf\_ext}]$   
**lemmas**  $\text{extf\_trans} = \text{ext.trans}[OF \ \text{extf\_ext\_trans}]$   
**lemmas**  $\text{extf\_irrefl\_from\_trans} =$   
 $\text{ext\_trans\_before\_irrefl.irrefl\_from\_trans}[OF \ \text{extf\_ext\_trans\_before\_irrefl}]$   
**lemmas**  $\text{extf\_compat\_list} = \text{ext\_compat\_list.compat\_list}[OF \ \text{extf\_ext\_compat\_list}]$

**lemmas**  $\text{extf\_compat\_cons} = \text{ext\_compat\_cons.compat\_cons}[OF \ \text{extf\_ext\_compat\_cons}]$   
**lemmas**  $\text{extf\_compat\_snoc} = \text{ext\_compat\_snoc.compat\_snoc}[OF \ \text{extf\_ext\_compat\_snoc}]$

**lemmas**  $\text{extf\_compat\_append\_right} = \text{ext\_compat\_snoc.compat\_append\_right}[OF \ \text{extf\_ext\_compat\_snoc}]$   
**lemmas**  $\text{extf\_compat\_append\_left} = \text{ext\_compat\_cons.compat\_append\_left}[OF \ \text{extf\_ext\_compat\_cons}]$

**lemma** *extf\_snoc*:  $\text{extf } f \text{ } gt \text{ } (xs @ [z]) \text{ } xs$   
 ⟨proof⟩

## 4.2 Inductive Definitions

**definition**

$\text{chkchop} :: ((s, v) \text{ } tm \Rightarrow (s, v) \text{ } tm \Rightarrow \text{bool}) \Rightarrow (s, v) \text{ } tm \Rightarrow (s, v) \text{ } tm \Rightarrow \text{bool}$

**where**

[simp]:  $\text{chkchop } gt \text{ } t \text{ } s \iff \text{is\_Hd } s \vee gt \text{ } t \text{ } (\text{chop } s)$

**definition**

$\text{chkchop\_same} :: ((s, v) \text{ } tm \Rightarrow (s, v) \text{ } tm \Rightarrow \text{bool}) \Rightarrow (s, v) \text{ } tm \Rightarrow (s, v) \text{ } tm \Rightarrow \text{bool}$

**where**

[simp]:  $\text{chkchop\_same } gt \text{ } t \text{ } s \iff$   
 (if  $\text{is\_Var } (\text{head } t)$   
 then  $\text{is\_App } t \wedge \text{chkchop } gt \text{ } (\text{chop } t) \text{ } s$   
 else  $\text{chkchop } gt \text{ } t \text{ } s$ )

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

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

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

$gt\_chop$ :  $\text{is\_App } t \implies \text{chop } t >_t s \vee \text{chop } t = s \implies t >_t s$

|  $gt\_diff$ :  $\text{head } t >_{hd} \text{head } s \implies \text{is\_Sym } (\text{head } s) \implies \text{chkchop } (>_t) \text{ } t \text{ } s \implies t >_t s$

|  $gt\_same$ :  $\text{head } t = \text{head } s \implies \text{chkchop\_same } (>_t) \text{ } t \text{ } s \implies$

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

**abbreviation** *ge* ::  $(s, v) \text{ } tm \Rightarrow (s, v) \text{ } tm \Rightarrow \text{bool}$  (**infix**  $\geq_t$  50) **where**

$t \geq_t s \equiv t >_t s \vee t = s$

**inductive** *gt\_chop* ::  $(s, v) \text{ } tm \Rightarrow (s, v) \text{ } tm \Rightarrow \text{bool}$  **where**

$gt\_chopI$ :  $\text{is\_App } t \implies \text{chop } t \geq_t s \implies gt\_chop \text{ } t \text{ } s$

**inductive** *gt\_diff* ::  $(s, v) \text{ } tm \Rightarrow (s, v) \text{ } tm \Rightarrow \text{bool}$  **where**

$gt\_diffI$ :  $\text{head } t >_{hd} \text{head } s \implies \text{is\_Sym } (\text{head } s) \implies \text{chkchop } (>_t) \text{ } t \text{ } s \implies gt\_diff \text{ } t \text{ } s$

**inductive** *gt\_same* ::  $(s, v) \text{ } tm \Rightarrow (s, v) \text{ } tm \Rightarrow \text{bool}$  **where**

$gt\_sameI$ :  $\text{head } t = \text{head } s \implies \text{chkchop\_same } (>_t) \text{ } t \text{ } s \implies$

$(\forall f \in \text{ground\_heads } (\text{head } t). \text{extf } f (>_t) (\text{args } t) (\text{args } s)) \implies gt\_same \text{ } t \text{ } s$

**lemma** *gt\_iff\_chop\_diff\_same*:  $t >_t s \iff gt\_chop \text{ } t \text{ } s \vee gt\_diff \text{ } t \text{ } s \vee gt\_same \text{ } t \text{ } s$   
 ⟨proof⟩

## 4.3 Transitivity

**lemma** *t\_gt\_chop\_t*:  $\text{is\_App } t \implies t >_t \text{chop } t$   
 ⟨proof⟩

**lemma** *gt\_trans*:  $u >_t t \implies t >_t s \implies u >_t s$   
 ⟨proof⟩

## 4.4 Irreflexivity

**theorem** *gt\_irrefl*:  $\neg s >_t s$   
 ⟨proof⟩

**lemma** *gt\_antisym*:  $t >_t s \implies \neg s >_t t$   
 ⟨proof⟩

## 4.5 Subterm Property

**lemma** *gt\_emb\_fun*:  $App\ s\ t >_t s$   
*<proof>*

**lemma** *gt\_emb\_arg*:  $App\ s\ t >_t t$   
*<proof>*

## 4.6 Compatibility with Contexts

**lemma** *gt\_compat\_fun*:  
 **assumes**  $t' >_t t$   
 **shows**  $App\ s\ t' >_t App\ s\ t$   
*<proof>*

**theorem** *gt\_compat\_arg*:  
 **shows**  $s' >_t s \implies t' \geq_t t \implies App\ s'\ t' >_t App\ s\ t$   
*<proof>*

**theorem** *gt\_compat\_fun\_strong*:  
 **assumes**  $t'_gt_t: t' >_t t$   
 **shows**  $apps\ s\ (t' \# us) >_t apps\ s\ (t \# us)$   
*<proof>*

**theorem** *gt\_or\_eq\_compat\_App*:  $s' \geq_t s \implies t' \geq_t t \implies App\ s'\ t' \geq_t App\ s\ t$   
*<proof>*

**theorem** *gt\_compat\_App*:  
 **shows**  $s' \geq_t s \implies t' >_t t \implies App\ s'\ t' >_t App\ s\ t$   
*<proof>*

## 4.7 Compatibility with Embedding Relation

**lemma** *gt\_embedding\_step\_property*:  
 **assumes**  $t \rightarrow_{emb} s$   
 **shows**  $t >_t s$   
*<proof>*

**lemma** *gt\_embedding\_property*:  
 **assumes**  $t \succeq_{emb} s\ t \neq s$   
 **shows**  $t >_t s$   
*<proof>*

## 4.8 Stability under Substitutions

**lemma** *extf\_map2*:  
 **assumes**  
  $\forall y \in set\ ys \cup set\ xs. \forall x \in set\ ys \cup set\ xs. y >_t x \longrightarrow (h\ y) >_t (h\ x)$   
 *extf*  $f (>_t) ys\ xs$   
 **shows**  
 *extf*  $f (>_t) (map\ h\ ys) (map\ h\ xs)$   
*<proof>*

**theorem** *gt\_sus*:  
 **assumes**  $\varrho\_wary: wary\_subst\ \varrho$   
 **assumes** *ghd*:  $\bigwedge x. ground\_heads\ (Var\ x) = UNIV$   
 **shows**  $t >_t s \implies subst\ \varrho\ t >_t subst\ \varrho\ s$   
*<proof>*

## 4.9 Totality on Ground Terms

**theorem** *gt\_total\_ground*:  
 **assumes** *extf\_total*:  $\bigwedge f. ext\_total\ (extf\ f)$   
 **shows**  $ground\ t \implies ground\ s \implies t >_t s \vee s >_t t \vee t = s$

*<proof>*

## 4.10 Well-foundedness

**lemma** *gt\_imp\_vars*:  $t >_t s \implies \text{vars } t \supseteq \text{vars } s$

*<proof>*

**abbreviation** *gtg* ::  $('s, 'v) \text{tm} \Rightarrow ('s, 'v) \text{tm} \Rightarrow \text{bool}$  (**infix**  $>_{tg}$  50) **where**

$(>_{tg}) \equiv \lambda t s. \text{ground } t \wedge t >_t s$

**theorem** *gt\_wf*:

**assumes** *ghd\_UNIV*:  $\bigwedge x. \text{ground\_heads\_var } x = \text{UNIV}$

**assumes** *extf\_wf*:  $\bigwedge f. \text{ext\_wf } (\text{extf } f)$

**shows** *wfP*  $(\lambda s t. t >_t s)$

*<proof>*

**end**

**end**