A transitive HORPO for curried systems

Liye Guo
Radboud University Nijmegen, Netherlands

Cynthia Kop
Radboud University Nijmegen, Netherlands

Abstract

The higher-order recursive path ordering is one of the oldest, but still very effective, methods to prove termination of higher-order TRSs. A limitation of this ordering is that it is not transitive (and its transitive closure is not computable). We will present a transitive variation of HORPO. Unlike previous HORPO definitions, this method can be used directly on terms in curried notation.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting

Keywords and phrases Higher-order term rewriting, termination, recursive path ordering

Funding The authors are supported by the NWO VIDI project "CHORPE", NWO VI.Vidi.193.075.

1 Introduction

Termination problems have been studied by the term rewriting community for decades. In *higher-order* termination, one of the earliest techniques was HORPO [2], a higher-order extension of the *recursive path ordering* [3]. This definition has seen a series of improvements over the years, culminating in the powerful *Computability Path Ordering* (CPO) [1].

Interestingly, the relations $\succ_{\text{horpo}}$ and $\succ_{\text{cpo}}$ are not transitive. To obtain a well-founded ordering, we must use the transitive closure $\succ^{+}_{\text{horpo}}$ (resp. $\succ^{+}_{\text{cpo}}$), but this is not computable. Hence, for many rules that can in theory be oriented by HORPO, this cannot be found in practice. This limitation is particularly problematic when HORPO is transposed to formalisms where lambda abstractions occur more often on the left-hand side of a rule, since we may for instance have $\mathtt{f}(\lambda x.\mathtt{g}(x), Y) \succ_{\text{cpo}} (\lambda x.\mathtt{g}(x)) \cdot Y \succ_{\text{cpo}} \mathtt{g}(Y)$, but not $\mathtt{f}(\lambda x.\mathtt{g}(x), Y) \succ_{\text{cpo}} \mathtt{g}(Y)$.

To address this issue, the second author explored an alternative HORPO in her PhD thesis [5]: following an idea from the *iterative* path ordering [4], we use an annotation $\star$ to mark an obligation to decrease a term. This can be harnessed to obtain a transitive definition. Like HORPO and CPO, *StarHorpo* was defined on a formalism with functional (uncurried) notation; application is encoded as a family of function symbols. Consequently, in curried specifications, the same few symbols occur over and over, making the method hard to apply.

In this paper, we adapt StarHorpo to a *curried* system. This is not just a notational matter: allowing function symbols to take a variable number of arguments poses new technical challenges. This is work in progress; we will focus on the core aspects of the method. For now, we omit lambda abstractions and type orderings as used in CPO. However, the eventual goal is to define a transitive ordering that strictly includes CPO for curried systems.

2 Preliminaries

2.1 Applicative TRS

For presentation, we shall consider an applicative term rewriting system. We assume that a set $\mathcal{S}$ of *base types* is given, and the set $\mathcal{T}$ of *simple types* is generated by the grammar $\mathcal{T} ::= \mathcal{S} \mid (\mathcal{T} \to \mathcal{T})$. Right-associativity is assigned to $\to$ so that some parentheses in types can be omitted. We postulate two disjoint sets $\mathcal{F}$ and $\mathcal{V}$, called the set of *function symbols* and the set of *variables*, respectively. We assume that every function symbol and variable

has exactly one simple type, and we write $\mathfrak{a} : A$ for $\mathfrak{a}$ of type A. In this paper, we let f and g range over the set $\mathcal{F}$, x over the set $\mathcal{V}$ and $\mathfrak{a}$ over $\mathcal{F} \cup \mathcal{V}$.

The set $\mathbb{T}$ of *pre-terms* is generated by the grammar $\mathbb{T} ::= \mathcal{F}(\mathbb{T}, \dots, \mathbb{T}) \mid \mathcal{V}(\mathbb{T}, \dots, \mathbb{T})$. The set of *terms* consists of pre-terms which can be given a simple type by the following rule:

$$\frac{\mathfrak{a} : A_1 \to \cdots \to A_n \to B \qquad t_1 : A_1 \qquad \dots \qquad t_n : A_n}{\mathfrak{a}(t_1, \dots, t_n) : B} \ (\mathfrak{a} \in \mathcal{F} \cup \mathcal{V})$$

A term has only one type. When $n = 0$, we omit the parentheses and write $\mathfrak{a}$ instead of $\mathfrak{a}()$.

The *application* of a term $t = \mathfrak{a}(t_1, \dots, t_n) : A \to B$ to another term $t_{n+1} : A$, denoted by $t \cdot t_{n+1}$, is defined to be $\mathfrak{a}(t_1, \dots, t_n, t_{n+1})$. We assign to $\cdot$ left-associativity. Type-preserving functions from variables to terms are called *substitutions*. Every substitution σ extends to a type-preserving function $\bar{\sigma}$ from terms to terms. We write $t\sigma$ for $\bar{\sigma}(t)$ and define it as follows: $f(t_1, \dots, t_n)\sigma = f(t_1\sigma, \dots, t_n\sigma)$, and $x(t_1, \dots, t_n)\sigma = \sigma(x) \cdot t_1\sigma \cdots t_n\sigma$.

A *rewrite rule* $\ell \to r$ is an ordered pair where ℓ and r are terms of the same type, variables occurring in r also occur in ℓ, and $\ell = f(t_1, \dots, t_n)$. Given a set $\mathcal{R}$ of rewrite rules, $t \to_{\mathcal{R}} t'$ if and only if one of the following conditions is true:

- $t = \ell\sigma$, $t' = r\sigma$ and $\ell \to r \in \mathcal{R}$ for some substitution σ.
- $t = t_1 \cdot t_2$, $t' = t_1{}' \cdot t_2$ and $t_1 \to_{\mathcal{R}} t_1{}'$.
- $t = t_1 \cdot t_2$, $t' = t_1 \cdot t_2{}'$ and $t_2 \to_{\mathcal{R}} t_2{}'$.

$\to_{\mathcal{R}}$ is called the *rewrite relation*. This paper concerns the well-foundedness of $\to_{\mathcal{R}}$.

In the above definition, the application operator $\cdot$ is distinct from the function symbols, and terms are lists headed by a function symbol or a variable. An equivalent and commonly used alternative is to consider $\cdot$ as part of term formation and terms as binary trees. In this view, we would for instance write $f \cdot t_1 \cdot t_2$, or just $f\ t_1\ t_2$, instead of $f(t_1, t_2)$. We favor our current presentation to stress the similarities to the existing recursive path orderings, which are typically defined on formalisms with functional notation. We do *not* consider application as function symbols, as this would be detrimental to our method.

2.2 HORPO

We review a simple *higher-order recursive path ordering* [2] reformulated for the above formalism. Given a well-founded ordering $\blacktriangleright$ on $\mathcal{F}$, called the *precedence*, $s \succ_{\text{horpo}} t$ if and only if s has the same type as t and one of the following conditions is true:

(1) $s = f(s_1, \dots, s_m)$ and $\exists i\ s_i \succeq_{\text{horpo}} t$.
(2) $s = f(s_1, \dots, s_m)$, $t = t_1 \cdot t_2 \cdots t_n$ and $f(s_1, \dots, s_m) \vdash \{t_1, \dots, t_n\}$.
(3) $s = f(s_1, \dots, s_m)$, $t = g(t_1, \dots, t_n)$, $f \blacktriangleright g$ and $f(s_1, \dots, s_m) \vdash \{t_1, \dots, t_n\}$.
(4) $s = f(s_1, \dots, s_m)$, $t = f(t_1, \dots, t_m)$, $(s_1, \dots, s_m) \succ^{\text{lex}}_{\text{horpo}} (t_1, \dots, t_m)$ and $f(s_1, \dots, s_m) \vdash \{t_1, \dots, t_m\}$.
(5) $s = s_1 \cdot s_2$, $t = t_1 \cdot t_2$, $s_1 \succeq_{\text{horpo}} t_1$, $s_2 \succeq_{\text{horpo}} t_2$ and $s \neq t$.

In the above definition, $\succeq_{\text{horpo}}$ is the reflexive closure of $\succ_{\text{horpo}}$, $\succ^{\text{lex}}_{\text{horpo}}$ lexicographically compares lists of the same length by $\succ_{\text{horpo}}$, and $f(s_1, \dots, s_m) \vdash \{t_1, \dots, t_n\}$ stands for $\forall i\ (f(s_1, \dots, s_m) \succ_{\text{horpo}} t_i \vee \exists j\ s_j \succeq_{\text{horpo}} t_i)$. We remark that the multiset extension in the definition [2] is omitted for simplicity's sake. The relation $\succ_{\text{horpo}}$ is *well-founded*, *monotonic* (i.e., $t_1 \succ_{\text{horpo}} t_1{}'$ implies $t_1 \cdot t_2 \succ_{\text{horpo}} t_1{}' \cdot t_2$, and $t_2 \succ_{\text{horpo}} t_2{}'$ implies $t_1 \cdot t_2 \succ_{\text{horpo}} t_1 \cdot t_2{}'$), and *stable* (i.e., $t \succ_{\text{horpo}} t'$ implies $t\sigma \succ_{\text{horpo}} t'\sigma$ for all substitutions σ). If in addition $\to_{\mathcal{R}}$ is *compatible* with $\succ_{\text{horpo}}$ (i.e., $\ell \succ_{\text{horpo}} r$ for all $\ell \to r \in \mathcal{R}$), then $\to_{\mathcal{R}}$ is well-founded.

As an example, consider the following definition of a recursor for the natural numbers:

$$\mathtt{rec}(\mathtt{0}, Y, F) \to Y \qquad \mathtt{rec}(\mathtt{s}(X), Y, F) \to F(X, \mathtt{rec}(X, Y, F))$$

where $\mathtt{0} : \mathtt{nat}$, $\mathtt{s} : \mathtt{nat} \to \mathtt{nat}$ and $\mathtt{rec} : \mathtt{nat} \to \mathtt{nat} \to (\mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}) \to \mathtt{nat}$ are the function symbols, and the types of the variables X, Y and F can be deduced. In order to show the well-foundedness of $\to_{\mathcal{R}}$, we need only to find a precedence $\blacktriangleright$ making $\to_{\mathcal{R}}$ compatible with the generated relation $\succ_{\text{horpo}}$. While $\mathtt{rec}(\mathtt{0}, Y, F) \succ_{\text{horpo}} Y$ follows from the first condition, $\mathtt{rec}(\mathtt{s}(X), Y, F) \succ_{\text{horpo}} F(X, \mathtt{rec}(X, Y, F))$ can be obtained as follows:

$$\dfrac{F \succeq_{\text{horpo}} F \qquad \dfrac{X \succeq_{\text{horpo}} X}{\mathtt{s}(X) \succeq_{\text{horpo}} X}\,1 \qquad \dfrac{\mathtt{s}(X) \succ_{\text{horpo}} X \qquad Y \succeq_{\text{horpo}} Y \qquad F \succeq_{\text{horpo}} F}{\mathtt{rec}(\mathtt{s}(X), Y, F) \succ_{\text{horpo}} \mathtt{rec}(X, Y, F)}\,4}{\mathtt{rec}(\mathtt{s}(X), Y, F) \succ_{\text{horpo}} F(X, \mathtt{rec}(X, Y, F))}\,2$$

The precedence can be any well-founded ordering on $\mathcal{F}$. The above process of finding the precedence can be automated by encoding the constraints $\ell \succ_{\text{horpo}} r$ in a propositional formula that is fed to a SAT solver, as demonstrated in [8] for the first-order RPO.

The usefulness of $\succ_{\text{horpo}}$ is limited by the type restriction—only terms of the same type can be compared. Let us extend the above example with the following rewrite rules:

$$\mathtt{add}(\mathtt{0}, Y) \to Y \qquad \mathtt{add}(\mathtt{s}(X), Y) \to \mathtt{s}(\mathtt{add}(X, Y)) \qquad \mathtt{sum}(X) \to \mathtt{rec}(X, \mathtt{0}, \mathtt{add})$$

where $\mathtt{add} : \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}$ and $\mathtt{sum} : \mathtt{nat} \to \mathtt{nat}$. If we ignore the rewrite rule on the right, we need only $\mathtt{add} \blacktriangleright \mathtt{s}$ to complete the proof. Since the rule on the right only removes occurrences of $\mathtt{sum}$, it seems harmless. However, $\mathtt{sum}(X) \succ_{\text{horpo}} \mathtt{rec}(X, \mathtt{0}, \mathtt{add})$ is not obtainable due to the type restriction: neither $\mathtt{sum}(X)$ nor X has the same type as $\mathtt{add}$ so the necessary premise $\mathtt{sum}(X) \vDash \{\mathtt{add}\}$ does not hold. This problem is addressed in [2] by introducing *computable closures*. We will provide an alternative in the next section.

3 StarHorpo

Let $\mathcal{F}^\star$ be $\mathcal{F} \uplus (\mathcal{F} \times \mathcal{T})$, in which a function symbol is either a function symbol $f \in \mathcal{F}$, or an ordered pair (f, A), written as $f^\star_A$, where $f \in \mathcal{F}$ and $A \in \mathcal{T}$. We assume $f^\star_A : A$ and $\mathcal{F}^\star \cap \mathcal{V} = \emptyset$. With $\mathcal{F}^\star$, terms are generated and typed likewise. Given a term $f(t_1, \dots, t_n)$ where $t_i : A_i$ for all i, the newly introduced function symbols allow us to have $f^\star(t_1, \dots, t_n) : B$ for any B, where $f^\star$ stands for $f^\star_{A_1 \to \cdots \to A_n \to B}$. We will omit the type and write just $f^\star$ whenever the type can be deduced from the context. In the above translation from $f(t_1, \dots, t_n)$ to $f^\star(t_1, \dots, t_n)$, *marking* the head symbol serves two purposes:

- $f^\star(t_1, \dots, t_n)$ can have a different type from the type of $f(t_1, \dots, t_n)$.
- $f^\star(t_1, \dots, t_n)$ encodes an obligation to make $f(t_1, \dots, t_n)$ smaller [4, 7].

We further assume that every marked function symbol $f^\star$ in a term is followed by at least $\mathsf{minar}(f)$ arguments, where the function $\mathsf{minar} : \mathcal{F} \to \mathbb{N}$ is called the *minimal arity*.

A term is said to be *unmarked* if it does not contain any marked function symbol. Given minar and the precedence $\blacktriangleright$ on $\mathcal{F}$, $s \succ_\star t$ if and only if s has the same type as t, t is unmarked, and one of the following conditions is true:

Put	$s = f(s_1, \dots, s_m)$, $m \geq \mathsf{minar}(f)$ and $f^\star(s_1, \dots, s_m) \succ_\star t$.
Select	$s = f^\star(s_1, \dots, s_m)$ and $\exists i\ s_i \cdot f^\star(s_1, \dots, s_m) \cdots f^\star(s_1, \dots, s_m) \succeq_\star t$.
Copy	$s = f^\star(s_1, \dots, s_m)$, $t = g(t_1, \dots, t_n)$, $f \blacktriangleright g$ and $\forall i\ f^\star(s_1, \dots, s_m) \succ_\star t_i$.
Lex	$s = f^\star(s_1, \dots, s_m)$, $t = f(t_1, \dots, t_n)$, $(s_1, \dots, s_{\mathsf{minar}(f)}) \succ_\star^{\text{lex}} (t_1, \dots, t_n)$ and $\forall i\ f^\star(s_1, \dots, s_m) \succ_\star t_i$.
Mono	$s = s_1 \cdot s_2$, $t = t_1 \cdot t_2$, $s_1 \succeq_\star t_1$, $s_2 \succeq_\star t_2$ and $s \neq t$.

In the above definition, $\succeq_\star$ is the union of $\succ_\star$ and the identity relation on unmarked terms, and $(s_1, \ldots, s_{\mathsf{minar}(f)}) \succ_\star^{\mathrm{lex}} (t_1, \ldots, t_n)$ if and only if $\exists i \leq \min(\mathsf{minar}(f), n)$ $(s_i \succ_\star t_i \wedge \forall j < i\ s_j = t_j)$. Occurrences of $f^\star(s_1, \ldots, s_m)$ in the conditions always have an appropriate type, and the number of occurrences of $f^\star(s_1, \ldots, s_m)$ in Select is determined by the type of s_i.

Put allows us to mark a function symbol without changing its type. When we do so, it is required that the marked function symbol $f^\star$ takes at least $\mathsf{minar}(f)$ arguments. Without this restriction, the ordering, denoted by $>$, will end up allowing the derivation in Figure 1. By Mono, we get an infinite sequence $\mathtt{f}(\mathtt{g}) > \mathtt{e}(\mathtt{f}(\mathtt{g},\mathtt{h})) > \mathtt{e}(\mathtt{e}(\mathtt{f}(\mathtt{g},\mathtt{h}),\mathtt{h})) > \cdots$, which shows that $>$ is not well-founded. The "minimal arity" restriction is necessary because Select may cause function symbols to take extra arguments, which is not the case for HORPO, and taking into account the extra arguments in a Lex step can break well-foundedness, as shown in Figure 1.

$$\mathtt{e} : \mathtt{a} \to \mathtt{a} \to \mathtt{a}$$
$$\mathtt{f} : ((\mathtt{a} \to \mathtt{a}) \to \mathtt{a} \to \mathtt{a}) \to \mathtt{a} \to \mathtt{a}$$
$$\mathtt{g} : (\mathtt{a} \to \mathtt{a}) \to \mathtt{a} \to \mathtt{a}$$
$$\mathtt{h} : \mathtt{a}$$

$$\dfrac{\mathtt{g} \blacktriangleright \mathtt{e} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\mathtt{g} \blacktriangleright \mathtt{h}}{\mathtt{g}^\star(\mathtt{f}^\star(\mathtt{g})) > \mathtt{h}}\,\mathsf{Copy}}{\mathtt{f}^\star(\mathtt{g}, \mathtt{g}^\star(\mathtt{f}^\star(\mathtt{g}))) > \mathtt{f}(\mathtt{g},\mathtt{h})}\,\mathsf{Lex}}{\mathtt{g}^\star(\mathtt{f}^\star(\mathtt{g})) > \mathtt{f}(\mathtt{g},\mathtt{h})}\,\mathsf{Select}}{\mathtt{g}^\star(\mathtt{f}^\star(\mathtt{g})) > \mathtt{e}(\mathtt{f}(\mathtt{g},\mathtt{h}))}\,\mathsf{Copy}}{\mathtt{g}(\mathtt{f}^\star(\mathtt{g})) > \mathtt{e}(\mathtt{f}(\mathtt{g},\mathtt{h}))}\,\mathsf{Put}}{\mathtt{f}^\star(\mathtt{g}) > \mathtt{e}(\mathtt{f}(\mathtt{g},\mathtt{h}))}\,\mathsf{Select}}{\mathtt{f}(\mathtt{g}) > \mathtt{e}(\mathtt{f}(\mathtt{g},\mathtt{h}))}\,\mathsf{Put}$$

Figure 1 Non-well-foundedness from dropping the minar restriction

We note that marked function symbols play a role only in generating $\succ_\star$. Because $\to_\mathcal{R}$ is a relation on unmarked terms, we should consider the restriction of $\succ_\star$ to unmarked terms when showing the well-foundedness of $\to_\mathcal{R}$. Like $\succ_{\mathrm{horpo}}$, the restriction of $\succ_\star$ to unmarked terms is well-founded, monotonic and stable, which means we only need to find a combination of $\blacktriangleright$ and minar that makes $\to_\mathcal{R}$ compatible with the generated relation $\succ_\star$.

For example, $\mathtt{sum}(X) \succ_\star \mathtt{rec}(X, \mathtt{0}, \mathtt{add})$ can be obtained as follows, with $\mathsf{minar}(\mathtt{sum}) = 1$:

$$\dfrac{\mathtt{sum} \blacktriangleright \mathtt{rec} \qquad \dfrac{X \succeq_\star X}{\mathtt{sum}^\star(X) \succ_\star X}\,\mathsf{Select} \qquad \dfrac{\mathtt{sum} \blacktriangleright \mathtt{0}}{\mathtt{sum}^\star(X) \succ_\star \mathtt{0}}\,\mathsf{Copy} \qquad \dfrac{\mathtt{sum} \blacktriangleright \mathtt{add}}{\mathtt{sum}^\star(X) \succ_\star \mathtt{add}}\,\mathsf{Copy}}{\mathtt{sum}^\star(X) \succ_\star \mathtt{rec}(X, \mathtt{0}, \mathtt{add})}\,\mathsf{Copy}}{\mathtt{sum}(X) \succ_\star \mathtt{rec}(X, \mathtt{0}, \mathtt{add})}\,\mathsf{Put}$$

Unlike $\succ_{\mathrm{horpo}}$, $\succ_\star$ is necessarily transitive, which we exemplify with the rewrite sequence

$$\mathtt{rec}(\mathtt{s}(\mathtt{s}(X)), Y, \mathtt{add}) \to_\mathcal{R} \mathtt{add}(\mathtt{s}(X), \mathtt{rec}(\mathtt{s}(X), Y, \mathtt{add})) \to_\mathcal{R} \mathtt{s}(\mathtt{add}(X, \mathtt{rec}(\mathtt{s}(X), Y, \mathtt{add}))).$$

With either $\succ_{\mathrm{horpo}}$ or $\succ_\star$, we can see that in each of the rewrite steps, the term on the left-hand side is greater than the one on the right-hand side, using only $\mathtt{add} \blacktriangleright \mathtt{s}$ and $\mathsf{minar}(\mathtt{rec}) = \mathsf{minar}(\mathtt{add}) = 1$. If we skip the term in the middle and try to directly compare the first and the last in the sequence, $\succ_{\mathrm{horpo}}$ fails unless we further impose $\mathtt{rec} \blacktriangleright \mathtt{s}$. This shows that $\succ_{\mathrm{horpo}}$ is not transitive as imposing extra assumptions can be problematic when there are other rewrite rules in the system. On the other hand, with $\succ_\star$, we do have the following derivation (with irrelevant part omitted):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathtt{add} \blacktriangleright \mathtt{s} \qquad \dfrac{\vdots}{\mathtt{add}^\star(\mathtt{rec}^\star(\ldots), \mathtt{rec}^\star(\ldots)) \succ_\star \mathtt{add}(X, \mathtt{rec}(\mathtt{s}(X), Y, \mathtt{add}))}\,\mathsf{Lex}}{\mathtt{add}^\star(\mathtt{rec}^\star(\ldots), \mathtt{rec}^\star(\ldots)) \succ_\star \mathtt{s}(\mathtt{add}(X, \mathtt{rec}(\mathtt{s}(X), Y, \mathtt{add})))}\,\mathsf{Copy}}{\mathtt{add}(\mathtt{rec}^\star(\ldots), \mathtt{rec}^\star(\ldots)) \succ_\star \mathtt{s}(\mathtt{add}(X, \mathtt{rec}(\mathtt{s}(X), Y, \mathtt{add})))}\,\mathsf{Put}}{\mathtt{rec}^\star(\mathtt{s}(\mathtt{s}(X)), Y, \mathtt{add}) \succ_\star \mathtt{s}(\mathtt{add}(X, \mathtt{rec}(\mathtt{s}(X), Y, \mathtt{add})))}\,\mathsf{Select}}{\mathtt{rec}(\mathtt{s}(\mathtt{s}(X)), Y, \mathtt{add}) \succ_\star \mathtt{s}(\mathtt{add}(X, \mathtt{rec}(\mathtt{s}(X), Y, \mathtt{add})))}\,\mathsf{Put}$$

In the above derivation, we "select" $\mathtt{add}$ in $\mathtt{rec}^\star(\mathtt{s}(\mathtt{s}(X)), Y, \mathtt{add})$ by Select. This reflects the derivation of $\mathtt{rec}(\mathtt{s}(X), Y, F) \succ_\star F(X, \mathtt{rec}(X, Y, F))$, in which F is selected. This step is not available with $\succ_{\text{horpo}}$. Also by Select, $\mathtt{add}$ gets two arguments, each with a marked head symbol. The arguments are later compared with some arguments on the right-hand side by Lex. This capacity to postpone comparison is vital to the transitivity of $\succ_\star$.

To automate StarHorpo, standard SAT encoding techniques (see, e.g., [8]) can be used. The only limitation is that, to ensure termination, the number of size-increasing applications of Select until the right-hand side is decreased should be bounded. This is implemented in the second author's termination tool Wanda [6], which features the original StarHorpo.

Finally, let us discuss what would happen if we encoded application as function symbols. In this perspective, we could ignore types and view a curried system as a first-order (uncurried) system with a single binary function symbol @. However, all complex terms are thus headed by the same binary function symbol, which sharply limits the applicability of recursive path orderings since we can rarely take advantage of the head symbol comparison, when we apply Copy. Consider the system with only one rewrite rule $\mathtt{f}(X) \to \mathtt{g}(X, X)$, where $\mathtt{f} : \mathtt{a} \to \mathtt{a}$ and $\mathtt{g} : \mathtt{a} \to \mathtt{a} \to \mathtt{a}$. This rule is easily oriented by $\succ_\star$ (or $\succ_{\text{horpo}}$) with $\mathtt{f} \blacktriangleright \mathtt{g}$, but its applicative first-order counterpart, $@(\mathtt{f}, X) \to @(@(\mathtt{g}, X), X)$, cannot be tackled. Even if we do not ignore types and introduce a separate symbol $@_{A,B} : (A \to B) \to A \to B$ for any types A and B, the same problem still arises: $@_{\mathtt{a},\mathtt{a}}(\mathtt{f}, X) \succ_\star @_{\mathtt{a},\mathtt{a}}(@_{\mathtt{a},\mathtt{a}\to\mathtt{a}}(\mathtt{g}, X), X)$ is not obtainable.

4 Conclusion

We have adapted StarHorpo to an applicative system. Changing the underlying formalism requires extra attention: the arity restriction of functional notation should not be dropped naively; instead, we impose the minimal arity, a weaker version of arity, on StarHorpo. Interestingly, on the same applicative system, our definition of HORPO does not seem in need of any kind of arity. While this definition is indeed more powerful in some cases, without Select, which can give function symbols extra arguments, HORPO is not necessarily transitive. We thus incorporate both Select and minar into StarHorpo to gain transitivity while maintaining well-foundedness.

In order to have CPO included in StarHorpo for curried systems, we still need to take into account lambda abstractions and type orderings, as well as the multiset extension, in our definition. Furthermore, another direction for future work is to apply StarHorpo to determine the termination of functional programs, which may require us to extend StarHorpo with support for real-world data types such as integers and floating-point numbers.

References

1 F. Blanqui, J. Jouannaud, and A. Rubio. The computability path ordering: The end of a quest. In *Proc. CSL*, 2008.
2 J. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *Proc. LICS*, 1999.
3 S. Kamin and J.-J. Lévy. Two generalizations of the recursive path ordering. Unpublished Manuscript, University of Illinois, 1980.
4 J.W. Klop, V. van Oostrom, and R. de Vrijer. Iterative lexicographic path orders. In *Algebra, Meaning, and Computation*. 2006.
5 C. Kop. *Higher Order Termination*. PhD thesis, VU Amsterdam, 2012.
6 C. Kop. WANDA – a higher-order termination tool. In *Proc. FSCD*, 2020.
7 C. Kop and F. van Raamsdonk. A higher-order iterative path ordering. In *Proc. LPAR*, 2008.
8 P. Schneider-Kamp, R. Thiemann, E. Annov, M. Codish, and J. Giesl. Proving termination using recursive path orders and SAT solving. In *Proc. FroCoS*, 2007.