Higher-Order Orderings for Normal Rewriting

Jean-Pierre Jouannaud[1⋆] and Albert Rubio[2⋆⋆]

[1] LIX, École Polytechnique, 91400 Palaiseau, France
[2] Technical University of Catalonia, Pau Gargallo 5, 08028 Barcelona, Spain

Abstract. We extend the termination proof methods based on reduction orderings to higher-order rewriting systems *à la Nipkow* using higher-order pattern matching for firing rules, and accommodate for any use of eta, as a reduction, as an expansion or as an equation. As a main novelty, we provide with a mechanism for transforming any reduction ordering including beta-reduction, such as the higher-order recursive path ordering, into a reduction ordering for proving termination of rewriting à la Nipkow. Non-trivial examples are carried out.

1 Introduction

Rewrite rules are used in logical systems to describe computations over lambda-terms used as a suitable abstract syntax for encoding functional objects like programs or specifications. This approach was pioneered in this context by Nipkow [18] and is available in Isabelle [21]. Its main feature is the use of higher-order pattern matching for firing rules. A recent generalization of Nipkow's setting allows one for rewrite rules of polymorphic, higher-order type [15], see also [10]. Besides, it is shown that using the η-rule as an expansion [20] or as a reduction [15] yields very similar confluence checks based on higher-order critical pairs.

A first contribution of this paper is a general setting for addressing termination of all variants of higher-order rewriting à la Nipkow, thanks to the notion of a *normal higher-order reduction ordering*. While higher-order reduction orderings actually *include* $\beta\eta$-reductions, normal higher-order reduction orderings must be compatible with $\beta\eta$-equality since higher-order rewriting operates on $\beta\eta$-equivalence classes of terms. This is done by computing with $\beta\eta$-normal forms as inputs. Since this may destroy stability under substitution, it becomes necessary to use higher-order reduction orderings enjoying a stronger stability property. Restricting the higher-order recursive path ordering [12] to achieve this property is our second contribution. Finally, the obtained ordering is used inside a

⋆ Project LogiCal, Pôle Commun de Recherche en Informatique du Plateau de Saclay, CNRS, École Polytechnique, INRIA, Université Paris-Sud.
⋆⋆ Project LogicTools (TIN2004-03382) of the Spanish Min. of Educ. and Science

powerful schema transforming an arbitrary higher-order reduction ordering satisfying the stronger stability property into a normal higher-order reduction ordering. This is our third contribution. The obtained ordering allows us to prove all standard examples of higher-order rules processing abstract syntax.

We describe our framework for terms in Section 2, and for higher-order rewriting in Section 3. The schema is introduced and studied in Section 4. The restricted higher-order recursive path ordering is given in Section 5. Two complex examples are carried out in Section 6. Significance of the results is briefly discussed in Section 7.

Readers are assumed familiar with the basics of term rewriting [9, 16] and typed lambda calculi [4, 5]. Most ideas presented here originate from [14], an unpublished preliminary draft. A full version is [13].

2 Higher-Order Algebras

Rewrite rules of polymorphic higher type are our target. To define them precisely, we need to recall the framework of higher-order algebras [12]. We will consider the non-polymorphic case for simplicity. The general case of polymorphic higher-order rewrite rules is carried out in [13].

2.1 Types

Given a set $\mathcal{S}$ of *sort symbols* of a fixed arity, denoted by $s : *^n \to *$, the set of *types* is generated by the constructor $\to$ for *functional types*:

$$\mathcal{T}_\mathcal{S} := s(\mathcal{T}_\mathcal{S}^n) \mid (\mathcal{T}_\mathcal{S} \to \mathcal{T}_\mathcal{S}) \qquad \text{for } s : *^n \to * \ \in \mathcal{S}$$

Types are *functional* when headed by the $\to$ symbol, and *data types* otherwise. $\to$ associates to the right. We use $\sigma, \tau, \rho, \theta$ for arbitrary types. The type $\sigma = \tau_1 \to \ldots \to \tau_n \to \tau$, τ not functional, has *arity* $ar(\sigma) = n$.

2.2 Signatures

Function symbols are meant to be algebraic operators equipped with a fixed number n of arguments (called the *arity*) of respective types $\sigma_1 \in \mathcal{T}_\mathcal{S}, \ldots, \sigma_n \in \mathcal{T}_\mathcal{S}$, and an *output type* $\sigma \in \mathcal{T}_\mathcal{S}$. Let

$$\mathcal{F} = \biguplus_{\sigma_1,\ldots,\sigma_n,\sigma} \mathcal{F}_{\sigma_1 \times \ldots \times \sigma_n \to \sigma}$$

The membership of a given function symbol f to $\mathcal{F}_{\sigma_1 \times \ldots \times \sigma_n \to \sigma}$ is called a *type declaration* and written $f : \sigma_1 \times \ldots \times \sigma_n \to \sigma$. A type declaration is *first-order* if it uses only sorts, and higher-order otherwise.

2.3 Terms

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

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

The raw term $\lambda x : \sigma.u$ is an *abstraction* and $@(u, v)$ is an application. We may omit σ in $\lambda x : \sigma.u$ and write $@(u, v_1, \ldots, v_n)$ or $u(v_1, \ldots, v_n)$, $n > 0$, omitting applications. The raw term $@(u, \overline{v})$ is a (partial) *left-flattening* u being possibly an application. $\mathcal{V}ar(t)$ is the set of free variables of t. A term t is *ground* if $\mathcal{V}ar(t) = \emptyset$. The notation $\overline{s}$ shall be ambiguously used to for a list, a multiset, or a set of raw terms $s_1, \ldots, s_n$.

Raw terms are identified with finite labeled trees by considering $\lambda x : \sigma.u$, for each variable x and type σ, as a unary function symbol taking u as argument to construct the raw term $\lambda x : \sigma.u$. *Positions* are strings of positive integers. Λ and $\cdot$ denote respectively the empty string (root position) and string concatenation. $\mathcal{P}os(t)$ is the set of positions in t. $t|_p$ denotes the *subterm* of t at position p. We use $t \trianglerighteq t|_p$ for the subterm relationship. The result of replacing $t|_p$ at position p in t by u is written $t[u]_p$. A raw term $t[x : \sigma]_p$ with a hole of type σ at position p is a *context*.

Given a binary relation $\longrightarrow$ on raw terms, a raw term s such that $s|_p \longrightarrow t$ for some position $p \in \mathcal{P}os(s)$ is called *reducible*. Irreducible raw terms are in *normal form*. A raw term s is *strongly normalizable* if there is no infinite sequence of $\longrightarrow$-steps issuing from s. The relation $\longrightarrow$ is *strongly normalizing*, or *terminating* or *well-founded*, if all raw terms are strongly normalizable. We denote by $\longleftrightarrow$ the symmetric closure of the relation $\longrightarrow$, by $\xrightarrow{*}$ its reflexive, transitive closure, and by $\xleftrightarrow{*}$ its reflexive, symmetric, transitive closure. The relation $\longrightarrow$ is *confluent* (resp. *Church-Rosser*) if $s \longrightarrow^* u$ and $s \longrightarrow^* v$ (resp. $u \longleftrightarrow^* v$) implies $u \longrightarrow^* t$ and $v \longrightarrow^* t$ for some t.

2.4 Typing rules

Definition 1. *An* environment *Γ is a finite set of pairs written as $\{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$, where x_i is a variable, σ_i is a type, and $x_i \neq x_j$ for $i \neq j$. $\mathcal{V}ar(\Gamma) = \{x_1, \ldots, x_n\}$ is the set of variables of Γ. Given two environments Γ and Γ', their* composition *is the environment $\Gamma \cdot \Gamma' = \Gamma' \cup \{x : \sigma \in \Gamma \mid x \notin \mathcal{V}ar(\Gamma')\}$. Two environments Γ and Γ' are* compatible *if $\Gamma \cdot \Gamma' = \Gamma \cup \Gamma'$.*

Our typing judgements are written as $\Gamma \vdash_{\mathcal{F}} s : \sigma$. A raw term s has type σ in the environment Γ if the judgement $\Gamma \vdash_{\mathcal{F}} s : \sigma$ is provable in our inference system given at Figure 1. Given an environment Γ, a raw term s is *typable* if there exists a type σ such that $\Gamma \vdash_{\mathcal{F}} s : \sigma$. Typable raw terms are called *terms*. An important property of our simple type system is that a raw term typable in a given environment has a unique type.

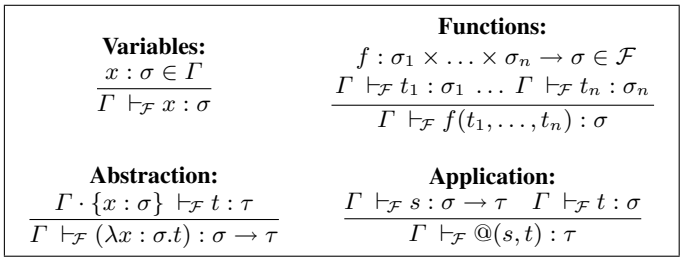

Fig. 1. The type system for monomorphic higher-order algebras

Because variables are typed, they must be replaced by typable terms:

Definition 2. *A* substitution $\gamma = \{(x_1 : \sigma_1) \mapsto (\Gamma_1, t_1), \ldots, (x_n : \sigma_n) \mapsto (\Gamma_n, t_n)\}$*, is a finite set of quadruples made of a variable symbol, a type, an environment and a term, such that*
(i) $\forall i \in [1..n]$*,* $t_i \neq x_i$ *and* $\Gamma_i \vdash_{\mathcal{F}} t_i : \sigma_i$*,*
(ii) $\forall i \neq j \in [1..n]$*,* $x_i \neq x_j$*, and*
(iii) $\forall i \neq j \in [1..n]$*,* Γ_i *and* Γ_j *are compatible environments.*
We may omit the type σ_i *and environment* Γ_i *in* $(x_i : \sigma_i) \mapsto (\Gamma_i, t_i)$*.*
The set of (input) variables of γ *is* $\mathcal{V}ar(\gamma) = \{x_1, \ldots, x_n\}$*, its* domain *is the environment* $\mathcal{D}om(\gamma) = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ *while its* range *is the environment* $\mathcal{R}an(\gamma) = \bigcup_{i \in [1..n]} \Gamma_i$*.*

Definition 3. *A substitution* γ *is* compatible *with the judgement* $\Gamma \vdash_{\mathcal{F}} s : \sigma$ *(or simply, with* Γ*) if (i)* $\mathcal{D}om(\gamma)$ *is compatible with* Γ*, and*
(ii) $\mathcal{R}an(\gamma)$ *is compatible with* $\Gamma \setminus \mathcal{D}om(\gamma)$*.*

A substitution γ compatible with a judgement $\Gamma \vdash_{\mathcal{F}} s : \sigma$ operates classically as an endomorphism on s, resulting in a term denoted by $s\gamma$.

Lemma 1. *Given a signature* $\mathcal{F}$ *and a substitution* γ *compatible with the judgement* $\Gamma \vdash_{\mathcal{F}} s : \sigma$*, then* $\Gamma \cdot \mathcal{R}an(\gamma) \vdash_{\mathcal{F}} s\gamma : \sigma$*.*

2.5 Conversions

We consider α-convertible terms as identical, and hence α-conversions are omitted. The congruence generated by the β- and η-equalities
$(\lambda x.u, v) =_\beta u\{x \mapsto v\}$ $\quad$ $\lambda x.@(u, x) =_\eta u$ if $x \notin \mathcal{V}ar(u)$
is written $=_{\beta\eta}$. An important property, *subject reduction*, is that typable terms u, v such that $u =_{\beta\eta} v$ have the same type. Both equalities can be oriented as rewrite rules. There are two possible choices for rewriting with η, either as a reduction or as an expansion, in which case termination is ensured by restricting its use to positions other than the first argument of an application. Typed lambda-calculi have all termination and confluence properties one may need, with respect to: $\beta\eta$-reductions; β-reductions and η-expansions; β-reductions modulo η-equality. Using the notations $u \longrightarrow_\beta v$ for one β-rewrite step, $u \longrightarrow_\beta^* v$ for its transitive closure, $u \downarrow_\beta$ ($u\downarrow$ for short) for the β-normal form of u, and $\longleftrightarrow_\eta^*$ or $=_\eta$ for η-equality, the Church-Rosser property of β-reductions modulo η-equality for typable terms can be phrased as

$$s =_{\beta\eta} t \text{ iff } s{\downarrow_\beta} =_\eta t{\downarrow_\beta}$$

3 Normal Higher-Order Rewriting of Higher Type

Normal higher-order rewriting [20, 18] allows defining computations on λ-terms used as a suitable abstract syntax for encoding functional objects like programs or specifications. Nipkow's framework assumes that rules are of basic type, and that left-hand sides of rules are patterns in the sense of Miller [19], assumptions which are not made here, nor in [15].

Nipkow's normal higher-order rewriting uses $\beta\eta$-equalities in two different ways: given a term s to be rewritten with a set R of rules, s is first normalized, using η-expanded β-normal forms, before to be searched for left-hand sides of rules in R via higher-order pattern matching, that is, matching modulo $=_{\beta\eta}$. In this section, we define higher-order rewriting so as to capture the different ways in which a term can be $\beta\eta$-normalized before pattern matching its subterm with a left-hand side of rule.

Definition 4. *A* normal rewrite rule *is a rewrite rule $\Gamma \vdash l \to r : \sigma$ such that l and r are higher-order terms in β-normal form satisfying $\Gamma \vdash_{\mathcal{F}} l : \sigma$ and $\Gamma \vdash_{\mathcal{F}} r : \sigma$. A* normal term rewriting system *is a set of normal rewrite rules.*

Given a normal term rewriting system R, an environment Γ, two β-normal terms s and t, and a type σ such that $\Gamma \vdash_{\mathcal{F}} s : \sigma$, we say that s

rewrites to t at position p with the normal rule $\Gamma_i \vdash l_i \to r_i : \sigma_i$ and the term substitution γ, written $\Gamma \vdash s \longrightarrow^{p}_{R_{\beta\eta}} t$, or $s \longrightarrow^{p}_{R_{\beta\eta}} t$ assuming the environment Γ, if the following conditions hold:

$$\begin{array}{ll} \textit{(i)}\ \mathcal{D}om(\gamma) \subseteq \Gamma_i & \textit{(iii)}\ s|_p =_{\beta\eta} l_i\gamma \\ \textit{(ii)}\ \Gamma_i \cdot \mathcal{R}an(\gamma) \subseteq \Gamma_{s|_p} & \textit{(iv)}\ t =_{\eta} s[r_i\gamma]_p{\downarrow_\beta} \end{array}$$

where $\Gamma_{s|_p}$ is the environment of the judgement $\Gamma_{s|_p} \vdash_{\mathcal{F}} s|_p : \sigma_i$, obtained as a subterm of the proof of the judgement $\Gamma \vdash_{\mathcal{F}} s : \sigma$.

Note that t is any term in the η-equivalence class of $s[r_i\gamma]_p{\downarrow_\beta}$. Higher-order rewriting is therefore defined up to η-equivalence of target terms. By providing a method for proving termination of this relation, we do provide a termination method for all variants of higher-order rewriting based on higher-order pattern matching. A key observation is this:

Lemma 2. *Assume $\Gamma \vdash_{\mathcal{F}} s : \sigma$ and $\Gamma \vdash s \to_{R_{\beta\eta}} t$. Then $\Gamma \vdash_{\mathcal{F}} t : \sigma$.*

Example 1. We present here an encoding of symbolic derivation in which functions are represented by λ-terms of a functional type. We give two typical rules of higher type. Both rules have the same environment $\Gamma = \{F : real \to real\}$, and x, y stand for real values. Let $\mathcal{S} = \{\mathsf{real}\}$, and

$$\begin{array}{l} \mathcal{F} = \{\ \mathsf{sin}, \mathsf{cos} : \mathsf{real} \to \mathsf{real}; \mathsf{diff} : (\mathsf{real} \to \mathsf{real}) \to \mathsf{real} \to \mathsf{real} \\ \qquad +, \times : (\mathsf{real} \to \mathsf{real}) \to (\mathsf{real} \to \mathsf{real}) \to \mathsf{real} \to \mathsf{real}\} \end{array}$$

$$\begin{array}{rl} \mathsf{diff}(\lambda\mathsf{x}.\,\mathsf{sin}(@(\mathsf{F},\mathsf{x}))) & \to \lambda\mathsf{x}.\,\mathsf{cos}(@(\mathsf{F},\mathsf{x})) \times \mathsf{diff}(\lambda\mathsf{x}.@(\mathsf{F},\mathsf{x})) \\ \mathsf{diff}(\lambda\mathsf{x}.@(\mathsf{F},\mathsf{x}) \times \lambda\mathsf{y}.@(\mathsf{F},\mathsf{y})) & \to (\mathsf{diff}(\lambda\mathsf{x}.@(\mathsf{F},\mathsf{x})) \times \lambda\mathsf{y}.@(\mathsf{F},\mathsf{y})) + \\ & \qquad (\lambda\mathsf{x}.@(\mathsf{F},\mathsf{x}) \times \mathsf{diff}(\lambda\mathsf{y}.@(\mathsf{F},\mathsf{y}))) \end{array}$$

This example makes sense when using normal higher-order rewriting, because using plain pattern matching instead would not allow to compute the derivative of all expressions: $\mathsf{diff}(\lambda\mathsf{x}.\mathsf{sin}(\mathsf{x})) =_\beta \mathsf{diff}(\lambda\mathsf{x}.\mathsf{sin}((\lambda\mathsf{y}.\mathsf{y})\ \mathsf{x}))$ does require higher-order pattern matching. We shall give a mechanical termination proof of both rules in Section 5.

3.1 Normal Higher-Order Reduction Orderings

We shall use well-founded relations for proving strong normalization properties. For our purpose, these relations may not be transitive, but their transitive closures will be well-founded orderings, justifying some abuse of terminology. Reduction orderings operating on judgements turn

out to be an adequate tool for showing termination of normal rewriting. We consider two classes of reduction orderings called *higher-order reduction ordering* when they include β-reductions and *normal higher-order reduction ordering* when they are compatible with $=_{\beta\eta}$.

Definition 5. *A binary relation $\succ$ on the set of judgements is*

- coherent *iff for all terms s, t such that $(\Gamma \vdash_{\mathcal{F}} s : \sigma) \succ (\Gamma \vdash_{\mathcal{F}} t : \sigma)$, and for all environment Γ' such that Γ and Γ' are compatible, $\Gamma' \vdash_{\mathcal{F}} s : \sigma$ and $\Gamma' \vdash_{\mathcal{F}} t : \sigma$, then $(\Gamma' \vdash_{\mathcal{F}} s : \sigma) \succ (\Gamma' \vdash_{\mathcal{F}} t : \sigma)$;*
- stable *iff for all terms s, t such that $(\Gamma \vdash_{\mathcal{F}} s : \sigma) \succ (\Gamma \vdash_{\mathcal{F}} t : \sigma)$, and all substitution γ whose domain is compatible with Γ, then $(\Gamma \cdot \mathcal{R}an(\gamma) \vdash_{\mathcal{F}} s\gamma : \sigma) \succ (\Gamma \cdot \mathcal{R}an(\gamma) \vdash_{\mathcal{F}} t\gamma : \sigma)$;*
- monotonic *iff for all terms s, t and type σ such that $(\Gamma \vdash_{\mathcal{F}} s : \sigma) \succ (\Gamma \vdash_{\mathcal{F}} t : \sigma)$, for all Γ' compatible with Γ and ground context $u[]$ such that $\Gamma' \vdash_{\mathcal{F}} u[x : \sigma] : \tau$, then $(\Gamma \cdot \Gamma' \vdash_{\mathcal{F}} u[s] : \tau) \succ (\Gamma \cdot \Gamma' \vdash_{\mathcal{F}} u[t] : \tau)$ (note the assumption that $u[]$ is ground);*
- normal-monotonic *iff for all terms s and t such that $(\Gamma \vdash_{\mathcal{F}} s : \sigma) \succ (\Gamma \vdash_{\mathcal{F}} t : \sigma)$, for all Γ' compatible with Γ and for all ground context $u[]$ such that $\Gamma' \vdash_{\mathcal{F}} u[x : \sigma] : \tau$ and $u[s]$ is in β-normal form, then $(\Gamma \cdot \Gamma' \vdash_{\mathcal{F}} u[s] : \tau) \succ (\Gamma \cdot \Gamma' \vdash_{\mathcal{F}} u[t] : \tau)$;*
- functional *iff for all terms s, t such that $(\Gamma \vdash_{\mathcal{F}} s : \sigma \longrightarrow_{\beta} t : \sigma)$, then $(\Gamma \vdash_{\mathcal{F}} s : \sigma) \succ (\Gamma \vdash_{\mathcal{F}} t : \sigma)$;*
- compatible *iff for all terms s', s, t, t' such that $(\Gamma \vdash_{\mathcal{F}} s' : \sigma =_{\beta\eta} s : \sigma)$, $(\Gamma \vdash_{\mathcal{F}} t : \sigma =_{\beta\eta} t' : \sigma)$ and $(\Gamma \vdash_{\mathcal{F}} s : \sigma) \succ (\Gamma \vdash_{\mathcal{F}} t : \sigma)$ then $(\Gamma \vdash_{\mathcal{F}} s' : \sigma) \succ (\Gamma \vdash_{\mathcal{F}} t' : \sigma)$.*

A higher-order reduction ordering $\succ$ *is a well-founded ordering of the set of judgements satisfying coherence, stability, monotonicity and functionality.*

A normal higher-order reduction ordering $\succ_n$ *is a well-founded ordering of the set of judgements satisfying coherence, stability, normal-monotonicity and compatibility.*

Let us show that no ordering $\succ$ can satisfy monotonicity, stability, compatibility and well-foundedness, therefore explaining the need for the weaker notion of normal-monotonicity. Assume $s : \sigma \succ t : \sigma$ (omitting judgements), where $s : \sigma$ is in β-normal form. Consider the term $\lambda y.a : \sigma \to \tau$ where $a : \tau$ is a constant. Then, by monotonicity, $@(\lambda y.a, s) : \tau \succ @(\lambda y.a, t) : \tau$ and by compatibility, $a : \tau \succ a : \tau$, contradicting well-foundedness. Normal-monotonicity removes the problem

since $@(\lambda y.a, s)$ is not in β-normal form. As a consequence, we cannot have $@(X, s) \succ @(X, t)$ when $s \succ t$, but only $@(X, s) = @(X, t)$.

Theorem 1. *Let $R = \{\Gamma_i \vdash l_i \to r_i : \sigma_i\}_i$ be a higher-order rewrite system and $\succ$ a normal higher-order reduction ordering s.t. $(\Gamma_i \vdash_{\mathcal{F}} l_i) \succ (\Gamma_i \vdash_{\mathcal{F}} r_i)$ $\forall i$. Then the relation $\longrightarrow_{R_{\beta\eta}}$ is strongly normalizing.*

Proof. Without loss of generality, let s be a ground normal term such that $\Gamma \vdash_{\mathcal{F}} s \xrightarrow[\Gamma_i \vdash l_i \to r_i : \sigma_i]{p} t$. By definition 4, t is a ground normal term. It therefore suffices to show that $\Gamma \vdash_{\mathcal{F}} s \succ t$, which we proceed to do now. By assumption, $\Gamma_i \vdash_{\mathcal{F}} l_i \succ r_i$. By stability, $\Gamma_i \cdot \mathcal{R}an(\gamma) \vdash_{\mathcal{F}} l_i\gamma \succ r_i\gamma$, therefore, by coherence, $\Gamma_{s|_p} \vdash_{\mathcal{F}} l_i\gamma \succ r_i\gamma$. By definition, $s|_p =_{\beta\eta} l_i\gamma$, hence, by compatibility, $\Gamma_{s|_p} \vdash_{\mathcal{F}} s|_p \succ r_i\gamma$. By monotonicity of $\succ$ for normal ground terms (of equal type), $\Gamma_{s|_p} \cdot \Gamma \vdash_{\mathcal{F}} s \succ s[r_i\gamma]_p$. By coherence $\Gamma \vdash_{\mathcal{F}} s \succ s[r_i\gamma]_p$, hence $\Gamma \vdash_{\mathcal{F}} s \succ t$ by compatibility. □

By lemma 2, higher-order rewriting can be seen as a type preserving relation on terms in a given environment Γ typing the term originating the sequence of rewrites. We can therefore simplify our notations by omitting the typing judgements unless they are really necessary.

4 Building Normal Higher-Order Reduction Orderings

In this section, we assume given a new function symbol $\bot_\sigma$ for every type σ and a function f_{new} for some of the function symbols in $\mathcal{F}$. We denote by $\mathcal{F}_{new}$ the augmented signature. We write $\bot_\sigma$ for $\bot_\sigma()$. The higher-order rules we want to prove terminating are built from terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, not in $\mathcal{T}(\mathcal{F}_{new}, \mathcal{X})$. We successively introduce neutralization, and the neutralized ordering schema. Neutralization replaces a term of functional type by its application to a term headed by a $\bot$-operator seen as a container for its arguments. Neutralizing an abstraction creates a redex which will be later eliminated by a β-normalization step.

Definition 6. *The* neutralization of level i *(i-*neutralization *in short) of a term $t : \tau \in \mathcal{T}(\mathcal{F}_{new}, \mathcal{X})$ w.r.t. a list of (typable) terms $\langle u_1 : \theta_1, \ldots, u_n : \theta_n\rangle$ in $\mathcal{T}(\mathcal{F}_{new}, \mathcal{X})$, is the term $\mathcal{N}_i(t, \langle u_1, \ldots, u_n\rangle)$ defined as follows:*

1. $\mathcal{N}_0(t, \langle u_1, \ldots, u_n\rangle) = t$;
2. $\mathcal{N}_{i+1}(t, \langle u_1, \ldots, u_n\rangle) = t$ *if* τ *is a data type;*
3. $\mathcal{N}_{i+1}(t, \langle u_1, \ldots, u_n\rangle) =$
   $\mathcal{N}_i(@(t, \bot_{\theta_1 \to \ldots \to \theta_n \to \sigma}(u_1, \ldots, u_n)), \langle u_1, \ldots, u_n\rangle)$ *if* $\tau = \sigma \to \rho$.

From now one, we shall very precisely control for each function symbol which of its arguments of a functional type are neutralized:

Definition 7. *A signature $\mathcal{F}$ is* neutralized *if each symbol $f : \sigma_1 \times \ldots \times \sigma_n \to \sigma \in \mathcal{F}$ comes along with, for each argument position $j \in [1..n]$:*

- *a natural number $\mathcal{L}_f^j \leq ar(\sigma_j)$, called* neutralization level *of f at position j. We call* neutralized *those positions j for which $\mathcal{L}_f^j > 0$.*
- *a subset $\mathcal{A}_f^j \subseteq [1..n]$ of argument positions of f used to filter out the list $\bar{t}$ of arguments of f by defining $\bar{t}_f^j = \langle t_k \mid k \in \mathcal{A}_f^j \rangle$.*

The role of full neutralization is to neutralize terms of functional type recursively from arguments of function symbols up to a given depth depending on the function symbol itself and its selected argument. This will allow us to eventually eliminate undesirable abstractions. This huge flexibility provided by levels and argument positions allows us to tune our coming normal higher-order ordering and carry out difficult and important examples taken from the literature. In most of them, the chosen level is 1, implying that neutralization applies to the top of the arguments only, and the set of argument positions is empty, implying that $\bot$ is a constant.

To neutralize terms recursively, we need to introduce new function symbols in the signature : for every declaration $f : \sigma_1 \times \ldots \times \sigma_n \to \sigma$, we assume given a new symbol $f_{new} : \sigma'_1 \times \ldots \times \sigma'_n \to \sigma$ whose type declaration depends upon the neutralization level of its argument positions: if $\sigma_i = \tau_1 \to \ldots \to \tau_k \to \tau$ and $\mathcal{L}_f^i = q \leq k$, then $\sigma'_i = \tau_{q+1} \to \ldots \to \tau_k \to \tau$.

Definition 8. *The* full neutralization *of a term t is the term $\mathcal{FN}(t)$ s.t.*

1. *if $t \in \mathcal{X}$, then $\mathcal{FN}(t) = t$;*
2. *if $t = \lambda x.u$, then $\mathcal{FN}(t) = \lambda x.\mathcal{FN}(u)$;*
3. *if $t = @(t_1, t_2)$, then $\mathcal{FN}(t) = @(\mathcal{FN}(t_1), \mathcal{FN}(t_2))$;*
4. *if $t = f(t_1, \ldots, t_n)$ with $f \in \mathcal{F}$, then*
   $\mathcal{FN}(t) = f_{new}(\mathcal{N}_{\mathcal{L}_f^1}(\mathcal{FN}(t_1), \bar{t}_f^1), \ldots, \mathcal{N}_{\mathcal{L}_f^n}(\mathcal{FN}(t_n), \bar{t}_f^n))$.

Our definition makes sense since, in all cases, $\mathcal{FN}(t)$ is typable with the same type as t. Note also that using Case 3 repeatedly for flattened applications yields $\mathcal{FN}(@(t_1, \ldots, t_n)) = @(\mathcal{FN}(t_1), \ldots, \mathcal{FN}(t_n))$.

Example 1 (continued). We show here the full neutralization of the left-hand and right-hand of the rules of Example 1 after β-normalizing. To

this end, we choose a neutralization level for each function symbol and argument position. The associated subsets of argument positions are all chosen empty. As a consequence, $\bot_{\mathsf{real}}$ is a constant abbreviated as $\bot$.

$\mathcal{L}^1_{\mathsf{diff}} = 1$ $\mathcal{A}^1_{\mathsf{diff}} = \{\}$	$\mathcal{L}^1_{\mathsf{sin}} = 0$ $\mathcal{A}^1_{\mathsf{sin}} = \{\}$	$\mathcal{L}^1_{\mathsf{cos}} = 0$ $\mathcal{A}^1_{\mathsf{cos}} = \{\}$	
$\mathcal{L}^1_{\times} = 1$ $\mathcal{A}^1_{\times} = \{\}$	$\mathcal{L}^2_{\times} = 1$ $\mathcal{A}^2_{\times} = \{\}$	$\mathcal{L}^1_{+} = 1$ $\mathcal{A}^1_{+} = \{\}$	$\mathcal{L}^2_{+} = 1$ $\mathcal{A}^2_{+} = \{\}$

We now compute the β-normalization of the full neutralization of both sides of the first rule:

$$\begin{array}{c}\mathcal{FN}(\ \ \mathsf{diff}\ \ (\ \lambda \mathsf{x}.\,\mathsf{sin}(@(\mathsf{F},\mathsf{x}))\ )\ )\downarrow \\ =\ \ \mathsf{diff}_{\mathsf{new}}\ (\ \ \mathsf{sin}(@(\mathsf{F},\bot))\ \ )\end{array}$$

$$\begin{array}{c}\mathcal{FN}(\ \lambda \mathsf{x}.\,\mathsf{cos}(@(\mathsf{F},\mathsf{x}))\ \ \times\ \ \mathsf{diff}\ (\ \lambda \mathsf{x}.@(\mathsf{F},\mathsf{x})\ )\ \ )\downarrow \\ =\ \ \mathsf{cos}(@(\mathsf{F},\bot))\ \ \times_{new}\ @(\mathsf{diff}_{\mathsf{new}}\ (\ \ @(\mathsf{F},\bot)\ \ )\ ,\bot)\end{array}$$

and of both sides of the second rule:

$$\begin{array}{c}\mathcal{FN}(\ \ \mathsf{diff}\ \ (\ \ \ \ \lambda \mathsf{x}.@(\mathsf{F},\mathsf{x})\ \ \times\ \ \lambda \mathsf{y}.@(\mathsf{F},\mathsf{y})\ \ \ \ )\ )\downarrow \\ =\ \ \mathsf{diff}_{\mathsf{new}}\ (\ @(\ \ @(\mathsf{F},\bot)\ \ \times_{new}\ \ @(\mathsf{F},\bot)\ \ ,\bot)\ )\end{array}$$

$$\mathcal{FN}((\mathsf{diff}(\lambda \mathsf{x}.@(\mathsf{F},\mathsf{x}))\times\lambda \mathsf{y}.@(\mathsf{F},\mathsf{y}))+(\lambda \mathsf{x}.@(\mathsf{F},\mathsf{x})\times\mathsf{diff}(\lambda \mathsf{y}.@(\mathsf{F},\mathsf{y}))))\downarrow=$$
$$@(@(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot)),\bot)\times_{\mathsf{new}}@(\mathsf{F},\bot),\bot)+_{\mathsf{new}}@(@(\mathsf{F},\bot)\times_{\mathsf{new}}@(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot)),\bot),\bot)$$

Definition 9. *Given a neutralized signature, two typable terms s, t and a higher-order ordering $\succ$, we define the neutralized ordering $\succ_n$ as:*

$$s \succ_n t \text{ if and only if } \mathcal{FN}(s)\downarrow \succ \mathcal{FN}(t)\downarrow$$

Note that normalization applies after neutralization: we will actually see that these two operations commute, therefore implying compatibility of $\succ_n$. Well-foundedness follows from well-foundedness of $\succ$. Stability and normal-monotonicity depend upon the particular ordering $\succ$ used in the construction, which must satisfy two stronger properties:

Definition 10. *An ordering $\succ$ on higher-order terms satisfies*

(i) schema-stability *if for all $\beta\eta$-normal terms s, t and substitutions γ, $s \succ t$ implies $t\gamma \longrightarrow^*_{\beta\eta} t'\gamma$ for some term t' such that $s\gamma\downarrow \succ t'\gamma$.*

(ii) schema-monotonicity *if for all $\beta\eta$-normal terms $\lambda x.v : \sigma \to \rho \succ t : \sigma \to \rho$, and for all sequences of $\beta\eta$-normal terms $\langle u_1, \ldots, u_n\rangle$,*

- *if $t = \lambda x.w$, then $v\{x \mapsto \bot_\sigma(u_1, \ldots, u_n)\} \succ w\{x \mapsto \bot_\sigma(u_1, \ldots, u_n)\}$*
- *otherwise, $v\{x \mapsto \bot_\sigma(u_1, \ldots, u_n)\} \succ @(t, \bot_\sigma(u_1, \ldots, u_n))$.*

Theorem 2. *Let $\succ$ be a higher-order reduction ordering fulfiling the schema-stability and schema-monotonicity properties. Then $\succ_n$ is a normal higher-order reduction ordering.*

The proof of this theorem requires several preliminary technical lemmas stating properties of neutralization with respect to normalization, before to start proving stability and normal-monotonicity of $\succ_n$.

5 Normal Higher-Order Recursive Path Orderings

While the higher-order recursive path ordering satisfies schema-monotonicity, it does not satisfy schema-stability. Fortunately, a simple natural restriction suffices in case of an application on left (Cases 5 and 7 of the coming definition). In order to ease the presentation, we present a simple version of the (restricted) higher-order recursive path ordering, which will be sufficient for all examples to come. We assume given:

1. a partition $Mul \uplus Lex$ of $\mathcal{F}$ and a quasi-ordering $\geq_{\mathcal{F}}$ on $\mathcal{F}$, called the *precedence*, such that $>_{\mathcal{F}}$ is well-founded;
2. a quasi-ordering $\geq_{\mathcal{T}_S}$ on types such that $>_{\mathcal{T}_S}$ is well-founded and preserves the functional structure of functional types [12].

Because of type comparisons, the higher-order recursive path ordering enjoys but a weak subterm property A used in its definition:

Definition 11. *Given* $s : \sigma$ *and* $t : \tau$, $\quad s \underset{rhorpo}{\succ} t$ *iff* $\sigma \geq_{\mathcal{T}_S} \tau$ *and*

1. $s = f(\overline{s})$ *with* $f \in \mathcal{F}$, *and* $u \underset{rhorpo}{\succeq} t$ *for some* $u \in \overline{s}$
2. $s = f(\overline{s})$ *and* $t = g(\overline{t})$ *with* $f >_{\mathcal{F}} g$, *and* A
3. $s = f(\overline{s})$ *and* $t = g(\overline{t})$ *with* $f =_{\mathcal{F}} g \in Mul$ *and* $\overline{s}(\underset{rhorpo}{\succ})_{mul}\overline{t}$
4. $s = f(\overline{s})$ *and* $t = g(\overline{t})$ *with* $f =_{\mathcal{F}} g \in Lex$ *and* $\overline{s}(\underset{rhorpo}{\succ})_{lex}\overline{t}$, *and* A
5. $s = @(s_1, s_2)$, s_1 *is not of the form* $@(X, \overline{w})$ *with* $X \in \mathcal{X}$ *and* $u \succeq_{rhorpo} t$ *for some* $u \in \{s_1, s_2\}$
6. $s = f(\overline{s})$, $@(\overline{t})$ *is an arbitrary left-flattening of* t, *and* A
7. $s = @(s_1, s_2)$, s_1 *is not of the form* $@(X, \overline{w})$ *with* $X \in \mathcal{X}$, $@(\overline{t})$ *is an arbitrary left-flattening of* t *and* $\{s_1, s_2\}$ $(\succ_{rhorpo})_{mul}$ $\overline{t}$
8. $s = \lambda x : \alpha.u$, $t = \lambda x : \beta.v$, $\alpha =_{\mathcal{T}_S} \beta$ *and* $u \underset{rhorpo}{\succ} v$
9. $s = @(\lambda x.u, v)$ *and* $u\{x \mapsto v\} \underset{rhorpo}{\succeq} t$

$$\text{where} \begin{cases} s \succeq_{rhorpo} t \text{ iff } s \succ_{rhorpo} t \text{ or } s =_\alpha t \text{ or} \\ \qquad\qquad s = @(X, u), t = @(X, v) \text{ and } u \succ_{rhorpo} v \\ A = \forall v \in \overline{t} \;\; s \succ_{rhorpo} v \text{ or } u \succeq_{rhorpo} v \text{ for some } u \in \overline{s} \end{cases}$$

Of course, making bound variables fit may need renaming in Case 8, and as usual, $\succ_{mul}$ and $\succ_{lex}$ denote respectively the multiset and lexicographic extensions of the relation $\succ$.

Theorem 3. *$(\succ_{rhorpo})^+$ is a higher-order reduction ordering satisfying schema-stability and schema-monotonicity.*

The relation $\succeq_{rhorpo}$ being non-transitive in general because of case 9, taking its transitive closure is needed to make it into an ordering.

The property of being a higher-order reduction ordering is inherited from the non-restricted version of the ordering, for which Cases 5 and 7 do not restrict the form of s_1 [12]. Without this restriction, we run into the aforementioned problem that any ordering satisfying monotonicity, compatibility and well-foundedness must violate stability. Note that the pairs which cause this violation do not become incomparable in the (quasi-) ordering: they are used to enrich its equality part.

Schema-monotonicity is straightforward, while schema-stability is by induction on the term structure. As a consequence of Theorems 2 and 3:

Theorem 4. *$(\succ_{rhorpo})^*_n$ is a normal higher-order reduction ordering.*

We will approximate $(\succ_{rhorpo})^*_n$ by $(\succ_{rhorpo})_n$ in all coming examples. As can be guessed, we need to define the precedence on the extended signature. In practice, we always make $\bot_\sigma$-function symbols small.

Example 1 (end). Let $\mathsf{diff}_{\mathsf{new}} >_{\mathcal{F}} \{\times_{\mathsf{new}}, +_{\mathsf{new}}, \mathsf{cos}, \bot\}$ and $\mathsf{diff}_{\mathsf{new}} \in \mathsf{Mul}$.
First rule: $s = \mathsf{diff}_{\mathsf{new}}(\mathsf{sin}(@(\mathsf{F}, \bot))) \succ_{\mathsf{rhorpo}} \mathsf{cos}(@(\mathsf{F}, \bot)) \times_{\mathsf{new}} @(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F}, \bot)), \bot)$
Applying first case 2, we recursively obtain two subgoals:
(i) $s \succ_{rhorpo} \mathsf{cos}(@(\mathsf{F}, \bot))$ and (ii) $s \succ_{rhorpo} @(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F}, \bot)), \bot)$.
(i): applying Case 2 yields $s \succ_{rhorpo} @(\mathsf{F}, \bot)$ shown by Case 1 twice.
(ii): applying Case 6 generates two new subgoals
(iii) $s \succ_{rhorpo} \mathsf{diff}_{\mathsf{new}}(@(\mathsf{F}, \bot))$, which holds by case 3, then case 1.
(iv) $\mathsf{diff}_{\mathsf{new}}(\mathsf{sin}(@(\mathsf{F}, \bot))) \succ_{\mathsf{rhorpo}} \bot$, which holds by case 2.
Second rule: $s = \mathsf{diff}_{\mathsf{new}}(@(@(\mathsf{F}, \bot) \times_{\mathsf{new}} @(\mathsf{F}, \bot), \bot)) \succ_{\mathsf{rhorpo}}$
$@(@(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot)),\bot) \times_{\mathsf{new}} @(\mathsf{F},\bot),\bot) +_{\mathsf{new}} @(@(\mathsf{F},\bot) \times_{\mathsf{new}} @(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot)),\bot),\bot)$
Case 2 generates two subgoals:
(i) $s \succ_{rhorpo} @(@(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F}, \bot)), \bot) \times_{\mathsf{new}} @(\mathsf{F}, \bot), \bot)$
(ii) $s \succ_{rhorpo} @(@(\mathsf{F}, \bot) \times_{\mathsf{new}} @(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F}, \bot)), \bot), \bot)$.
By Case 6, (i) generates two new subgoals:
(iii) $s \succ_{rhorpo} @(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F}, \bot)), \bot) \times_{\mathsf{new}} @(\mathsf{F}, \bot)$ and (iv) $s \succ_{rhorpo} \bot$.
The latter holds by case 1 and then case 5. By Case 2, (iii) yields two subgoals:
(v) $s \succ_{rhorpo} @(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F}, \bot)), \bot)$ and (vi) $s \succ_{rhorpo} @(\mathsf{F}, \bot)$.
By Case 6, (v) generates (vii) $s \succ_{rhorpo} \mathsf{diff}_{\mathsf{new}}(@(\mathsf{F}, \bot))$ and (viii) $s \succ_{rhorpo} \bot$
(vii) is solved by Case 3, 5, and 1 successively, and (viii) is solved by cases 1 and 5.

6 Examples

We present two complex examples proven terminating with $(\succ_{rhorpo})^*_n$. For all of them, we give the necessary ingredients for computing the appropriate neutralizations and comparisons. The precise computations can be found in the full version of the paper available from the web. An implementation is available for the original version of the ordering which will be extended to the present one in a near future.

As a convention, missing neutralization levels are equal to 0, in which case the corresponding subset of argument positions will be empty. In all examples, we use a simple type ordering $\geq_{\mathcal{T}_\mathcal{S}}$ equating all data types, which satisfies the requirements given in Section 5. Precedence on function symbols and statuses will be given in full.

In all examples we write $F(X)$ instead of $@(F, X)$ to ease the reading.

Example 2. The coming encoding of first-order prenex normal forms is adapted from [20], where its local confluence is proved via the computation of its (higher-order) critical pairs. Formulas are represented as λ-terms with sort $form$. The idea is that quantifiers are higher-order constructors binding a variable via the use of a functional argument.

$$\mathcal{S} = \{form\}, \quad \mathcal{F} = \{ \wedge, \vee : form \times form \to form; \neg : form \to form; \\ \forall, \exists : (form \to form) \to form\}.$$

$$\begin{array}{ll} P \wedge \forall(\lambda x.Q(x)) \to \forall(\lambda x.(P \wedge Q(x))) & P \wedge \exists(\lambda x.Q(x)) \to \exists(\lambda x.(P \wedge Q(x))) \\ \forall(\lambda x.Q(x)) \wedge P \to \forall(\lambda x.(Q(x) \wedge P)) & \exists(\lambda x.Q(x)) \wedge P \to \exists(\lambda x.(Q(x) \wedge P)) \\ P \vee \forall(\lambda x.Q(x)) \to \forall(\lambda x.(P \vee Q(x))) & P \vee \exists(\lambda x.Q(x)) \to \exists(\lambda x.(P \vee Q(x))) \\ \forall(\lambda x.Q(x)) \vee P \to \forall(\lambda x.(Q(x) \vee P)) & \exists(\lambda x.Q(x)) \vee P \to \exists(\lambda x.(Q(x) \vee P)) \\ \neg(\forall(\lambda x.Q(x))) \to \exists(\lambda x.\neg(Q(x))) & \neg(\exists(\lambda x.Q(x))) \to \forall(\lambda x.\neg(Q(x))) \end{array}$$

Ingredients for neutralization: $\mathcal{L}^1_\forall = 1$, $\mathcal{L}^1_\exists = 1$, $\mathcal{A}^1_\forall = \{\}$, $\mathcal{A}^1_\exists = \{\}$.
Precedence: $\wedge >_\mathcal{F} \{\forall_{\mathsf{new}}, \exists_{\mathsf{new}}\}$, $\vee >_\mathcal{F} \{\forall_{\mathsf{new}}, \exists_{\mathsf{new}}\}$, $\neg >_\mathcal{F} \{\forall_{\mathsf{new}}, \exists_{\mathsf{new}}\}$.
Statuses: $\forall_{\mathsf{new}}, \exists_{\mathsf{new}} \in \mathsf{Mul}$ □

Example 3. Encoding of natural deduction, taken from [6].
Let $\mathcal{S} = \{o,\ c : * \times * \to *\}$. Because we did not consider polymorphism, the following signature and rules is parameterized by all possible types $\sigma, \tau, \rho \in \mathcal{T}_\mathcal{S}$.

$$\begin{array}{l} \mathcal{F} = \{\ app_{\sigma,\tau} : (\sigma \to \tau) \times \sigma \to \tau;\ \ abs_{\sigma,\tau} : (\sigma \to \tau) \to (\sigma \to \tau); \\ \qquad \mathit{\Pi}_{\sigma,\tau} : \sigma \times \tau \to c(\sigma,\tau);\ \mathit{\Pi}^0_{\sigma,\tau} : c(\sigma,\tau) \to \sigma;\ \mathit{\Pi}^1_{\sigma,\tau} : c(\sigma,\tau) \to \tau; \\ \qquad \exists^+_\sigma : o \times \sigma \to c(o,\sigma);\ \ \exists^-_{\sigma,\tau} : c(o,\sigma) \times (o \to \sigma \to \tau) \to \tau\ \ \}. \end{array}$$

$$\begin{array}{l} \mathcal{X} = \{\ X : \sigma;\ Y : \tau;\ Z : o;\ T : c(o,\rho),\ F : \sigma \to \tau;\ G : o \to \sigma \to \tau, \\ \qquad H : o \to \rho \to (\sigma \to \tau), I : o \to \rho \to c(\sigma,\tau), J : o \to \rho \to c(o,\sigma)\}. \end{array}$$

$$
\begin{array}{c}
app_{\sigma,\tau}(abs_{\sigma,\tau}(F), X) \to F(X) \\
\Pi^0_{\sigma,\tau}(\Pi_{\sigma,\tau}(X, Y)) \to X \\
\Pi^1_{\sigma,\tau}(\Pi_{\sigma,\tau}(X, Y)) \to Y \\
\exists^-_{\sigma,\tau}(\exists^+_{\sigma}(Z, X), G) \to G(Z, X)
\end{array}
$$

$$
\begin{array}{c}
app_{\sigma,\tau}(\exists^-_{\rho,\sigma\to\tau}(T, H), X) \to \exists^-_{\rho,\tau}(T, \lambda x : o\, y : \rho.app_{\sigma,\tau}(H(x, y), X)) \\
\Pi^0_{\sigma,\tau}(\exists^-_{\rho,c(\sigma,\tau)}(T, I)) \to \exists^-_{\rho,\tau}(T, \lambda x : o\, y : \rho.\Pi^0_{\sigma,\tau}(I(x, y))) \\
\Pi^1_{\sigma,\tau}(\exists^-_{\rho,c(\sigma,\tau)}(T, I)) \to \exists^-_{\rho,\tau}(T, \lambda x : o\, y : \rho.\Pi^1_{\sigma,\tau}(I(x, y))) \\
\exists^-_{\sigma,\tau}(\exists^-_{\rho,c(o,\sigma)}(T, J), G) \to \exists^-_{\rho,\tau}(T, \lambda x : o\, y : \rho.\exists^-_{\sigma,\tau}(J(x, y), G))
\end{array}
$$

Neutralization: $\mathcal{L}^2_{\exists^-_{\sigma,\tau}} = 2$ and $\mathcal{A}^2_{\exists^-_{\sigma,\tau}} = \{1\}$ for all possible types σ and τ.
Precedence: $\{app_{\sigma,\tau}, \Pi^0_{\sigma,\tau}, \Pi^1_{\sigma,\tau}\} >_{\mathcal{F}} \exists^-_{new\ \rho,\tau}$ and $\exists^-_{new\ \rho,\tau} = \exists^-_{new\ \sigma,\tau}$ for all possible types ρ, σ and τ.
Statuses: $\exists^-_{\mathsf{new}\ \sigma,\tau} \in \mathsf{Lex}$ and $\mathsf{app}_{\sigma,\tau}, \Pi^0_{\sigma,\tau}, \Pi^1_{\sigma,\tau} \in \mathsf{Mul}$ for all ρ σ and τ; $\Box$

7 Conclusion

Proving termination properties of Nipkow's rewriting was considered in [8] and [2]. The former yields a *methodology* needing important user-interaction to prove that the constructed ordering has the required properties. Here, our method does provide with an ordering having automatically all desired properties. The user has to provide with a precedence and statuses as usual with the recursive path ordering. He or she must also provide with neutralization levels together with filters selecting appropriate arguments for each function symbols. This requires of course some expertise, but can be implemented by searching non-deterministically for appropriate neutralization levels and filters, as done in many implementations of the recursive path ordering for the precedence and statuses.

The higher-order recursive path ordering generalizes the notion of general schema as formulated in [3] where the notion of computability closure was introduced. However, what can be done with the schema can be done with the higher-order recursive path ordering when using the computability closure of $f(\bar{t})$ in the subterm case, instead of simply the set of subterms $\bar{t}$ itself. The general definition of the higher-order recursive path ordering with closure is given in [12]. It is however interesting to notice that the neutralization mechanism is powerful enough so as to dispense us with using the closure for all these complex examples taken from the literature that we have considered here and in [13]. It remains to be seen whether the closure plays in the context of normal higher-order rewriting, a role as important as for proving termination of recursor rules

for inductive types for which plain pattern matching is used instead of higher-order pattern matching.

References

1. F. Blanqui, J.-P. Jouannaud, and M. Okada. The Calculus of Algebraic Constructions. In Narendran and Rusinowitch, Proc. RTA'99, 1999.
2. F. Blanqui. Termination and Confluence of Higher-Order Rewriting Systems. In Proc. RTA'00, 2000.
3. F. Blanqui, J.-P. Jouannaud, and M. Okada. Inductive Data Type Systems. *Theoretical Computer Science*, 272(1-2):41–68. 2002.
4. H. Barendregt. Functional Programming and Lambda Calculus. In [22], pages 321–364.
5. H. Barendregt. *Handbook of Logic in Computer Science*, chapter Typed lambda calculi. Oxford Univ. Press, 1993. eds. Abramsky et al.
6. J. van de Pol. Termination of Higher-Order Rewrite Systems. PhD thesis, Department of Philosophy, Utrecht University, 1996.
7. N. Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17(3):279–301, March 1982.
8. J. van de Pol and H. Schwichtenberg. Strict functional for termination proofs. In *Typed Lambda Calculi and Applications, Edinburgh*. Springer-Verlag, 1995.
9. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In [22], pages 321–364.
10. J.-P. Jouannaud. Higher-Order rewriting: Framework, Confluence and termination. In A. Middeldorp, V. van Oostrom, F. van Raamsdonk and R. de Vrijer eds., *Processes, Terms and Cycles: Steps on the road to infinity*. Essays Dedicated to Jan Willem Klop on the occasion of his 60th Birthday. LNCS 3838. Springer Verlag, 2005.
11. J.-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In Giuseppe Longo, editor, *Fourteenth Annual IEEE Symposium on Logic in Computer Science*, Trento, Italy, July 1999. IEEE Comp. Soc. Press.
12. J.-P. Jouannaud and A. Rubio. Polymorphic Higher-Order Recursive Path Orderings. 2005. Submitted to JACM. http://www.lix.polytechnique.fr/Labo/Jean-Pierre.Jouannaud.
13. J.-P. Jouannaud and A. Rubio. Higher-Order Orderings for Normal Rewriting. 2005. Full version. http://www.lix.polytechnique.fr/Labo/Jean-Pierre.Jouannaud.
14. J.-P. Jouannaud and A. Rubio. Higher-Order Recursive Path Orderings à la carte. 2001. http://www.lix.polytechnique.fr/Labo/Jean-Pierre.Jouannaud.
15. J.-P. Jouannaud, F. van Raamsdonk and A. Rubio Higher-order rewriting with types and arities. 2005. http://www.lix.polytechnique.fr/Labo/Jean-Pierre.Jouannaud.
16. J. W. Klop. Combinatory Reduction Relations. Mathematical Centre Tracts 127. Mathematisch Centrum, Amsterdam, 1980.
17. J. W. Klop. Term Rewriting Systems. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum eds., *Handbook of Logic in Computer Science*, vol. 2:2–116. Oxford University Press, 1992.
18. R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(1):3–29, February 1998.
19. D. Miller. A Logic Programming Language with Lambda-Abstraction, Function Variables, and Simple Unification. In *Journal and Logic and Computation* 1(4):497–536, 1991.
20. T. Nipkow. Higher-order critical pairs. In *6th IEEE Symp. on Logic in Computer Science*, pages 342–349. IEEE Computer Society Press, 1991.
21. L. C. Paulson. Isabelle: the next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*. Academic Press, 1990.
22. J. van Leeuwen, ed. Handbook of Theoretical Computer Science, vol. B. North-Holland, 1990.