Higher-Order Recursive Path Orderings

Jean-Pierre Jouannaud† and **Albert Rubio**‡
† LRI, Bat. 490, CNRS/Université de Paris Sud, 91405 Orsay, FRANCE
Email: Jean-Pierre.Jouannaud@lri.fr
‡ Technical University of Catalonia, Pau Gargallo 5, 08028 Barcelona, SPAIN
Email: rubio@lsi.upc.es

Abstract: This paper extends the termination proof techniques based on reduction orderings to a higher-order setting, by adapting the recursive path ordering definition to higher-order simply-typed λ-terms. The main result is that this ordering is well-founded, compatible with β-reductions, and with polymorphic typing. We also restrict the ordering so as to obtain a new ordering operating on higher-order terms in η-long β-normal form. Both orderings are powerful enough to allow for complex examples, including the polymorphic version of Gödel's recursor for simple inductive types.

Key words: *higher-order rewriting*; *typed lambda calculus*; *Gödel's polymorphic recursor*; *termination orderings.*

1 Introduction

Rewrite rules are increasingly used in programming languages and logical systems, with two main goals: defining functions by pattern matching; describing rule-based decision procedures. ML, Elf [16] and Isabelle [15] examplify the first use. A future version of Coq [8] will examplify the second use [7]. In Elf and Isabelle, rules operate on terms in β-normal, η-expanded form. In ML and Coq, they operate on arbitrary terms. In the future version of Coq, both kinds will probably coexist.

Our ambition is to develop for the higher-order case the kind of semi-automated termination proof techniques that are available for the first-order case, of which the most popular one is the recursive path ordering [5].

Our first important contribution is a reduction ordering for typed higher-order terms which conservatively extends Dershowitz's recursive path ordering for first-order terms. In the latter, the precedence rule allows to decrease from the term $s = f(s_1, \ldots, s_n)$ to the term $g(t_1, \ldots, t_n)$, provided that (i) f is bigger than g in the given precedence on function symbols, and (ii) s is bigger than every t_i. For typing reasons, in our ordering the latter condition becomes: (ii) for every t_i, either s or some s_j is bigger than or equal to t_i. Indeed, we can instead allow t_i to be obtained from the subterms of s by computability preserving operations. Here, computability refers to Tait and Girard's strong normalization proof technique which we have used to show that our ordering is well-founded and compatible with β-reductions. To hint at the strength of the ordering, let us mention that the polymorphic version of Gödel's recursor for the natural numbers is easily oriented. And indeed, our ordering applies to arbitrary polymorphic signatures. Several other examples are given which examplify the expressive power of the ordering.

As a second contribution, we then restrict the previous ordering so as to obtain a reduction ordering operating on terms in η-long β-normal form. Only very weak orderings had been obtained so far for that purpose [12, 13, 4, 10, 11].

The framework we use is described in Section 2. The ordering is defined and studied in Section 3, where several examples are also given. The ordering for terms in η-long β-normal form is discussed in Section 4. A comparison with previous work is given in conclusion, where future work is also discussed.

We expect the reader to be familiar with the basic concepts and notations of term rewriting systems [6] and typed lambda calculi [1, 2].

2 Preliminaries

2.1 Algebraic λ-Terms

2.2 Types

We are first given a set $\mathcal{S}$ of *basic sorts* and a set $\mathcal{S}^\forall$ of *type variables*. The set $\mathcal{T}_{\mathcal{S}^\forall}$ of *polymorphic types* is generated from these sets by the constructor $\rightarrow$ for *functional types*:

$$\mathcal{T}_{\mathcal{S}^\forall} := \mathcal{S} \mid \mathcal{S}^\forall \mid (\mathcal{T}_{\mathcal{S}^\forall} \rightarrow \mathcal{T}_{\mathcal{S}^\forall})$$

We will denote by $\mathcal{T}_\mathcal{S}$ the set of ground types. By a *basic type*, we mean a basic sort or a type variable. Other types are *arrow types*.

In the following, we use σ, τ and ρ to denote types. *Type declarations* are expressions of the form $\sigma_1 \times \ldots \times \sigma_n \rightarrow \sigma$, where $\sigma_1, \ldots, \sigma_n, \sigma$ are types. Types occurring before the arrow are called input types, while the type occurring after the arrow is called the output type. A type declaration is *polymorphic* if it uses some non-ground type, otherwise, it is *monomorphic*. Type declarations are not types, although they are used for typing purposes. The notion of input and output types will also be used for types, considering that the output type of a sort or a type variable is the type itself and for a type $\sigma_1 \rightarrow \ldots \rightarrow \sigma_n \rightarrow \sigma$, again σ is the output type and all others are the input types.

2.3 Signatures

We are given a set of function symbols which are meant to be algebraic operators, equipped with a fixed number n of arguments (called the *arity*) of respective types $\sigma_1, \ldots, \sigma_n$, and an output type σ:

$$\mathcal{F} = \bigcup_{\sigma_1,\ldots,\sigma_n,\sigma} \mathcal{F}_{\sigma_1 \times \ldots \times \sigma_n \rightarrow \sigma}$$

$\mathcal{F}$ is called a *polymorphic signature* if some type declaration is polymorphic, and a monomorphic signature otherwise. Given a *type substitution* ξ extending to types a mapping from $\mathcal{S}^\forall$ to $\mathcal{T}_\mathcal{S}$, we denote by $\mathcal{F}\xi$ the monomorphic signature $\{f : s_1\xi \rightarrow s_2\xi \rightarrow \ldots \rightarrow s_n\xi \mid f : s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_n \in \mathcal{F}\}$. Polymorphic signatures capture infinitely many monomorphic ones via type instantiation.

In addition, function symbols may be overloaded, that is, they may have several arities and type declarations.

2.4 Terms

The set of *untyped algebraic λ-terms* is generated from the signature $\mathcal{F}$ and a denumerable set $\mathcal{X}$ of variables according to the grammar:

$$\mathcal{T} := \mathcal{X} \mid (\lambda\mathcal{X}.\mathcal{T}) \mid \mathcal{T}(\mathcal{T}) \mid \mathcal{F}(\mathcal{T}, \ldots, \mathcal{T}).$$

$u(v)$ denotes the application of u to v. We will sometimes write the application operator explicitely, as in $@(u, v)$. As a matter of convenience, we may write $u(v_1, \ldots, v_n)$, or $@(u, v_1, \ldots, v_n)$ for

$u(v_1)\ldots(v_n)$, assuming $n \geq 1$. $@(u, \overline{v})$ is called a (partial) *left-flattening* of $s = u(v_1)\ldots(v_n)$, u being possibly an application itself.

We denote by $\mathcal{V}ar(t)$ the set of free variables of t, and by $\mathcal{BV}ar(t)$ its set of bound variables. We may assume for convenience (and without further notice) that bound variables in a term are all different, and are different from the free ones.

Terms are identified with finite labeled trees by considering $\lambda x.$, for each variable x, as a unary function symbol. A term (context) is *ground* if it contains no free variable. *Positions* are strings of positive integers. Λ and $\cdot$ denote respectively the empty string (root position) and the concatenation of strings. The latter may sometimes be omitted. The *subterm* of t at position p is denoted by $t|_p$, and we write $t \trianglerighteq t|_p$. The result of replacing $t|_p$ at position p in t by u is denoted by $t[u]_p$. We use $t[u]$ to indicate that u is a subterm of t, and simply $t[]$ for a term with a hole, also called a context. We will also use multiple replacements at a non-empty set $\overline{p}$ of incomparable positions, writing $u[\overline{t}]_{\overline{p}}$. The notation $\overline{s}$ wil be ambiguously used to denote a list, or a multiset, or a set of terms $s_1, \ldots, s_n$.

Substitutions are written as in $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ where t_i is assumed different from x_i. We use the letter γ for substitutions and postfix notation for their application. Remember that substitutions behave as endomorphisms defined on free variables (avoiding captures).

2.5 Typing Rules

Typing rules restrict the set of terms by constraining them to follow a precise discipline. Environments are sets of pairs of the form $x : \sigma$, where x is a variable and σ is a type. Our typing judgements are written as $\Gamma \vdash M : \sigma$ if the term M can be proved to have the type σ in the environment Γ:

Variables:
$$\frac{X : \sigma \in \Gamma}{\Gamma \vdash X : \sigma}$$

Functions:
$$\frac{F : \sigma_1 \times \ldots \times \sigma_n \to \sigma \in \mathcal{F} \quad \Gamma \vdash t_1 : \sigma_1 \ \ldots \ \Gamma \vdash t_n : \sigma_n}{\Gamma \vdash F(t_1, \ldots, t_n) : \sigma}$$

Abstraction:
$$\frac{\Gamma \cup \{x : \sigma\} \vdash M : \tau}{\Gamma \vdash (\lambda x : \sigma.M) : \sigma \to \tau}$$

Application:
$$\frac{\Gamma \cup \{x : \sigma\} \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M(N) : \tau}$$

A term M has type σ in the environment Γ if $\Gamma \vdash M : \sigma$ is provable in the above inference system. A term M is typable in the environment Γ if there exists a (necessarily unique) type σ such that M has type σ in the environment Γ. A term M is typable if it is typable in some environment Γ. Note that function symbols are uncurried, hence must come along with all their arguments.

2.6 Conversion rules

Three particular equations originate from the λ-calculus, α-, β- and η-equality:

$$\begin{array}{ccll} \lambda x.v & =_\alpha & \lambda y.v\{x \mapsto y\} & \text{if } y \notin \mathcal{BV}ar(v) \cup (\mathcal{V}ar(v) \setminus \{x\}) \\ (\lambda x.v)(u) & =_\beta & u\{x \mapsto v\} & \\ \lambda x.u(x) & =_\eta & u & \text{if } x \notin \mathcal{V}ar(u) \end{array}$$

As usual, we do not distinguish α-convertible terms. We use $\xleftrightarrow[\beta\eta]{*}$ for the congruence generated by the latter two equations. Both equalities are easily oriented to give the β- and η-reduction rules:

$$(\lambda x.v)(u) \xrightarrow[\beta]{} u\{x \mapsto v\} \qquad \lambda x.u(x) \xrightarrow[\eta]{} u \text{ if } x \notin \mathcal{V}ar(u)$$

The simply typed λ-calculus is confluent and terminating with respect to $\beta\eta$-reductions.

2.7 Higher-order rewrite rules

A (possibly higher-order) *term rewriting system* is a set of rewrite rules $R = \{\Gamma_i \vdash l_i \to r_i : \sigma_i\}_i$, where l_i and r_i are higher-order terms such that l_i and r_i have the same type σ_i in the environment Γ_i. Given a term rewriting system R, a term s rewrites to a term t at position p with the rule $l \to r$ and the substitution γ, written $s \xrightarrow[l \to r]{p} t$, or simply $s \to_R t$, if $s|_p = l\gamma$ and $t = s[r\gamma]_p$.

A term s such that $s \xrightarrow[R]{p} t$ is called *reducible* (with respect to R). $s|_p$ is a *redex* in s, and t is the *reduct* of s. Irreducible terms are said to be in *R-normal form*. A substitution γ is in R-normal form if $x\gamma$ is in R-normal form for all x. We denote by $\xrightarrow[R]{*}$ the reflexive, transitive closure of the rewrite relation $\xrightarrow[R]{}$, and by $\longleftrightarrow_R^*$ its reflexive, symmetric, transitive closure. We are actually interested in the relation $\longrightarrow_{R\beta\eta} = \longrightarrow_R \cup \longrightarrow_\beta \cup \longrightarrow_\eta$.

Given a rewrite relation $\longrightarrow$, a term s is strongly normalizing if there is no infinite sequence of rewrites issuing from s. The rewrite relation itself is *strongly normalizing*, or *terminating*, if all terms are strongly normalizing, in which case it is called a *reduction*. It is confluent if $s \longrightarrow^* u$ and $s \longrightarrow^* v$ implies that $u \longrightarrow^* t$ and $v \longrightarrow^* t$ for some t.

Several examples of higher-order rewrite systems are developped in section 3.

2.8 Reduction Orderings

We will make intensive use of well-founded orderings for proving strong normalization properties. We will use the vocabulary of rewrite systems for orderings. For our purpose, an ordering, usually denoted by $\geq$, is the union of its strict part with α-conversion, and a strict ordering, usually denoted by $>$, is an irreflexive and transitive relation. We will essentially use strict orderings, and hence, we will use the word ordering for them too. *Rewrite orderings* are *monotonic* and *stable* orderings, and *reduction orderings* are in addition *well-founded*. Monotonicity of $>$ is defined as $u > v$ implies $s[u]_p > s[v]_p$ for all contexts $s[\ ]_p$. Stability of $>$ is defined as $u > v$ implies $s\gamma > t\gamma$ for all substitutions γ. Reduction orderings are used to prove termination of rewrite systems by simply comparing the left and right-hand sides of rules. The following results will play a key role, see [6]:

Assume $>_1, \ldots, >_n$ are well-founded orderings on sets $S_1, \ldots, S_n$. Then $(>_1, \ldots, >_n)_{lex}$ is a well-founded ordering on $S_1 \times \ldots \times S_n$. We write $>_{lex}$ in case all sets and all orderings are equal.

Assume $>$ is a well-founded ordering on a set S. Then $>_{mul}$ is a well-founded ordering on the set of multisets of elements of S. It is defined as the transitive closure of the following relation on multisets (using $\cup$ for multiset union):

$$M \cup \{s\} \;>>\; M \cup \{t_1, \ldots, t_n\} \quad \text{if } s : \sigma > t_i : \sigma \ \forall i \in [1..n]$$

We end up with the definition of a reduction ordering operating on higher-order terms, and allowing one's to show that the relation $\longrightarrow_R \cup \longrightarrow_\beta$ is well-founded by simply comparing the lefthand and righthand sides of the rules in R:

Definition 1 *A polymorphic higher-order reduction ordering is a monotonic and stable polimorphic ordering $>$ of the set of higher-order terms, such that $> \cup \longrightarrow_\beta$ is well-founded.*

3 Higher-order recursive path ordering

In this section, we will always write the application operator explicitely. Let $Mul \uplus Lex$ be a partition of $\mathcal{F}$, s.t. all symbols in Lex have a fixed arity, and let $>_{\mathcal{F}}$ be a well-founded ordering on $\mathcal{F}$, called the precedence.

As usual, variables will be considered as constants incomparable among themselves and with other function symbols. We will also assume without loss of generality that there exists a single basic type σ, since, equating them all in case there are several of them, preserves non-termination.

3.1 Definition of the higher-order recursive path ordering

The following higher-order recursive path ordering (HORPO) is essentially the same as Dershowitz's recursive path ordering for first-order terms. The only difference is that we take care of higher-order arguments on the smaller side by having a corresponding bigger higher-order argument on the lefthand side.

Definition 2 *$s : \tau \succ_{horpo} t : \tau$ iff*

1. *$s = f(\overline{s})$ with $f \in \mathcal{F}$, and $s_i \succeq_{horpo} t$ for some $s_i : \tau \in \overline{s}$.*
2. *$f, g \in \mathcal{F}$, $f >_{\mathcal{F}} g$ and $t = g(\overline{t})$ and $\forall t_i \in \overline{t}$
   if $t_i : \tau$ then $s \succ_{horpo} t_i$ otherwise $s_j \succeq_{horpo} t_i : \sigma$ for some $s_j : \sigma \in \overline{s}$.*
3. *$f = g \in Mul$ and $\overline{s}(\succ_{horpo})_{mul}\overline{t}$.*
4. *$f = g \in Lex$ and $\overline{s}(\succ_{horpo})_{lex}\overline{t}$, and $\forall t_i \in \overline{t}$
   if $t_i : \tau$ then $s \succ_{horpo} t_i$ otherwise $s_j \succeq_{horpo} t_i : \sigma$ for some $s_j : \sigma \in \overline{s}$.*
5. *$f \in \mathcal{F}$, $t = @(\overline{t})$ and $\forall t_i \in \overline{t}$ if $t_i : \tau$ then $s \succ_{horpo} t_i$ otherwise $s_j \succeq_{horpo} t_i : \sigma \neq \tau$ for some $s_j : \sigma \in \overline{s}$, where $@(\overline{t})$ is some partial left-flattening of t.*
6. *$s = @(s_1, s_2)$, $t = @(t_1, t_2)$ and $\{s_1, s_2\}(\succ_{horpo})_{mul}\{t_1, t_2\}$.*
7. *$s = \lambda x.u$ and $t = \lambda x.v$ and $u \succ_{horpo} v$.*

There are a few important remarks about this definition:

- Cases 1, 2, 3, and 4 reduce to the usual recursive path ordering for first-order terms.
- Case 7 compares two abstractions. No other case applies to an abstraction on the bigger side. Case 1 may compare a term headed by a function symbol on the bigger side with an abstraction on the smaller side.
- In Case 6, type considerations show that the first arguments are compared together as well as the second's. On the other hand, Case 5 compares a term s headed by a function symbol with an application on the smaller side, for which an appropriate (partial) left-flattening has to be chosen non-deterministically. Type considerations may help finding one quickly. This non-deterministic choice is essential for ensuring stability.
- Multiset symbols may be overloaded while lexicographic symbols must be of fixed arity, this is due to the proof technique we use (the use of Kruskal's tree theorem does not need this assumption).

Example 3 *Taking $map \in Mul$ and $map >_{\mathcal{F}} cons$ we have,*

$$map(cons(x, l), X) \underset{horpo}{\succ} cons(@(X, x), map(l, X)).$$

Since $map >_{\mathcal{F}} cons$, applying case 2, we need to show that $map(cons(x, l), X) \succ_{horpo} @(X, x)$ and $map(cons(x, l), X) \succ_{horpo} map(l, X)$. The latter is true by case 3, since $cons(x, l) \succ_{horpo} l$ by case 1. The first works by case 2, as X is an argument of the first term and $map(cons(x, l), X) \succ_{horpo} x$ by case 1 (applied twice).

3.2 Properties of the higher-order recursive path ordering

The proofs of decidability, transitivity, monotonicity and stability are very similar to the usual proofs for the recursive path ordering while irreflexivity will follow later from strong normalization.

Lemma 4 *$\succ_{horpo}$ is decidable.*

Proof: We check that all recursive calls generated from the initial comparison $s \succeq_{horpo} t$ terminate. This is proved by comparing pairs (s, t) in the well-founded ordering $(\trianglerighteq, \trianglerighteq)_{lex}$. □

The complexity of the recursive path ordering is known to be n^2. We believe that the use of partial flattening should not impact the complexity.

Property 5 *Assume that $s \succ_{horpo} t$ by case 6. Then, for any left-flattening $@(\overline{s})$ of s, there exists a left-flattening $@(\overline{t})$ of t such that $s_i \succeq_{horpo} t_i$ for every $t_i \in \overline{t}$.*

Proof: Letting $\overline{s} = (s_1, \ldots, s_n)$, we proceed by induction on n. Since s is headed by @, case 6 applies, hence $@(s_1, \ldots, s_{n_1}) \succeq_{horpo} t_1$ and $s_2 \succeq_{horpo} w_2$. We conclude by induction hypothesis. □

Property 5 suggest the use of a maximal left-flattening for s in case 6 of the definition. This would however complicate the proof of strong normalization to come.

Property 6 *Let $s = f(\overline{s}) : \tau \succ_{horpo} g(\overline{t})$ by case 2-4, or 5 (in which case $g(\overline{t})$ is a partial left-flattening). Then, for all $t_i \in \overline{t}$, $s \succ_{horpo} t_i$ if $t_i : \tau$ and $s_j \succeq_{horpo} t_i$ for some $s_j \in \overline{s}$, otherwise.*

Lemma 7 *$\succ_{horpo}$ is transitive.*

Proof: Let $s = f(\overline{s}) \succ_{horpo} t = g(\overline{t}) \succ_{horpo} w = h(\overline{w})$. We prove that $s \succ_{horpo} w$ by induction on $|s| + |t| + |w|$. Note that s, t, w must have the same type τ.

1. Let $s \succ_{horpo} t$ by case 1. Then, $s_i \succ_{horpo} t$ for some $s_i : \tau$ and $t \succ_{horpo} w$. By induction hypothesis $s_i \succ_{horpo} w$, which implies $s \succ_{horpo} w$, by case 1.

2. let $t \succ_{horpo} w$ by case 1 and $s \succ_{horpo} t$ by cases 2-4 (note that since $g \in \mathcal{F}$ cases 5-7 cannot apply). Then, $t_i \succ_{horpo} w$ for some $t_i : \tau$. By property 6 $s \succ_{horpo} t_i$. By induction hypothesis, $s \succ_{horpo} w$.

3. Let $s \succ_{horpo} t$ by case 2 and $t \succ_{horpo} w = h(\overline{w})$ by cases 2-4, or 5 (note that cases 6 and 7 cannot apply). By property 6, for every $w_i \in \overline{w}$, either $w_i : \tau$ in which case $t \succ_{horpo} w_i$, or else $w_i : \sigma \neq \tau$ and $t_k \succeq_{horpo} w_i$ for some $t_k : \sigma \in \overline{t}$. In the former case, $s \succ_{horpo} w_i$ by induction hypothesis. In the latter case, by property 6, there exists some $s_j : \sigma$ such that $s_j \succeq_{horpo} t_k$, and therefore, by induction hypothesis, $s_j \succeq_{horpo} w_i$. If $h \in \mathcal{F}$, then $f >_{\mathcal{F}} h$ and $s \succ_{horpo} w$ follows by case 2, otherwise $s \succ_{horpo} w$ follows by case 5.

4. Let $s \succ_{horpo} t$ by case 3 or 4 and $t \succ_{horpo} w$ by case 2 or 5 (note that cases 6 and 7 cannot apply). As in the previous case, by property 6 and induction hypothesis, we have that for every $w_i \in \overline{w}$ either $s \succ_{horpo} w_i$ or $s_j \succeq_{horpo} w_i$ for some $s_j \in \overline{s}$, which, since $f >_{\mathcal{F}} h$, implies $s \succ_{horpo} w$ by case 2 or 5.

5. Let $s \succ_{horpo} t$ and $t \succ_{horpo} w$ by case 3. By induction hypothesis $\overline{s}(\succ_{horpo})_{mul}\overline{t}$ and $\overline{t}(\succ_{horpo})_{mul}\overline{w}$ imply $\overline{s}(\succ_{horpo})_{mul}\overline{w}$. Therefore, $s \succ_{horpo} w$ by case 3.

6. Let $s \succ_{horpo} t$ and $t \succ_{horpo} w$ by case 4. By induction hypothesis $\overline{s}(\succ_{horpo})_{lex}\overline{t}$ and $\overline{t}(\succ_{horpo})_{lex}\overline{w}$ imply $\overline{s}(\succ_{horpo})_{lex}\overline{w}$. As in the previous cases, property 6 and the induction hypothesis imply that for every $w_i \in \overline{w}$, either $s \succ_{horpo} w_i$ or $s_j \succeq_{horpo} w_i$ for some $s_j \in \overline{s}$. Therefore, $s \succ_{horpo} w$ by case 4.

7. Let $s \succ_{horpo} t$ by case 5 and $t \succ_{horpo} w$ by case 6. Let $t = @(\overline{t})$ be the left-flattening of t used in the proof that $s \succ_{horpo} t$. For every $t_i \in \overline{t}$, if $t_i : \tau$ then $s \succ_{horpo} t_i$, and otherwise $s_j \succeq_{horpo} t_i : \sigma \neq \tau$ for some $s_i : \sigma \in \overline{s}$. By Property 5, $w = @(\overline{w})$ and $t_i \succeq_{horpo} w_i$ for every $w_i \in \overline{w}$. By induction hypothesis, $s \succ_{horpo} w_i$ for every $w_i : \tau \in \overline{w}$ and for every $w_i : \sigma \in \overline{w}$, $s_j \succeq_{horpo} w_i : \sigma$ for some $s_j : \sigma$. We conclude by case 5.

8. Let $s \succ_{horpo} t$ and $t \succ_{horpo} w$ by case 6. By induction hypothesis $\overline{s}(\succ_{horpo})_{mul}\overline{t}$ and $\overline{t}(\succ_{horpo})_{mul}\overline{w}$ imply $\overline{s}(\succ_{horpo})_{mul}\overline{w}$. Therefore, $s \succ_{horpo} w$ by case 3.

9. If $s = \lambda x.s'$ then $t = \lambda x.t'$ and $w = \lambda x.w'$ and $s' \succ_{horpo} t'$ and $t' \succ_{horpo} w'$, and by induction hypothesis $s' \succ_{horpo} w'$ which implies $s = \lambda x.s' \succ_{horpo} \lambda x.w'$ by case 7. □

Lemma 8 $\succ_{horpo}$ *is monotonic.*

Proof: Assume that $s \succ_{horpo} t$, and let $h(\ldots, [\,], \ldots)$ be a context. We discuss according to h.

- If $h \in Mul$, then the result follows by case 3.
- If $h \in Lex$, then $\langle \ldots s \ldots \rangle (\succ_{horpo})_{lex} \langle \ldots t \ldots \rangle$. Now, for every $v \in \langle \ldots t \ldots \rangle$, there exists $u \in \langle \ldots s \ldots \rangle$ such that $u \succeq_{horpo} v$. Besides, in case $h(\ldots, s, \ldots)$ and v have the same type, then, $h(\ldots, s, \ldots) \succ_{horpo} v$ by case 1. Therefore, $h(\ldots s \ldots) \succ_{horpo} h(\ldots t \ldots)$ by case 4.
- If $h = @$, $@(s, u) \succ_{horpo} @(t, u)$ and $@(u, s) \succ_{horpo} @(u, t)$ follow by case 6.
- $\lambda x.s \succ_{horpo} \lambda x.t$ follows by case 7. □

Lemma 9 $\succ_{horpo}$ *is stable under substitutions.*

Proof: We prove that $s : \tau \succ_{horpo} t$ implies $s\gamma \succ_{horpo} t\gamma$ for all substitution γ, by induction on $|s| + |t|$. There are several cases according to the definition.

1. If $s \succ_{horpo} t$ by case 1, then $s_i \succeq_{horpo} t$, and by induction hypothesis $s_i\gamma \succeq_{horpo} t\gamma$, and therefore, $s\gamma \succ_{horpo} t\gamma$ by case 1.

2. If $s \succ_{horpo} t$ by case 2, then $s = f(\overline{s})$, $t = g(\overline{t})$, $f >_{\mathcal{F}} g$ and for all $t_i \in \overline{t}$ either $t_i : \tau$ in which case $s \succ_{horpo} t_i$ or $t_i : \sigma \neq \tau$ and $s_j \succeq_{horpo} t_i$ for some $s_j : \sigma \in \overline{s}$. By induction hypothesis, for all $t_i\gamma \in \overline{t\gamma}$ either $t_i\gamma : \tau$ in which case $s\gamma \succ_{horpo} t_i\gamma$ or $t_i\gamma : \sigma \neq \tau$ and $s_j\gamma \succeq_{horpo} t_i\gamma$ for some $s_j\gamma : \sigma \in \overline{s\gamma}$. Theerfore, $s\gamma = f(\overline{s\gamma}) \succ_{horpo} g(\overline{t\gamma}) = t\gamma$ by case 2.

3. If $s \succ_{horpo} t$ by case 3, then $s = f(\overline{s})$, $t = f(\overline{t})$, $f \in Mul$ and $\overline{s}(\succ_{horpo})_{mul}\overline{t}$. By induction hypothesis $\overline{s\gamma}(\succ_{horpo})_{mul}\overline{t\gamma}$, and hence $s\gamma \succ_{horpo} t\gamma$ by case 3.

4. If $s \succ_{horpo} t$ by case 4, then $s = f(\overline{s})$, $t = f(\overline{t})$, $f \in Lex$, $\overline{s}(\succ_{horpo})_{lex}\overline{t}$, and for all $t_i \in \overline{t}$ either $t_i : \tau$ in which case $s \succ_{horpo} t_i$ or $t_i : \sigma \neq \tau$ and $s_j \succeq_{horpo} t_i$ for some $s_j : \sigma \in \overline{s}$. By induction hypothesis $\overline{s\gamma}(\succ_{horpo})_{lex}\overline{t\gamma}$, and as in the precedence case, by induction hypothesis, for all $t_i\gamma \in \overline{t\gamma}$ either $t_i\gamma : \tau$ in which case $s\gamma \succ_{horpo} t_i\gamma$ or $t_i\gamma : \sigma \neq \tau$ and $s_j\gamma \succeq_{horpo} t_i\gamma$ for some $s_j\gamma : \sigma \in \overline{s\gamma}$. Therefore $s\gamma \succ_{horpo} t\gamma$ by case 4.

5. If $s \succ_{horpo} t = @(\overline{t})$ by case 5, then for every $t_i \in \overline{t}$, either $s \succ_{horpo} t_i : \tau$, and $s\gamma \succ_{horpo} t_i\gamma$ by induction hypothesis, or $s_j \succeq_{horpo} t_i : \sigma \neq \tau$ for some $s_j : \sigma \in \overline{s}$, and $s_j\gamma \succ_{horpo} t_i\gamma$ by induction hypothesis. Therefore, since $t\gamma = @\overline{t\gamma}$, we have $s\gamma \succ_{horpo} t\gamma$ by case 5.

6. If $s = @(s_1, s_2) \succ_{horpo} t = @(t_1, t_2)$ by case 6, then $s_1\gamma \succeq_{horpo} t_1\gamma$ and $s_2\gamma \succeq_{horpo} t_2\gamma$ by induction hypothesis, and we conclude by case 6.

7. If $s \succ_{horpo} t$ by case 7, then $s = \lambda x.u$ and $t = \lambda x.v$ and $u \succ_{horpo} v$. Therefore, by induction hypothesis $u\gamma \succ_{horpo} v\gamma$. Assuming that $x \notin \mathcal{D}om(\gamma)$, then $s\gamma = \lambda x.u\gamma \succ_{horpo} \lambda x.v\gamma = t\gamma$ by case 7. □

Now we prove that $\succ_{horpo}$ is a polymorphic ordering. To this end, the ordering $\succ_{horpo}$ is superscripted by the signature in which the proof is performed.

Lemma 10 *Let $\mathcal{F}$ be a polymorphic signature, s and t be two terms of the same type τ with respect to the polymorphic signature $\mathcal{F}$ and such that $s \succ^{\mathcal{F}}_{horpo} t$. Let now ξ be a type substitution. Then s and t are two terms of the same type $\tau\xi$ in the instantiated signature $F\xi$ and $s \succ^{\mathcal{F}\xi}_{horpo} t$.*

Proof: That s and t are well-typed in $\mathcal{F}\xi$ is straightforward. Now, two terms of the same type in $\mathcal{F}$ are still of the same type in $\mathcal{F}\xi$, but two terms of different types in $\mathcal{F}$ may end up being of the same type. This may impact the comparisons in cases 2, 4, and 5. The proof that it does not is by induction on the proof that $s \succ^{\mathcal{F}}_{horpo} t$. We do it for case 2. For every $t_i : \sigma \neq \tau$, there exists $s_j : \sigma$ such that $s_j \succ^{\mathcal{F}}_{horpo} t_i$, hence, by induction hypothesis, $s_j \succ^{\mathcal{F}\xi}_{horpo} t_i$. If $\sigma\xi \neq \tau\xi$, then we are done. Otherwise, s, s_j and t_i have the same type $\sigma\xi$, hence $s \succ^{\mathcal{F}\xi}_{horpo} t_i$ by case 1. Therefore $s \succ^{\mathcal{F}\xi}_{horpo} t$ by case 2.

3.3 Strong normalization of $\succ_{horpo} \cup \longrightarrow_\beta$

The well-foundedness of the recursive path ordering is usually proved by using the fact that it contains the embedding relation which is a well-order by Kruskal's tree theorem. Since we do not know of any non-trivial extension of Kruskal's tree theorem for higher-order terms that includes β-reductions, we will adopt a completely different method, the computability predicate proof method due to Tait and Girard. As a side effect, we obtain a new proof of well-foundedness for the recursive path ordering on first-order terms. Indeed, this proof method will also suggest improvements of our ordering, that will be discussed in section 3.5.

We now start proving well-foundedness of $\longrightarrow \; = \; \succ_{horpo} \cup \longrightarrow_\beta$ for any precedence $>_{\mathcal{F}}$. Our definition of computability for typed terms is standard:

Definition 11 *An algebraic λ-term s is computable iff*

1. *s is a variable,*
2. *s has a basic type and s is strongly normalizable,*
3. *or else s has an arrow type $\sigma \to \tau$ and $@(s,t)$ is computable for every computable term t of type σ.*

We say that a vector $\overline{s}$ of terms is computable iff every component of $\overline{t}$ is computable.

We first recall the properties of the computability predicate. Those depending on the type structure, but not on the actual form of the relation $\longrightarrow$, are given without proof.

Lemma 12 *Every computable term is strongly normalizable.*

Lemma 13 *Assume that t is computable and $t \longrightarrow s$. Then s is computable.*

We call *neutral* a term which is not an abstraction.

Lemma 14 *Let t be a neutral term. Then t is computable iff s is computable for all s such that $t \longrightarrow s$.*

Lemma 15 *Let $\overline{t}$ be a vector of at least two computable terms. Then $@(\overline{t})$ is computable.*

Proof: Easy induction on the length of $\overline{t}$.

Lemma 16 *Let $f \in \mathcal{F}$ and let $\overline{t}$ be a set of terms. If $\overline{t}$ is computable, then $f(\overline{t})$ is computable.*

Proof: Let $t_i : \tau_i$ and $f(\overline{t}) : \tau$. Since terms in $\overline{t}$ are computable, by Lemma 13, they are strongly normalizable. We will use this remark for building our induction argument: we prove that $f(\overline{t})$ is computable by induction on the triple $\langle f, Max_i|\tau_i|, \overline{t}\rangle$ ordered lexicographically by $\langle >_{\mathcal{F}}, >_{\mathbb{N}}, (\longrightarrow)_{stat}\rangle$ where $stat$ is either mul or lex, depending on the symbol f.

Since $f(\overline{t})$ is neutral, by Lemma 14, it is computable iff every s such that $f(\overline{t}) \longrightarrow s$ is computable, which we prove by an inner induction on the size of s. There are several cases to be considered:

1. Let $s = f(t_1, \ldots, t_i', \ldots, t_n)$, with $t_i \longrightarrow_\beta t_i'$. By Lemma 13, t_i' is computable, and since $\longrightarrow_\beta$ preserves types and $\overline{t}(\longrightarrow)_{stat}\overline{s}$ for any status $stat$, s is computable by application of the outer induction hypothesis.

2. Let $f(\overline{t}) \succ_{horpo} s$ by case 1, hence $t_i \succeq_{horpo} s$ for some $t_i \in \overline{t}$. Since t_i is computable, s is computable by Lemma 13.

3. Let $t = f(\overline{t}) \succ_{horpo} s$ by case 2. Then $s = g(\overline{s})$, $f >_{\mathcal{F}} g$ and for every $s_i \in \overline{s}$ either $s_i : \tau$ in which case $t \succ_{horpo} s_i$ or $s_i : \sigma \neq \tau$ and $t_j \succeq_{horpo} s_i$ for some $t_j : \sigma \in \overline{t}$. In the latter case, s_i is computable by Lemma 13. In the former case, s_i is computable by the inner induction hypothesis. Therefore, $\overline{s}$ is computable, and since $f >_{\mathcal{F}} g$, s is computable by the outer induction hypothesis.

For the two remaining cases, $s = f(\overline{s})$, with $s_i : \sigma_i$.

5. If $f(\overline{t}) \succ_{horpo} s$ by case 3, then $\overline{t}(\succ_{horpo})_{mul}\overline{s}$. By definition of the multiset comparison, for every $s_i \in \overline{s}$ there is some $t_j \in \overline{t}$, s.t. $t_j \succeq_{horpo} s_i$, hence, by Lemma 13, s_i is computable. Besides, since comparable terms must have the same type, then the set $\{\sigma_i\}_i$ is a subset of the set $\{\tau_j\}_j$, hence $Max_j|\tau_j| \geq_{\mathbb{N}} Max_i|\sigma_i|$. This allows us to conclude by the outer induction hypothesis that s is computable, using the second component of the measure if $Max_j|\tau_j| = Max_i|\sigma_i|$, and the third otherwise.

6. If $f(\overline{t}) \succ_{horpo} s$ by case 4, then $\overline{t}(\succ_{horpo})_{lex}\overline{s}$ and for every $s_i \in \overline{s}$ either $\sigma_i = \tau$ in which case $t \succ_{horpo} s_i$ or $\sigma_i \neq \tau$ and $t_j \succeq_{horpo} s_i$ for some $t_j : \sigma_i \in \overline{t}$. As in the precedence case, this implies that $\overline{s}$ is computable. Now, since f must have a fixed arity, $Max_j|\tau_j| = Max_i|\sigma_i|$. Then, since $\overline{t}(\succ_{horpo})_{lex}\overline{s}$, s is computable by the outer induction hypothesis.

7. If $f(\overline{t}) \succ_{horpo} s$ by case 5, let $@(s_1, \ldots, s_n)$ be the partial left-flattening of s used in that proof. By the same token as in case 2, every term in $\overline{s}$ is computable, hence s is computable by Lemma 15.

8. If $t \succ_{horpo} s$ by case 6 then $t = @(u, v)$ and $s = @(u', v')$. By definition, $u \succeq_{horpo} u'$ and $v \succeq_{horpo} v'$, hence, by Lemma 13, u' and v' are computable. It follows that s is computable. □

Lemma 17 *$\lambda x : \sigma.u$ is computable iff $u\{x \mapsto w\}$ is computable for every well-typed computable term $w : \sigma$.*

Proof: Again, the only if part is Lemma 13. For the converse, we prove that $@(\lambda x.u, w)$ is computable for an arbitrary term $w : \sigma$ if $u\{x \mapsto w\}$ and w are computable.

Since variables are computable, $u = u\{x \mapsto x\}$ is computable by assumption. By Lemma 12, u and w are strongly normalizable. We prove that $@(\lambda x.u, w)$ is computable by induction on the set $\{u, w\}$ ordered by $(\longrightarrow)_{mul}$.

By Lemma 14, since $@(\lambda x.u, w)$ is neutral, it is computable iff v is computable for all v such that $@(\lambda x.u, w) \longrightarrow v$. There are several cases to be considered.

1. Let $v = @(\lambda x.u', w)$ with $u \longrightarrow u'$. Then, $u\{x \mapsto w\} \longrightarrow u'\{x \mapsto w\}$. Hence, by assumption and Lemma 13, $u'\{x \mapsto w\}$ is computable for every computable $w : \sigma$. Since $\{u, w\}(\longrightarrow)_{mul}\{u', w\}$, by induction hypothesis, v is computable.

2. Let $v = @(\lambda x.u, w')$ with $w \longrightarrow_{\beta} w'$. Then $\{u, w\}(\longrightarrow)_{mul}\{u, w'\}$, and by induction hypothesis, v is computable.

3. Let $v = u\{x \mapsto w\}$. Then v is computable by assumption.

4. Let $t = @(\lambda x.u, w) \succ_{horpo} v = @(v', w')$ by case 6. For typing reasons, $w \succeq_{horpo} w'$ and $\lambda x.u \succeq_{horpo} v'$. Hence, w' is computable by Lemma 13. On the other hand, by property of the ordering, $v' = \lambda x.u'$ with $u \succeq_{horpo} u'$. Now, since $u\{x \mapsto w\}$ is computable for every computable $w : \sigma$ by assumption, $u'\{x \mapsto w\}$ is computable for every computable $w : \sigma$ by Lemma 13. Hence $\lambda x.u'$ is computable, and therefore v is computable.

□

Lemma 18 *Let γ be a computable substitution and let t be a linear algebraic λ-term. Then $t\gamma$ is computable.*

Proof: The proof proceeds by induction on $|t|$.

1. t is a variable x. Then $x\gamma$ is computable by assumption.

2. t is an abstraction $\lambda x.u$. By Lemma 17, $t\gamma$ is computable if $u\gamma\{x \mapsto w\}$ is computable for every well-typed computable term w. Taking $\delta = \gamma \cup \{x \mapsto w\}$, we have $u\gamma\{x \mapsto w\} = u(\gamma \cup \{x \mapsto w\})$ since x may not occur in γ.

   We proceed to prove that $u\delta$ is computable. Let ξ be a bijection between the set of pairs (x, p) of a variable and a position, and the set of variables $\mathcal{X} \setminus \mathcal{BV}ar(u)$. We now linearize u according to ξ, resulting in u', by replacing every occurence of a free variable x at position p in u by the fresh variable $\xi(x, p)$. Let δ' the substitution s.t. $\xi(x, p)\delta' = x\delta$. Clearly, $u\delta = u'\delta'$.

   Then, since $|t| > |u'|$, by induction hypothesis, $u'\delta' = u\delta$ is computable.

3. $t = @(t_1, t_2)$ or $t = f(t_1, \ldots, t_n)$, and some t_i is not a variable. Let $s = f(x_1, \ldots, x_n)$ or $s = @(x_1, x_2)$ and $\delta = \{x_i \mapsto t_i \mid 1 \leq i \leq n\}$. By induction hypothesis $t_i\gamma$ is computable for all i, hence δ is computable. Since not all t_i are variables, $|t| > |s|$ and, by induction hypothesis $s\delta$ is computable.

4. $t = @(x_1, x_2)$ or $t = f(x_1, \ldots, x_n)$. Then, by Lemma 16, $t\gamma$ is computable. □

Theorem 19 $\succ_{horpo}$ *is a decidable polymorphic higher-order reduction ordering.*

Proof: Monotonicity, stability under substitutions and polymorphism follow from the previous Lemmas. For the well-foundedness and compatibility with β-reduction we need to show that every term is strongly normalizable with respect to $\longrightarrow$. This follows by using a computable variable renaming substitution γ and a linear algebraic λ-term s such that $s\gamma = t$. Then, since γ is computable, $t = s\gamma$ is computable by Lemma 18, and strongly normalizable by Lemma 12.

□

Note that we did not consider the η conversion rule in this section. Indeed, the above arguments go through with η as well, used as either a contraction or an expansion.

3.4 Examples

Example 20 *Gödel's recursor.*

$$rec(0,u,X) \rightarrow u$$
$$rec(S(x),u,X) \rightarrow @(X,x,rec(x,u,X))$$

The first rule succeeds immediately by case 1. For the second rule, we apply the precedence rule, and need to show recursively that (i) $X \succeq_{horpo} X$, *(ii)* $rec(S(x),u,X) \succ_{horpo} x$, *and (iii)* $rec(S(x),u,X) \succ_{horpo} rec(x,u,X)$. *(i) is trivial. (ii) is by case 1 applied twice. (iii) is by case 3, calling recursively for* $S(x) \succ_{horpo} x$, *which succeeds by case 1.*

We can of course now add some defining rules for sum and product:

$$x+0 \rightarrow 0$$
$$x+S(y) \rightarrow S(x+y)$$
$$x*y \rightarrow rec(y,0,\lambda z.x+z)$$

The first two first-order rules are easily taken care of. For the third, we use the precedence $* >_{\mathcal{F}} rec$ *to eliminate the rec operator. But the computation will fail, since there is no higher-order subterm of* $x*y$ *to take care of the righthand side subterm* $\lambda z.x+z$.

Note that we have indeed proved Gdel's polymorphic recursor, for which the output type of *rec* is any given type. This is because we do not care about types in our comparisons, provided two compared terms are typable with the same type.

The following example presents a set of rewrite rules defining the insertion algorithm for the (ascending or descending) sort of a list of natural numbers.

Example 21 *Insertion Sort.*
$\mathcal{S} = \{Nat, List\}$,
$\mathcal{F}_{List} = \{nil\}, \mathcal{F}_{Nat\times Nat\rightarrow Nat} = \{max, min\}, \mathcal{F}_{Nat\times List\rightarrow List} = \{cons\}$,
$\mathcal{F}_{Nat\times List\times(Nat\times Nat\rightarrow Nat)\times(Nat\times Nat\rightarrow Nat)\rightarrow List} = \{insert\}$,
$\mathcal{F}_{List\times(Nat\times Nat\rightarrow Nat)\times(Nat\times Nat\rightarrow Nat)\rightarrow List} = \{sort\}$,
$\mathcal{F}_{List\rightarrow List} = \{ascending\_sort, descending\_sort\}, \mathcal{X}_{Nat\times Nat\rightarrow Nat} = \{X,Y\}$.

$$\begin{array}{rclrcl} max(0,x) & \rightarrow & x & \qquad min(0,x) & \rightarrow & 0 \\ max(x,0) & \rightarrow & x & min(x,0) & \rightarrow & 0 \\ max(s(x),s(y)) & \rightarrow & s(max(x,y)) & min(s(x),s(y)) & \rightarrow & s(min(x,y)) \end{array}$$

We simply need the precedence $max, min >_{\mathcal{F}} 0$ *for these first-order rules.*

$$\begin{array}{l} insert(n,nil,X,Y) \rightarrow cons(n,nil) \\ insert(n,cons(m,l),X,Y) \rightarrow \\ \quad cons(X(n,m),insert(Y(n,m),l,X,Y)) \end{array}$$

Note that the lefthand and righthand sides of each rule are written in η*-expanded form. The first insert rule is easily taken care of by applying case 2 with the precedence insert* $>_{\mathcal{F}}$ *cons. For the second insert rule, we apply first case 2, and we recursively need to show that*

$$insert(n,cons(m,l),X,Y) \underset{horpo}{\succ} @(X,n,m)$$

which follows by applying rule 2, and showing recursively that

$$insert(n,cons(m,l),X,Y) \underset{\text{hos}}{\longrightarrow} insert(Y(n,m),l,X,Y)$$

which succeeds as well, with a right-to-left lexicographic status for insert, by calling recursively the (sucessfull by rule 2) computation of $insert(n, cons(m,l), X, Y) \succ_{horpo} @(Y, n, m)$.

$$\begin{array}{l} sort(nil, X, Y) \to nil \\ sort(cons(n,l), X, Y) \to \\ \quad insert(n, sort(l, X, Y), X, Y) \end{array}$$

Again, these rules are easily oriented by $\succ_{horpo}$*, by using the precedence sort* $>_{\mathcal{F}}$ *insert.*

$$\begin{array}{l} ascending\_sort(l) \to sort(l, \lambda xy.min(x,y), \lambda xy.max(x,y)) \\ descending\_sort(l) \to sort(l, \lambda xy.max(x,y), \lambda xy.min(x,y)) \end{array}$$

On the other hand, $\succ_{horpo}$ *fails to orient these last two seemingly easy rules.*

Let us also point out that we have always written our higher-order variables in η*-long form. Doing otherwise would also have led to failures.*

3.5 Improvements

The ordering is quite sensitive to innocent variations of the language, like adding (higher-order) dummy arguments to righthand sides, or η-converting expressions. We will now solve these problems by improving our definition in the light of the strong normalization proof. In that proof, it was crucial to show the computability of the righthand side subterms by using the lefthand side subterms. In our definition, we actually require that for each righthand side subterm v, there exists a lefthand side subterm u such that $u \succeq_{horpo} v$. Assuming u is computable, then v is computable by Lemma 13. But any computability preserving operation applied to the lefthand side subterms in order to construct a term of the appropriate type would do as well. For example, the higher-order variable X is computable if and only if $\lambda x.X(x)$ is computable. Therefore, both forms may coexist.

Definition 22 *Given a term* $t = f(\overline{t})$*, we define its computable closure* $\mathcal{CC}(t)$ *as* $\mathcal{CC}(t, \emptyset)$*, where* $\mathcal{CC}(t, \mathcal{V})$*, with* $\mathcal{V} \cap \mathcal{V}ar(t) = \emptyset$*, is the smallest set of well-typed terms containing all variables in* $\mathcal{V}$ *and all terms in* $\overline{t}$ *as well as their basic sort subterms, and closed under the following operations:*

1. *precedence: let* g *such that* $f >_{\mathcal{F}} g$*, and* $\overline{s} \in \mathcal{CC}(t, \mathcal{V})$*; then* $g(\overline{s}) \in \mathcal{CC}(t, \mathcal{V})$*;*

2. *recursive call: let* $\overline{s}$ *be a sequence of computable terms such that* $f(\overline{s})$ *is well typed and* $\overline{t}(\longrightarrow_{\beta} \cup \rhd)_{stat_f} \overline{s}$*; then* $f(\overline{s}) \in \mathcal{CC}(t, \mathcal{V})$*;*

3. *application: let* $s : \sigma_1 \to \ldots \to \sigma_n \to \sigma \in \mathcal{CC}(t, \mathcal{V})$ *and* $u_i : \sigma_i \in \mathcal{CC}(t, \mathcal{V})$ *for every* $i \in [1..n]$*; then* $@(s, u_1, \ldots, u_n) \in \mathcal{CC}(t, \mathcal{V})$*;*

4. *abstraction: let* $x \notin \mathcal{V}ar(t)$ *and* $s \in \mathcal{CC}(t, \mathcal{V} \cup \{x\})$*; then* $\lambda x.s \in \mathcal{CC}(t, \mathcal{V})$*;*

5. *reduction: let* $u \in \mathcal{CC}(t, \mathcal{V})$*, and* $u \longrightarrow_{\beta} v$*; then* $v \in \mathcal{CC}(t, \mathcal{V})$*.*

Because of rules 1, 2 and 3, the compuation closure of a term may be infinite. We can of course put a bound on the number of times these rule can be applied. We call it the bounded computable closure.

Lemma 23 *For every* t*, the bounded computable closure of* t *is finite*

Proof: By induction on the size of V. □

Lemma 24 *Assume that* $u \in \mathcal{CC}(\overline{t})$*. Then,* $u\gamma \in \mathcal{CC}(\overline{t\gamma})$ *for every substitution* γ*.*

Proof: Easy induction on the definition of the computable closure. □

We now show that terms in the computable closure of a term are computable under the appropriate hypotheses for its use.

Lemma 25 *Assume $\overline{t} : \overline{\tau}$ is computable, as well as every term smaller than $t = f(\overline{t})$ in the ordering $\langle >_{\mathcal{F}}, >_{I\!N}, (\longrightarrow \cup \rhd)_{stat} \rangle$ operating on triples $\langle f, Max_i|\tau_i|, \overline{t} \rangle$ where stat is the status of f. Then, every term in $\mathcal{CC}(t)$ is computable.*

The precise formulation of this statement arises from its forthcoming use inside the proof of Lemma 16. Note that we have replaced the ordering $(\longrightarrow)_{stat}$ used in the proof of lemma 16 by the ordering $(\longrightarrow \cup \rhd)_{stat}$. This is possible, because the latter is well-founded when so is the former, since $\longrightarrow$ is indeed a rewrite relation,see [6].

Proof: Let $u \in \mathcal{CC}(t, \mathcal{V})$. We show by induction on the construction of u that $u\gamma$ is computable for every computable substitution γ of domain $\mathcal{V}ar(t)$ (note that $\mathcal{V} \cap \mathcal{V}ar(t) = \emptyset$). If $u \in \overline{t}$, by assumption, $u\gamma$ is computeble, and if $u \in \mathcal{V}$ then $u\gamma = u$ is computable by definition. If u is a basic sort subterm of some $v \in \overline{t}$. By assumption, $v\gamma$ is computable, hence it is strongly normalizable by Lemma 12. Then, $u\gamma$ is strongly normalizable as well, and since it is of basic sort, it is computable. Now we discuss the successive cases closure of definition 22.

1. case 1: by assumption, since $f >_{\mathcal{F}} g$.
2. case 3. By induction and application of the definition.
3. case 2. By assumption, since $\overline{t}(\longrightarrow_{\beta} \cup \rhd)_{stat_f}\overline{s}$.
4. case 4. By induction hypothesis, $s(\gamma \cup \{x \mapsto w\})$ is computable for any w, and by Lemma 17, $(\lambda x.s)\gamma$ is computable.
5. case 5. By Lemma 13. □

We can now modify our ordering as follows:

Definition 26 $s : \tau \succ_{horpo} t : \tau$ *iff*

1. *$s = f(\overline{s})$ with $f \in \mathcal{F}$, and $s_i \succeq_{horpo} t$ for some $s_i : \tau \in \overline{s}$.*
2. *$f, g \in \mathcal{F}$, $f >_{\mathcal{F}} g$ and $t = g(\overline{t})$ and $\forall t_i \in \overline{t}$ either $t_i \in \mathcal{CC}(s)$ or if $t_i : \tau$ then $s \succ_{horpo} t_i$ otherwise $s_j \succeq_{horpo} t_i$ for some $s_j \in \overline{s}$*
3. *$f = g \in Mul$ and $\overline{s}(\succ_{horpo})_{mul}\overline{t}$.*
4. *$f = g \in Lex$ and $\overline{s}(\succ_{horpo})_{lex}\overline{t}$, and $\forall t_i \in \overline{t}$ either $t_i \in \mathcal{CC}(s)$ or if $t_i : \tau$ then $s \succ_{horpo} t_i$ otherwise $s_j \succeq_{horpo} t_i$ for some $s_j \in \overline{s}$*
5. *$f \in \mathcal{F}$, $t = @(\overline{t})$, and $\forall t_i \in \overline{t}$ either $t_i \in \mathcal{CC}(s)$ or if $t_i : \tau$ then $s \succ_{horpo} t_i$ otherwise $s_j \succeq_{horpo} t_i$ for some $s_j \in \overline{s}$ or $t_i \in \mathcal{CC}(s)$, where $@(\overline{t})$ is some partial left-flattening of t.*
6. *$s = @(\overline{s})$, $t = @(\overline{s})$ and $\overline{s}(\succ_{horpo})_{mul}\overline{t}$.*
7. *$s = \lambda x.u$ and $t = \lambda x.v$ and $u \succ_{horpo} v$.*

Since the computational closure of a term is in general infinite, it is not clear whether the membership of a term to the closure is decidable. Of course, the membership becomes decidable for the bounded closure. We will therefore assume this restriction for practice.

Lemma 27 $\succ_{horpo}$ *is monotonic, stable, well-founded, and polymorphic.*

Proof:

1. monotonicity: the same induction as previously goes through.
2. stability: we replace the induction of the sum $|s| + |t|$ by an induction on the complexity of the proof of $s \succ_{horpo} t$, and use of Lemma 24.
3. well-foundedness: same proof as previously, and use of Lemma 25.
4. polymorphism: same proof as before, and noticing that the membership to $\mathcal{CC}(\overline{t})$ does not depend on types except when taking basic sort subterms which do not disappear by type instantiation.

We can now conclude:

Theorem 28 $\succ_{horpo}$ *is decidable and included in a polymorphic higher-order reduction ordering.*

Proof: First, the same induction argument as in the proof of lemma 4, together with Lemma 23, can be used for showing decidability, Thanks to Lemma 25, the strong normalization proof of this improved ordering is exactly the same as previously. Then by lemma 27 we conclude that the transitive closure of $\succ_{horpo}$ is a higher-order polymorphic reduction ordering. $\square$

Our definition of the higher-order recursive path ordering can be improved again in two different ways.

(i) By replacing the conditions $t_i \in \mathcal{CC}(s)$ or $s_j \succeq_{horpo} t_i$ for some $s_j \in \overline{s}$ in cases 2 and 4 of the definition of the improved higher-order recursive path ordering by the weaker condition $u \succeq_{horpo} t_i$ for some $u \in \mathcal{CC}s$.

(ii) By replacing the condition $u \longrightarrow_\beta v$ in case 5 of the definition of the computable closure, by the weaker condition $u \longrightarrow_\beta \cup \succ_{horpo} v$.

(i) does not complicate the proofs at all, but is of very little practical value while increasing the complexity of the computation.

(ii) may have a theoretical impact. We believe that this may yield a transitive relation, hence an ordering, to the price of important complications in the proofs, since, with this modification, the ordering and the computational closure become mutually inductive definitions. On the other hand, transitivity is not so important, since well-foundedness does not rely on transitivity. It is essentially a matter of aesthetics.

This new definition is much stronger than the previous one. In addition to prove the strong normalization property of the remaining rules of the sorting example, and for the same reason, we can easily add new rules to the recusor example, such as the following:

Example 29 $n * m \quad \rightarrow \quad rec(n, 0, \lambda xy.m + y)$

can be proved terminating with the precedence: $* >_{\mathcal{F}} \{+, 0\}$, *since* $\lambda xy.m + y \in CS(n * m)$.

4 Normalized rewriting

We will first assume that all function symbols are monomorphic, and that the output type of a function symbol in $\mathcal{F}$ is a basic sort, before to remove these assumption in Section 4.4.

4.1 η-long normal forms

We now consider rewriting modulo β and η conversions, by using η-long normal forms, see also [11]. These normal forms are obtained by applying η as an expansion. Since it may not terminate, its use is restricted by spelling out in which context it applies:

$$C[u]_p \underset{\eta}{\longrightarrow} C[\lambda x.u(x)]_p \ \textbf{if} \begin{cases} u : \sigma \rightarrow \tau \\ x \notin \mathcal{V}ar(u) \\ u \text{ is not an abstraction} \\ C[u]|_q \text{ is not an application in case } p = q \cdot 1 \end{cases}$$

The simply typed λ-calculus is confluent and terminating with respect to β-reductions and η-expansions. We write $s{\downarrow_\beta}$, $s{\downarrow}$ for short, for the unique β-normal form of a term, $s\uparrow^\eta$, for the unique η-long form of a term wrt. η-expansion, and $u \updownarrow_\beta^\eta$, $s \updownarrow$ for short, for its unique (up to α-equivalence) normal form with respect to β-reductions and η-expansions, also called η-long normal form.

Definition 30 *An algebraic λ-term s is* normalized *if $s = s \updownarrow$.*

Since β-reduction and η-expansion are confluent and η-expansion does not create β-redexes, normalized terms can be obtained by β-normalization followed by η-expansion. As a consequence, these normal forms satisfy the following property, which justifies the terminology:

Lemma 31 *Normalized terms are of the form $\lambda x_1 \ldots x_m.F(u_1, \ldots, u_n)$, for some $m \geq 0$, $F \in \mathcal{F} \cup \mathcal{X}$ and normalized terms $u_1, \ldots, u_n$, such that $F(u_1, \ldots, u_n)$ is of basic type.*

Corollary 32 *Let $s|_p$ and t be terms of the same type. Then $s[t]_p$ is normalized if so are s and t.*

The use of the extensionality rule as an expansion, and the assumption that the output type of each algebraic symbol is basic allows us to hide applications except when $F \in \mathcal{X}$. Abstraction-free normalized terms will be called *algebraic*. Algebraic terms are first-order when they do not contain higher-order variables. One advantage of η-expansions is that higher-order algebraic terms look and sometimes behave as first-order ones. The only relevant difference, discussed now, appear when substitutions on higher-order variables are applied:

Property 33 *If s and t are in η-long form then $s\{x \mapsto t\}$ is in η-long form.*

Proof: It is trivial since any η-expansion that can be applied to $s\{x \mapsto t\}$ can either be applied to s or to t, which contradicts the assumption. □

Property 34 *If t is in η-long form and $t \longrightarrow_\beta t'$ then t' is in η-long form.*

Proof: Let $t|_p = (\lambda x.u)v$ for some position p and $t' = t[u\{x \mapsto v\}]_p$. Since t is in η-long for then u and v are in η-long form (note that none of them is the first argument of an application), and hence, by property 35, $u\{x \mapsto v\}$ is in η-long form. Now p cannot be an η-redex in t' since as β-reduction preserves types, p should be an η-redex in t. Finnaly any other redex position in t' should be trivially an η-redex position in t. □

This last property implies that normalized terms can also be obtained by η-expansion followed by β-normalization.

Lemma 35 *If t is a normalized term and γ is a normalized substitution then $(t\gamma) \updownarrow_\beta^\eta = (t\gamma){\downarrow_\beta}$.*

Proof: By a simple induction on the number of variables non-trivially assigned in γ and property 35, we have that $t\gamma$ is in η-long form. Then, by a simple induction on the number of β-reduction steps and property 34 we have that $t\gamma{\downarrow_\beta}$ is in η-long form. □ In words, normalization of a substitution instance requires β-normalization only.

4.2 Normalized higher-order rewriting [11]

A (possibly higher-order) *term rewriting system modulo* $\beta\eta$ is a set of rewrite rules $R = \{\Gamma_i \vdash l_i \to r_i : \sigma_i\}_i$, where l_i and r_i are normalized terms such that l_i and r_i have the same type σ_i in the environment Γ_i. Given a term rewriting system R, a normalized term s rewrites to a term t at position p with the rule $l \to r$ and the substitution γ, written $s \xrightarrow[l \to r]{p} t$, or simply $s \to_{R_{\beta\eta}} t$, if $s|_p \longleftrightarrow^*_{\beta\eta} l\gamma$ and $t = s[r\gamma\downarrow]_p$. By Corollary 32, t is normalized since s is.

Note that if the applied rule $l \to r$ has an arrow type σ then we have to require $s|_p$ to have type σ too, since otherwise $s|_p$ and $r\gamma\downarrow$ may have different types. If the type of the rule is basic then this restriction always holds.

Definition 36 *An ordering* $>$ *on normalized higher-order terms is* ground monotonic*, (that is,* stable under normalized ground contexts*), if* $u > v$ *implies* $s[u]_p > s[v]_p$ *for all terms* u, v, s *and positions* p *for which* $s[u]_p$ *and* $s[v]_p$ *are typable ground terms in normal form.*

An ordering $>$ *on normalized higher-order terms is* ground stable *if* $u > v$ *implies* $(u\gamma)\downarrow > (v\gamma)\downarrow$ *for all terms* u, v *in normal form and normalized substitutions* γ *such that* $u\gamma$ *and* $v\gamma$ *are typable ground terms.*

A normalized higher-order rewrite ordering *is an ordering on higher-order terms which is ground monotonic and ground stable. A* normalized higher-order reduction ordering *is a well-founded normalized higher-order rewrite ordering.*

Theorem 37 *Let* R *be a set of normalized higher-order rewrite rules. Then* R *is terminating for normalized higher-order rewriting iff there exists a normalized higher-order reduction ordering* $>$ *such that* $l > r$ *for each rewrite rule* $l \to r \in R$*.*

4.3 The normalized higher-order recursive path ordering

We now adapt $\succ_{horpo}$ to normalized higher-order rewriting. A first idea is to use $\succ_{horpo}$ followed by β-normalization (η-expansions are actually not needed),

$$s > t \quad \text{iff} \quad s \underset{horpo}{\succ} t' \text{ for some } t' \text{ such that } t = t'\downarrow_\beta$$

which as we have seen is well-founded and monotonic. Unfortunately it is not stable under *normalized* substitutions. The problem comes from applications, as shown in the example:

Example 38 $@(X, f(a)) = X(f(a)) \succ_{horpo} X(a) = @(X, a)$ *and* $X(a) \succ_{horpo} a$
while instantiating these comparisons with the substitution $\gamma = \{X \mapsto \lambda y.a\}$ *yields:*
$X(f(a))\gamma\downarrow \;=\; X(a)\gamma\downarrow \;=\; a$ *which contradicts stability under normalized substitutions.*

Fortunatelly this is the only case in which this may happen. By Lemma 31, since s is a normalized term, it should be of the form $\lambda x_1 \dots x_m.F(u_1, \dots, u_n)$, for some $m \geq 0$, $F \in \mathcal{F} \cup \mathcal{X}$ and normalized terms $u_1, \dots, u_n$, such that $F(u_1, \dots, u_n)$ is of basic type. Hence s can be headed by @, only if its first argument is a variable. Another problem of the previous definition is that it is not computable, since we have to try $s \succ_{horpo} t'$ for all terms that β-normalize into t.

Now we restrict $\succ_{horpo}$, obtaining the *normalized higher-order recursive path ordering* in which these comparisons are disallowed and which is adapted to work on normalized terms. First, we restrict the notion of computable closure to deal with normalized terms.

Definition 39 *Given a normalized term* $s = f(\overline{s})$ *with* $f \in \mathcal{F}$*, we define its normalized computable closure* $\mathcal{NCC}(s)$ *as* $\mathcal{NCC}(s, \emptyset)$*, where* $\mathcal{NCC}(s, \mathcal{V})$*, with* $\mathcal{V} \cap \mathcal{V}ar(s) = \emptyset$*, is the smallest set of well-typed terms containing all variables in* $\mathcal{V}$ *and all terms in* $\overline{s}$ *as well as their basic sort subterms* u *such that* $t \rhd_{\mathcal{X}} u$ *for some* $t \in \overline{t}$*, and closed under the following operations:*

1. *precedence: let g such that $f >_{\mathcal{F}} g$, and $\overline{t} \in \mathcal{NCC}(s, \mathcal{V})$; then $g(\overline{t}) \in \mathcal{NCC}(s, \mathcal{V})$;*
2. *recursive call: let $\overline{t} \in \mathcal{NCC}(s, \mathcal{V})$ and $\overline{s}(\rhd_{\mathcal{X}})_{stat_f}\overline{t}$; then $f(\overline{t}) \in \mathcal{NCC}(s, \mathcal{V})$;*
3. *abstraction: let $t \in \mathcal{NCC}(s, \mathcal{V} \cup \{x\})$ for some $x \notin \mathcal{V}ar(s)$; then $\lambda x.t \in \mathcal{NCC}(s, \mathcal{V})$;*

Definition 40 *Let $>_{\mathcal{F}}$ a precedence on $\mathcal{F} = Mul \cup Lex$. Then $s : \tau \succ_{nhorpo} t : \tau$ for normalized terms s and t iff*

1. *$s = f(\overline{s}), f \in \mathcal{F}$ and $s_i \succeq_{nhorpo} t$ for some $s_i : \tau \in \overline{s}$*
2. *$s = f(\overline{s}), t = g(\overline{t})$, $f >_{\mathcal{F}} g$ and $\forall t_i \in \overline{t}$ either $t_i \in \mathcal{NCC}(s)$ or if $t_i : \tau \in \mathcal{S}$ then $s \succ_{nhorpo} t_i$ otherwise $s_j \succeq_{nhorpo} t_i : \sigma$ for some $s_j : \sigma \in \overline{s}$.*
3. *$s = f(\overline{s})$ for some $f \in \mathcal{F}$, $t = u[\overline{v}]_{\overline{p}}$ with $\forall p \in \overline{p},\ p \neq \Lambda$ and $\forall v' \in \overline{v}$, $\mathcal{V}ar(v') \subseteq \mathcal{V}ar(t)$, and for all $v' \in \overline{v} \cup \{\lambda\overline{x}.u[\overline{x\uparrow}]_{\overline{p}}$ with $\overline{x}$ fresh$\}$, either $v' \in \mathcal{NCC}(s)$, or if $v' : \tau \in \mathcal{S}$ then $s \succ_{nhorpo} v'$ otherwise $s_j \succeq_{nhorpo} v' : \sigma$ for some $s_j : \sigma \in \overline{s}$.*
4. *$s = f(\overline{s}), t = f(\overline{t})$, $f \in Mul$ and $\overline{s}(\succ_{nhorpo})_{mul}\overline{t}$.*
5. *$s = f(\overline{s}), t = f(\overline{t})$, $f \in Lex$ and $\overline{s}(\succ_{nhorpo})_{lex}\overline{t}$, and $\forall t_i \in \overline{t}$ either $t_i \in \mathcal{NCC}(s)$ or if $t_i : \tau \in \mathcal{S}$ then $s \succ_{nhorpo} t_i$ otherwise $s_j \succeq_{nhorpo} t_i : \sigma$ for some $s_j : \sigma \in \overline{s}$.*
6. *$s = \lambda x.u$ and $t = \lambda x.v$ and $u \succ_{nhorpo} v$.*

There are some remarks to be considered:

- In cases 2, 3 and 5, the condition $\tau \in \mathcal{S}$ is a tautology, by our assumption that the output type of a function symbol is a sort. This assumption will be removed in Section 4.4, making the condition necessary at that time.
- No case applies when s or t are headed by @, since @ $\notin \mathcal{F}$;
- In case 3 (used in example 55) all variables in $\overline{x}$ occur in u as leaves and hence $u' = \lambda\overline{x}.u[\overline{x\uparrow}]_{\overline{p}}$ is normalized; Note that finitely many u need to be considered;
- As function symbols have a basic output type, no case but case 6 applies to an abstraction t.

Property 41 *Let t be a normalized term. Then membership to $\mathcal{NCC}(t)$ is decidable.*

Proof: We prove that given a normalized term t' and a set of variables $\mathcal{V}$, s.t. $\mathcal{V} \cap \mathcal{V}ar(t) = \emptyset$, it is decidable whether $t' \in \mathcal{NCC}(t, \mathcal{V})$ or not by induction on —t'— and case analysis.

First, it is decidable if t' is in $\mathcal{V}$ or in $\overline{t}$. Then by induction hypothesis, it is decidable whether $\overline{t'} \in \mathcal{NCC}(t, \mathcal{V})$ or not, and hence it is decidable whether case 1 applies or not.

Since it is decidable whether $\overline{t}(\rhd_x)_{stat_f}\overline{t'}$, it is decidable whether case 2 applies or not.

Finally if $t' = \lambda x.u$ with $x \notin \mathcal{V}ar(t)$, by induction hypothesis it is decidable whether $s \in \mathcal{NCC}(t, \mathcal{V} \cup \{x\})$ or not, and hence it is decidable whether case 3 applies or not. □

Lemma 42 *$\succ_{nhorpo}$ is well-defined on normalized terms and is decidable.*

Proof: It is enough to check that all recursive calls generated from the initial comparison $s \succeq_{nhorpo} t$ terminate and operate on normalized terms. This is proved by comparing pairs (s, t) in the well-founded ordering $(\unrhd, \unrhd)_{lex}$. Since, by property 41, membership to $\mathcal{NCC}(s)$ is decidable, all cases are as for $\succ_{horpo}$ except case 3 in this case for all $v' \in \overline{v}$ we have $t \rhd v'$ and hence v' is a normalized term. Therefore by induction hypothesis, we have that all recursive calls generated from the comparison $s \succ_{nhorpo} v'$ or $s_j \succeq_{nhorpo} v'$ terminate and operate on normalized terms. Finally for $u' = \lambda\overline{x}.u[\overline{x\uparrow}]_{\overline{p}}$, since it cannot have type τ and u' is normalized, we have that, by induction hypothesis, all recursive calls generated from the comparison $s_j \succeq_{nhorpo} u'$ terminate and operate on normalized terms. □

Lemma 43 *Let t and t' be normalized terms. If $t' \in \mathcal{NCC}(t)$ then $t'\gamma\!\downarrow \in \mathcal{NCC}(t\gamma\!\downarrow)$.*

Proof: Let $\mathcal{V}$ be a set of variables s.t. $\mathcal{V} \cap \mathcal{V}ar(t) = \emptyset$. We prove that $t' \in \mathcal{NCC}(t, \mathcal{V})$ implies $t'\gamma\!\downarrow \in \mathcal{NCC}(t\gamma\!\downarrow, \mathcal{V})$, by induction on the construction of t'. We assume that $\mathcal{D}om(\gamma) \subseteq \mathcal{V}ar(t)$. We discuss on the cases of definition 22.

If $t' \in \mathcal{V}$ then $t'\gamma\!\downarrow = t' \in \mathcal{V}$, and hence $t'\gamma\!\downarrow \in \mathcal{NCC}(t\gamma\!\downarrow, \mathcal{V})$. If $t' \in \overline{t}$ then, since $t\gamma\!\downarrow = f(\overline{t\gamma\!\downarrow})$, we have $t'\gamma\!\downarrow \in \overline{t\gamma\!\downarrow}$ and hence $t'\gamma\!\downarrow \in \mathcal{NCC}(t\gamma\!\downarrow, \mathcal{V})$. If t' is a basic sort subterm of t not below a variable, then $t'\gamma\!\downarrow$ is a basic sort subterm of $t\gamma\!\downarrow$ not below a variable. Otherwise, we have the following cases.

1. If $t' \in \mathcal{NCC}(t, \mathcal{V})$ by case 1 then $t' = g(\overline{t'})$ and $f \succ_{\mathcal{F}} g$ and $\overline{t'} \in \mathcal{NCC}(t, \mathcal{V})$. Then by induction hypothesis $\overline{t'\gamma\!\downarrow} \in \mathcal{NCC}(t\gamma\!\downarrow, \mathcal{V})$, which implies $t'\gamma\!\downarrow = g(\overline{t'\gamma\!\downarrow}) \in \mathcal{NCC}(t\gamma\!\downarrow, \mathcal{V})$ by case 1.

2. If $t' \in \mathcal{NCC}(t, \mathcal{V})$ by case 2 then $t' = f(\overline{t'})$ and $\overline{t}(\rhd_x)_{stat_f}\overline{t'}$, where $\rhd_x$ is subterm not below a variable. Therefore $\overline{t\gamma\!\downarrow}(\rhd_x)_{stat_f}\overline{t'\gamma\!\downarrow}$, which implies $t'\gamma\!\downarrow = f(\overline{t'\gamma\!\downarrow}) \in \mathcal{NCC}(t\gamma\!\downarrow, \mathcal{V})$ by case 2.

3. If $t' \in \mathcal{NCC}(t, \mathcal{V})$ by case 3 then $t' = \lambda x.u$ for some $x \notin \mathcal{V}ar(t)$ and $u \in \mathcal{NCC}(t, \mathcal{V} \cup \{x\})$. Then, since $x \notin \mathcal{D}om(\gamma)$, we have $t'\gamma\!\downarrow = \lambda x.u\gamma\!\downarrow$, and by induction hypothesis $u\gamma\!\downarrow \in \mathcal{NCC}(t\gamma\!\downarrow, \mathcal{V} \cup \{x\})$, which implies $t'\gamma\!\downarrow = \lambda x.u\gamma\!\downarrow \in \mathcal{NCC}(t\gamma\!\downarrow, \mathcal{V} \cup \{x\})$, by case 3. □

Lemma 44 *$\succ_{nhorpo}$ is stable under substitution.*

Proof: Let s be a normalized term and $s \succ_{nhorpo} t$. Then, $s\gamma\!\downarrow \succ_{nhorpo} t\gamma\!\downarrow$ for every normalized substitution γ. Note that by lemma 35 we only need to normalize wrt. β.

We proceed by induction on $(|s|, |t|)$ ordered lexicographically, and case analysis. s cannot be either a variable nor headed by a variable, otherwise no case would apply. Let $s = f(\overline{s})$ with $f \in \mathcal{F}$ or $s = \lambda x.u$.

1. $s \succ_{nhorpo} t$ by case 1. Then $f \in \mathcal{F}$ and $s_i \succeq_{nhorpo} t$. By induction hypothesis $s_i\gamma\!\downarrow \succeq_{nhorpo} t\gamma\!\downarrow$ (note that if $s_i = t$ then $s_i\gamma\!\downarrow = t\gamma\!\downarrow$). Therefore since $f(\overline{s})\!\downarrow = f(\overline{s\!\downarrow})$ we have $f(\overline{s\!\downarrow}) \succ_{nhorpo} t\gamma\!\downarrow$, by case 1.

2. $s \succ_{nhorpo} t$ by case 2. Then $f \in \mathcal{F}$, $t = g(\overline{t})$, $g \in \mathcal{F}$ and $f >_{\mathcal{F}} g$ and for all $t_i \in \overline{t}$ either $t_i \in \mathcal{NCC}(s)$ or, if $t_i : \tau$ then $s \succ_{nhorpo} t_i$, otherwise $s_j \succeq_{nhorpo} t_i : \sigma$ for some $s_j : \sigma \in \overline{s}$. Then, since $s\!\downarrow = f(\overline{s\!\downarrow})$, by induction hypothesis for all $t_i\gamma\!\downarrow \in \overline{t\gamma\!\downarrow}$, either, by lemma 43, $t_i\gamma\!\downarrow \in \mathcal{NCC}(s\!\downarrow)$, or if $t_i\gamma\!\downarrow : \tau$ then $s\gamma\!\downarrow \succ_{nhorpo} t_i\gamma\!\downarrow$, otherwise $s_j\gamma\!\downarrow \succeq_{nhorpo} t_i\gamma\!\downarrow : \sigma$ for some $s_j\gamma\!\downarrow : \sigma \in \overline{s\gamma\!\downarrow}$. Then, by case 2, $s\gamma\!\downarrow = f(\overline{s\!\downarrow}) \succ_{nhorpo} g(\overline{t\gamma\!\downarrow})t\gamma\!\downarrow$.

3. $s \succ_{nhorpo} t$ by case 3. Then $f \in \mathcal{F}$ and $t = u[\overline{v}]_{\overline{p}}$ with $\forall p \in \overline{p}$, $p \neq \Lambda$ and $\forall v' \in \overline{v}$, $\mathcal{V}ar(v') \subseteq \mathcal{V}ar(t)$, and
   for all $v' \in \overline{v} \cup \{\lambda\overline{x}.u[\overline{x\uparrow}]_{\overline{p}}$ with $\overline{x}$ fresh$\}$, either $v' \in \mathcal{NCC}(s)$, or if $v' : \tau$ then $s \succ_{nhorpo} v'$ otherwise $s_j \succeq_{nhorpo} v' : \sigma$ for some $s_j : \sigma \in \overline{s}$. Note that for typing reasons $u' = \lambda\overline{x}.u[\overline{x\uparrow}]_{\overline{p}}$ has a type different from τ, and hence either $s_j \succeq_{nhorpo} u'$ or $u' \in \mathcal{NCC}(s)$.

   On the other hand, $t\gamma = u\gamma[\overline{v\gamma}]_{\overline{p}}$ and, by Corollary 32, $t\gamma\!\downarrow = u\gamma\!\downarrow [\overline{v\gamma\!\downarrow}]_{\overline{p}}$ (note that since t is normalized then u and $\overline{v}$ are normalized and since γ is a normalized substitution, by lemma 35, $u\gamma\!\downarrow$ and $\overline{v\gamma\!\downarrow}$ are normalized). Moreover, since all variables in $\overline{x}$ are fresh, $u'\gamma = \lambda\overline{x}.u\gamma[\overline{x\uparrow}]_{\overline{p}}$, and hence $u'\gamma\!\downarrow = \lambda\overline{x}.u\gamma\!\downarrow [\overline{x\uparrow}]_{\overline{p}}$. Now by lemma 43 and induction hypothises, for all $v' \in \overline{v\gamma\!\downarrow} \cup \{u'\gamma\!\downarrow\}$, either $v' \in \mathcal{NCC}(s)$, or if $v' : \tau$ then $s\gamma\!\downarrow \succ_{nhorpo} v'$ otherwise $s_j\gamma\!\downarrow \succeq_{nhorpo} v' : \sigma$ for some $s_j : \sigma \in \overline{s}$. Therefore, since $s\gamma\!\downarrow = f(\overline{s\gamma\!\downarrow})$ and $t\gamma\!\downarrow = u\gamma\!\downarrow \{\overline{x} \mapsto \overline{v\gamma\!\downarrow}\}$, we have $s\gamma\!\downarrow \succ_{nhorpo} t\gamma\!\downarrow$, by case 3.

4. $s \succ_{nhorpo} t$ by case 4. Then $f \in \mathcal{F}$, $t = f(\overline{t})$ and $\overline{s}(\succ_{nhorpo})_{mul}\overline{t}$, which implies, by induction hypothesis, that $\overline{s\gamma\!\downarrow}(\succ_{nhorpo})_{mul}\overline{t\gamma\!\downarrow}$ and hence $s\gamma\!\downarrow = f(\overline{s\gamma\!\downarrow}) \succ_{nhorpo} f(\overline{t\gamma\!\downarrow}) = s\gamma\!\downarrow$ by case 4.

5. $s \succ_{nhorpo} t$ by case 5. Then $f \in \mathcal{F}$, $t = f(\overline{t})$ and $\overline{s}(\succ_{nhorpo})_{lex}\overline{t}$,and for all $t_i \in \overline{t}$, either $t_i \in \mathcal{NCC}(s)$, or if $t_i : \tau$ then $s \succ_{nhorpo} t_i$ otherwise $s_j \succeq_{nhorpo} t_i : \sigma$ for some $s_j : \sigma \in \overline{s}$. Therefore by induction hypothesis, $\overline{s\gamma\downarrow}(\succ_{nhorpo})_{lex}\overline{t\gamma\downarrow}$, and, by lemma 43 and induction hypothises, for all $t_i\gamma\downarrow \in \overline{t\gamma\downarrow}$, either $t_i\gamma\downarrow \in \mathcal{NCC}(\underline{s\downarrow})$, or if $t_i\gamma\downarrow : \tau$ then $s\gamma\downarrow \succ_{nhorpo} t_i\gamma\downarrow$, otherwise $s_j\gamma\downarrow \succeq_{nhorpo} t_i\gamma\downarrow : \sigma$ for some $s_j\gamma\downarrow : \sigma \in \overline{s\gamma\downarrow}$. Then, by case 5, $s\gamma\downarrow = f(\overline{s\downarrow}) \succ_{nhorpo} g(\overline{t\gamma\downarrow}) t\gamma\downarrow$.

6. $s \succ_{nhorpo} t$ by case 6. Then $s = \lambda x.u$ and $t = \lambda x.v$ and $u \succ_{nhorpo} v$. By induction hypothesis $u\gamma\downarrow \succ_{nhorpo} v\gamma\downarrow$. Therefore, since $s\gamma\downarrow = \lambda x.u\gamma\downarrow$ and $t\gamma\downarrow = \lambda x.v\gamma\downarrow$, we have $s\gamma\downarrow \succ_{nhorpo} t\gamma\downarrow$ by case 6. □

Lemma 45 *Let s be a normalized term. If $s \succ_{nhorpo} t$, then $C[s] \succ_{nhorpo} C[t]$ for every ground context C such that $C[s]$ is normalized.*

Proof: As in lemma 8, $s \succ_{nhorpo} t$ implies $\lambda x.s \succ_{nhorpo} \lambda x.t$ by case 6 and $f(\ldots s \ldots) \succ_{nhorpo} f(\ldots t \ldots)$ for every $f \in \mathcal{F}$ either by case 4 or by case 5, depending on the status of f. Then we conclude by a straightforward induction on the size of C. □

Property 46 $\mathcal{NCC}(t) \subseteq \mathcal{CC}(t)$.

Lemma 47 *Let s and t be normalized terms. If $s \succ_{nhorpo} t$ then $s \succ_{horpo} t' \longrightarrow_\beta^* t$ for some term t'.*

Proof: We proceed by induction on $(|s|, |t|)$ ordered lexicographically, and case analysis on the proof that $s \succ_{nhorpo} t$.

1. $s \succ_{nhorpo} t$ by case 1. Then $f \in \mathcal{F}$ and $s_i \succeq_{nhorpo} t$. By induction hypothesis, $s_i \succeq_{horpo} t' \longrightarrow_\beta^* t$ for some t' and hence by case 1 of $\succ_{horpo}$ we have $s \succ_{horpo} t' \longrightarrow_\beta^* t$.

2. $s \succ_{nhorpo} t$ by case 2. Then $f \in \mathcal{F}$, $t = g(\overline{t})$, $g \in \mathcal{F}$ and $f >_\mathcal{F} g$ and for all $t_i \in \overline{t}$ either $t'_i = t_i \in \mathcal{CC}(s)$ or, if $t_i : \tau$ then $s \succ_{nhorpo} t_i$, otherwise $s_j \succeq_{nhorpo} t_i : \sigma$ for some $s_j : \sigma \in \overline{s}$. Then, by induction hypothesis and property 46, we have for all $t_i \in \overline{t}$ either $t'_i = t_i \in \mathcal{CC}(s)$ or, if $t_i : \tau$ then $s \succ_{horpo} t'_i \longrightarrow_\beta^* t_i$, otherwise $s_j \succeq_{nhorpo} t'_i \longrightarrow_\beta^* t_i$ for some $s_j : \sigma \in \overline{s}$. Then by case 2 of $\succ_{horpo}$ we have $s \succ_{horpo} g(\overline{t'}) = t'$ and then $t' = g(\overline{t'}) \longrightarrow_\beta^* g(\overline{t})$.

3. $s \succ_{nhorpo} t$ by case 3. Then $f \in \mathcal{F}$ and $t = u[\overline{v}]_{\overline{p}}$ with $\forall p \in \overline{p}$, $p \neq \Lambda$ and $\forall v' \in \overline{v}$, $\mathcal{V}ar(v') \subseteq \mathcal{V}ar(t)$, and for all $v_i \in \overline{v}$ either $v_i \in \mathcal{NCC}(s)$ or, if $v_i : \tau$ then $s \succ_{nhorpo} v_i$ otherwise $s_j \succeq_{nhorpo} v_i : \sigma$ for some $s_j : \sigma \in \overline{s}$, and finally, for typing reasons, for $u' = \lambda\overline{x}.u[\overline{x\uparrow}]_{\overline{p}}$, either $u' \in \mathcal{NCC}(s)$ or $s_j \succeq_{nhorpo} u'$.

   By induction hypothesis and property 46, we have for all $v_i \in \overline{v}$ either $v'_i = v_i \in \mathcal{CC}(s)$ or if $v_i : \tau$ then $s \succ_{horpo} v'_i \longrightarrow_\beta^* v_i$ otherwise $s_j \succeq_{horpo} v'_i \longrightarrow_\beta^* v_i : \sigma$ for some $s_j : \sigma \in \overline{s}$, and, either $u'' = u' \in \mathcal{CC}(s)$ or $s_j \succeq_{horpo} u'' \longrightarrow_\beta^* u' : \sigma$ for some $s_j : \sigma \in \overline{s}$. Then, by case 5 of $\succ_{horpo}$, we have $s \succ_{horpo} @(u'', \overline{v'}) = t'$ and then $t' = @(u'', \overline{v'}) \longrightarrow_\beta^* @(u', \overline{v})$. Now, thanks to our assumption that $\forall v' \in \overline{v}$, $\mathcal{V}ar(v') \subseteq \mathcal{V}ar(t)$, then $@(u', \overline{v}) \longrightarrow_\beta^+ t$.

4. $s \succ_{nhorpo} t$ by case 4. Then $f \in \mathcal{F}$, $t = f(\overline{t})$ and $\overline{s}(\succ_{nhorpo})_{mul}\overline{t}$, which implies, by induction hypothesis, that $\overline{s}(\succ_{horpo})_{mul}\overline{t'}$, and $\overline{t'} \longrightarrow_\beta^* \overline{t}$. Therefore, $s \succ_{horpo} f(\overline{t'}) = t'$ by case 3 of $\succ_{horpo}$ and $t' = f(\overline{t'}) \longrightarrow_\beta^* t$.

5. $s \succ_{nhorpo} t$ by case 5. Then $f \in \mathcal{F}$, $t = f(\overline{t})$ and $\overline{s}(\succ_{nhorpo})_{lex}\overline{t}$,and for all $t_i \in \overline{t}$ either $t_i \in \mathcal{NCC}(s)$ or, if $t_i : \tau$ then $s \succ_{nhorpo} t_i$, otherwise $s_j \succeq_{nhorpo} t_i : \sigma$ for some $s_j : \sigma \in \overline{s}$. Assume we have lexicographic status left-to-right (the right-to-left case is symmetric). Then let k be the first position in which $s_k \succ_{nhorpo} t_k$ then for all $i < k$ we have $s_i = t_i$. Then

by induction hypothesis and property 46, $s_k \succ_{horpo} t'_k \longrightarrow^*_\beta t_k$ and for all $i > k$ if $t_i : \tau$ then $s \succ_{horpo} t'_i \longrightarrow^*_\beta t_i$, otherwise $s_j \succeq_{horpo} t'_i \longrightarrow^*_\beta t_i : \sigma$ for some $s_j : \sigma \in \overline{s}$ or $t'_i = t_i \in \mathcal{NCC}(s)$. Now taking $t'_i = s_i$ for all $i < k$ we have $\overline{s}(\succ_{horpo})_{lex}\overline{t'} \longrightarrow^*_\beta \overline{t}$, and for all for all $t'_i \in \overline{t}$ either $t'_i \in \mathcal{CC}(s)$ or, if $t'_i : \tau$ then $s \succ_{horpo} t'_i$, otherwise $s_j \succeq_{nhorpo} t'_i : \sigma$ for some $s_j : \sigma \in \overline{s}$ (note that for the $t'_i : \tau$ with $i < k$ we apply case 1 of $\succ_{horpo}$), which implies $s \succ_{horpo} f(\overline{t'}) = t'$ by case 4 of $\succ_{horpo}$, and then $t' = f(\overline{t'}) \longrightarrow^*_\beta f(\overline{t}) = t$.

6. $s \succ_{nhorpo} t$ by case 6. Then $s = \lambda x.u$ and $t = \lambda x.v$ and $u \succ_{nhorpo} v$. By induction hypothesis we have $u \succ_{horpo} v' \longrightarrow^*_\beta v$, which implies $s \succ_{horpo} \lambda x.v' = t'$ by case 7 of $\succ_{horpo}$, and then $t' = \lambda x.v' \longrightarrow^*_\beta \lambda x.v = t$. □

Theorem 48 *$\succ_{nhorpo}$ is a well-defined, decidable relation of the set of normalized terms whose transitive closure is a polymorphic normalized higher-order reduction ordering.*

Proof: Monotonicity under ground normalized contexts and stability under normalized substitutions follow from previous lemmas, and well-foundedness folows from lemma 47. Finally for polymorphism, since the output type of all function symbols is a basic sort, the only terms that has a non-ground type are either variables if it is a basic type or terms headed by lambda if it is an arrow type. Lets us consider both cases. If it is a variable then it is only comparable with the same variable, then after instantiation and η-expansion we will have the same term again and we can conclude. If it is a term headed, since all function symbols has a basic sort only case 6 applies and then again after instantiation and η-expansion we conclude with the same case again (maybe applied several times). □

Corollary 49 *Let R be a set of normalized higher-order rewrite rules. Then R is terminating for normalized rewriting if $l \succ_{nhorpo} r$ for each rewrite rule $l \to r \in R$.*

We end up this section with an important example of an encoding of logics.

Example: Prenex Normal Forms

This example is adapted from [14], where its local confluence is proved via the computation of its (higher-order) critical pairs. We give below an automatic termination proof with our ordering.

Formulas are represented as λ-terms with basic sort ϕ. The idea is that quantifiers bind variables via the use of a functional argument, that is, an abstraction. We have

$\mathcal{S} = \{\phi\} \qquad \mathcal{F}_{\phi\to\phi} = \{\neg\}, \mathcal{F}_{\phi\to\phi\to\phi} = \{\wedge,\vee\}, \mathcal{F}_{(\phi\to\phi)\to\phi} = \{\exists,\forall\} \qquad \mathcal{X}_\phi = \{X\}, \mathcal{X}_{\phi\to\phi} = \{Y\}.$

$$\begin{array}{rcl}
X \wedge \forall(\lambda x.Y(x)) & \to & \forall(\lambda x.(X \wedge Y(x))) \\
X \vee \forall(\lambda x.Y(x)) & \to & \forall(\lambda x.(X \vee Y(x))) \\
X \wedge \exists(\lambda x.Y(x)) & \to & \exists(\lambda x.(X \wedge Y(x))) \\
X \vee \exists(\lambda x.Y(x)) & \to & \exists(\lambda x.(X \vee Y(x))) \\
\neg(\forall(\lambda x.Y(x))) & \to & \exists(\lambda x.\neg(Y(x))) \\
\neg(\exists(\lambda x.Y(x))) & \to & \forall(\lambda x.\neg(Y(x)))
\end{array}$$

The missing rules are obtained by symmetry. We detail the termination proof for the first rule. We take $\wedge, \vee \in Mul$ and use the following precedence:

$\wedge >_\mathcal{F} \forall, \quad \wedge >_\mathcal{F} \exists, \quad \wedge >_\mathcal{F} \lambda_{\phi\to\phi}, \quad \vee >_\mathcal{F} \forall, \quad \vee >_\mathcal{F} \exists,$
$\vee >_\mathcal{F} \lambda_{\phi\to\phi}, \quad \neg >_\mathcal{F} \forall, \quad \neg >_\mathcal{F} \exists, \quad \neg >_\mathcal{F} \lambda_{\phi\to\phi}$

Using rule 2, we must show that $\lambda x.(X \wedge Y(x)) \in \mathcal{NCC}(X \wedge \forall(\lambda x.Y(x)), \emptyset)$, that is, using rule 3 of the computational closure, that $X \wedge Y(x) \in \mathcal{NCC}(X \wedge \forall(\lambda x.Y(x)), \{x\})$. We can now use rule 2, because both X and $Y(x)$ are subterms of $X \wedge \forall(\lambda x.Y(x))$ not below variables.

4.4 Curried polymorphic function symbols

Polymorphic function symbols such as Gdel's recursor have monomorphic instances which violate the assumption that the output type of a function symbol is a sort. Removing the latter restriction will allow us to remove the former as well. Note that a term u may be normalized with respect to a polymorphic signature $\mathcal{F}$, but not with respect to some of its monomorphic instances. We will therefore write $u\uparrow^{\eta^{\mathcal{F}}}$ for the η-long normal form of u with respect to $\mathcal{F}$.

The idea for handling Curried, polymorphic function symbols is quite simple. Assume that $f \in \mathcal{F}$ has type $\sigma \to (\sigma \to \sigma)$, where σ is a sort, or a type variable. Treating type variables as if they were sorts, a well-typed term of the form $f(u)$ is written $\lambda x.@(f(u), x)$ in η-long normal form. The idea is to write it instead in the form $\lambda x.f(u, x)$, where f has now the type $\sigma \times \sigma \to \sigma$, with an output type which is a sort or a type variable. In this new signature, we can of course compare terms with the normalized higher-order recursive path ordering already defined.

Definition 50 *Let $\mathcal{F}$ be an arbitrary polymorphic signature. We define the* uncurried signature $\eta\mathcal{F} = \{f : \sigma_1 \times \ldots \times \sigma_p \to \sigma \mid f : \sigma_1 \times \ldots \times \sigma_n \to (\sigma_{n+1} \to \ldots \to (\sigma_m \to \ldots \to \sigma_p \to \sigma)\ldots) \in \mathcal{F}\}$, *and the uncurrying rewrite system* $\mathcal{F}\uparrow^{\eta} = \{@(f(u_1, \ldots, u_n), u_{n+1}, \ldots, u_p) \to f(u_1, \ldots, u_p) \mid f : \sigma_1 \times \ldots \times \sigma_n \to (\sigma_{n+1} \to \ldots \to (\sigma_m \to \ldots \to \sigma_p \to \sigma)\ldots) \in \mathcal{F}\}$, *where* $\sigma \in \mathcal{S} \cup \mathcal{S}^{\forall}$.

To any normalized term $u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, we associate its normal form with respect to $\mathcal{F}\uparrow^{\eta}$, denoted by $u\downarrow_{\mathcal{F}\uparrow^{\eta}}$, and say that an ordering $\succ$ on normalized terms is polymorphic if $s \succ t$ implies $s\uparrow^{\eta^{\mathcal{F}\xi}}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} \succ t\uparrow^{\eta^{\mathcal{F}\xi}}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}$ for an arbitrary type instantiation ξ.

Remember that variables are indeed distinguished constants. Therefore, if x is a variable of polymorphic type τ, its normal form with respect to a type instantiation $\xi(\tau) = \sigma_1 \to \ldots \sigma_p \to \sigma$ where $\sigma \in \mathcal{S}$ is the abstraction $\lambda x_1 \ldots x_p.x(x_1, \ldots, x_n)$.

The definition of $\mathcal{NCC}$ needs to be modified to cope with this new situation, while the definition of $\succ_{nhorpo}$ remains the same, modulo the previous change.

Definition 51 *Given a normalized term $s = f(\overline{s}) : \tau$ with $f \in \mathcal{F}$, we define its normalized computable closure $\mathcal{NCC}(s)$ as $\mathcal{NCC}(s, \emptyset)$, where $\mathcal{NCC}(s, \mathcal{V})$, with $\mathcal{V} \cap \mathcal{V}ar(s) = \emptyset$, is the smallest set of well-typed terms containing all variables in $\mathcal{V}$ and all terms in $\overline{s}$ as well as their basic sort subterms not below a variable, and closed under the following operations:*

1. *precedence: let g such that $f >_{\mathcal{F}} g$, and $\overline{t} \in \mathcal{NCC}(s, \mathcal{V})$; then $g(\overline{t}) \in \mathcal{NCC}(s, \mathcal{V})$;*
2. *recursive call: assume $\tau \in \mathcal{S}$ or $f \in Lex$ (left-to-right), and let $s : \tau$, $\overline{t} \in \mathcal{NCC}(s, \mathcal{V})$ and $\overline{s}(\rhd_{\mathcal{X}})_{stat_f}\overline{t}$; then $f(\overline{t}) \in \mathcal{NCC}(s, \mathcal{V})$;*
3. *abstraction: let $x \notin \mathcal{V}ar(s)$ and $t \in \mathcal{NCC}(s, \mathcal{V} \cup \{x\})$; then $\lambda x.t \in \mathcal{NCC}(s, \mathcal{V})$.*

To show that $\succ_{nhorpo}$ is polymorphic, let $\mathcal{F}$ be a polymorphic signature, and ξ be a type instantiation. We need to show that $u \succ^{\mathcal{F}} v$ implies $u \succ^{\mathcal{F}\xi} v$ for every type instantiation ξ. First,

Lemma 52 *Let $s = f(\overline{s})$, $f \in \mathcal{F}$, and $t \in \mathcal{NCC}(s)$ be normalized with respect to the polymorphic signature $\mathcal{F}$ and ξ be an arbitrary type instantiation. Then $t\uparrow^{\eta^{\mathcal{F}\xi}}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} \in \mathcal{NCC}(f(\overline{s}\uparrow^{\eta^{\mathcal{F}\xi}}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x}))$.*

Proof: We prove that for all set of variables $\mathcal{V}$ s.t. $\mathcal{V} \cap \mathcal{V}ar(s) = \emptyset$, we have $t \in \mathcal{NCC}(s, \mathcal{V})$ implies $t\uparrow^{\eta^{\mathcal{F}\xi}}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} \in \mathcal{NCC}(f(\overline{s}\uparrow^{\eta^{\mathcal{F}\xi}}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x}), \mathcal{V})$. By induction on the definition of $\mathcal{NCC}$ and case analysis. Then if t is in $\mathcal{V}$, or in $\overline{s}$ or is a subterm of basic sort of s then trivially, $t\uparrow^{\eta^{\mathcal{F}\xi}}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} \in \mathcal{NCC}(f(\overline{s}\uparrow^{\eta^{\mathcal{F}\xi}}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x}), \mathcal{V})$.

1. if $t = g(\overline{t})$ and $t \in \mathcal{NCC}(s, \mathcal{V})$ because $f >_{\mathcal{F}} g$, and $\overline{t} \in \mathcal{NCC}(s, \mathcal{V})$. Then by induction hypothesis $\overline{t}\uparrow^{\eta^{\mathcal{F}\xi}}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} \in \mathcal{NCC}(f(\overline{s}\uparrow^{\eta^{\mathcal{F}\xi}}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x}), \mathcal{V})$ and hence $\overline{t}\uparrow^{\eta^{\mathcal{F}\xi}}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} \cup \overline{y} \in \mathcal{NCC}(f(\overline{s}\uparrow^{\eta^{\mathcal{F}\xi}}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x}), \mathcal{V} \cup \overline{y})$, which implies $g(\overline{t}\uparrow^{\eta^{\mathcal{F}\xi}}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{y}) \in \mathcal{NCC}(f(\overline{s}\uparrow^{\eta^{\mathcal{F}\xi}}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x}), \mathcal{V} \cup \overline{y})$ by case 1, and, finally, $t\uparrow^{\eta^{\mathcal{F}\xi}}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} = \lambda\overline{y}.g(\overline{t}\uparrow^{\eta^{\mathcal{F}\xi}}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{y}) \in \mathcal{NCC}(f(\overline{s}\uparrow^{\eta^{\mathcal{F}\xi}}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x}), \mathcal{V})$, by repeatedly applying case 3.

2. if $t = f(\overline{t})$, $\overline{t} \in \mathcal{NCC}(s, \mathcal{V})$ and $\overline{s}(\rhd_{\mathcal{X}})_{stat_f}\overline{t}$ then we have $\overline{s}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} (\rhd_{\mathcal{X}})_{stat_f}\overline{t}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}$ and by induction hypothesis $\overline{t}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} \in \mathcal{NCC}(f(\overline{s}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x}), \mathcal{V})$. If the type of s is sort it holds by case 2, since $\overline{x} = \emptyset$ and $t\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} = f(\overline{t}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}})$. Otherwise, $f \in Lex$ and it is left to right. Since $\overline{t}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} \cup \overline{y} \in \mathcal{NCC}(f(\overline{s}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x}), \mathcal{V} \cup \overline{y})$, and $\overline{s}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x}(\rhd_{\mathcal{X}})_{lex}\overline{t}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x}$, since it is compared left to right, by case 2, $f(\overline{t}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{y}) \in \mathcal{NCC}(f(\overline{s}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x}), \mathcal{V} \cup \overline{y})$. Now as in the previous case we have $t\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} = \lambda\overline{y}.f(\overline{t}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{y}) \in \mathcal{NCC}(f(\overline{s}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x}), \mathcal{V})$, by repeatedly applying case 3.

3. if $t = \lambda y.u$ with $y \notin \mathcal{V}ar(s)$ and $u \in \mathcal{NCC}(s, \mathcal{V} \cup \{y\})$. Then by induction hypothesis $u\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} \in \mathcal{NCC}(f(\overline{s}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x}), \mathcal{V} \cup \{x\})$, and hence $t\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} = \lambda x.u\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} \in \mathcal{NCC}(f(\overline{s}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x}), \mathcal{V})$, by case 3. $\square$

Lemma 53 *The normalized higher-order recursive path ordering is polymorphic.*

Proof: Let $s : \tau$ and $t : \tau$ be normalized terms with respect to the polymorphic signature $\mathcal{F}$. We need to show that $s \succ_{nhorpo} t$ implies $s\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} \succ_{nhorpo} t\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}$. The proof is by induction on $|s| + |t|$ and case analysis on the proof that $s \succ_{nhorpo} t$.

1. Let $\tau \in \mathcal{S}$. Then, $s = f(\overline{s})$ and $t = g(\overline{t})$. Since $\tau \in \mathcal{S}$ we have $f(\overline{s})\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} = f(\overline{s}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}})$ and $g(\overline{t})\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} = g(\overline{t}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}})$ and then in all cases we conclude easily by induction hypothesis and lemma 52 (note that in this case $s\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} = f(\overline{s}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}})$).

2. Let now $\tau = \tau_1 \to \ldots \to \tau_k \to \tau$. Then $s = \lambda x.u$ and $t = \lambda x.v$ and $u \succ_{nhorpo} v$. Now since $s\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} = \lambda x.u\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}$ and $t\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} = \lambda x.v\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}$, and, by induction hypothesis $u\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} \succ_{nhorpo} v\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}$, we have $s\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} \succ_{nhorpo} t\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}$ by case 6.

3. Let $\tau \in \mathcal{S}^{\forall}$. Then $s \succ_{nhorpo} t$ by cases 1-5. Then let s be $f(\overline{s}) : \tau$ and t be $g(\overline{t}) : \tau$, where $f \in \mathcal{F}$ and $g \in \mathcal{F} \cup \mathcal{X}$. If $\xi(\tau) = \sigma_{n+1} \to \ldots \to \sigma_r \to \sigma$ with $\sigma \in \mathcal{S}$, then $s\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} = \lambda\overline{x}.f(\overline{s}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x})$. Now, t has also type τ, therefore $t = g(\overline{t})$ for some $g \in \mathcal{F} \cup \mathcal{X}$, and $t\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} = \lambda\overline{x}.g(\overline{t}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x})$. To show that $s\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} \succ_{nhorpo} t\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}$, we therefore apply case 6 until we get $s' = f(\overline{s}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x}) \succ_{nhorpo} g(\overline{t}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x}) = t'$. We distinguish several cases according to the proof of $s \succ_{nhorpo} t$.

   - Case 1. Then $s_i \succeq_{nhorpo} t$ for some $s_i : \tau \in \overline{s}$. If $s_i = h(\overline{s_i})$, with $h \in \mathcal{F} \cup \mathcal{X}$ then we have $s_i\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} = \lambda\overline{y}.h(\overline{s_i}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{y}) \in \overline{s}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}$, and since $t' = g(\overline{t}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{[]})[\overline{x}]$ and by induction hypothesis $s_i\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} \succ_{nhorpo} \lambda\overline{y}.g(\overline{t}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{y}) = t\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}$ and for every $x \in \overline{x}$ we have $x \in \overline{s}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}$ we can conclude $s' \succ_{nhorpo} t'$ by case 3.
   - Case 2. Since $\tau \notin \mathcal{S}$, we have $s = f(\overline{s}), t = g(\overline{t})$, $f >_{\mathcal{F}} g$ and $\forall t_i \in \overline{t}$ either $t_i \in \mathcal{NCC}(s)$ or $s_j \succeq_{nhorpo} t_i : \sigma$ for some $s_j : \sigma \in \overline{s}$. Then by induction hypothesys and lemma 52 we have that $\forall t'_i \in \overline{t}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}$ either $t'_i \in \mathcal{NCC}(s')$ or $s'_j \succeq_{nhorpo} t'_i : \sigma$ for some $s'_j : \sigma \in \overline{s}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}$, and since all variables in $\overline{x}$ are also arguments of s' we have $s' \succ_{nhorpo} t'$ by case 2.
   - Case 3. Then, $t = u[\overline{v}]_{\overline{p}}$ with $\forall p \in \overline{p}$, $p \neq \Lambda$ and $\forall v' \in \overline{v}$, $\mathcal{V}ar(v') \subseteq \mathcal{V}ar(t)$, and, since $\tau \notin \mathcal{S}$, $\forall v' \in \overline{v} \cup \{\lambda\overline{y}.u[\overline{y\uparrow}]_{\overline{p}}$ with $\overline{y}$ fresh$\}$, either $v' \in \mathcal{NCC}(s)$ or $s_j \succeq_{nhorpo} v' : \sigma$ for some $s_j : \sigma \in \overline{s}$.
   Since $t = g(\overline{t})$, then $t = g(\overline{u})[\overline{v}]_{\overline{p}}$ and $t' = g(\overline{u}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x})[\overline{v}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}]_{\overline{p}}$ and, letting $u' = g(\overline{u}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{[]})$, we get $t' = u'[\overline{v}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{x}]_{\overline{p},n11,\ldots,r}$. Now since $u[\overline{y\uparrow}]_{\overline{p}}$ has type τ we have $\lambda\overline{y}.g(u[\overline{y\uparrow}]_{\overline{p}})\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}} = \lambda\overline{zy}.g(u[\overline{y\uparrow}]_{\overline{p}}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{z})$, which is equal to $w' = \lambda\overline{zy}.u'[\overline{y\uparrow}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}, \overline{z}]_{\overline{p},n+1,\ldots,r}$. Then by lemma 52 or by induction hypothesis (depending on $\lambda\overline{y}.g(u[\overline{y\uparrow}]_{\overline{p}})$), we get either $w' \in \mathcal{NCC}(s')$ or $s'_j \succeq_{nhorpo} w'$ for some $s'_j \in \overline{s}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}$, and similarly for all $v' \in \overline{v}\uparrow^{\eta^{\mathcal{F}}\xi}\downarrow_{\mathcal{F}\xi\uparrow^{\eta}}$ we have either $v' \in \mathcal{NCC}(s')$ or

$s'_j \succeq_{nhorpo} v'$ for some $s'_j \in \overline{s}\uparrow^{\eta\mathcal{F}\xi}\downarrow_{\mathcal{F}\xi\uparrow^\eta}$. Therefore, since all variables in $\overline{x}$ are also arguments of s', we conclude $s' \succ_{nhorpo} t'$ by case 3.

- Case 4. Then $f \in Mul$ and $\overline{s}(\succ_{nhorpo})_{mul}\overline{t}$. Therefore, since by induction hypothesis $\overline{s}\uparrow^{\eta\mathcal{F}\xi}\downarrow_{\mathcal{F}\xi\uparrow^\eta} (\succ_{nhorpo})_{mul}\overline{t}\uparrow^{\eta\mathcal{F}\xi}\downarrow_{\mathcal{F}\xi\uparrow^\eta}$, we have $s' \succ_{nhorpo} t'$ by case 4.
- Case 5. The arguments are similar to those used in cases 4 and 2. □

Theorem 54 $\succ_{nhorpo}$ *is a polymorphic normalized higher-order reduction ordering.*

Example 55 *Normalized Gödel's recursor for the natural numbers:*

$$\begin{array}{rcl} rec(0, m, \lambda yz.X(y,z)) & \rightarrow & m \\ rec(S(x), m, \lambda yz.X(y,z)) & \rightarrow & X(x, rec(x, m, \lambda yz.X(y,z))) \end{array}$$

The first rule succeeds immediately by case 1. For the second rule, we apply case 3, splitting $X(x, rec(x, u, \lambda yz.X(y,z)))$ *into* $u = X([],[])$ *and* $\overline{v} = \{x, rec(x, m, \lambda yz.X(y,z))\}$. *Then we have* $\lambda yz.X(y,z) \succeq_{nhorpo} \lambda yz.X(x,y)$ *and* $x \in \mathcal{NCC}(rec(S(x), u, \lambda yz.X(y,z)))$ *since it is a basic sort subterm and* $rec(x, u, \lambda yz.X(y,z)) \in \mathcal{NCC}(rec(S(x), u, \lambda yz.X(y,z)))$ *applying case 2, giving status lexicographic left-to-right to rec.*

5 Conclusion

We have defined two powerful orderings. Operating on higher-order terms, the first one is a conservative extension of Dershowitz's recursive path ordering, and is a reduction ordering compatible with β-reductions and polymorphic. To our knowledge, this is the first such ordering ever. Operating on higher-order terms in η-long normal form, the second is a well-founded normalized reduction ordering. This ordering overpowers the very few attempts that we know of [12, 13, 4, 10, 11]. In particular, none of them was able to order the recursor for natural numbers, which we can do easily. On the other hand, the obtained ordering is not fully polimorphic as the first one, and we are still working on an improved fully polymorphic version.

The idea of computable closure is also used in a different context in [9, 3]. The goal there is to define a syntactic class of higher-order rewrite rules that are compatible with beta reductions and with recursors for arbitrary positive inductive types. The language is indeed the calculus of inductive constructions generated by a monomorphic signature. The usefulness of the notion of closure in this different context shows the strenght of this concept.

Finally, we like to point out a few possible developments of the ideas of the present paper. First, it should be easy to develop a higher-order version of any variation of the recusive path ordering. In particular, an associative-commutative version, since this kind of rewriting is common in practice. Along the same line, it should be interesting to investigate whether these ideas apply to stronger ordinal notations, such as Takeuti's ordinal diagrams. Since we do not rely on the subterm property, this is likely to work. As important, but surely more difficult should be the development of a version able to compare terms of the calculus of constructions. Of course, we should aim at handling rules on (dependent) types. Since the well-foundedness proof is based on the use of Tait and Girard's method, this aim should not be out of hands.

References

[1] Henk Barendregt. *Handbook of Theoretical Computer Science*, volume B, chapter Functional Programming and Lambda Calculus, pages 321–364. North-Holland, 1990. J. van Leeuwen ed.

[2] Henk Barendregt. *Handbook of Logic in Computer Science*, chapter Typed lambda calculi. Oxford Univ. Press, 1993. eds. Abramsky et al.

[3] F. Blanqui, J.-P. Jouannaud, and M. Okada. Strong normalization of inductive data type systems, november 1998.

[4] J. Van de Pol and H. Schwichtenberg. Strict functional for termination proofs. In *Typed Lambda Calculi and Applications, Edinburgh*. Springer-Verlag, 1995.

[5] Nachum Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17(3):279–301, March 1982.

[6] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–309. North-Holland, 1990.

[7] G. Dowek, T. Hardin, and C. Kirchner. Theorem proving modulo, 1998. Rapport de recherche INRIA 3400.

[8] Gilles Dowek, Amy Felty, Hugo Herbelin, Gérard Huet, Christine Paulin-Mohring, and Benjamin Werner. The coq proof assistant user's guide version 5.6. INRIA Rocquencourt and ENS Lyon.

[9] Jean-Pierre Jouannaud and Mitsuhiro Okada. Abstract data type systems. Theoretical Computer Science, 173(2):349-391, February 1997.

[10] Jean-Pierre Jouannaud and Albert Rubio. A recursive path ordering for higher-order terms in η-long β-normal form. In Harald Ganzinger, editor, *7th International Conference on Rewriting Techniques and Applications*, volume 1103 of *Lecture Notes in Computer Science*, pages 108–122, New Brunswick, NJ, USA, July 1996. Springer-Verlag.

[11] Jean-Pierre Jouannaud and Albert Rubio. Rewrite orderings for higher-order terms in η-long β-normal form and the recursive path ordering. *Theoretical Computer Science*, 208(1–2):3–31, November 1998.

[12] C. Loría-Sáenz and J. Steinbach. Termination of combined (rewrite and λ-calculus) systems. In *Proc. 3rd Int. Workshop on Conditional Term Rewriting Systems, Pont-à-Mousson, LNCS 656*, volume 656 of *Lecture Notes in Computer Science*, pages 143–147. Springer-Verlag, 1992.

[13] Olav Lysne and Javier Piris. A termination ordering for higher order rewrite systems. In *Proc. 6th Rewriting Techniques and Applications, Kaiserslautern, LNCS 914*, Kaiserslautern, Germany, 1995.

[14] T. Nipkow. Higher order critical pairs. In *Proc. IEEE Symp. on Logic in Comp. Science*, Amsterdam, 1991.

[15] Lawrence C. Paulson. Isabelle: the next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*. Academic Press, 1990.

[16] Frank Pfenning. Elf: A language for logic definition and verified meta-programming. In *Proc. 4th IEEE Symp. Logic in Computer Science*, pages 313–322, 1989.