


A Higher-Order Iterative Path Ordering

View metadata, citation and similar papers at core.ac.uk

brought to you by  CORE

provided by DSpace at VU

De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
kop@few.vu.nl, femke@few.vu.nl

Abstract. The higher-order recursive path ordering (HORPO) defined by Jouannaud and Rubio provides a method to prove termination of higher-order rewriting. We present an iterative version of HORPO by means of an auxiliary term rewriting system, following an approach originally due to Bergstra and Klop. We study well-foundedness of the iterative definition, discuss its relationship with the original HORPO, and point out possible ways to strengthen the ordering.

1 Introduction

This paper is about termination of higher-order rewriting, where bound variables may be present. An important method to prove termination of first-order term rewriting is provided by the recursive path ordering (RPO) defined by Dershowitz [4]. Jouannaud and Rubio [6] define the higher-order recursive path ordering (HORPO) which extends RPO to the higher-order case. The starting point is a well-founded ordering on the function symbols which is lifted to a relation \succ on terms, such that if $l \succ^+ r$ for every rewrite rule $l \rightarrow r$, then rewriting terminates.

Klop, van Oostrom and de Vrijer [10] present, following an approach originally due to Bergstra and Klop [1], the iterative lexicographic path ordering (ILPO) by means of an auxiliary term rewriting system. ILPO can be understood as an iterative definition of the lexicographic path ordering (LPO), a variant of RPO [9]. They show that ILPO is well-founded, and that ILPO and LPO coincide if the underlying relation on function symbols is transitive.

The starting point of the present work is the question whether also for HORPO an iterative definition can be given. We present HOIPO, a higher-order iterative path ordering, which is defined by means of an auxiliary (higher-order) term rewriting system, following the approach of [10]. HOIPO can be considered as an extension of ILPO obtained by a generalization to the higher-order case and the addition of comparing arguments as multisets.

We show well-foundedness of HOIPO as in [6] using the notion of computability and the proof technique due to Buchholz [3], see also [5]. It then follows that HOIPO provides a method for proving termination of higher-order rewriting. Further, we show that HOIPO includes HORPO but not vice versa. So HOIPO is (slightly) stronger than HORPO as a termination method; the reason is that the fine-grained approach permits one to postpone the choice of smaller terms.

2 Preliminaries

In this paper we mainly consider higher-order rewriting as defined by Jouannaud and Okada [5], also called Algebraic Functional Systems (AFSs) [16, Chapter 11]. The terms are simply typed λ -terms with typed constants. Every system has β as one of its rewrite rules. That is, in AFSs we do not work modulo β as for instance in HRSs [12]. Below we recall the main definitions.

Definition 1 (Types). We assume a set \mathcal{S} of *sorts* also called *base types*. The set \mathcal{T} of *simple types* is defined by the grammar $\mathcal{T} ::= \mathcal{S} \mid (\mathcal{T} \rightarrow \mathcal{T})$.

Types are denoted by $\sigma, \tau, \rho, \dots$. As usual \rightarrow associates to the right and we omit outermost parentheses. A *type declaration* is an expression of the form $(\sigma_1 \times \dots \times \sigma_m) \rightarrow \sigma$ with $\sigma_1, \dots, \sigma_m$ and σ types. If $m = 0$ then such a type declaration is shortly written as σ . Type declarations are not types, but are used for typing terms. In the remainder of this paper we assume a set \mathcal{V} of typed variables, denoted by $x : \sigma, y : \tau, z : \rho, \dots$, with countably many variables of every type. In the following definitions we assume in addition a set \mathcal{F} of function symbols, each equipped with a unique type declaration, denoted by $f : (\sigma_1 \times \dots \times \sigma_m) \rightarrow \sigma, g : (\tau_1 \times \dots \times \tau_n) \rightarrow \tau, \dots$

Definition 2 (Terms). The set $\mathbb{T}(\mathcal{F}, \mathcal{V})$ of *terms* over \mathcal{F} and \mathcal{V} is the smallest set consisting of all expressions s for which we can infer $s : \sigma$ for some type σ using the following clauses:

- (var) $x : \sigma$ if $x : \sigma \in \mathcal{V}$
- (app) $@(u, t) : \sigma$ if $u : \tau \rightarrow \sigma$ and $t : \tau$
- (abs) $\lambda x : \tau. t : \rho$ if $x : \tau \in \mathcal{V}$ and $t : \rho$
- (fun) $f(s_1, \dots, s_m) : \sigma$ if $f : (\sigma_1 \times \dots \times \sigma_m) \rightarrow \sigma \in \mathcal{F}$
and $s_1 : \sigma_1, \dots, s_m : \sigma_m$

The application of n terms is sometimes written as $@(t_1, \dots, t_n)$; here $n \geq 2$ and t_1 may be an application itself. Note that a function symbol $f : (\sigma_1 \times \dots \times \sigma_m) \rightarrow \sigma$ must get exactly m arguments, and that σ is not necessarily a base type. Occurrences of x in t in the term $\lambda x : \tau. t$ are bound. We consider equality on terms modulo α -conversion, denoted by $=$. If we want to mention explicitly the type of a (sub)term then we write $s : \sigma$ instead of simply s .

A *substitution* $[x := s]$, with \mathbf{x} and \mathbf{s} finite vectors of equal length, is the homomorphic extension of the type-preserving mapping $\mathbf{x} \mapsto \mathbf{s}$ from variables to terms. Substitutions are denoted by γ, δ, \dots , and the result of applying the substitution γ to the term s is denoted by $s\gamma$. Substitutions do not capture free variables; we assume that bound variables are renamed if necessary.

Definition 3 (Rewrite rule). A *rewrite rule* over \mathcal{F} and \mathcal{V} is a pair of terms (l, r) in $\mathbb{T}(\mathcal{F}, \mathcal{V})$ of the same type, such that all free variables of r occur in l .

A rewrite rule (l, r) is usually written as $l \rightarrow r$. A *higher-order rewrite system* is specified by a set \mathcal{F} of function symbols with type declarations, and a set of rewrite rules over \mathcal{F} and \mathcal{V} . The rewrite rules induce a rewrite relation which is defined as follows; note that matching is modulo α , not modulo $\alpha\beta$ nor $\alpha\beta\eta$.

Definition 4 (Rewrite relations). Given a set of rewrite rules \mathcal{R} over \mathcal{F} and \mathcal{V} , the rewrite relation $\rightarrow_{\mathcal{R}}$ is defined by the following clauses:

(head) $l\gamma \rightarrow_{\mathcal{R}} r\gamma$	if $l \rightarrow r \in \mathcal{R}$ and γ a substitution
(fun) $f(s_1, \dots, s_i, \dots, s_n) \rightarrow_{\mathcal{R}} f(s_1, \dots, s'_i, \dots, s_n)$	if $s_i \rightarrow_{\mathcal{R}} s'_i$
(app-l) $@(s, t) \rightarrow_{\mathcal{R}} @(s', t)$	if $s \rightarrow_{\mathcal{R}} s'$
(app-r) $@(s, t) \rightarrow_{\mathcal{R}} @(s, t')$	if $t \rightarrow_{\mathcal{R}} t'$
(abs) $\lambda x : \sigma. s \rightarrow_{\mathcal{R}} \lambda x : \sigma. s'$	if $s \rightarrow_{\mathcal{R}} s'$

The β -reduction relation, denoted by \rightarrow_{β} , is induced by the β -reduction rule $@(\lambda x : \sigma. s, t) \rightarrow_{\beta} s[x := t]$. The *rewrite relation* of $(\mathcal{F}, \mathcal{R})$, denoted by \rightarrow , is defined as the union of $\rightarrow_{\mathcal{R}}$ and β -reduction: $\rightarrow = \rightarrow_{\mathcal{R}} \cup \rightarrow_{\beta}$. As usual, the transitive closure of \rightarrow is denoted by \rightarrow^+ and the reflexive-transitive closure of \rightarrow is denoted by \rightarrow^* .

Example 1 (Recursor). The rewrite system **Rec** for recursor on natural numbers uses a base type \mathbb{N} and function symbols $0 : \mathbb{N}$, $S : (\mathbb{N}) \rightarrow \mathbb{N}$, $\text{rec} : (\mathbb{N} \times \mathbb{N} \times (\mathbb{N} \rightarrow \mathbb{N})) \rightarrow \mathbb{N}$. The rewrite rules of **Rec** are as follows:

$$\begin{aligned} \text{rec}(0, y, z) &\rightarrow y \\ \text{rec}(S(x), y, z) &\rightarrow @(z, x, \text{rec}(x, y, z)) \end{aligned}$$

Addition of natural numbers can now be represented by the following term:

$$\lambda x : \mathbb{N}. \lambda y : \mathbb{N}. \text{rec}(x, y, \lambda u : \mathbb{N}. \lambda v : \mathbb{N}. S(v))$$

Example 2 (Map). The rewrite system **Map** uses base types \mathbb{N} for natural numbers, and **natlist** for lists of natural numbers. The function symbols are $\text{nil} : \text{natlist}$, $\text{cons} : (\mathbb{N} \times \text{natlist}) \rightarrow \text{natlist}$, $\text{map} : (\text{natlist} \times (\mathbb{N} \rightarrow \mathbb{N})) \rightarrow \text{natlist}$. The rewrite rules of **Map** are as follows:

$$\begin{aligned} \text{map}(\text{nil}, z) &\rightarrow \text{nil} \\ \text{map}(\text{cons}(h, t), z) &\rightarrow \text{cons}(@(z, h), \text{map}(t, z)) \end{aligned}$$

3 The Higher-Order Recursive Path Ordering

The starting point of the recursive path ordering due to Dershowitz [4] is a well-founded ordering on the function symbols, which is lifted to a reduction ordering \succ_{rpo} on the set of terms. That is, the rewriting system is terminating if $l \succ_{rpo} r$ for every rewrite rule $l \rightarrow r$. Jouannaud and Rubio [6] present an extension of RPO to the higher-order case, here called HORPO. Below we recall the definition of HORPO and in the next section we introduce its fine-grained iterative version.

We have chosen to work with the definition as in [6] and not with later versions as for instance [8,2]; we will come back to this issue in the last section.

In the following, multisets are denoted by $\{\{ \dots \}\}$. If $>$ is a binary relation, then the *multiset extension* of $>$, denoted by $>_{MUL}$, is defined as follows: $XUY >_{MUL} X \cup Z$, with \cup the disjoint union of multisets, if $Y \neq \emptyset$ and $\forall z \in Z. \exists y \in Y. y > z$.

Sequences are denoted by $[..]$. The *lexicographic extension* of $>$, denoted by $>_{LEX}$, is defined as follows: $[s_1, \dots, s_m] >_{LEX} [s'_1, \dots, s'_m]$ if either $s_1 > s'_1$ or $s_1 = s'_1$ and $[s_2, \dots, s_m] >_{LEX} [s'_2, \dots, s'_m]$. If $>$ is a well-founded relation, then both $>_{MUL}$ and $>_{LEX}$ are well-founded.

We assume that the set of function symbols \mathcal{F} is the disjoint union of \mathcal{F}_{MUL} and \mathcal{F}_{LEX} . If $f \in \mathcal{F}_{MUL}$ then its arguments will be compared with the multiset extension of HORPO, and if $f \in \mathcal{F}_{LEX}$ then its arguments will be compared with the lexicographic extension of HORPO. We assume a well-founded precedence \triangleright on \mathcal{F} . Finally, in the remainder we identify all base types.

Definition 5 (HORPO). We have $s \succ t$ for terms $s : \sigma$ and $t : \sigma$ if one of the following conditions holds:

- (H1) $s = f(s_1, \dots, s_m)$
there is an $i \in \{1, \dots, m\}$ such that $s_i \succeq t$
- (H2) $s = f(s_1, \dots, s_m)$, $t = g(t_1, \dots, t_n)$
 $f \triangleright g$
 $s \succ \{t_1, \dots, t_n\}$
- (H3LEX) $s = f(s_1, \dots, s_m)$, $t = f(t_1, \dots, t_m)$, $f \in \mathcal{F}_{LEX}$
 $[s_1, \dots, s_m] \succ_{LEX} [t_1, \dots, t_m]$
 $s \succ \{t_1, \dots, t_m\}$
- (H3MUL) $s = f(s_1, \dots, s_m)$, $t = f(t_1, \dots, t_m)$, $f \in \mathcal{F}_{MUL}$
 $\{\{s_1, \dots, s_m\}\} \succ_{MUL} \{\{t_1, \dots, t_m\}\}$
- (H4) $s = f(s_1, \dots, s_m)$, $t = @ (t_1, \dots, t_n)$ with $n \geq 2$
 $s \succ \{t_1, \dots, t_n\}$
- (H5) $s = @(s_1, s_2)$, $t = @(t_1, t_2)$
 $\{\{s_1, s_2\}\} \succ_{MUL} \{\{t_1, t_2\}\}$
- (H6) $s = \lambda x : \sigma. s_0$, $t = \lambda x : \sigma. t_0$
 $s_0 \succ t_0$

Here \succeq denotes the reflexive closure of \succ , and \succ_{MUL} and \succ_{LEX} denote the multiset and lexicographic extension of \succ . Further, following the notation from [11], the relation \succ between a functional term and a set of terms is defined as follows: $s = f(s_1, \dots, s_m) \succ \{t_1, \dots, t_n\}$ if for every $i \in \{1, \dots, n\}$ we have either $s \succ t_i$, or there exists $j \in \{1, \dots, m\}$ such that $s_j \succeq t_i$.

The first four clauses of the definition stem directly from the first-order definition of RPO, with the difference that instead of the requirement $s \succ \{t_1, \dots, t_n\}$ for HORPO, we have for RPO the simpler $s \succ t_i$ for every i . This is not possible for the higher-order case because of the type requirements; the relation \succ is only defined on terms of equal type (after identifying all base types).

Jouannaud and Rubio prove well-foundedness of $\succ \cup \rightarrow_\beta$. HORPO provides a method to prove termination of higher-order rewriting: a higher-order rewrite system $(\mathcal{F}, \mathcal{R})$ is terminating if $l \succ r$ for every rewrite rule $l \rightarrow r \in \mathcal{R}$.

Example 3 (Recursor). We prove termination of the recursor on natural numbers using HORPO. For the first rewrite rule, we have $\text{rec}(0, y, z) \succ y$ by (H1). We use (H4) to show $\text{rec}(S(x), y, z) \succ @(z, x, \text{rec}(x, y, z))$. There are three remaining

proof obligations. First, we have $z \succeq z$ by reflexivity of \succeq . Second, we have $S(x) \succeq x$ by clause (H1). Third, we have $\text{rec}(S(x), y, z) \succ \text{rec}(x, y, z)$ by assuming rec to be a lexicographic function symbol and using again $S(x) \succ x$, and reflexivity.

Example 4 (Map). The first rewrite rule is oriented using (H1). In order to orient the second rewrite rule we apply (H2) with $\text{map} \triangleright \text{cons}$. Then first we need to show $\text{map}(\text{cons}(h, t), z) \succ @ (z, h)$ which follows from (H4). Second we need to show $\text{map}(\text{cons}(h, t), z) \succ \text{map}(t, z)$ using (H3MUL). (Alternatively one can assume map to be a lexicographic function symbol.) Note that in this example we need the collapse of all base types to one.

4 An Iterative Version of HORPO

In this section we present an iterative version of HORPO, called HOIPO. HOIPO is defined by means of a term rewriting system that intuitively step by step transforms a term into a term that is smaller with respect to HORPO. We will add marked function symbols: if \mathcal{F} is a set of function symbols, then \mathcal{F}^* is defined to be a copy of \mathcal{F} which contains for every $f \in \mathcal{F}$ a symbol f^* with the same type declaration. We follow the approach and notations as in [10].

Definition 6 (HOIPO). We assume a set of function symbols \mathcal{F} divided in \mathcal{F}_{MUL} and \mathcal{F}_{LEX} , and a relation \triangleright on \mathcal{F} . The rewriting system $\mathcal{H}(\mathcal{F}, \triangleright)$ uses function symbols in $\mathcal{F} \cup \mathcal{F}^*$ and contains the following rules:

$$\begin{array}{ll}
f(x_1, \dots, x_m) \rightarrow_{\text{put}} f^*(x_1, \dots, x_m) & \\
f^*(x_1, \dots, x_m) \rightarrow_{\text{select}} x_i & \text{(a)} \\
f^*(x_1, \dots, x_m) \rightarrow_{\text{copy}} g(l_{\tau_1}, \dots, l_{\tau_n}) & \text{(b)(f)} \\
f^*(x_1, \dots, s_i, \dots, x_m) \rightarrow_{\text{lex}} f(x_1, \dots, x_{i-1}, s'_i, l_{\sigma_{i+1}}, \dots, l_{\sigma_m}) & \text{(c)(d)(f)} \\
f^*(x_1, \dots, s_i, \dots, x_m) \rightarrow_{\text{mul}} f(r_1, \dots, r_{i-1}, s'_i, r_{i+1}, \dots, r_m) & \text{(c)(e)(g)} \\
f^*(x_1, \dots, x_m) \rightarrow_{\text{ord}} f^*(x_{\pi^{-1}1}, \dots, x_{\pi^{-1}m}) & \text{(e)(h)} \\
f^*(x_1, \dots, x_m) \rightarrow_{\text{appl}} @(l_{\rho_1 \rightarrow \dots \rightarrow \rho_k \rightarrow \sigma}, l_{\rho_1}, \dots, l_{\rho_k}) & \text{(f)}
\end{array}$$

We assume that the typing and arity constraints are met (after identifying all base types). In particular the left- and right-hand sides of rewrite rules must have the same type, and $f : (\sigma_1 \times \dots \times \sigma_m) \rightarrow \sigma$, and $g : (\tau_1 \times \dots \times \tau_n) \rightarrow \sigma$. Further, the rules are subject to the following conditions:

- (a) $i \in \{1, \dots, m\}$,
- (b) $f \triangleright g$,
- (c) $s_i \rightarrow_{\text{put}} s'_i$,
- (d) $f \in \mathcal{F}_{\text{LEX}}$,
- (e) $f \in \mathcal{F}_{\text{MUL}}$,
- (f) we use the notation l_ρ for some term of type ρ ; either $l_\rho = f^*(x_1, \dots, x_m)$ or $l_\rho = x_j$ for some j , as long as the type constraints are met; it is not a fixed term: we can choose different values for l_{σ_i} and l_{σ_j} even if $\sigma_i = \sigma_j$,
- (g) we use the notation r_j for some term of type σ_j : either $s_i \rightarrow_{\text{put}} r_j$, or $r_j = x_j$
- (h) π a type-preserving permutation of $1, \dots, m$.

The first four rewrite rules stem directly from the first-order term rewriting system $\mathcal{L}ex$ defined in [10] which is used to define ILPO, an iterative definition of the lexicographic path order. They are adapted because of the typing constraints, just as the clauses (H1), (H2), (H3MUL), (H3LEX) in the definition of HORPO are typed versions of the clauses of the definition of first-order RPO. We now first discuss the intuitive meaning of the rewrite rules of HOIPO and then give some examples.

- The put-rule can be considered as the start of a proof obligation for HORPO. It expresses the intention to make a functional term smaller. It is exactly the same as the put-rule of the first-order rewriting system for ILPO.
- The select-rule expresses that selecting a direct argument of a functional term makes it smaller. It roughly corresponds to clause (H1) of the definition of HORPO. It is exactly the same as the select-rule in the rewriting system for ILPO. However, the use of the rule in the higher-order setting is weaker because of the typing constraints. For instance, with $f : (o \rightarrow o) \rightarrow o$, $g : o \rightarrow (o \rightarrow o)$, $a : o$, we cannot reduce $f(g(a)) : o$ to $a : o$ in \mathcal{H} using the rules put and select, because we would need to go via $g(a)$ which has type $o \rightarrow o$. In the first-order setting we have $f(g(a)) \rightarrow_{\text{put}} f^*(g(a)) \rightarrow_{\text{select}} g(a) \rightarrow_{\text{put}} g^*(a) \rightarrow_{\text{select}} a$.
- The copy-rule makes copies of the original term under a function symbol that is smaller with respect to \triangleright . This corresponds to clause (H2) of the definition of HORPO. The choice for l_{σ_i} in the right-hand side of the rule corresponds to the \succ relation used in (H2). The first-order version of the rule is

$$f^*(\mathbf{x}) \rightarrow_{\text{copy}} g(f^*(\mathbf{x}), \dots, f^*(\mathbf{x})) \quad (\text{if } f \triangleright g)$$

There the left-hand side is copied at all argument positions of g , which cannot be done in the higher-order case because of the typing constraints.

- The lex-rule implements the lexicographic extension of HORPO and can be applied to lexicographic functional terms only. The first $i - 1$ arguments are not changed. The i th argument is marked, meaning we will make it smaller. The arguments $i + 1$ till m may increase, but are bounded by the left-hand side. That is, on those positions we put either the original term (left-hand side) or a direct argument. The first-order version of this rewrite rule is:

$$f^*(\mathbf{x}, g(\mathbf{y}), \mathbf{z}) \rightarrow_{\text{lex}} f(\mathbf{x}, g^*(\mathbf{y}), l, \dots, l) \quad (\text{for } l = f^*(\mathbf{x}, g(\mathbf{y}), \mathbf{z}))$$

Here the put-reduct of the argument $g(\mathbf{y})$ can be given directly, and at all positions thereafter the left-hand side can be put. For the higher-order case this is not possible because instead of a functional term we can also have an application or an abstraction.

- The mul- and ord-rule implement the multiset extension of HORPO and can be applied to multiset functional terms only. With $a \triangleright b$ we have for instance $f(x, a, c) \rightarrow_{\text{put}} f^*(x, a, c) \rightarrow_{\text{ord}} f^*(x, c, a) \rightarrow_{\text{mul}} f(a^*, c, a^*) \rightarrow_{\text{copy}} f(b, c, a^*) \rightarrow_{\text{copy}} f(b, c, b)$. We cannot reduce $f(x, a, c)$ to $f(b, c, b)$ using the first-order rules put, select, copy, and lex, so the mul-rule cannot be derived from those rules.

The ord-rule expresses that the order of the arguments of a multiset functional term do not matter (but remain subject to the typing constraints). This rule does not express a decrease in HORPO.

- The appl-rule corresponds to clause (H4) of the definition of HORPO and is typical for the higher-order case. The idea is that the application of terms that are all smaller than the original term is smaller than the original term. We have $l_{\rho_1 \rightarrow \dots \rightarrow \rho_k \rightarrow \sigma} = x_i$ and $\rho_1 \rightarrow \dots \rightarrow \rho_k \rightarrow \sigma = \sigma_i$ for some i .

The rewrite relation induced by the rules of $\mathcal{H}(\mathcal{F}, \triangleright)$ is denoted by $\rightarrow_{\mathcal{H}}$, and the union of $\rightarrow_{\mathcal{H}}$ and β -reduction is denoted by $\rightarrow_{\mathcal{H}\beta}$. How can HOIPO be used to prove termination? The claim is that a system $(\mathcal{F}, \mathcal{R})$ is terminating if we have $l \rightarrow_{\mathcal{H}\beta}^+ r$ for every rewrite rule $l \rightarrow r \in \mathcal{R}$. This is proved in the following section; here we first look at three examples of the use of HOIPO.

Example 5 (Recursor). For the first rewrite rule we have:

$$\text{rec}(0, y, z) \rightarrow_{\text{put}} \text{rec}^*(0, y, z) \rightarrow_{\text{select}} y$$

For the second rewrite rule we assume $\text{rec} \in \mathcal{F}_{\text{LEX}}$. Then:

$$\begin{aligned} \text{rec}(\mathbf{S}(x), y, z) &\rightarrow_{\text{put}} \text{rec}^*(\mathbf{S}(x), y, z) \rightarrow_{\text{appl}} @ (z, \mathbf{S}(x), \text{rec}^*(\mathbf{S}(x), y, z)) \rightarrow_{\text{put}} \\ @ (z, \mathbf{S}^*(x), \text{rec}^*(\mathbf{S}(x), y, z)) &\rightarrow_{\text{select}} @ (z, x, \text{rec}^*(\mathbf{S}(x), y, z)) \rightarrow_{\text{lex}} \\ @ (z, x, \text{rec}(\mathbf{S}^*(x), y, z)) &\rightarrow_{\text{select}} @ (z, x, \text{rec}(x, y, z)) \end{aligned}$$

Example 6 (Map). We take $\text{map} \triangleright \text{cons}$ and $\text{map} \in \mathcal{F}_{\text{LEX}}$. For the first rewrite rule we have:

$$\text{map}(\text{nil}, z) \rightarrow_{\text{put}} \text{map}^*(\text{nil}, z) \rightarrow_{\text{select}} \text{nil}$$

For the second rewrite rule we have (base types are identified):

$$\begin{aligned} \text{map}(\text{cons}(h, t), z) &\rightarrow_{\text{put}} \text{map}^*(\text{cons}(h, t), z) \rightarrow_{\text{copy}} \\ \text{cons}(\text{map}^*(\text{cons}(h, t), z), \text{map}^*(\text{cons}(h, t), z)) &\rightarrow_{\text{appl}} \\ \text{cons}(@ (z, \text{cons}(h, t)), \text{map}^*(\text{cons}(h, t), z)) &\rightarrow_{\text{put}} \\ \text{cons}(@ (z, \text{cons}^*(h, t)), \text{map}^*(\text{cons}(h, t), z)) &\rightarrow_{\text{select}} \\ \text{cons}(@ (z, h), \text{map}^*(\text{cons}(h, t), z)) &\rightarrow_{\text{lex}} \text{cons}(@ (z, h), \text{map}(\text{cons}^*(h, t), z)) \rightarrow_{\text{select}} \\ \text{cons}(@ (z, h), \text{map}(t, z)) &\end{aligned}$$

5 Termination

In this section we prove the following theorem:

Theorem 1. *A system $(\mathcal{F}, \mathcal{R})$ is terminating if there exists a well-founded ordering \triangleright on the terms over \mathcal{F} such that $l \rightarrow_{\mathcal{H}\beta}^+ r$ in $\mathcal{H}(\mathcal{F}, \triangleright)$.*

This means that HOIPO provides a termination method, as was already claimed in the previous section. The proof of Theorem 1 proceeds as follows; we follow the approaches of [10,6]. We define a labelled rewrite relation $\rightarrow_{\mathcal{H}\omega}$ (Definition 7) and consider its union with β -reduction, denoted by $\rightarrow_{\mathcal{H}\omega\beta}$. It is shown that

$\rightarrow_{\mathcal{H}\omega\beta}^+$ and $\rightarrow_{\mathcal{H}\beta}^+$ coincide on the set of terms over \mathcal{F} , so without marks or labels (Lemma 1). Then we show termination of $\rightarrow_{\mathcal{H}\omega\beta}$ (Theorem 3). It follows that $\rightarrow_{\mathcal{H}\beta}^+$ is a transitive relation on the terms over \mathcal{F} , that is closed under contexts and substitutions, and that is moreover well-founded. Hence it is a reduction ordering, that is, $l \rightarrow_{\mathcal{H}\beta}^+ r$ for every rewrite rule $l \rightarrow r$ implies that rewriting is terminating. HOIPO is more fine-grained than HORPO and also its termination proof is more fine-grained than the one for HORPO. We continue by presenting the labelled version of HOIPO, which is used to prove termination of $\rightarrow_{\mathcal{H}\beta}$.

5.1 The Labelled System

We assume a set \mathcal{F} of function symbols, which is the disjoint union of lexicographic and multiset function symbols, and a well-founded ordering \triangleright on \mathcal{F} . We define a copy \mathcal{F}^ω of \mathcal{F} as follows: for every $f \in \mathcal{F}$, the set \mathcal{F}^ω contains the labelled function symbol f^n for every natural number n . An unlabelled function symbol f is also denoted as f^ω ; then every function symbol in $\mathcal{F} \cup \mathcal{F}^\omega$ can be denoted by f^α with α an ordinal of at most ω . The usual ordering on \mathbb{N} is extended by $n < \omega$ for every $n \in \mathbb{N}$.

Definition 7 (Labelled HOIPO). The rewriting system $\mathcal{H}\omega(\mathcal{F}, \triangleright)$ uses function symbols in $\mathcal{F} \cup \mathcal{F}^\omega$ and contains the following rules:

$$\begin{array}{ll}
f^\omega(x_1, \dots, x_m) \rightarrow_{\text{put}} f^p(x_1, \dots, x_m) & \text{(a)} \\
f^p(x_1, \dots, x_m) \rightarrow_{\text{select}} x_i & \text{(b)} \\
f^{p+1}(x_1, \dots, x_m) \rightarrow_{\text{copy}} g^\omega(l_{\tau_1}, \dots, l_{\tau_n}) & \text{(f)} \\
f^{p+1}(x_1, \dots, s_i, \dots, x_m) \rightarrow_{\text{lex}} f^\omega(x_1, \dots, x_{i-1}, s'_i, l_{\sigma_{i+1}}, \dots, l_{\sigma_m}) & \text{(c)(d)(f)} \\
f^{p+1}(x_1, \dots, s_i, \dots, x_m) \rightarrow_{\text{mul}} f^\omega(r_1, \dots, r_{i-1}, s'_i, r_{i+1}, \dots, r_m) & \text{(c)(e)(g)} \\
f^{p+1}(x_1, \dots, x_m) \rightarrow_{\text{ord}} f^p(x_{\pi^{-1}1}, \dots, x_{\pi^{-1}m}) & \text{(e)(h)} \\
f^{p+1}(x_1, \dots, x_m) \rightarrow_{\text{appl}} @ (l_{\rho_1 \rightarrow \dots \rightarrow \rho_k \rightarrow \sigma}, l_{\rho_1}, \dots, l_{\rho_k}) & \text{(f)}
\end{array}$$

Here p is an arbitrary natural number. As in Definition 6 we assume that the type- and arity constraints are met. In addition we have the following conditions; here (a), (b), (d), (e), and (h) are exactly the same as in Definition 6, and (c), (f) and (g), provide the crucial differences.

- (a) $i \in \{1, \dots, m\}$,
- (b) $f \triangleright g$,
- (c) $s_i \rightarrow_{\text{put}} s'_i$ with label p
- (d) $f \in \mathcal{F}_{\text{LEX}}$,
- (e) $f \in \mathcal{F}_{\text{MUL}}$,
- (f) we use the notation l_ρ for some term of type ρ ; either $l_\rho = f^p(x_1, \dots, x_m)$ or $l_\rho = x_j$ for some j , as long as the type constraints are met; it is not a fixed term: we can choose different values for l_{σ_i} and l_{σ_j} even if $\sigma_i = \sigma_j$,
- (g) we use the notation r_j for some term of type σ_j : either $s_i \rightarrow_{\text{put}} r_j$ with label p , or $r_j = x_j$,
- (h) π a type-preserving permutation of $1, \dots, m$.

The rewrite relation induced by the rewrite rules of $\mathcal{H}\omega(\mathcal{F}, \triangleright)$ is denoted by $\rightarrow_{\mathcal{H}\omega}$, and the union of $\rightarrow_{\mathcal{H}\omega}$ and β -reduction is denoted by $\rightarrow_{\mathcal{H}\omega\beta}$. The following examples illustrate that in $\mathcal{H}\omega$ the labels provide more control over the reduction relation than the marks.

Example 7 (Map). We take $\text{map} \triangleright \text{cons}$ and $\text{map} \in \mathcal{F}_{\text{LEX}}$. For the first rewrite rule we have:

$$\text{map}(\text{nil}, z) \rightarrow_{\text{put}} \text{map}^0(\text{nil}, z) \rightarrow_{\text{select}} \text{nil}$$

For the second rewrite rule we have:

$$\begin{aligned} & \text{map}(\text{cons}(h, t), z) \rightarrow_{\text{put}} \text{map}^2(\text{cons}(h, t), z) \rightarrow_{\text{copy}} \\ & \text{cons}(\text{map}^1(\text{cons}(h, t), z), \text{map}^1(\text{cons}(h, t), z)) \rightarrow_{\text{appl}} \\ & \text{cons}(@z, \text{cons}(h, t)), \text{map}^1(\text{cons}(h, t), z)) \rightarrow_{\text{put}} \\ & \text{cons}(@z, \text{cons}^0(h, t)), \text{map}^1(\text{cons}(h, t), z)) \rightarrow_{\text{select}} \\ & \text{cons}(@z, h), \text{map}^1(\text{cons}(h, t), z)) \rightarrow_{\text{lex}} \text{cons}(@z, h), \text{map}(\text{cons}^0(h, t), z)) \rightarrow_{\text{select}} \\ & \text{cons}(@z, h), \text{map}(t, z) \end{aligned}$$

Example 8. As remarked in [10], the rewriting system $\rightarrow_{\mathcal{H}}$ may contain loops. For instance, for $a : \sigma$ and $f : \sigma \rightarrow \sigma$ with $a \triangleright f$ we have $a \rightarrow_{\text{put}} a^* \rightarrow_{\text{copy}} f(a^*) \rightarrow_{\text{put}} f^*(a^*) \rightarrow_{\text{select}} a^*$. The loop on marked terms has no counterpart in the labelled system.

We now show that the reflexive closures $\rightarrow_{\mathcal{H}\beta}^+$ and $\rightarrow_{\mathcal{H}\omega\beta}^+$ coincide on the set of terms without labels.

Lemma 1. *We have $\rightarrow_{\mathcal{H}\beta}^+ = \rightarrow_{\mathcal{H}\omega\beta}^+$ on the set of terms over \mathcal{F} .*

Proof. First note that the marks and labels do not control β -reduction.

In order to show $\rightarrow_{\mathcal{H}\beta}^+ \subseteq \rightarrow_{\mathcal{H}\omega\beta}^+$ we consider a rewrite sequence $s \rightarrow_{\mathcal{H}\beta}^+ t$ that does not consist of β -reduction steps only. The first step with respect to HOIPO is induced by the put-rule. The total number of HOIPO steps is finite, say n . We now lift the rewrite sequence in $\mathcal{H}\beta$ to a rewrite sequence in $\mathcal{H}\omega\beta$ by using in the first put-step of the latter the label n . The values for the other labels then follow easily.

In order to show $\rightarrow_{\mathcal{H}\omega\beta}^+ \subseteq \rightarrow_{\mathcal{H}\beta}^+$, note that a step in labelled HOIPO is mapped to a step in HOIPO by replacing a label p (for any natural number p) by the mark $*$. A label ω is just a notational device and in fact denotes ‘no label’. \square

5.2 Computability

We will make use of the notion of computability due to Tait and Girard [15]. Here we define computability with respect to $\rightarrow_{\mathcal{H}\omega\beta}$.

Definition 8. A term $s : \sigma$ is said to be *computable* with respect to $\rightarrow_{\mathcal{H}\omega\beta}$ if:

- σ is a base type, and s is strongly normalizing with respect to $\rightarrow_{\mathcal{H}\omega\beta}$,
- $\sigma = \tau \rightarrow \rho$ for some τ and ρ , and for every $t : \tau$ with t computable with respect to $\rightarrow_{\mathcal{H}\omega\beta}$ we have that $@(s, t)$ is computable with respect to $\rightarrow_{\mathcal{H}\omega\beta}$.

As in [6,11] variables are not by definition computable, but are proved to be computable. We do not consider computability modulo a convertibility relation on terms as in [11] because we work with typed variables and not with environments. The following two lemma's are concerned with (standard) properties of computability and correspond to Property 3.4 (i), (ii), (iii), (v) in [6] and to Lemma 6.3 and Lemma 6.5 in [11].

Lemma 2

- (a) *If $s : \sigma$ is computable then $s : \sigma$ is strongly normalizing with respect to $\rightarrow_{\mathcal{H}\omega\beta}$.*
- (b) *If $s : \sigma$ is computable and $s \rightarrow_{\mathcal{H}\omega\beta} t$ then t is computable.*
- (c) *If $s : \sigma$ is not an abstraction (or: $s : \sigma$ is neutral) and t is computable for every t with $s \rightarrow_{\mathcal{H}\omega\beta} t$, then s is computable.*

The three items are proved simultaneously by induction on the structure of type. A consequence of the third point is that variables are computable.

Lemma 3. *Consider an abstraction $\lambda x : \sigma. s : \sigma \rightarrow \tau$. If $s[x := t]$ is computable for every computable $t : \sigma$, then $\lambda x : \sigma. s$ is computable.*

The proof proceeds by showing that all one-step reducts of $@(\lambda x : \sigma. s, t)$ are computable and then applying Lemma 2(c).

We proceed by showing that a functional term $f(s_1, \dots, s_m)$ is computable if all its direct arguments are computable. The proof of the following lemma employs a technique due to Buchholz [3], also already present in [5], that is for instance also used in [6,10].

Lemma 4. *If $s_1 : \sigma_1, \dots, s_m : \sigma_m$ are computable, then $f^\alpha(s_1, \dots, s_m)$ is computable.*

Proof. Assume computable terms $s_1 : \sigma_1, \dots, s_m : \sigma_m$ and a function symbol $f : (\sigma_1 \times \dots \times \sigma_m) \rightarrow \tau$, and an ordinal α with $\alpha \leq \omega$. We prove that $f^\alpha(s_1, \dots, s_m)$ is computable by well-founded induction on the triple (f, \mathbf{s}, α) , ordered by the lexicographic product of the following three orderings: first \triangleright on function symbols, second the lexicographic (for $f \in \mathcal{F}_{\text{LEX}}$) or multiset (for $f \in \mathcal{F}_{\text{MUL}}$) extension of $\rightarrow_{\mathcal{H}\omega\beta}$ on vectors of computable terms, and third the ordering $>$ on natural numbers extended with $\omega > n$ for every $n \in \mathbb{N}$. We denote this ordering by $>>$.

The induction hypothesis is: If $(f, \mathbf{s}, \alpha) >> (g, \mathbf{t}, \beta)$ with $\mathbf{t} = t_1, \dots, t_n$ computable terms, then $g^\beta(t_1, \dots, t_n)$ is computable.

Because $s = f^\alpha(s_1, \dots, s_m)$ is neutral (i.e. not an abstraction), by Lemma 2 (c) it is sufficient to prove that all one-step reducts of s are computable. So we suppose that $f^\alpha(s_1, \dots, s_m) \rightarrow_{\mathcal{H}\omega\beta} t$ and proceed by showing computability of t . If the rewrite step takes place inside one of the s_i , then t is computable by the induction hypothesis for the second component (note that f does not increase). Otherwise, $f^\alpha(s_1, \dots, s_k) \rightarrow_{\mathcal{H}\omega\beta} t$ is a head step. We consider 4 of the 7 possibilities.

- Suppose that $f^\omega(s_1, \dots, s_m) \rightarrow_{\text{put}} f^p(s_1, \dots, s_m)$. (Note that in this case $\alpha = \omega$.) Then $t = f^p(s_1, \dots, s_m)$ is computable by the induction hypothesis for the third component, because the first two components of the triple do not change, and $\omega > p$ for every natural number p .
- Suppose that $f^{p+1}(s_1, \dots, s_m) \rightarrow_{\text{copy}} g^\omega(t_1, \dots, t_n)$. For every t_j with $j \in \{1, \dots, n\}$ we have either $t_j = f^p(s_1, \dots, s_m)$ or $t_j = s_k$ for some $k \in \{1, \dots, m\}$. In the first case t_j is computable by the induction hypothesis on the third component. In the second case t_j is computable by assumption. Therefore (t_1, \dots, t_n) consists of computable terms. Now computability of $t = g(t_1, \dots, t_n)$ follows by the induction hypothesis on the first component.
- Suppose that $f^{p+1}(s_1, \dots, s_m) \rightarrow_{\text{mul}} f^\omega(t_1, \dots, t_{i-1}, s'_i, t_{i+1}, \dots, t_m)$. For every $j \in \{1, \dots, i-1, i+1, \dots, m\}$ we have either $s_i \rightarrow_{\text{put}} t_j$ or $t_j = s_j$. In the first case t_j is computable by the assumption that s_i is computable and Lemma 2(b). In the second case t_j is computable by assumption. Further, s'_i is a put-reduct of s_i and hence computable by assumption and Lemma 2(b). We conclude that $(t_1, \dots, t_{i-1}, s'_i, t_{i+1}, \dots, t_m)$ consists of computable terms. Now computability of $t = f^\omega(t_1, \dots, t_{i-1}, s'_i, t_{i+1}, \dots, t_m)$ follows by the induction hypothesis on the second component, because the multiset $\{\{s_1, \dots, s_m\}\}$ is greater than the multiset $\{\{t_1, \dots, t_{i-1}, s'_i, t_{i+1}, \dots, t_m\}\}$ in the multiset extension of $\rightarrow_{\mathcal{H}\omega\beta}$.
- Suppose that $f^{p+1}(s_1, \dots, s_m) \rightarrow_{\text{appl}} @ (t_0, t_1, \dots, t_n)$. For every $j \in \{0, \dots, n\}$ we have either $t_j = f^p(s_1, \dots, s_m)$ or $t_j = s_k$ for some $k \in \{1, \dots, m\}$. In the first case, t_j is computable by the induction hypothesis on the third component of the triple. In the second case, t_j is computable by assumption. Now computability of t follows because by definition the application of computable terms is computable.

From the complete case analysis follows that all one-step reducts of s are computable. Hence by Lemma 2(c) the term s is computable. □

We now show that all terms (possibly with labels) are computable, by showing the stronger statement that the application of a computable substitution to an arbitrary term yields a computable term. A substitution is said to be *computable* if all terms in its range are computable. The proof of the following theorem proceeds by induction on the definition of terms.

Theorem 2 (Computability of all terms). *Let $s : \sigma$ be an arbitrary term over $\mathcal{F} \cup \mathcal{F}^\omega$ and let γ be a computable substitution. Then $s\gamma$ is computable.*

Because the empty substitution is computable, it follows from this theorem that all terms over $\mathcal{F} \cup \mathcal{F}^\omega$ are computable. By Lemma 2 it then follows that all terms are strongly normalizing with respect to $\rightarrow_{\mathcal{H}\omega\beta}$.

Theorem 3 (Termination). *The rewriting relation $\rightarrow_{\mathcal{H}\omega\beta}$ is terminating on the set of terms over $\mathcal{F} \cup \mathcal{F}^\omega$.*

This concludes the proof of Theorem 1.

6 HOIPO Contains HORPO

The rewriting system \mathcal{H} follows the definition of HORPO quite closely. In this section we show that indeed HOIPO contains HORPO. We assume a set of function symbols \mathcal{F} and work also with marked terms over $\mathcal{F} \cup \mathcal{F}^*$.

Theorem 4. *Let s and t be terms over \mathcal{F} . If $s \succ t$ then $s \rightarrow_{\mathcal{H}}^* t$.*

Proof. Assume $s \succ t$. We prove by induction on the structure of s and t that there is some s' such that $s \rightarrow_{\text{put}} s' \rightarrow_{\mathcal{H}}^* t$. The induction hypothesis (IH) is: for all q and r , if q is a subterm of s , or ($q = s$ and r is a subterm of t), then $q \succ r$ implies $q \rightarrow_{\text{put}} q' \rightarrow_{\mathcal{H}}^* r$ (for some term q'). We consider all possible cases for $s \succ t$.

- (H1) We have $s = f(s_1, \dots, s_m)$ and there is an $i \in \{1, \dots, m\}$ such that $s_i \succeq t$. If $s_i \succ t$ then by the IH (first component) $s_i \rightarrow_{\text{put}} s'_i \rightarrow_{\mathcal{H}}^* t$. If $s_i = t$ then also $s_i \rightarrow_{\mathcal{H}}^* t$. Hence in both cases $s \rightarrow_{\text{put}} f^*(s_1, \dots, s_m) \rightarrow_{\text{select}} s_i \rightarrow_{\mathcal{H}}^* t$.
- (H2) We have $s = f(s_1, \dots, s_m)$, $t = g(t_1, \dots, t_n)$, $f \triangleright g$, and $s \succ \triangleright \{t_1, \dots, t_n\}$.

For every $i \in \{1, \dots, n\}$ we have either $s \succ t_i$ or $s_j \succeq t_i$ for some $j \in \{1, \dots, m\}$. In the first case we define $l_i = f^*(s_1, \dots, s_m)$. In the second case we define $l_i = s_j$.

In the first case we have by the IH $s \rightarrow_{\text{put}} s' \rightarrow_{\mathcal{H}}^* t_i$. Because for this s' (which is a put-reduct of s) either $s' = f^*(s_1, \dots, s_m)$ or $f^*(s_1, \dots, s_m) \rightarrow_{\text{lex/mul}} s'$, this yields $l_i = f^*(s_1, \dots, s_m) \rightarrow_{\mathcal{H}}^* t_i$.

In the second case, if $s_j \succ t_i$ for some $j \in \{1, \dots, m\}$, we have by the IH $s_j \rightarrow_{\text{put}} s'_j \rightarrow_{\mathcal{H}}^* t_i$. Hence we have $l_i = s_j \rightarrow_{\mathcal{H}}^* t_i$ (also if $s_j = t_i$).

Now we have $s \rightarrow_{\text{put}} f^*(s_1, \dots, s_m) \rightarrow_{\text{copy}} g(l_1, \dots, l_n) \rightarrow_{\mathcal{H}}^* g(t_1, \dots, t_n)$.

- (H3LEX) We have $s = f(s_1, \dots, s_m)$ and $t = f(t_1, \dots, t_m)$ with $f \in \mathcal{F}_{\text{LEX}}$, and $[s_1, \dots, s_m] \succ_{\text{LEX}} [t_1, \dots, t_m]$ and $s \succ \triangleright \{t_1, \dots, t_m\}$.

There is an $i \in \{1, \dots, m\}$ such that $s_1 = t_1, \dots, s_{i-1} = t_{i-1}, s_i \succ t_i$. Moreover, for every $j \in \{i+1, \dots, m\}$ we have either $s \succ t_j$ or $s_k \succeq t_j$ for some k . By an analysis as in (H2) we can define for every $j \in \{i+1, \dots, m\}$ a term l_j such that $l_j \rightarrow_{\mathcal{H}}^* t_j$. Because $s_i \succ t_i$ we have by the IH some term s'_i such that $s_i \rightarrow_{\text{put}} s'_i \rightarrow_{\mathcal{H}}^* t_i$. Combining this yields $s \rightarrow_{\text{put}} f^*(s_1, \dots, s_m) \rightarrow_{\text{lex}} f(s_1, \dots, s_{i-1}, s'_i, l_{i+1}, \dots, l_m) \rightarrow_{\mathcal{H}}^* f(s_1, \dots, s_{i-1}, t_i, t_{i+1}, \dots, t_m)$.

- (H3MUL) We have $s = f(s_1, \dots, s_m)$, $t = f(t_1, \dots, t_m)$, $f \in \mathcal{F}_{\text{MUL}}$, and moreover $\{\{s_1, \dots, s_m\}\} \succ_{\text{MUL}} \{\{t_1, \dots, t_m\}\}$.

By definition of the multiset ordering, we can write $\{\{s_1, \dots, s_m\}\} = A \cup B \cup C$, $\{\{t_1, \dots, t_m\}\} = A \cup D$, for all $t_i \in D$ there is some $s_j \in B$ such that $s_j \succ t_i$, and, since we are working with multisets of equal size, $|B| \geq 1$.

Suppose $|B| = 1$; write $A = \{\{s_{i_1}, \dots, s_{i_k}\}\} = \{\{t_{j_1}, \dots, t_{j_k}\}\}$, $B = \{\{s_n\}\}$. Since $s_n \succ t_x$ for all $x \notin \{j_1, \dots, j_k\}$ we can derive by the induction hypothesis that for such x there is some s'_n where $s_n \rightarrow_{\text{put}} s'_n \rightarrow_{\mathcal{H}}^* t_x$.

Now, let π be a permutation that maps each i_x to j_x ; then always $s_{\pi^{-1}j_x} = t_{j_x}$, and $s_{\pi^{-1}(\pi n)} > t_{\pi n}$ (because πn is not one of the j_x). So $f(s_1, \dots, s_m) \rightarrow_{\text{put}} f^*(s_1, \dots, s_m) \rightarrow_{\text{ord}} f^*(s_{\pi^{-1}1}, \dots, s_{\pi^{-1}m}) \rightarrow_{\text{mul}} f(r_1, \dots, r_m) \rightarrow_{\mathcal{H}}^* f(t_1, \dots, t_m)$, where $r_i = s_{\pi^{-1}i} = t_i$ if i is one of the j_x , $s_n \rightarrow_{\text{put}} r_i \rightarrow_{\mathcal{H}}^* t_i$ otherwise.

For the case $|B| > 1$, we observe that a comparison $X >_{\text{MUL}} Y$ can always be decomposed into a sequence $X = X_1 \rightsquigarrow X_2 \rightsquigarrow \dots \rightsquigarrow X_n = Y$ where each $X_i \rightsquigarrow X_{i+1}$ is an atomic $>_{\text{MUL}}$ step as described above, and that $\rightarrow_{\mathcal{H}}^*$ is transitive.

(H4) We have $s = f(s_1, \dots, s_m)$, $t = @ (t_1 : \rho_1, \dots, t_n : \rho_n)$ with $n \geq 2$, and $s \succ \{t_1, \dots, t_n\}$. By an analysis as in (H2), we can define for every $i \in \{1, \dots, n\}$ a term l_i such that $l_i \rightarrow_{\mathcal{H}}^* t_i$. Note that $\rho_1 = \rho_2 \rightarrow \dots \rightarrow \rho_n \rightarrow \sigma$. Therefore, $s \rightarrow_{\text{put}} f^*(s_1, \dots, s_m) \rightarrow_{\text{appl}} @(l_1, \dots, l_n) \rightarrow_{\mathcal{H}}^* @(t_1, \dots, t_n)$.

(H5) We have $s = @(s_1, s_2)$, $t = @(t_1, t_2)$ and $\{\{s_1, s_2\}\} \succ_{\text{MUL}} \{\{t_1, t_2\}\}$.

Because of the typing constraints either $s_1 \succ t_1$ and $s_2 = t_2$ or $s_1 \succeq t_1$ and $s_2 \succ t_2$. In the first case, we have by the IH $s_1 \rightarrow_{\text{put}} s'_1 \rightarrow_{\mathcal{H}}^* t_1$. Therefore, $s \rightarrow_{\text{put}} @(s'_1, s_2) \rightarrow_{\mathcal{H}}^* t_1 s_2 = t$. In the second case, we have by the IH $s_2 \rightarrow_{\text{put}} s'_2 \rightarrow_{\mathcal{H}}^* t_2$. Additionally, $s_1 \rightarrow_{\mathcal{H}}^* t_1$, although this may be in 0 steps. Therefore, $s \rightarrow_{\text{put}} @(s_1, s'_2) \rightarrow_{\mathcal{H}}^* @(s_1, t_2) \rightarrow_{\mathcal{H}}^* @(t_1, t_2) = t$.

(H6) We have $s = \lambda x : \sigma. s_0$, $t = \lambda x : \sigma. t_0$, and $s_0 \succ t_0$. By the IH, $s_0 \rightarrow_{\text{put}} s'_0 \rightarrow_{\mathcal{H}}^* t_0$. Hence $s = \lambda x : \sigma. s_0 \rightarrow_{\text{put}} \lambda x : \sigma. s'_0 \rightarrow_{\mathcal{H}}^* \lambda x : \sigma. t_0 = t$. \square

7 HORPO Does Not Contain HOIPO

In this section we present a rewrite rule $l \rightarrow r$ for which $l \succ^+ r$ in HORPO does not hold, but for which we can prove $l \rightarrow_{\mathcal{H}\beta}^+ r$. This shows that HORPO does not contain HOIPO; the crux is that postponing the choice for a smaller term can sometimes be useful.

We consider the the following rewrite rule $l \rightarrow r$, using function symbols $G, H : o \rightarrow o \rightarrow o$ and $A, B : o$ and $f : (o \times o \rightarrow o) \rightarrow o$ (so f is the only function symbol that takes arguments):

$$f(B, \lambda z : o. @(G, z, z)) \rightarrow @(G, f(B, \lambda z : o. @(H, z, z)), f(A, \lambda z : o. @(G, z, z)))$$

In addition assume that there are rewrite rules that enforce $G \triangleright H$ and $f \triangleright B \triangleright A$. We have $l \rightarrow_{\mathcal{H}\beta}^+ r$:

$$\begin{aligned} l &= f(B, \lambda z : o. @(G, z, z)) \\ \rightarrow_{\text{put}} & f^*(B, \lambda z : o. @(G, z, z)) \\ \rightarrow_{\text{appl}} & @(\lambda z : o. @(G, z, z), f^*(B, \lambda z : o. @(G, z, z))) \\ \rightarrow_{\beta} & @(G, f^*(B, \lambda z : o. @(G, z, z)), f^*(B, \lambda z : o. @(G, z, z))) \\ \rightarrow_{\text{lex}} & @(G, f(B, \lambda z : o. @(G^*, z, z)), f^*(B, \lambda z : o. @(G, z, z))) \\ \rightarrow_{\text{lex}} & @(G, f(B, \lambda z : o. @(G^*, z, z)), f(B^*, \lambda z : o. @(G, z, z))) \\ \rightarrow_{\text{copy}} & @(G, f(B, \lambda z : o. @(H, z, z)), f(B^*, \lambda z : o. @(G, z, z))) \\ \rightarrow_{\text{copy}} & @(G, f(B, \lambda z : o. @(H, z, z)), f(A, \lambda z : o. @(G, z, z))) = r \end{aligned}$$

It is easy to see that $l \succ r$ does not hold. But do we have $l \succ^+ r$? We show that this is not the case. Suppose that $l \succ^+ t$ for some term t . We can prove by induction, first over the length of the \succ -sequence, second on the size of t , that t must have one of the following forms:

- (a) $f(A, \lambda z : o.@(L, z, z))$ with $L \in \{G, H\}$
- (b) $f(B, \lambda z : o.@(H, z, z))$
- (c) $@(\lambda z : o.@(L, z, z), K)$ with $L \in \{G, H\}$ and $l \succ^+ K$
- (d) $@(L, K_1, K_2)$ with $L \in \{G, H\}$ and there exists some K such that $l \succ^+ K$ and $K \succ^* K_1, K \succ^* K_2$.

The reason that there are so few possibilities is that A and H are minimal with respect to \succ , and B, G only compare to A, H .

Now, r has form (d). So there has to be some K of one of the forms above such that $K \succ^* K_1, K \succ^* K_2$. However, using induction we can see that whenever $s \succ t$ it can not hold that t contains f, G or B without s containing that symbol too. So if K has form (a) it can not reduce to a term with B in it, form (b) will not reduce to a term with G in it, and the last two forms will never reduce to a term with G in it.

So we see that following the same idea as in the iterative version does not work because there is no term t satisfying the following three conditions: $f(B, \lambda z : o.@(G, z, z)) \succ^* t$, and $t \succ^* f(B, \lambda z : o.@(H, z, z))$, and $t \succ^* f(A, \lambda z : o.@(G, z, z))$.

A similar phenomenon seems to occur when comparing HOIPO to the stronger definition of \succ as given in [2]. The following system can be proven to be termination using HOIPO. However, it does not seem to have an easy termination proof using \succ as defined in [2]. The system consists of the following rewrite rules: $\{g(x, y, z) \rightarrow f(f(x, z), z), f(x, z) \rightarrow B, B \rightarrow A, f(B, \lambda x : o.g(x, x, z)) \rightarrow g(f(A, \lambda x : o.g(x, x, z)), f(B, \lambda x : o.B), z)\}$ using the function symbols $f : (o \times (o \rightarrow o)) \rightarrow o$, $g : (o \times o \times (o \rightarrow o)) \rightarrow o$, $B : o$, and $A : o$.

Finally, note that the problem illustrated above is easily solved by adding a pairing operator to the system which is smaller than the other function symbols.

8 Concluding Remarks

We have defined an iterative version of HORPO as defined in [6]. Other, more advanced, definitions of HORPO are given in [7,2]. We have on purpose chosen for the definition from [6], because it is conceptually and technically very clear; it even has been proof-checked in Coq [11]. Moreover, because it is the most basic variant, it is the best starting point for an investigation to stronger orderings. In fact, the present definition of HOIPO already lead to some ideas about strengthening the ordering, which will be developed in more detail.

Having said that, clearly the termination method provided by HOIPO is, just as the one provided by HORPO, rather weak. For instance, it cannot be used to prove termination of developments in the untyped λ -calculus, which can be done in a higher-order iterative ordering in longer versions of [10], following [13]. Therefore, in further work we intend to consider extensions of HOIPO which may

or may not be defined as iterative definitions of more sophisticated variants of HORPO. We then need to compare those extensions with the more sophisticated versions of HORPO [7,2], as well as with the long version of [10].

One of the natural next steps is to extend HOIPO using the notion of computable closure, to define iterative versions of HORPO for other frameworks of higher-order rewriting (CRSs or HRSs) [14], or to replace the ordering on function symbols by interpretations on terms.

Acknowledgements. We are very grateful to the referees of an earlier version and to the referees of LPAR 2008 for their remarks.

References

1. Bergstra, J., Klop, J.W.: Algebra of communicating processes. *Theoretical Computer Science* 37(1), 171–199 (1985)
2. Blanqui, F., Jouannaud, J.-P., Rubio, A.: The computability path ordering: The end of a quest. In: Kaminski, M., Martini, S. (eds.) *CSL 2008*. LNCS, vol. 5213, pp. 1–14. Springer, Heidelberg (2008)
3. Buchholz, W.: Proof-theoretic analysis of termination proofs. *Annals of Pure and Applied Logic* 75(1–2), 57–65 (1995)
4. Dershowitz, N.: Orderings for term rewriting systems. *Theoretical Computer Science* 17(3), 279–301 (1982)
5. Jouannaud, J.-P., Okada, M.: A computation model for executable higher-order algebraic specification languages. In: *Proceedings of LICS 1991*, Amsterdam, The Netherlands, pp. 350–361 (July 1991)
6. Jouannaud, J.-P., Rubio, A.: The higher-order recursive path ordering. In: *Proceedings of LICS 1999*, Trento, Italy, pp. 402–411 (July 1999)
7. Jouannaud, J.-P., Rubio, A.: Higher-order orderings for normal rewriting. In: Pfenning, F. (ed.) *RTA 2006*. LNCS, vol. 4098, pp. 387–399. Springer, Heidelberg (2006)
8. Jouannaud, J.-P., Rubio, A.: Polymorphic higher-order recursive path orderings. *Journal of the ACM* 54(1), 1–48 (2007)
9. Kamin, S., Lévy, J.-J.: Two generalizations of the recursive path ordering. University of Illinois (1980)
10. Klop, J.W., van Oostrom, V., de Vrijer, R.: Iterative lexicographic path orders. In: Futatsugi, K., Jouannaud, J.-P., Meseguer, J. (eds.) *Algebra, Meaning, and Computation*. LNCS, vol. 4060, pp. 541–554. Springer, Heidelberg (2006)
11. Koprowski, A.: Coq formalization of the higher-order recursive path ordering. Technical Report CSR-06-21, Eindhoven University of Technology (August 2006)
12. Nipkow, T.: Higher-order critical pairs. In: *Proceedings of LICS 1991*, Amsterdam, The Netherlands, pp. 342–349 (July 1991)
13. van Oostrom, V.: Personal communication (2008)
14. van Raamsdonk, F.: On termination of higher-order rewriting. In: Middeldorp, A. (ed.) *RTA 2001*. LNCS, vol. 2051, pp. 261–275. Springer, Heidelberg (2001)
15. Tait, W.W.: Intensional interpretations of functionals of finite type I. *Journal of Symbolic Logic* 32(2), 198–212 (1967)
16. *Terese: Term Rewriting Systems*. Cambridge Tracts in Theoretical Computer Science, vol. 55. Cambridge University Press, Cambridge (2003)