Normal Higher-Order Termination

Jean-Pierre Jouannaud, Albert Rubio

Normal Higher-Order Termination

Jean-Pierre Jouannaud, LIX, École Polytechnique, Palaiseau, France
Albert Rubio, Technical University of Catalonia, Barcelona, Spain

We extend the termination proof methods based on reduction orderings to higher-order rewriting systems based on higher-order pattern matching. We accommodate, on the one hand, a weakly polymorphic, algebraic extension of Church's simply typed λ-calculus, and on the other hand, any use of eta, as a reduction, as an expansion or as an equation. The user's rules may be of any type in this type system, either a base, functional, or weakly polymorphic type.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic**]: Lambda calculus and related systems; F.4.2 [**Grammars and Other Rewriting Systems**]

General Terms: Theory

Additional Key Words and Phrases: Higher-order rewriting, higher-order orderings, higher-order patterns, typed lambda calculus

ACM Reference Format:
Jean-Pierre Jouannaud and Albert Rubio. 2013. Normal Higher-Order Termination *ACM* 79, 3, Article 3 (May 2014), 46 pages.
DOI : http://dx.doi.org/10.1145/0000000.0000000

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 has been pioneered in this context by Nipkow [Mayr and Nipkow 1998] and is available in Isabelle [Nipkow and Paulson 1992]. Its main feature is the use of higher-order pattern matching for firing rules. A generalization of Nipkow's setting allows using rewrite rules of higher type [Middeldorp et al. 2005; Jouannaud and Li 2012]. Besides, it is shown there that using the extensionality rule as an expansion [Nipkow 1991] or as a reduction [Jouannaud 2005] 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, called *normal termination*, thanks to the notion of a *normal higher-order reduction ordering*. While higher-order reduction orderings target termination of higher-order rewriting based on plain pattern matching, hence must *include* β-reductions, normal higher-order or-

This work has been partly carried out in the INRIA-LIAMA project Formes and the UPC projects SweetLogics and DAMAS, supported by Spanish MEC/MICINN grants TIN2010-21062-C02-01 and TIN2013-45732-C4-3-P respectively. The first author is now Professor emeritus at Université Paris-Sud, although still working at École Polytechnique. This paper is based on a preliminary version published by the same authors in the Proceedings of the 17th International Conference on Term Rewriting and Applications.

DOI : http://dx.doi.org/10.1145/0000000.0000000

derings target termination of higher-order rewriting based on higher-order pattern matching, hence must be *compatible* with $\beta\eta$-equality. We show however that, in general, monotonicity, stability, compatibility and well-foundedness cannot be satisfied at the same time. This leads to the definition that higher-order rewrite orderings are *normal* when they include a subrelation enjoying some stability property for terms in $\beta\eta$-normal form. Checking the rewrite rules against such a subrelation then ensures normal termination.

How to build normal higher-order reduction orderings from higher-order reduction orderings is our second contribution, that we illustrate with the case of the higher-order recursive path ordering HORPO [Jouannaud and Rubio 2007], yielding its normal version NHORPO. Some examples of use are presented. The method would also apply to other similar orders, such as CPO [Blanqui et al. 2008].

This method does not work well in the presence of abstractions in the righthand side of rules. Our third contribution is an interpretation-based framework that solves this problem, at an abstract level first, where the properties of such interpretations are specified which imply preservation of normal termination, and then at a concrete level, by defining a particular termination-preserving interpretation, *neutralization*, whose role is to remove abstractions that can never be applied. Combining neutralization with NHORPO allows us to solve complex examples from the literature.

So far, these methods target the use of higher-order reduction orderings. In the recent years, other methods have popped up that allow us to show higher-order termination, like higher-order dependency pairs. The question arises whether these methods that aim at proving higher-order termination can also be turned into methods aiming at proving normal higher-order termination? Our fourth contribution is a transformation of a higher-order rewrite system R into a new system R' containing β-reductions, such that R is normal terminating if R' is terminating. Our second contribution then appears as a refinement of the latter when using a method based on a higher-order reduction ordering like HORPO.

We describe our framework for simply typed terms in Section 2, and for higher-order rewriting in Section 3. NHORPO is given in Section 4 and illustrated by an example. Termination-preserving interpretations, including neutralization, are introduced and studied in Section 5. Our implementation is described briefly in Section 7, several examples are carried out there. Section 6 covers weak polymorphism, and is illustrated by yet another example. Our general transformation reducing normal higher-order termination to higher-order termination is introduced and illustrated in Section 8. The relevant literature is discussed in Section 9 where it is compared with the present work whose significance is discussed at this occasion. Brief concluding remarks are given in Section 10.

Readers are assumed familiar with the basics of term rewriting [Dershowitz and Jouannaud 1990; Terese 2003] and typed lambda calculi [Barendregt 1993].

2. HIGHER-ORDER ALGEBRAS

This section recalls the notion of monomorphic higher-order algebras (HOAs). We discuss in Section 6 how our results can be lifted to polymorphic higher-order algebras. Both frameworks originate from [Jouannaud and Rubio 2007].

2.1. Types and Signatures

Given a set $\mathcal{S}$ of *sort symbols* of a fixed arity, denoted by $s : *^n \Rightarrow *$, the set $\mathcal{T}_\mathcal{S}$ of *simple types* is generated by the following grammar:

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

Types are *functional* when headed by the $\to$ symbol, and *data types* when headed by a sort symbol. As usual, $\to$ associates to the right. We will often make explicit the functional structure of an arbitrary type τ by writing it in the *canonical form* $\sigma_1 \to \ldots \to \sigma_n \to \sigma$, with $n \geq 0$, where n is the *arity* of τ and σ is a data type called *canonical output type* of τ. We use $\sigma, \tau, \rho, \theta$ for arbitrary types.

We are given a set of function symbols denoted by the letters f, g, h, which 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 *output type* $\sigma \in \mathcal{T}_\mathcal{S}$. Let $\mathcal{F} = \biguplus_{\sigma_1,\ldots,\sigma_n,\sigma} \mathcal{F}_{\sigma_1\times\ldots\times\sigma_n\Rightarrow\sigma}$ be the set of all function symbols. The membership of a given function symbol f to a set $\mathcal{F}_{\sigma_1\times\ldots\times\sigma_n\Rightarrow\sigma}$ is called a *type declaration* and written $f : \sigma_1 \times \ldots \times \sigma_n \Rightarrow \sigma$. In case $n = 0$, the declaration is written $f : \sigma$. A type declaration is *first-order* if its constituent types are data types, and higher-order otherwise. $\mathcal{F}$ is said to be *first-order* if all its type declarations are first-order, and *higher-order* otherwise.

The pair $\mathcal{S}; \mathcal{F}$ is called the *signature*. We say that the signature is *first-order, higher-order* when $\mathcal{F}$ satisfies the corresponding property.

2.2. Raw 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 following grammar rules:

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

Raw terms of the forms $\lambda x : \sigma.u$, $@(u, v)$, and $f(t_1, \ldots, u_n)$ are respectively called *abstractions, applications*, and *pre-algebraic*. As a matter of convenience, we may sometimes omit the type σ in $\lambda x : \sigma.u$ and write $@(u, v_1, \ldots, v_n)$ for $@(\ldots @(u, v_1), \ldots, v_n)$, assuming $n \geq 1$.

Our syntax requires using explicit applications for variables, since they cannot take arguments. In the examples, we shall however allow ourselves writing $X(v_1, \ldots, v_n)$ for $@(X, v_1, \ldots, v_n)$ when X is a higher-order variable. On the other hand, function symbols have arities, eliminating the need for an explicit application of a function symbol to its arguments. Currying a function symbol is possible by taking zero for its arity.

Raw terms are identified with finite labeled trees by considering $\lambda x : \sigma._-$, for each variable x and type σ, as a unary function symbol taking a raw term u as argument to construct the raw term $\lambda x : \sigma.u$. We denote the set of free variables of the raw term t by $\mathcal{V}ar(t)$, and its size (the number of symbols occurring in t) by $|t|$. A raw term is *closed* if it has no free variable. The notation $\overline{s}$ will be ambiguously used to denote a list, or a multiset, or a set of raw terms $s_1, \ldots, s_n$.

Positions are strings of positive integers. $\wedge$ and $\cdot$ denote respectively the empty string (root position) and the concatenation of strings. We use $\mathcal{P}os(t)$ for the set of positions in t, and $>$ for the prefix ordering on positions. Incomparable positions are called *disjoint*. The *subterm* of t at position p is denoted by $t|_p$, and we write $t \trianglerighteq t|_p$ for the subterm relationship. The result of replacing $t|_p$ at position p in t by u is denoted by $t[u]_p$, and we denote it by $t[u]_{p_1 \ldots p_n}$ if the same replacement is made in several positions $p_1 \ldots p_n$ at once. Accordingly, we use $t[x : \sigma]_p$ for a raw term with a unique hole of type σ at position p, called a *context* (and respectively $t[x : \sigma]_{p_1 \ldots p_n}$ when we have several holes). Both p and $x : \sigma$ may be omitted.

2.3. Typing

Definition 2.1. An *environment* Γ is a finite set of pairs written as $\{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ such that $x_i \in \mathcal{X}$, $\sigma_i \in \mathcal{T}_\mathcal{S}$, 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. We may write $\Gamma(x_i)$ for σ_i. Given two environments Γ and Γ', their *composition* is the environment $\Gamma \cdot \Gamma' = \Gamma' \cup \{x : \sigma \in \Gamma \mid x \notin \mathcal{V}ar(\Gamma')\}$. Γ' is *compatible* with Γ if $\Gamma \cdot \Gamma' = \Gamma \cup \Gamma'$.

We now collect all declarations into a single *environment* $\Sigma; \Gamma$, of which the signature $\Sigma = \mathcal{S}; \mathcal{F}$ is the fixed part. We assume that there is exactly one declaration for each symbol in an environment $\Sigma; \Gamma$. We may omit Σ in our examples, using the word *environment* for Γ alone.

Typing rules restrict the set of raw terms by constraining them to follow a precise discipline. Our typing judgments are written as $\Gamma \vdash_\Sigma s : \sigma$, and read "$s$ has type σ in the environment Γ". The typing judgments are displayed in Figure 1.

Variables:	**Abstraction:**
$\dfrac{x : \sigma \in \Gamma}{\Gamma \vdash_\Sigma x : \sigma}$	$\dfrac{\Gamma \cdot \{x : \sigma\} \vdash_\Sigma t : \tau}{\Gamma \vdash_\Sigma (\lambda x : \sigma.t) : \sigma \to \tau}$
Application:	**Functions:**
$\dfrac{\Gamma \vdash_\Sigma s : \tau \to \sigma \quad \Gamma \vdash_\Sigma t : \tau}{\Gamma \vdash_\Sigma @(s,t) : \sigma}$	$\dfrac{f : \sigma_1 \times \ldots \times \sigma_n \Rightarrow \sigma \in \mathcal{F} \quad \Gamma \vdash_\Sigma t_1 : \sigma_1 \ \ldots \ \Gamma \vdash_\Sigma t_n : \sigma_n}{\Gamma \vdash_\Sigma f(t_1, \ldots, t_n) : \sigma}$

Fig. 1. Typing judgments in higher-order algebras

Given an environment $\Sigma; \Gamma$, a raw term s has type σ if the judgment $\Gamma \vdash_\Sigma s : \sigma$ is provable in our type system. Given an environment $\Sigma; \Gamma$, a raw term s is *typable*,

and said to be a *term* if there exists a type σ such that $\Gamma \vdash_\Sigma s : \sigma$. Classically, we consider the proof of a given judgment as a tree whose nodes are labeled by the judgments derived in the proof and the edges by the names of the rules used in the proof. We use as usual $\mathcal{P}os(P)$ for the set of positions of the proof tree P. We can therefore speak of the rule used at a position $p \in \mathcal{P}os(P)$ whose conclusion labels the node at p while its premises, displayed above the conclusion, label its sons.

We shall use without mentioning that a term has a unique type in a given environment. We further recall two important typing properties of HOAs used later:

LEMMA 2.2 (SUBTERM). *Let P denote the proof of the judgment $\Gamma \vdash_\Sigma s : \sigma$ and p be a position in $\mathcal{P}os(s)$. Then, there exists a unique position $q \in \mathcal{P}os(P)$ such that the subproof $P|_q$ is a proof of a judgment of the form $\Gamma_{s|_p} \vdash_\Sigma s|_p : \tau$ in which τ is the type of $s|_p$ in $\Gamma_{s|_p}$.*

LEMMA 2.3 (REPLACEMENT). *Assume given an environment $\Sigma; \Gamma$, two terms s and v, two types σ and τ, and a position $p \in \mathcal{P}os(s)$ such that $\Gamma \vdash_\Sigma s : \sigma$, $\Gamma_{s|_p} \vdash_\Sigma s|_p : \tau$, and $\Gamma_{s|_p} \vdash_\Sigma v : \tau$. Then, $\Gamma \vdash_\Sigma s[v]_p : \sigma$.*

2.4. Substitutions

Definition 2.4. A *substitution* $\gamma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ is a finite set of pairs made of a variable in its domain $\mathcal{D}om(\gamma) = \{x_1, \ldots, x_n\}$ and a term in its range $\mathcal{R}an(\gamma) = \{t_1, \ldots, t_n\}$, such that $\forall i \in [1..n]$, $t_i \neq x_i$ and $\forall i \neq j \in [1..n]$, $x_i \neq x_j$. We denote by $\gamma_{\setminus V}$ the restriction of γ to the variables in $\mathcal{V}ar(\gamma) \setminus V$. We use the letter γ for substitutions and postfix notation for their application.

Definition 2.5. A substitution γ operates as an endomorphism on s and yields the term $s\gamma$ defined as:

If	$s = x \in \mathcal{X}$ and $x \notin \mathcal{V}ar(\gamma)$	then	$s\gamma = x$
If	$s = x \in \mathcal{X}$ and $x \mapsto t \in \gamma$	then	$s\gamma = t$
If	$s = @(u, v)$	then	$s\gamma = @(u\gamma, v\gamma)$
If	$s = f(u_1, \ldots, u_n)$	then	$s\gamma = f(u_1\gamma, \ldots, u_n\gamma)$
If	$s = \lambda x : \tau.u$	then	$s\gamma = \lambda z : \tau.u\gamma_x$

where $\gamma_x = \gamma_{\setminus\{x\}} \cup \{x \mapsto z\}$ for some fresh variable z.

A notion of typed substitution is used in [Jouannaud and Rubio 2007]. The present more convenient untyped notion is borrowed from [Blanqui et al. 2013]. As expected, substitutions preserve typability:

LEMMA 2.6 (TYPE INVARIANCE). *Let s, t be two terms and γ be a substitution such that $t = s\gamma$. Then,*

(i) if $\Gamma \vdash_\Sigma s : \sigma$ and $\forall x \in \mathcal{V}ar(s)$, $\Theta \vdash_\Sigma x\gamma : \Gamma(x)$ for some environments Γ, Θ and type σ, then $\Theta \vdash_\Sigma s\gamma : \sigma$.

(ii) if $\Gamma \vdash_\Sigma t : \tau$ for some environment Γ and type τ, then there is some environment Θ such that $\Theta \vdash_\Sigma s : \tau$ and $\Gamma \vdash_\Sigma x\gamma : \Theta(x)$ for all $x \in \mathcal{V}ar(s)$.

2.5. Conversions

The following three equations originate from the (pure) λ-calculus, and are called α-, β- and η-equality respectively:

$$\begin{array}{rl} \lambda x:\alpha.u & =_\alpha \ \lambda y:\alpha.u\{x\mapsto y\} \text{ if } y\notin \mathcal{B}Var(v)\ \cup (\mathcal{V}ar(v)\setminus\{x\}) \\ @(\lambda x:\sigma.v,w) & =_\beta \ v\{x\mapsto w\} \\ \lambda x:\sigma.@(u,x) & =_\eta \ u \text{ if } x\notin \mathcal{V}ar(u) \qquad \text{(usually called extensionality)} \end{array}$$

In the above equations, u, v and w stand for arbitrary terms to which substitutions $\{x \mapsto y\}$ and $\{x \mapsto u\}$ apply. We consider α-convertible terms as identical, and therefore omit α-conversions in the sequel. Both β and η-equalities can be oriented as rewrite rules. There are two possible choices for rewriting with η, either as a reduction (from left to right) or as an expansion (from right to left), 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 Church-Rosser properties one may need, as recalled below.

We consistently use the following important notations: $u \longrightarrow_\beta v$ for one β-rewrite step, $u \longrightarrow^*_\beta v$ for its reflexive, transitive closure, and $=_\beta$ for its symmetric, reflexive, transitive closure, also called β-equality. Similarly $u \longrightarrow_\eta v$, $u \longrightarrow^*_\eta v$, and $=_\eta$ are the corresponding notations for η used as a reduction. We also use $u{\downarrow_\beta}$, $u{\downarrow_\eta}$ and $u{\downarrow_{\beta\eta}}$ for the β-normal form, η-normal form and $\beta\eta$-normal form of a typable term u. The notation $u{\downarrow}$ is exclusively used as a short form of $u{\downarrow_{\beta\eta}}$. We say that a term is β-*normal* (resp., η-normal, normal) if it is in normal form for β-reduction (resp., η-reduction, $\beta\eta$-reductions). We shall not need a notation for η-expansion, η-long forms or β-normal η-long forms. We shall use, possibly without mentioning, several related properties of the simply typed λ-calculus:

(1) Subject reduction: Assume $\Gamma \vdash_\Sigma s:\sigma$ and $s \longrightarrow_\beta t$. Then, $\Gamma \vdash_\Sigma t:\sigma$.
(2) η-preservation: Assume $\Gamma \vdash_\Sigma s:\sigma$ and $s \longleftrightarrow_\eta t$. Then, $\Gamma \vdash_\Sigma t:\sigma$.
(3) Postponement of η: $(\forall u,v,w)\ u =_\eta v \longrightarrow_\beta w$, $(\exists v')\ u \longrightarrow^*_\beta v' =_\eta w$.
(4) Church-Rosser property (i) of β-reductions: $s =_\beta t$ implies $s{\downarrow_\beta} = t{\downarrow_\beta}$
(5) Church-Rosser property (ii) of η-reductions: $s =_\eta t$ implies $s{\downarrow_\eta} = t{\downarrow_\eta}$
(6) Church-Rosser property (iii) of β-reductions modulo η: $s =_{\beta\eta} t$ iff $s{\downarrow_\beta} =_\eta t{\downarrow_\beta}$
(7) Church-Rosser property (iv) of $\beta\eta$-reductions: $s =_{\beta\eta} t$ iff $s{\downarrow} = t{\downarrow}$
(8) Closedness of η-normal forms by instantiation: $(t\gamma){\downarrow_\eta} = t{\downarrow_\eta}\ \gamma{\downarrow_\eta}$, a consequence of (5), since variables bound in t cannot occur free in γ.

3. NORMAL HIGHER-ORDER REWRITING OF HIGHER TYPE

Introduced in [Nipkow 1991], normal higher-order rewriting allows defining computations over λ-terms used as a suitable abstract syntax for encoding functional objects like programs or specifications. It differs from the more traditional definition of plain higher-order rewriting from [Jouannaud and Okada 1991]:

Definition 3.1. A *rewrite rule* is a pair of terms written $l \to r$, such that $l \notin \mathcal{X}$, $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$ and $(\forall \Theta, \theta, \gamma)\, \Theta \vdash_\Sigma l\gamma : \theta$ iff $\Theta \vdash_\Sigma r\gamma : \theta$. A *higher-order term rewriting system* is a set of rules containing both rules β and η.

Given a higher-order term rewriting system R, an environment Γ, two higher-order terms s and t, and a type σ such that $\Gamma \vdash_\Sigma s : \sigma$, we say that s rewrites to t at position p with the rule $l \to r$ and the term substitution γ, written $\Gamma \vdash s \longrightarrow^p_R t$, or $s \longrightarrow^p_R t$ assuming Γ, if $s|_p = l\gamma$ and $t = s[r\gamma]_p$.

Nipkow's formulation of normal higher-order rewriting, uses β and η in two different ways, both as rules and equalities: given a term s to be rewritten with a set R of rules, s is first normalized, using η-long β-normal forms, before being searched for lefthand sides of rules in R via higher-order pattern matching [Nipkow 1991; Mayr and Nipkow 1998]. In [Middeldorp et al. 2005; Jouannaud and Li 2012], Nipkow and Mayr's confluence result is generalized by allowing using η as a reduction on the one hand, and rules of higher type on the other hand.

We now define normal higher-order rewriting so as to capture the different ways in which a term can be $\beta\eta$-normalized before pattern-matching a subterm with a lefthand side of rule. This definition is not meant to be used for computing, as is the case of the previous just mentioned two. The fact that it has poor operational properties (including confluence) is therefore irrelevant. Its sole purpose is to study the termination properties of Nipkow's normal higher-order rewriting and its variants: by providing a method for proving termination of this more general relation, we do provide a method for all variants of higher-order rewriting based on higher-order pattern matching, independently of a particular orientation of extensionality.

From now on, by *normal higher-order rewriting* (possibly omitting normal), we mean the coming definition, while Nipkow's specific definition is systematically referred to as *Nipkow's higher-order rewriting*.

Definition 3.2. A *normal rewrite rule* is a pair of β-normal terms written $l \to r$, such that $l{\downarrow_\eta} \notin \mathcal{X}$, $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$ and $(\forall \Theta, \theta, \gamma)\, \Theta \vdash_\Sigma l\gamma{\downarrow_\beta} : \theta$ iff $\Theta \vdash_\Sigma r\gamma{\downarrow_\beta} : \theta$.

A *normal term rewriting system* is a set of normal rules. Given a normal term rewriting system R, an environment Γ, two β-normal terms s and t, and a type σ such that $\Gamma \vdash_\Sigma s : \sigma$, we say that s rewrites to t at position p with the normal rule $l \to r$ and the β-normal term substitution γ, written $\Gamma \vdash s \longrightarrow^p_{R_{\beta\eta}} t$, or $s \longrightarrow^p_{R_{\beta\eta}} t$ assuming Γ, if $s|_p =_{\beta\eta} l\gamma$ and $t =_\eta s[r\gamma]_p{\downarrow_\beta}$.

Note that our definition does not require rules to be typable, but does require that in case they are, then, their normalized instances have the same type. This property is weaker than usual, but suffices for our purpose and is indeed more convenient to work with. A key observation is:

LEMMA 3.3 (TYPE PRESERVATION). *Let s be a β-normal term such that $\Gamma \vdash_\Sigma s : \sigma$ and $\Gamma \vdash s \longrightarrow^p_{R_{\beta\eta}} t$. Then $\Gamma \vdash_\Sigma t : \sigma$.*

PROOF. By Lemma 2.2, $\Gamma_{s|_p} \vdash_\Sigma s|_p : \tau$. Let $l \to r$ be the rule of R matching $s|_p$, hence $s|_p =_{\beta\eta} l\gamma$. Since s is β-normal, so is $s|_p$, hence, by Property (6) of untyped λ-calculus, $(l\gamma)\!\downarrow_\beta =_\eta s|_p$. By η-preservation, $\Gamma_{s|_p} \vdash_\Sigma (l\gamma)\!\downarrow_\beta : \tau$. By Definition 3.2, $\Gamma_{s|_p} \vdash_\Sigma (r\gamma)\!\downarrow_\beta : \tau$. By Lemma 2.3, $\Gamma \vdash_\Sigma s[(r\gamma)\!\downarrow_\beta]_p : \tau$. By confluence and subject reduction, $\Gamma \vdash_\Sigma s[r\gamma]_p\!\downarrow_\beta : \tau$. We conclude by η-preservation. □

We often consider type preserving higher-order rewriting as a relation on terms instead of on judgments, therefore simplifying our notations.

Example 3.4 (*differentiation*). 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. The free variable F stands for a function over the reals, while x, y stand for real values. Let $\mathcal{S} = \{\mathsf{real}\}$, and

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

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

This example makes sense when using normal higher-order rewriting, because using plain pattern matching instead would not allow for computing the derivative of all expressions: rewriting the expression $\mathsf{diff}(\lambda \mathsf{x}.\mathsf{sin}(\mathsf{x}))$ does require higher-order pattern matching, since taking for F the identity substitution $\lambda y.y$, we have $\mathsf{diff}(\lambda \mathsf{x}.\mathsf{sin}(@(\lambda \mathsf{y}.\mathsf{y}, \mathsf{x})) \longrightarrow_\beta \mathsf{diff}(\lambda \mathsf{x}.\mathsf{sin}(\mathsf{x}))$. Note that we use η-expanded forms here. We shall give a mechanical termination proof of these two rules in Section 4.

3.1. Higher-Order Reduction Orderings

We shall use well-founded relations for proving strong normalization properties [Jouannaud and Rubio 2007]. For our purpose, these relations may not be transitive, but their transitive closures will be well-founded orderings, justifying our abuse of terminology. We shall consider two kinds of higher-order reduction orderings, dubbed *plain* when they include β-reductions and *normal* when they are compatible with $\beta\eta$-equivalence.

Definition 3.5. A binary relation $\succ$ on the set of judgments is

- *coherent* iff for all terms s, t such that $(\Gamma \vdash_\Sigma s : \sigma) \succ (\Gamma \vdash_\Sigma t : \sigma)$, and for all environment Γ' such that $\Gamma' \vdash_\Sigma s : \sigma$ and $\Gamma' \vdash_\Sigma t : \sigma$, then $(\Gamma' \vdash_\Sigma s : \sigma) \succ (\Gamma' \vdash_\Sigma t : \sigma)$;
- *stable* iff for all terms s, t such that $(\Gamma \vdash_\Sigma s : \sigma) \succ (\Gamma \vdash_\Sigma t : \sigma)$, and all substitutions γ such that $\forall x \in \mathcal{V}ar(s)$, $\Theta \vdash_\Sigma x\gamma : \Gamma(x)$ for some environment Θ, then $(\Theta \vdash_\Sigma s\gamma : \sigma) \succ (\Theta \vdash_\Sigma t\gamma : \sigma)$;

– *monotonic* iff for all environments Γ, contexts $u[]$, variable x, type σ and terms s, t s.t. $\Gamma \cdot \{x : \sigma\} \vdash_\Sigma u[x : \sigma] : \tau$ and $(\Gamma \vdash_\Sigma s : \sigma) \succ (\Gamma \vdash_\Sigma t : \sigma)$, then $(\Gamma \vdash_\Sigma u[s] : \tau) \succ (\Gamma \vdash_\Sigma u[t] : \tau)$;
– *functional* iff for all terms s, t, t' such that $(\Gamma \vdash_\Sigma s : \sigma) \succ (\Gamma \vdash_\Sigma t : \sigma)$ and $(\Gamma \vdash_\Sigma t : \sigma \longrightarrow_\beta t' : \sigma)$ then $(\Gamma \vdash_\Sigma s : \sigma) \succ (\Gamma \vdash_\Sigma t' : \sigma)$.

Note that our definition of functionality here is slightly weaker than the one in [Jouannaud and Rubio 2007] which does not assume that $s \succ t$.

Coherence is a natural property ensuring that a comparison does not depend upon the type of the variables in the compared terms. For example $@(x, y) \succ y$ should not depend upon the type of x, y. HORPO and other similar orders are coherent.

Definition 3.6. A *higher-order reduction ordering* is a well-founded ordering of the set of judgments satisfying coherence, stability and monotonicity. A higher-order ordering is *plain* if it also satisfies functionality.

Plain higher-order orderings are an adequate tool for proving strong normalization of plain higher-order rewriting. One may wonder whether they should also include some form of extensionality, as in [Jouannaud and Rubio 2007]. This is indeed not possible if one wants to be compatible with both syntactic choices of having extensionality as either η-reduction (our choice in [Jouannaud and Rubio 2007]) or η-expansion (Di Cosmo-Kesner's choice in [Cosmo and Kesner 1994], which includes a discussion of their respective advantages).

3.2. Normal Higher-Order Reduction Orderings

We would need higher-order orderings satisfying the *compatibility* property:

$$\forall s', s, t, t' \text{ s.t. } (\Gamma \vdash_\Sigma s' : \sigma =_{\beta\eta} s : \sigma), (\Gamma \vdash_\Sigma t : \sigma =_{\beta\eta} t' : \sigma) \text{ and } (\Gamma \vdash_\Sigma s : \sigma) \succ (\Gamma \vdash_\Sigma t : \sigma) \text{ then } (\Gamma \vdash_\Sigma s' : \sigma) \succ (\Gamma \vdash_\Sigma t' : \sigma)$$

Unfortunately, no ordering $\succ$ can satisfy monotonicity, stability, compatibility and well-foundedness together: assume $s : \sigma \succ t : \sigma$ (omitting judgments), where $s : \sigma$ is in β-normal form. Given a variable $X : \sigma \to \tau$, $@(X, s)$ is in β-normal form as well. By monotonicity, $@(X, s) : \tau \succ @(X, t) : \tau$. Consider the substitution $\gamma = \{X \mapsto \lambda y.a\}$ where $a : \tau$ is a constant. By stability, $@(\lambda y.a, s) : \tau \succ @(\lambda y.a, t) : \tau$. By compatibility, $a : \tau \succ a : \tau$, contradicting well-foundedness.

We therefore introduce a new kind of ordering for the normal case:

Definition 3.7. A normal higher-order reduction ordering (*normal ordering*) is a pair $\langle >, \succ \rangle$ of a plain higher-order reduction ordering $\succ$ and a relation $>$ satisfying:

(i) normal η-compatibility: for all β-normal typed terms s, s', t s.t. $s' =_\eta s \succ t$ there exists some β-normal term t' such that $s' \succ t' =_\eta t$.

(ii) *normal stability*: for all β-normal typed terms s, t and β-normal substitutions γ, $s > t$ implies $(s\gamma)\downarrow_\beta \succ (t\gamma)\downarrow_\beta$.

Normal stability is a property that combines: inclusion of $>$ into $\succ$ hence well-foundedness (by taking the identity substitution for γ) ; β-compatibility by taking β-normal forms ; and a form of stability by instantiation.

THEOREM 3.8. *Let $R = \{l_i \to r_i\}_i$ be a normal higher-order rewrite system, and $(>, \succ)$ be a normal higher-order reduction ordering satisfying the property $(\forall l \to r \in R, \forall \Gamma, \forall \sigma)\, \Gamma \vdash l : \sigma$ implies $l' > r'$ for some $l' =_\eta l$ and $r' =_\eta r$.*
Then, the relation $\longrightarrow_{R_{\beta\eta}}$ is strongly normalizing.

PROOF. First, strong normalization is true for arbitrary terms provided it is true for closed terms, since adding new constants at every type does not influence termination. Note that η-equivalence classes of closed terms contains only closed terms.

Let therefore s be a closed β-normal term such that $\Gamma \vdash_\Sigma s : \sigma$. The result is by noetherian induction on s with $\succ$, showing the property, that will be useful to have as our induction argument, that no term in the η-equivalence of s can originate an infinite sequence of R-reductions.

If s is in $R_{\beta\eta}$-normal form, then so are all its subterms $s|_p$, for $p \in \mathcal{P}os(s)$. Now, all terms in the η-equivalence class of $s|_p$ are in $R_{\beta\eta}$-normal form by definition of higher-order rewriting, and we are done with this case.

Otherwise, by definition of higher-order rewriting, s rewrites at some position $p \in \mathcal{P}os(s)$ with rule $l \to r \in R$ and β-normal substitution γ, hence $s|_p =_{\beta\eta} l\gamma$ and $t =_\eta s[r\gamma]_p{\downarrow_\beta}$. Because rewriting cannot introduce free variables, t must be closed as well. We then show that $s \succ t'$ for some β-normal term $t' =_\eta t$. By induction hypothesis, no term in the η-equivalence class of t' can originate an infinite sequence of R-reductions. The property is therefore true of t, hence of s.

We are left showing that $\Gamma \vdash_\Sigma s \succ t'$ for some $t' =_\eta t$. Since s is β-normal, by Church-Rosser property (iii) and η-postponement, we have $s|_p =_\eta (l\gamma){\downarrow_\beta}$, hence $s =_\eta s[(l\gamma){\downarrow_\beta}]_p$. Using additionally Church-Rosser property (i), $t =_\eta s[(r\gamma){\downarrow_\beta}]_p{\downarrow_\beta}$. By assumption, $l' > r'$ for some $l' =_\eta l$ and $r' =_\eta r$, hence, by normal stability applied with the β-normal substitution $\gamma{\downarrow_\beta}$, $(l'\gamma{\downarrow_\beta}){\downarrow_\beta} \succ (r'\gamma{\downarrow_\beta}){\downarrow_\beta}$. By the Church-Rosser property (i), $(l'\gamma{\downarrow_\beta}){\downarrow_\beta} = (l'\gamma){\downarrow_\beta}$ and $(r'\gamma{\downarrow_\beta}){\downarrow_\beta} = (r'\gamma){\downarrow_\beta}$, hence $(l'\gamma){\downarrow_\beta} \succ (r'\gamma){\downarrow_\beta}$. Since s is closed, $s[(l'\gamma){\downarrow_\beta}]_p$ is closed as well. Further, $\mathcal{V}ar((r'\gamma){\downarrow_\beta}) \subseteq \mathcal{V}ar(r'\gamma) \subseteq \mathcal{V}ar(l'\gamma)$ since $\succ$ is stable, hence $s[(l'\gamma){\downarrow_\beta}]_p$ is closed. Therefore, by monotonicity of $\succ$, $s[(l'\gamma){\downarrow_\beta}]_p \succ s[(r'\gamma){\downarrow_\beta}]_p$ and, by functionality, $s[(l'\gamma){\downarrow_\beta}]_p \succ s[(r'\gamma){\downarrow_\beta}]_p{\downarrow_\beta}$. By Church-Rosser property (i) again $s[(l'\gamma){\downarrow_\beta}]_p \succ s[r'\gamma]_p{\downarrow_\beta}$. Finally, by normal η-compatibility, $s \succ t' =_\eta s[r'\gamma]_p{\downarrow_\beta} =_\eta t$ where t' is β-normal and we are done. □

The actual meaning of this result is that rules should be checked with the restriction $>$ of the higher-order reduction ordering $\succ$ instead of with $\succ$ itself.

This result improves over [Jouannaud and Rubio 2006] in two different ways: it applies to all forms of plain higher-order rewriting, independently of the orientation

of η for normalizing terms, and of the type of the rules, basic or functional; by using simpler properties, it requires simpler arguments to justify the construction of normal higher-order orderings, making it an easier task. Further, lefthand sides of rules need not be patterns, an assumption that only becomes necessary to check the Church-Rosser property of higher-order rewrite rules via the computation of their higher-order critical pairs [Nipkow 1991; Jouannaud and Li 2012].

4. THE NORMAL HIGHER-ORDER RECURSIVE PATH ORDERING

Our purpose here is to define a particular normal ordering $(>, \succ)$. There are several possible candidates for $\succ$, notably HORPO [Jouannaud and Rubio 2007], the computability closure ordering CCO [Blanqui 2007], the computability path ordering CPO [Blanqui et al. 2008] and the recursive path and polynomial ordering RPPO [Bofill et al. 2012].

Our choice here for $\succ$ is the higher-order recursive path ordering HORPO, and therefore, in the remainder of this section, $\succ$ denotes the ordering HORPO. The reason is that HORPO is the simplest one, but we believe that our techniques would apply to CPO and CCO (and probably RPPO) as well. As all these orderings, HORPO is not an appropriate answer per se, that is, the pair $\langle \succ, \succ \rangle$ is not a normal ordering:

(i) HORPO does not satisfy normal η-compatibility: we shall enforce it easily by comparing η-normal forms of terms in the first place.

(ii) HORPO does not satisfy normal stability, in particular because of applications $@(X, v)$ which become a redex if X is instantiated by an abstraction. We shall forbid such cases.

Definition 4.1. A β-normal term $u = f(\overline{u})$ with $f \in \mathcal{F} \cup \{@, \lambda\}$ is *non-versatile* if $u\gamma{\downarrow} = f((\overline{u}\gamma){\downarrow})$ for all normal substitutions γ. Other β-normal terms are *versatile*.

Variables are versatile. Here is a sufficient syntactic condition for non-versatility:

LEMMA 4.2. *The following terms are non-versatile: (i) β-normal $\mathcal{F}$-headed terms; (ii) β-normal typable applications $@(v, w)$ with non-versatile v; (iii) $\beta\eta$-normal typable abstractions $\lambda x.v$ with non-versatile application-headed subterms.*

This lemma provides an inductive way of checking non-versatility, which terminates since terms it applies recursively to are of decreasing size, and is stronger than the definition of non-versatility in [Jouannaud and Rubio 2006].

PROOF. We consider the three possible cases in turn. The case of β-normal $\mathcal{F}$-headed term is straightforward.

Let $u = @(v, v')$. Since u is β-normal, v cannot be an abstraction. And since v is non-versatile, then $(v\gamma){\downarrow}$ is not an abstraction either, concluding the proof.

Let now $u = \lambda x.v$, assuming without loss of generality that $x \notin \mathcal{D}om(\gamma)$. In this case we prove the stronger property that for every term $\beta\eta$-normal term t such that all its subterms headed by application are non-versatile, we have that $t\gamma{\downarrow} = t(\gamma{\downarrow})$.

We proceed by induction on the size of t. If t is a variable we are done. Otherwise, if $t = f(\bar{t})$ with $f \in \mathcal{F}$, we have $t\gamma\downarrow = f((\bar{t}\gamma)\downarrow)$ and we can conclude by induction. If $t = @(t_1, t_2)$, since it is non-versatile, we have $t\gamma\downarrow = @(t_1\gamma\downarrow, t_2\gamma\downarrow)$, and again we conclude by induction hypothesis. Finally, if $t = \lambda x.u$, then, by induction hypothesis, we have that $u\gamma\downarrow = u(\gamma\downarrow)$. Hence, since $t\gamma = \lambda x.(u\gamma)$ and t is in $\beta\eta$-normal form, we have that $\lambda x.u(\gamma\downarrow)$ is $\beta\eta$-normal form. □

Fortunately, a simple restriction $>$ of $\succ$ suffices to take care of versatile terms.

We assume given: a partition $Mul \uplus Lex$ of $\mathcal{F}$; a quasi-ordering $\geq_{\mathcal{F}}$ on $\mathcal{F}$, called the *precedence*, such that $>_{\mathcal{F}}$ is well-founded; a quasi-ordering $\geq_{\mathcal{T}_S}$ on types, which is taken here to be a recursive path ordering RPO on the type structure generated by $\geq_{\mathcal{F}}$ modified so as to satisfy all conditions given in [Jouannaud and Rubio 2007][1]: $\rightarrow$ cannot be bigger than or equal to sorts, and the subterm and status cases must be restricted appropriately for $\rightarrow$. Here is our restriction of HORPO:

Definition 4.3 (*NHORPO*). Given two terms, $s : \sigma$ and $t : \tau$, we define:
$s{>}t$ iff s is normal, non-versatile , $\sigma{\geq_{\mathcal{T}_S}}\tau$ and

(1) $s = f(\overline{s})$ with $f \in \mathcal{F}$, and $u \geq t$ for some $u \in \overline{s}$
(2) $s = f(\overline{s})$, $t = g(\overline{t})$ with $f >_{\mathcal{F}} g$, and A
(3) $s = f(\overline{s})$, $t = g(\overline{t})$ with $f =_{\mathcal{F}} g \in Mul$, and $\overline{s}\,(>)_{mul}\,\overline{t}$
(4) $s = f(\overline{s})$, $t = g(\overline{t})$ with $f =_{\mathcal{F}} g \in Lex$, $\overline{s} >_{lex} \overline{t}$, and A
(5) $s = f(\overline{s})$, $t = @(u, v)$, and A
(6) $s = f(\overline{s})$, $t = \lambda x.v$, $x \notin \mathcal{V}ar(v)$ and $s > v$
(7) $s = @(u, v)$ and $w \geq t$ for some $w \in \{u, v\}$
(8) $s = @(u, v)$, $t = @(u', v')$, and $\{u, v\} >_{mul} \{u', v'\}$
(9) $s = \lambda x : \sigma.u$ and $u\{x \mapsto z\} \geq t$ with z fresh
(10) $s = \lambda x : \alpha.u$, $t = \lambda x : \beta.v$, $\alpha{=}\beta$ and $u{>}v$
(11) $s = \lambda x.u$, t is not a variable nor an abstraction, and $u\{x \mapsto z\}{\geq}@(t, z)$, z fresh

where
- $A = \forall v \in \bar{t}\ \ s{>}v$ or $u{\geq}v$ for some $u \in \overline{s}$
- $s \geq t$ iff $s > t$ or $s \equiv t$ where
- $\equiv$ is the congruence on typed terms generated by the following axiom schemas (typed in an appropriate environment):

$$
\begin{array}{lll}
f(\overline{x}) = g(\overline{x}) & & f =_{\mathcal{F}} g, f \in Lex, g \in Lex \\
f(\overline{x}) = g(\xi(\overline{x})) & f =_{\mathcal{F}} g, f \in Mul, g \in Mul, & \xi \text{ a permutation} \\
\lambda x.u = \lambda z.u\{x \mapsto z\} & & z \notin \mathcal{V}ar(\lambda x.u)
\end{array}
$$

Although $>$ is defined on arbitrary terms, the computation proceeds only in case s is normal. Note that s is non-versatile in Cases (1) to (6). It is therefore sufficient to check the non-versatility requirement for s in cases (7) to (11) only.

[1] This is Definition 5.1 there. The proof of transitivity is missing, but follows the same scheme as that of RPO [Dershowitz 1982].

Besides forbidding versatile terms in lefthand sides of a comparison, we also abandon flattening of the lefthand sides as in [Jouannaud and Rubio 2007], which makes the obtained order slightly less powerful but simpler. On the other hand, Case 11 enhances NHORPO's strength by replacing η-reduction with a combination of Cases 10 and 9 (called the middle term trick in [Jouannaud and Rubio 2007]). This is indeed needed to ensure that recursive calls apply to β-normal terms. On the other hand, the equivalence associated to $\geq$ is the same as the one for $\succeq$.

We now define the pair $(>_\eta, \succ_\eta)$ using an extra ordering $\succ_\mathcal{T}$:

Definition 4.4. Given two typable terms $s : \sigma$ and $t : \tau$, we define:

— $s : \sigma \succ_\mathcal{T} t : \tau$ if $s \succ t$ and $\sigma = \tau$.
— $s \succ_\eta t$ iff $s{\downarrow_\eta} \succ^+_\mathcal{T} t{\downarrow_\eta}$ and if $s{\downarrow_\eta} = @(u, x)$ and x is a free variable such that $x \notin \mathcal{V}ar(u)$ then $u \succ_\eta \lambda x.t$.
— $s : \sigma >_\eta t : \tau$ iff $s{\downarrow_\eta} > t{\downarrow_\eta}$, $\sigma = \tau$ and if $s{\downarrow_\eta} = @(u, v)$ then either v is non-versatile or v is a variable x and $u >_\eta \lambda x.t$.

We use $\succ^+_\mathcal{T}$ instead of $\succ^+$, which would suffice for transitivity, because we need $\succ_\eta$ to be also monotonic, and $\succ$ is only monotonic on terms with the same type.

Note that we could make $>_\eta$ marginally stronger by replacing $u >_\eta \lambda x.t$ by $u \succ_\eta \lambda x.t$. We prefer having this stand-alone definition at no practical cost.

LEMMA 4.5. *Let s', s, t and t' be terms. Then $s' =_\eta s \succ_\eta t =_\eta t'$ implies $s' \succ_\eta t'$.*

PROOF. We proceed by induction on the size of s. Since $s \succ_\eta t$, $s'{\downarrow_\eta} = s{\downarrow_\eta} \succ^+_\mathcal{T} t{\downarrow_\eta} = t'{\downarrow_\eta}$. In case $s'{\downarrow_\eta} = @(u, x)$ with $x \notin \mathcal{V}ar(u)$, then $s{\downarrow_\eta} = @(u, x)$ with $x \notin \mathcal{V}ar(u)$, hence $u \succ_\eta \lambda x.t$ by definition of $s \succ_\eta t$. Now, $\lambda x.t =_\eta \lambda x.t'$, hence, by induction hypothesis, $u \succ_\eta \lambda x.t'$. □

LEMMA 4.6. *$\succ_\eta$ is a plain higher-order reduction ordering of the set of terms.*

PROOF. The relation $\succ_\eta$ is clearly a well founded relation of the set of terms, since so is $\succ$. Coherence is clear. We are left with four properties proved in turn.

Monotonicity: if $s : \sigma \succ_\eta t : \sigma$ then $u[s] \succ_\eta u[t]$ for all u, s and t. We proceed by induction on the size of u, using monotonicity of HORPO [Jouannaud and Rubio 2007]. If u is empty we are done otherwise there are four cases:

(1) $u[] = f(\ldots u'[] \ldots)$ with $f \in \mathcal{F}$. By induction hypothesis, $u'[s] \succ_\eta u'[t]$, hence $u'[s]{\downarrow_\eta} \succ^+_\mathcal{T} u'[t]{\downarrow_\eta}$. The result follows by monotonicity of HORPO.
(2) $u[] = @(u'[], v)$. By induction hypothesis, $u'[s] \succ_\eta u'[t]$, hence $u'[s]{\downarrow_\eta} \succ^+_\mathcal{T} u'[t]{\downarrow_\eta}$. Therefore $@(u'[s], v){\downarrow_\eta} = @(u'[s]{\downarrow_\eta}, v{\downarrow_\eta}) \succ^+_\mathcal{T} @(u'[t]{\downarrow_\eta}, v{\downarrow_\eta}) = @(u'[t], v){\downarrow_\eta}$, by monotonicity of HORPO. Assume $@(u'[s], v){\downarrow_\eta} = @(u'[s]{\downarrow_\eta}, v{\downarrow_\eta})$ with $v{\downarrow_\eta} = x \notin \mathcal{V}ar(u'[s]{\downarrow_\eta}) \subseteq \mathcal{V}ar(u'[t]{\downarrow_\eta})$ by stability of HORPO. We then further need to show that $u'[s]{\downarrow_\eta} \succ_\eta \lambda x.@(u'[t], v)$. This follows from the fact that $u'[s] \succ_\eta u'[t] =_\eta \lambda x.@(u'[t], x)$ and Lemma 4.5.

(3) $u[] = @(v, u'[])$. This case is similar to the first part of the previous case.
(4) $u[] = \lambda x.u'[]$. By induction hypothesis, $u'[s] \succ_\eta u'[t]$, hence $u'[s]{\downarrow_\eta} \succ^+_{\mathcal{T}} u'[t]{\downarrow_\eta}$. By Church-Rosser property (ii), $(\lambda x.u'[s]){\downarrow_\eta} = (\lambda x.u'[s]{\downarrow_\eta}){\downarrow_\eta}$. We get two cases:
— $(\lambda x.u'[s]\ {\downarrow_\eta})\ {\downarrow_\eta} = \lambda x.(u'[s]\ {\downarrow_\eta})$.By monotonicity of HORPO, we have $\lambda x.(u'[s]{\downarrow_\eta}) \succ^+_{\mathcal{T}} \lambda x.(u'[t]{\downarrow_\eta})$, hence $(\lambda x.u'[s]{\downarrow_\eta}){\downarrow_\eta} \succ^+_{\mathcal{T}} (\lambda x.u'[t]{\downarrow_\eta}){\downarrow_\eta}$, since HORPO includes η-reductions, which preserve types, and we are done.
— $u'[s{\downarrow_\eta}] = @(u, x)$, where x is a free variable such that $x \notin \mathcal{V}ar(u'[s]{\downarrow_\eta})$ and $(\lambda x.u'[s]{\downarrow_\eta}){\downarrow_\eta} = u$. Then $u \succ_\eta \lambda x.u'[t]$, and since $\lambda x.u'[t] =_\eta (\lambda x.u'[t]{\downarrow_\eta}){\downarrow_\eta}$, then $u \succ_\eta (\lambda x.u'[t]{\downarrow_\eta}){\downarrow_\eta}$ by Lemma 4.5, and we are done again

Transitivity: $s \succ_\eta t \succ_\eta u$ implies $s \succ_\eta u$. The proof is by induction on the size of s. By definition $s \downarrow_\eta \succ^+_{\mathcal{T}} t \downarrow_\eta \succ^+_{\mathcal{T}} u \downarrow_\eta$, hence $s \downarrow_\eta \succ^+_{\mathcal{T}} u \downarrow_\eta$. Assuming $s{\downarrow_\eta} = @(w, x)$ with $x \notin \mathcal{V}ar(w)$, then $w \succ_\eta \lambda x.t$. By monotonicity, $\lambda x.t \succ_\eta \lambda x.u$. Hence $w \succ_\eta \lambda x.t \succ_\eta \lambda x.u$. By induction hypothesis, $w \succ_\eta \lambda x.u$, and we are done.

Stability follows from the property, given s and γ, that $s\gamma{\downarrow_\eta} = (s{\downarrow_\eta})(\gamma{\downarrow_\eta})$.

Functionality: assuming $s \succ_\eta t \longrightarrow_\beta u$, we prove that $s \succ_\eta u$ by induction on the size of s. By definition, $s{\downarrow_\eta} \succ^+_{\mathcal{T}} t{\downarrow_\eta}$ and $t \longrightarrow_\beta u$. By Church-Rosser property (iii), $t{\downarrow_\eta} \longrightarrow^*_{\beta\eta} u{\downarrow_\eta}$, and since HORPO includes β- and η-reductions [Jouannaud and Rubio 2007], and both preserve types, then $s{\downarrow_\eta} \succ^+_{\mathcal{T}} u{\downarrow_\eta}$. Assume that $s{\downarrow_\eta} = @(w, x)$ with $x \notin \mathcal{V}ar(w)$. Then $w \succ_\eta \lambda x.t$. Therefore $w \succ_\eta \lambda x.t \longrightarrow_\beta \lambda x.u$, hence, by induction hypothesis, $w \succ_\eta \lambda x.u$ and we are done. □

THEOREM 4.7. *The pair $(>_\eta, \succ_\eta)$ is a normal higher-order reduction ordering.*

PROOF. Using Lemma 4.6, we are left proving normal η-compatibility, which follows from Lemma 4.5, and normal stability: for all β-normal terms $s : \sigma, t : \tau$ and β-normal substitutions γ, $s : \sigma >_\eta t : \tau$ implies $(s\gamma){\downarrow_\beta} : \sigma \succ_\eta (t\gamma){\downarrow_\beta} : \tau$. We proceed by induction on the size of s. By definition of $\succ_\eta$, this requires $(s\gamma){\downarrow_\beta}{\downarrow_\eta} \succ^+_{\mathcal{T}} (t\gamma){\downarrow_\beta}{\downarrow_\eta}$ and if $(s\gamma){\downarrow_\beta}{\downarrow_\eta} = @(u', y)$ and y is a free variable such that $y \notin \mathcal{V}ar(u')$ then $u' \succ_\eta \lambda y.((t\gamma){\downarrow_\beta})$. We first prove the second part.

Assume $(s\gamma){\downarrow_\beta}{\downarrow_\eta} = @(u', y)$ and y is a free variable such that $y \notin \mathcal{V}ar(u')$. By Church-Rosser properties (ii)-(iv), we have that $(s{\downarrow_\eta}\ \gamma){\downarrow} = (s\gamma){\downarrow_\beta}{\downarrow_\eta}$, and since $s{\downarrow_\eta} > t{\downarrow_\eta}$ implies that $s{\downarrow_\eta}$ is non-versatile, we have that if $(s{\downarrow_\eta}\ \gamma){\downarrow} = @(u', y)$ then $s{\downarrow_\eta} = @(u, v)$, with $(u\gamma){\downarrow} = u'$ and $(v\gamma){\downarrow} = y$. Moreover, since $s >_\eta t$ and $s{\downarrow_\eta} = @(u, v)$, we have that either v is non-versatile or v is a variabe x and $u >_\eta \lambda x.t$. Since $(v\gamma){\downarrow} = y$, we have that v cannot be non-versatile, and hence v must be a free variable x and $x(\gamma{\downarrow}) = y$, which, by Church-Rosser property (iii), implies that $x(\gamma{\downarrow_\beta}) =_\eta y$, and hence, by confluence, $(t\{x \mapsto y\}\gamma){\downarrow_\beta} =_\eta (t\gamma){\downarrow_\beta}$. Then, by induction hypothesis, $(u\gamma){\downarrow_\beta} \succ_\eta ((\lambda x.t)\gamma){\downarrow_\beta}$, which is, by α-conversion, equal to $((\lambda y.t\{x \mapsto y\})\gamma)\,{\downarrow_\beta} = (\lambda y.(t\{x \mapsto y\}\gamma))\,{\downarrow_\beta} = \lambda y.((t\{x \mapsto y\}\gamma)\,{\downarrow_\beta}) =_\eta \lambda y.((t\gamma){\downarrow_\beta})$, and by η-compatibility, $(u\gamma){\downarrow_\beta}{\downarrow_\eta} \succ_\eta \lambda y.((t\gamma){\downarrow_\beta})$.

We are left with proving that $(s\gamma)\downarrow_\beta\downarrow_\eta \succ^+_\mathcal{T} (t\gamma)\downarrow_\beta\downarrow_\eta$, that is $(s\gamma)\downarrow \succ^+_\mathcal{T} (t\gamma)\downarrow$ under the assumption that $s\downarrow_\eta > t\downarrow_\eta$ and $\sigma = \tau$. Since $(s\gamma)\downarrow\colon \sigma$ and $(t\gamma)\downarrow\colon \tau$ and, by confluence, $(s\gamma)\downarrow = (s\downarrow_\eta \gamma)\downarrow$ and $(t\gamma)\downarrow = (t\downarrow_\eta \gamma)\downarrow$, we can prove $(s\gamma)\downarrow\colon \sigma \succ^+_\mathcal{T} (t\gamma)\downarrow\colon \sigma$ under the assumption that $s : \sigma > t\sigma$ for normal terms s and t. Additionally, since HORPO includes β- and η-reductions [Jouannaud and Rubio 2007] and both preserve types, it suffices to show the following property:
(i) $s > t$ implies $(s\gamma)\downarrow \succ w$ for some w such that $t\gamma \longrightarrow^*_{\beta\eta} w$. Showing (i) will need
(ii) $s \equiv t$ implies $(s\gamma)\downarrow \equiv (s\gamma)\downarrow$. Further, (i) and (ii) both together imply
(iii) $s \geq t$ implies $(s\gamma)\downarrow \succeq w$ for some w such that $t\gamma \longrightarrow^*_{\beta\eta} w$.

We prove (ii) by induction on the size of s. Assume $s \equiv t$. There are four cases.

— $s = t$. Then $(s\gamma)\downarrow = (t\gamma)\downarrow$ and we are done.
— $s = f(\overline{s})$, $t = g(\overline{t})$, $f =_\mathcal{F} g$, $f \in Lex$, $g \in Lex$ and $\overline{s} \equiv \overline{t}$. Then, by induction hypothesis, $(\overline{s}\gamma)\downarrow \equiv (\overline{t}\gamma)\downarrow$ and therefore $(s\gamma)\downarrow = f((\overline{s}\gamma)\downarrow) \equiv g((\overline{s}\gamma)\downarrow) = (t\gamma)\downarrow$.
— $s = f(\overline{s})$, $t = g(\xi(\overline{s}))$, $f =_\mathcal{F} g$, $f \in Mul$, $g \in Mul$, ξ a permutation. Similarly, $(s\gamma)\downarrow = f((\overline{s}\gamma)\downarrow) \equiv g(\xi(\overline{s}\gamma)\downarrow) = (t\gamma)\downarrow$.
— $s = \lambda x.u$, $t = \lambda z.v\{x \mapsto z\}$ with $z \notin \mathcal{V}ar(\lambda x.v)$, and $u \equiv v$. By induction hypothesis, $(u\gamma)\downarrow \equiv (v\gamma)\downarrow$, and since $z \notin \mathcal{V}ar(\lambda x.(v\gamma)\downarrow)$, it follows that $(s\gamma)\downarrow = \lambda x.(u\gamma)\downarrow \equiv \lambda z.(v\gamma)\downarrow\ \{x \mapsto z\} = \lambda z.(v\{x \mapsto z\}\gamma)\downarrow = (t\gamma)\downarrow$.

We now prove (i) by induction on the pair $\langle s, t\rangle$ ordered lexicographically by size. There are 11 cases according to the definition of NHORPO:

(1) $s = f(\overline{s})$ with $f \in \mathcal{F}$, and $u \geq t$ for some $u \in \overline{s}$. By, induction hypothesis (i) and property (ii), $(u\gamma)\downarrow \succeq w$ for some w such that $t\gamma \longrightarrow^*_{\beta\eta} w$, and since $(f(\overline{s})\gamma)\downarrow = f((\overline{s}\gamma)\downarrow)$, we conclude by Case 1 of HORPO.
(2) $s = f(\overline{s})$, $t = g(\overline{t})$ with $f >_\mathcal{F} g$, and for all $v_i \in \overline{t}$, $s > v_i$ or $u \geq v_i$ for some $u \in \overline{s}$. By induction hypothesis (i) and property (ii), for all $v_i \in \overline{t}$, $(s\gamma)\downarrow \succ w_i$ or $(u\gamma)\downarrow \succeq w_i$ for some w such that $v_i\gamma \longrightarrow^*_{\beta\eta} w_i$. Since $(f(\overline{s})\gamma)\downarrow = f((\overline{s}\gamma)\downarrow)$, then $(s\gamma)\downarrow \succ g(w_1\gamma, \ldots, w_i) = w$ by Case 2 of HORPO. Since $t\gamma = g(v_1\gamma, \ldots, v_n\gamma) \longrightarrow^*_{\beta\eta} w$, we are done with this case.
(3) $s = f(\overline{s})$, $t = g(\overline{t})$ with $f =_\mathcal{F} g \in Mul$, and $\overline{s}\,(>)_{mul}\,\overline{t}$. Then, by induction hypothesis on (i) and property (ii), $(\overline{s}\gamma)\downarrow\ (\succ)_{mul}\ \overline{w}$ for some $\overline{w}$ such that $\overline{t} \longrightarrow^*_{\beta\eta} \overline{w}$. We conclude by Case 3 of HORPO.
(4) $s = f(\overline{s})$, $t = g(\overline{t})$ with $f =_\mathcal{F} g \in Lex$, $\overline{s} >_{lex} \overline{t}$, and for all $v_i \in \overline{t}$, then $s > v_i$ or $u \geq v_i$ for some $u \in \overline{s}$. This case is similar to Case 2.
(5) $s = f(\overline{s})$, $t = @(t_1, t_2)$, and for all t_i, then $s > t_i$ or $u \geq t_i$ for some $u \in \overline{s}$. By induction hypothesis (i) and property (ii), for all t_i, then $(s\gamma)\downarrow \succ w_i$ or $(u\gamma)\downarrow \succeq w_i$ for some w_i such that $t_i \longrightarrow^*_{\beta\eta} w_i$. We conclude thanks to Case 5 of HORPO.
(6) $s = f(\overline{s})$, $t = \lambda x.v$, $x \notin \mathcal{V}ar(v)$ and $s > v$. By induction hypothesis (i), $(s\gamma)\downarrow \succ w'$ with $v\gamma \longrightarrow^*_{\beta\eta} w'$. Since, $(f(\overline{s})\gamma)\downarrow = f((\overline{s}\gamma)\downarrow)$, we conclude as in case 3.

(7) $s = @(u, v)$ is non-versatile and $s' \geq t$ for some $s' \in \{u, v\}$. Since s is non-versatile then $(s\gamma)\downarrow = @((u\gamma)\downarrow, (v\gamma)\downarrow)$, and we conclude as in Case 3.

(8) $s = @(u, v)$, $t = @(u', v')$, and $\{u, v\} >_{mul} \{u', v'\}$. Since s is non-versatile, then $(s\gamma)\downarrow = @((u\gamma)\downarrow, (v\gamma)\downarrow)$ and we conclude as in Case 3.

(9) $s = \lambda x : \sigma.u$ and $u\{x \mapsto z\} \geq t$ with z fresh. Since s is non-versatile, then $(s\gamma)\downarrow = \lambda x.((u\gamma)\downarrow)$. By, induction hypothesis (i) and property (ii), $(u\{x \mapsto z\}\gamma)\downarrow \succeq w$ with $t\gamma \longrightarrow^*_{\beta\eta} w$, and, since z is fresh, $(u\{x \mapsto z\}\gamma)\downarrow = ((u\gamma)\downarrow)\{x \mapsto z\}$. We conclude as in case 3.

(10) $s = \lambda x : \alpha.u$, $t = \lambda x : \beta.v$, $\alpha = \beta$ and $u > v$. Since s is non-versatile, then $(s\gamma)\downarrow = \lambda x.((u\gamma)\downarrow)$. By induction hypothesis (i), $(u\gamma)\downarrow \succeq w'$ for some w' such that $v\gamma \longrightarrow^*_{\beta\eta} w'$, and we conclude as in Case 3.

(11) $s = \lambda x.u$, t is not an abstraction and $u\{x \mapsto z\} \geq @(t, z)$, z fresh. Since s is non-versatile, then $(s\gamma)\downarrow = \lambda x.((u\gamma)\downarrow)$. By induction hypothesis (i) and property (ii), $(u\{x \mapsto z\}\gamma)\downarrow \succeq w'$ with $@(t, z)\gamma \longrightarrow^*_{\beta\eta} w'$. Since z is fresh, $(u\{x \mapsto z\}\gamma)\downarrow = (u\gamma)\downarrow \{x \mapsto z\}$. Since t is not a variable nor an abstraction, $w' = @(w, z)$ and $t\gamma \longrightarrow^*_{\beta\eta} w$. Therefore, $(u\gamma)\downarrow \{x \mapsto z\} \succeq @(w, z)$, which implies by monotonicity and α-conversion $\lambda x.(u\gamma)\downarrow \succeq \lambda x.@(w, x)$. Since HORPO includes η-reduction, we get $(s\gamma)\downarrow = \lambda x.(u\gamma)\downarrow \succ w$. We assume here that this can be done with HORPO in one step, which could be achieved by adding the necessary case without modifying HORPO as an order. □

We can therefore use $(>_\eta, \succ_\eta)$ as our normal higher-order ordering for checking whether a given higher-order rewrite system R is terminating. However, note that the lefthand side l of a rule $l \to r \in R$ must be in β-normal form in order to use NHORPO via $\succ_\eta$, since NHORPO needs lefthand sides of comparisons to be normal to have a chance of success, which will be the case by η-postponement.

We strongly believe that the very same changes as done here work equally well for similarly defined higher-order reduction orderings such as CPO [Blanqui et al. 2008], with a similar proof as for HORPO, although details have not checked.

Example 4.8 (*differentiation continued*). Symbols have multiset status, sin and cos are equal in the precedence, diff is bigger than all others. We consider only the first rule. Our goal is: $\mathsf{diff}(\lambda\mathsf{x}.\,\mathsf{sin}(@(\mathsf{F}, \mathsf{x}))) > \lambda\mathsf{x}.\,\mathsf{cos}(@(\mathsf{F}, \mathsf{x})) \times \mathsf{diff}(\mathsf{F})$, which, by Case 2, yields two subgoals:

Subgoal 1: $\mathsf{diff}(\lambda\mathsf{x}.\,\mathsf{sin}(@(\mathsf{F}, \mathsf{x}))) > \lambda\mathsf{x}.\,\mathsf{cos}(@(\mathsf{F}, \mathsf{x}))$ which succeeds by Case 1 since $\lambda\mathsf{x}.\,\mathsf{sin}(@(\mathsf{F}, \mathsf{x})) = \lambda\mathsf{x}.\,\mathsf{cos}(@(\mathsf{F}, \mathsf{x}))$, sin and cos having the same precedence.

Subgoal 2: $\mathsf{diff}(\lambda\mathsf{x}.\,\mathsf{sin}(@(\mathsf{F}, \mathsf{x}))) > \mathsf{diff}(\mathsf{F})$ which, by Case 3 yields

Subgoal 3: $\lambda\mathsf{x}.\,\mathsf{sin}(@(\mathsf{F}, \mathsf{x})) > \mathsf{F}$. We cannot apply Case 9 here as one might have expected, the type of F is too large. We use instead Case 11, which yields:

Subgoal 4: $\mathsf{sin}(@(\mathsf{F}, \mathsf{x})) > @(\mathsf{F}, \mathsf{x})$ which succeeds by Case 1.

Since $\langle \succ_\eta, > \rangle$ is a normal higher-order reduction ordering, we conclude normal termination of (the first rule of) our example by using theorems 3.8 and 4.7 together.

Many examples cannot be treated by normal HORPO, because the only rule able to handle abstractions on the righthand side are the weak Rule 6 and the monotonicity Rule 10. Some of these examples would work with normal CPO, which was designed to remedy this very weakness of HORPO. This question is indeed at the heart of the next section, where a powerful technique is presented that makes it possible to solve complex goals, even when CPO itself would fail.

5. INTERPRETATIONS FOR NORMAL HIGHER-ORDER REWRITING

The profusion of abstractions on both the lefthand and righthand sides of rules in normal higher-order rewrite rules is a challenge for termination proofs, even when using CPO instead of HORPO. This is the question addressed in this section. To this end, we introduce *neutrlization*, a particular higher-order interpretation aimed at eliminating abstractions which cannot originate reductions. Higher-order term interpretations that preserve termination of higher-order rewriting are studied first, making no assumption whatsoever on the method used for carrying out such a termination proof. On the other hand, we make an assumption on higher-order rules: their lefthand side is either of basic type or headed by an algebraic function symbol.

Definition 5.1. A higher-order interpretation is a mapping from higher-order terms to higher-order terms which is the identity on free variables and preserves types in a given environment: $(\forall \Gamma, u, \sigma)$ such that $\Gamma \vdash_\Sigma u : \sigma$, then $\Gamma \vdash_\Sigma I(u) : \sigma$.

Higher-order interpretations are extended to terms, substitutions, rewrite rules, and rewrite systems in the natural way.

The role of our assumption that types are preserved ensures that term manipulations in interpreted terms preserve typability. More general assumptions are of course possible, provided they achieve the same property and ensure that if two terms the same type σ, then their interpretations have the same type τ for some τ.

Definition 5.2. A higher-order interpretation $I : \mathcal{T}(\mathcal{F}, \mathcal{X}) \to \mathcal{T}(\mathcal{F}', \mathcal{X})$ is *normal* if it satisfies the following properties:

- β-*compatibility*: $s =_\beta t$ implies $I(s) =_\beta I(t)$ for every two typed terms s and t ;
- η-*compatibility*: $s =_\eta t$ implies $I(s) =_\eta I(t)$ for every two typed terms s and t ;
- preservation of β-normal forms: $I(t)$ is β-normal whenever t is ;
- preservation of algebraic head: $I(t)$ is $\mathcal{F}'$-headed if t is $\mathcal{F}$-headed ;
- *monotonicity*: $\forall \beta$-normal u, $\forall p \in \mathcal{P}os(u)$, there exists a set $\{q_1, \ldots, q_n\}_{n>0}$ of disjoint positions in $I(u)$ such that, $\forall \beta$-normal t, $I(u[t]_p) =_{\beta\eta} I(u)[I(t)]_{q_1 \ldots q_n}$;
- *stability*: for all β-normal term t and substitution γ, $I(t\gamma) =_{\beta\eta} I(t)I(\gamma)$.

LEMMA 5.3. *Let t and u be higher-order terms. Then, for every set $\{p_1 \ldots p_m\}$ of disjoint positions in $\mathcal{P}os(t)$ there is a set $\{q_1, \ldots, q_n\}$ of disjoint positions in $\mathcal{P}os(t{\downarrow_\beta})$ such that $t[u]_{p_1 \ldots p_m}{\downarrow_\beta} = (t{\downarrow_\beta}\ [u{\downarrow_\beta}]_{q_1,\ldots,q_n}){\downarrow_\beta}$. Further, if u has a basic type or is headed by an algebraic symbol, then $t[u]_{p_1 \ldots p_m}{\downarrow_\beta} = t{\downarrow_\beta}\ [u{\downarrow_\beta}]_{q_1 \ldots q_n}$.*

PROOF. Follows from the Church-Rosser property of β-reductions, and the fact that $u{\downarrow_\beta}$ has a basic type or is headed by an algebraic symbol if this is true of u. □

LEMMA 5.4. *Let u be a β-normal term, R a terminating higher-order rewrite system whose lefthand sides are either headed by an algebraic symbol or of base type, $\overline{p}$ be a non-empty sequence of disjoint positions of $\mathcal{P}os(u)$, and $l \to r$ a rule in R such that $(\forall p \in \overline{p})\, u|_p =_{\beta\eta} l\gamma$. Then $u \longrightarrow_R^+ u[r\gamma]_{\overline{p}}{\downarrow_\beta}$.*

PROOF. The difficulty is that higher-order rewriting does not satisfy the so-called disjoint redex axiom because of the β-normalization which applies at each step, but all β-normalization steps can indeed be postponed and grouped together.

Since l hence $l\gamma$ is not an abstraction and $u|_p$ is β-normal and it is not an abstraction either, $u =_\eta u[z]_{\overline{p}}\{z \mapsto l\gamma{\downarrow_\beta}\}$ where z is a fresh variable, $\overline{p}$ is a set of disjoint positions in $\mathcal{P}os(u[z]_{\overline{p}})$ and $u[z]_{\overline{p}}$ is β-normal. We prove that $u =_\eta u[z]_{\overline{p}}\{z \mapsto l\gamma{\downarrow_\beta}\} \longrightarrow_R^+ (u[z]_{\overline{p}}\{z \mapsto r\gamma\}){\downarrow_\beta}$ by induction on u compared in the well-founded order $\longrightarrow_R^+$.

If $|\overline{p}| = 1$, the result is clear. Otherwise, $\overline{p} = p, \overline{q}$. We choose to rewrite p first: $u \longrightarrow_R^p v =_\eta (u[z]_p\{z \mapsto r\gamma\}[z]_{\overline{q}}\{z \mapsto l\gamma{\downarrow_\beta}\}){\downarrow_\beta} = (u[r\gamma]_p[z]_{\overline{q}}\{z \mapsto l\gamma{\downarrow_\beta}\}){\downarrow_\beta}$. Because $l\gamma{\downarrow_\beta}$ is not an abstraction and is β-normal, $v =_\eta (u[r\gamma]_p[z]_{\overline{q}}){\downarrow_\beta}\ \{z \mapsto l\gamma{\downarrow_\beta}\}$, and z occurs at a non-empty set $\overline{q}$ of disjoint positions of $\mathcal{P}os(u[r\gamma]_p[z]_{\overline{q}}{\downarrow_\beta})$. By induction hypothesis, $v \longrightarrow_R^+ v' =_\eta ((u[r\gamma]_p[z]_{\overline{q}}){\downarrow_\beta}\ \{z \mapsto r\gamma\}){\downarrow_\beta}$. By confluence of β, $v' =_\eta ((u[r\gamma]_p[z]_{\overline{q}}){\downarrow_\beta}\ \{z \mapsto r\gamma\}){\downarrow_\beta} = u[z]_{p,\overline{q}}\{z \mapsto r\gamma\}{\downarrow_\beta}$ and we are done. □

THEOREM 5.5. *Let R be a normal higher-order rewrite system all whose lefthand sides are either headed by an algebraic symbol or have a basic type, and I be a normal higher-order interpretation. Then termination of $I(R)$ implies termination of R.*

PROOF. Let s be a β-normal term such that $s \longrightarrow_R t$. By definition of higher-order rewriting, $s = s{\downarrow_\beta}$, $s|_p =_{\beta\eta} l\gamma$ for some position p, β-normal substitution γ and rule $l \to r \in R$, and t is a β-normal term such that $t =_\eta s[r\gamma]_p{\downarrow_\beta}$. By definition of an interpretation, $I(s)$ and $I(t)$ are β-normal. We now show that $I(s) \longrightarrow_{I(R)}^+ I(t)$, which implies termination of R assuming termination of $I(R)$.

The assumption that interpretations preserve types ensures that all terms constructed in the coming proof are typable: environments can be dispensed with.

Since s, hence $s|_p$ is β-normal, and $s|_p =_{\beta\eta} l\gamma$, by confluence of β-reductions modulo η, $l\gamma \longrightarrow_\beta^* l\gamma{\downarrow_\beta} =_\eta s|_p$. By assumption, l is headed by an algebraic symbol or is a term of basic type, hence so are $l\gamma$ and therefore $l\gamma{\downarrow_\beta}$. It follows that $s|_p = \lambda\overline{x}.@(s|_q, \overline{x})$ and $\overline{x} \cap \mathcal{V}ar(s|_q) = \emptyset$, and either $s|_q$ has basic type or $s|_q = f(\overline{u})$. Now, $s|_q =_\eta l\gamma{\downarrow_\beta}$, hence $s \longrightarrow^q t' =_\eta (s[\lambda\overline{x}.@(r\gamma, \overline{x})]_p){\downarrow_\beta}$. Further, $s|_q$ and $l\gamma$ being $\beta\eta$-equivalent, $\mathcal{V}ar(l\gamma) = \mathcal{V}ar(s|_q)$, and therefore $\mathcal{V}ar(l\gamma) \cap \overline{x} = \emptyset$. Since $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$, $\mathcal{V}ar(r\gamma) \subseteq \mathcal{V}ar(l\gamma)$, implying that $\mathcal{V}ar(r\gamma) \cap \overline{x} = \emptyset$. It follows that $t' =_\eta (s[r\gamma]_q){\downarrow_\beta} =_\eta (s[r\gamma]_p){\downarrow_\beta} =_\eta t$.

We proceed with the calculation of $I(s)$ first and then $I(t)$, which are both β-normal since I preserves β-normal forms.

By monotonicity of I, there are disjoint positions $q_1 \dots q_n$ (depending on s and q only) such that $I(s) =_{\beta\eta} I(s)[I(s|_q)]_{q_1 \dots q_m}$. By preservation of β-normal forms $I(s)$ and $I(s|_q)$ are β-normal. Further, since $s|_q$ has basic type or is headed by an algebraic symbol, the same is true of $I(s|_q)$ by preservation of either types or algebraic symbols. Hence, by Lemma 5.3, since $I(s)$ and $I(s|_q)$ are β-normal, we have $I(s) =_\eta I(s)[I(s|_q)]_{q_1 \dots q_n}$.

By η-compatibility and β-compatibility, $I(s|_q) =_{\beta\eta} I(l\gamma)$. By stability, $I(l\gamma) =_{\beta\eta} I(l)I(\gamma)$. Hence $I(s|_q) =_{\beta\eta} I(l)I(\gamma)$. Hence $I(s) \longrightarrow^+_{I(R)} I(s)[I(r)I(\gamma)]_{q_1 \dots q_n}\downarrow_\beta$ by Lemma 5.4.

We move to $I(t)$. By successively η-compatibility and β-compatibility of I, we get $I(t) =_\eta I(s[r\gamma]_q\downarrow_\beta) =_\beta I(s\downarrow_\beta [(r\gamma)\downarrow_\beta]_q)$. Further, by monotonicity of I, $I(s\downarrow_\beta [r\gamma\downarrow_\beta]_q) =_{\beta\eta} I(s\downarrow_\beta)[I((r\gamma)\downarrow_\beta)]_{q_1,\dots,q_n}$. Finally, by stability and β-compatibility of I, it follows that $I(t) =_{\beta\eta} I(s\downarrow_\beta)[I(r)I(\gamma)]_{q_1,\dots,q_n}$. Since t is β-normal and I preserves normal forms, and β is Church-Rosser modulo η, we finally get $I(t) =_\eta I(s)[I(r)I(\gamma)]_{q_1,\dots,q_n}\downarrow_\beta$, which achieves the proof. □

5.1. Neutralization and Normalization

We present now a particular example of termination-preserving interpretation for normal rewriting, originally introduced in [Jouannaud and Rubio 2006]. The construction relies on a specific treatment, called *neutralization*, of those abstractions which should not be involved in a beta-redex, that is, which should not be or become after reductions the first argument of an application. To this end, a term built from the signature $\mathcal{F}$ is transformed into a term built from an enlarged signature $\mathcal{F}_{new}$ obtained from $\mathcal{F}$ by adding (if necessary) a minimal constant type o, a coercion symbol $m_\sigma : \sigma \to o$ for every type σ to the minimal type o, a function symbol $\bot^n_\sigma : \tau_1 \times \dots \times \tau_n \to \tau$ for every type $\sigma = \tau_1 \to \dots \to \tau_n \to \tau$, and a function symbol f_{new} for some of the function symbols in $\mathcal{F}$. We omit the superscript n in $\bot^n_\sigma$ when it is clear from the context. The higher-order rules to be proved terminating are of course built from terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, not in $\mathcal{T}(\mathcal{F}_{new}, \mathcal{X})$.

Definition 5.6. The *neutralization of level* i (*i-neutralization* for short) of a typable term $t \in \mathcal{T}(\mathcal{F}_{new}, \mathcal{X})$ with respect to the list of (typable) terms $\langle u_1 : \theta_1, \dots, u_n : \theta_n\rangle$ in $\mathcal{T}(\mathcal{F}_{new}, \mathcal{X})$, is the term $\mathcal{N}_i(t, \langle u_1, \dots, u_n\rangle)$ defined as follows:

(1) $\mathcal{N}_i(t : \tau, \langle u_1, \dots, u_n\rangle) = m_\tau(t)$ if $i < 0$;
(2) $\mathcal{N}_0(t : \tau, \langle u_1, \dots, u_n\rangle) = t$;
(3) $\mathcal{N}_{i+1}(t : \tau, \langle u_1, \dots, u_n\rangle) = t$ if τ is a data type;
(4) $\mathcal{N}_{i+1}(t : \sigma \to \tau, \langle u_1, \dots, u_n\rangle) = \mathcal{N}_i(@(t, \bot^n_{\theta_1 \to \dots \to \theta_n \to \sigma}(u_1, \dots, u_n)))$.

Terms of a functional type are neutralized by applying them to the $\bot$-expression of the appropriate type, therefore reducing their functionality as long as the level

is non-zero. The role of $\mathcal{N}_{-1}$ is specific to HORPO, we shall understand it in examples. The role of the sequence of terms $\langle u_1 : \theta_1, \ldots, u_n : \theta_n \rangle$ is to become arguments of $\perp^n_\sigma$ in case that symbol has a functional type. Note also that neutralization is the identity function if all levels are 0. Further, for each function symbol, we control which arguments of a functional type can be neutralized:

Definition 5.7. To each symbol $f : \sigma_1 \times \ldots \times \sigma_n \to \sigma \in \mathcal{F}$ and each argument position $j \in [1..n]$, we associate:

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

We now neutralize terms recursively. To this end, for each symbol $f : \sigma_1 \times \ldots \times \sigma_n \to \sigma$ such that $\sigma_i = \tau_1 \to \ldots \to \tau_k \to \tau$, where τ is a basic type, we add to the signature a new function symbol $f_{new} : \sigma'_1 \times \ldots \times \sigma'_n \to \sigma$ such that $\sigma'_i = \tau_{q+1} \to \ldots \to \tau_k \to \tau$ if $nl^i_f = q < k$ and $\sigma'_i = \tau$ if $nl^i_f = k$.

Definition 5.8. The *full neutralization* of a term t is the term $\mathcal{FN}(t)$ s.t.

$$
\begin{array}{ll}
\mathcal{FN}(x) & = x \text{ if } x \in \mathcal{X} \\
\mathcal{FN}(\lambda x.u) & = \lambda x.\mathcal{FN}(u) \\
\mathcal{FN}(@(t_1, t_2)) & = @(\mathcal{FN}(t_1), \mathcal{FN}(t_2)) \\
\mathcal{FN}(f(t_1, \ldots, t_n)) & = \\
\multicolumn{2}{r}{f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1), \mathcal{FN}(\bar{t}^{\,1}_f)), \ldots, \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n), \mathcal{FN}(\bar{t}^{\,n}_f)))}
\end{array}
$$

where $\mathcal{FN}(\langle u_1, \ldots, u_n \rangle) = \langle \mathcal{FN}(u_1), \ldots, \mathcal{FN}(u_n) \rangle$.

Our definition makes sense since, in all cases, $\mathcal{FN}(t)$ is typable with the same type as t. Note also that the filtered list of arguments $\bar{t}^{\,i}_f$ is itself recursively neutralized before using it to neutralize t_i in the last case.

Example 5.9 (*differentiation continued*). We illustrate here the full neutralization of the lefthand and righthand sides of the rules of Example 3.4. To this end, we choose a neutralization level for each function symbol and argument position. The associated subsets of argument positions are chosen empty, hence $\perp^0_{\mathsf{real}}$ is a constant abbreviated as $\perp$:

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

We can now compute the full neutralizations of both sides of the first rule:

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

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

and normalizing both sides we get

$$\mathsf{diff}_{\mathsf{new}}(\mathsf{sin}(@(\mathsf{F},\bot))) \to \mathsf{cos}(@(\mathsf{F},\bot)) \times_{\mathsf{new}} @(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot)),\bot)$$

and of the second rule:

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

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

and normalizing both sides we get

$$\begin{array}{c} \mathsf{diff}_{\mathsf{new}}(@(@(\mathsf{F},\bot)\times_{\mathsf{new}}@(\mathsf{F},\bot),\bot))\to \\ @(@(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot)),\bot)\times_{\mathsf{new}}@(\mathsf{F},\bot),\bot)+_{\mathsf{new}}@(@(\mathsf{F},\bot)\times_{\mathsf{new}}@(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot)),\bot),\bot) \end{array}$$

An example with a non-empty set of argument positions is given at Example 7.2.

5.2. Properties of neutralization

We investigate here the interactions between neutralization, monotonicity and instantiation.

LEMMA 5.10. *Let $t:\tau$ and $u_1,\ldots,u_n$ be higher-order terms and γ be a substitution. Then $\mathcal{N}_i(t\gamma,\langle u_1\gamma,\ldots,u_n\gamma\rangle)=\mathcal{N}_i(t,\langle u_1,\ldots,u_n\rangle)\gamma$.*

PROOF. We proceed by induction on i. There are three cases.

(1) If $i<0$. Then, we have $t\gamma:\tau$, and hence
$\mathcal{N}_i(t\gamma,\langle u_1\gamma,\ldots,u_n\gamma\rangle)=m_\tau(t\gamma)=m_\tau(t)\gamma=\mathcal{N}_i(t,\langle u_1,\ldots,u_n\rangle)\gamma$.
(2) If $i=0$ or τ is a data type. Then, we have $t\gamma:\tau$, and hence
$\mathcal{N}_i(t\gamma,\langle u_1\gamma,\ldots,u_n\gamma\rangle)=t\gamma=\mathcal{N}_i(t,\langle u_1,\ldots,u_n\rangle)\gamma$.
(3) Otherwise $i>0$ and $\tau=\sigma\to\rho$. Then, $\mathcal{N}_i(t\gamma,\langle u_1\gamma,\ldots,u_n\gamma\rangle)=$
$\mathcal{N}_{i-1}(@(t\gamma,\bot_\theta(u_1\gamma,\ldots,u_n\gamma)))=\mathcal{N}_{i-1}(@(t,\bot_\theta(u_1,\ldots,u_n))\gamma)$.
By induction hypothesis, this is equal to
$\mathcal{N}_{i-1}(@(t,\bot_\theta(u_1,\ldots,u_n)))\gamma=\mathcal{N}_i(t,\langle u_1,\ldots,u_n\rangle)\gamma$. □

LEMMA 5.11. *Let t be a higher-order term and γ be a substitution. Then $\mathcal{FN}(t\gamma)=\mathcal{FN}(t)\mathcal{FN}(\gamma)$.*

PROOF. We proceed by induction on $|t|$. There are four cases:

(1) t is a variable x. Then $\mathcal{FN}(x\gamma) = x\mathcal{FN}(\gamma) = \mathcal{FN}(x)\mathcal{FN}(\gamma)$.
(2) $t = \lambda x.u$. Then $\mathcal{FN}(t\gamma) = \mathcal{FN}(\lambda x.u\gamma) = \lambda x.\mathcal{FN}(u\gamma)$. By induction hypothesis, $\mathcal{FN}(u\gamma) = \mathcal{FN}(u)\mathcal{FN}(\gamma)$, and therefore $\lambda x.\mathcal{FN}(u\gamma) = \lambda x.\mathcal{FN}(u)\mathcal{FN}(\gamma) = (\lambda x.\mathcal{FN}(u))\mathcal{FN}(\gamma)$, since x is not occurring in $\mathcal{R}an(\mathcal{FN}(\gamma))$, and which is equal to $\mathcal{FN}(\lambda x.u)\mathcal{FN}(\gamma) = \mathcal{FN}(t)\mathcal{FN}(\gamma)$.
(3) $t = @(t_1, t_2)$. Then $\mathcal{FN}(t\gamma) = \mathcal{FN}(@(t_1, t_2)\gamma) = \mathcal{FN}(@(t_1\gamma, t_2\gamma)) = @(\mathcal{FN}(t_1\gamma), \mathcal{FN}(t_2\gamma))$. By induction hypothesis, it is equal to $@(\mathcal{FN}(t_1)\mathcal{FN}(\gamma), \mathcal{FN}(t_2)\mathcal{FN}(\gamma)) = @(\mathcal{FN}(t_1)\mathcal{FN}(\gamma), \mathcal{FN}(t_2)\mathcal{FN}(\gamma)) = @(\mathcal{FN}(t_1), \mathcal{FN}(t_2))\mathcal{FN}(\gamma) = \mathcal{FN}(@(t_1, t_2))\mathcal{FN}(\gamma) = \mathcal{FN}(t)\mathcal{FN}(\gamma)$

(4) $t = f(t_1, \ldots, t_n)$ with $f \in \mathcal{F}$. Then $\mathcal{FN}(t\gamma) = \mathcal{FN}(f(t_1, \ldots, t_n)\gamma) = \mathcal{FN}(f(t_1\gamma, \ldots, t_n\gamma)) =$
$f_{new}(\mathcal{N}_{nl_f^1}(\mathcal{FN}(t_1\gamma), \mathcal{FN}(\bar{t}_f^{\,1}\gamma)), \ldots, \mathcal{N}_{nl_f^n}(\mathcal{FN}(t_n\gamma), \mathcal{FN}(\bar{t}_f^{\,n}\gamma)))$.
By induction hypothesis, this is equal to
$f_{new}(\mathcal{N}_{nl_f^1}(\mathcal{FN}(t_1)\mathcal{FN}(\gamma), \mathcal{FN}(\bar{t}_f^{\,1})\mathcal{FN}(\gamma)), \ldots,$
$\mathcal{N}_{nl_f^n}(\mathcal{FN}(t_n)\mathcal{FN}(\gamma), \mathcal{FN}(\bar{t}_f^{\,n})\mathcal{FN}(\gamma)))$, and, by Lemma 5.10, to
$f_{new}(\mathcal{N}_{nl_f^1}(\mathcal{FN}(t_1), \mathcal{FN}(\bar{t}_f^{\,1}))\mathcal{FN}(\gamma), \ldots, \mathcal{N}_{nl_f^n}(\mathcal{FN}(t_n), \mathcal{FN}(\bar{t}_f^{\,n}))\mathcal{FN}(\gamma)) =$
$f_{new}(\mathcal{N}_{nl_f^1}(\mathcal{FN}(t_1), \mathcal{FN}(\bar{t}_f^{\,1})), \ldots, \mathcal{N}_{nl_f^n}(\mathcal{FN}(t_n), \mathcal{FN}(\bar{t}_f^{\,n})))\mathcal{FN}(\gamma) =$
$\mathcal{FN}(f(t_1, \ldots, t_n))\mathcal{FN}(\gamma) = \mathcal{FN}(t)\mathcal{FN}(\gamma)$ □

LEMMA 5.12. *Let s, t, $u_1, \ldots, u_m$ and w be higher-order terms and let p be a position in t and $q_1 \ldots q_n$ positions in $\mathcal{FN}(t)$, such that $\mathcal{FN}(t[w]_p) = \mathcal{FN}(t)[\mathcal{FN}(w)]_{q_1 \ldots q_n}$. Then we have that there are disjoint positions $r_1 \ldots r_o$ such that*

—$\mathcal{N}_i(\mathcal{FN}(t[w]_p), \langle \mathcal{FN}(u_1), \ldots, \mathcal{FN}(t[w]_p), \ldots, \mathcal{FN}(u_m)\rangle) =$
$\mathcal{N}_i(\mathcal{FN}(t), \langle \mathcal{FN}(u_1), \ldots, \mathcal{FN}(t), \ldots, \mathcal{FN}(u_m)\rangle)[\mathcal{FN}(w)]_{r_1 \ldots r_o}$
if $t : \tau_1 \to \ldots \to \tau_k \to \tau$ *and* $i \leq k$.
—$\mathcal{N}_i(\mathcal{FN}(t[w]_p), \langle \mathcal{FN}(u_1), \ldots, \mathcal{FN}(u_m)\rangle) =$
$\mathcal{N}_i(\mathcal{FN}(t), \langle \mathcal{FN}(u_1), \ldots, \mathcal{FN}(u_m)\rangle)[\mathcal{FN}(w)]_{r_1 \ldots r_o}$
if $t : \tau_1 \to \ldots \to \tau_k \to \tau$ *and* $i \leq k$.
—$\mathcal{N}_i(\mathcal{FN}(s), \langle \mathcal{FN}(u_1), \ldots, \mathcal{FN}(t[w]_p), \ldots, \mathcal{FN}(u_m)\rangle) =$
$\mathcal{N}_i(\mathcal{FN}(s), \langle \mathcal{FN}(u_1), \ldots, \mathcal{FN}(t), \ldots, \mathcal{FN}(u_m)\rangle)[\mathcal{FN}(w)]_{r_1 \ldots r_o}$
if $s : \tau_1 \to \ldots \to \tau_k \to \tau$ *and* $i \leq k$.

PROOF. We only provide the proof of the first case, the other two are easily obtained from this one. $\mathcal{N}_i(\mathcal{FN}(t[w]_p), \langle \mathcal{FN}(u_1), \ldots, \mathcal{FN}(t[w]_p), \ldots, \mathcal{FN}(u_m)\rangle) =$
$@(\ldots @(\mathcal{FN}(t[w]_p),\ \perp_{\tau_1}(\mathcal{FN}(u_1), \ldots, \mathcal{FN}(t[w]_p), \ldots, \mathcal{FN}(u_m))), \ldots$
$\perp_{\tau_i}(\mathcal{FN}(u_1), \ldots, \mathcal{FN}(t[w]_p), \ldots, \mathcal{FN}(u_m))))$
By assumption, we have that $\mathcal{FN}(t[w]_p) = \mathcal{FN}(t)[\mathcal{FN}(w)]_{q_1 \ldots q_n}$, hence is equal to

$$@(\ldots @(\mathcal{FN}(t)[\mathcal{FN}(w)]_{q_1\ldots q_n}),\ \bot_{\tau_1}(\mathcal{FN}(u_1),\ldots,\mathcal{FN}(t)[\mathcal{FN}(w)]_{q_1\ldots q_n},\ldots,\mathcal{FN}(u_m))),\ldots \bot_{\tau_i}(\mathcal{FN}(u_1),\ldots,\mathcal{FN}(t)[\mathcal{FN}(w)]_{q_1\ldots q_n},\ldots,\mathcal{FN}(u_m))))$$

Now adding the corresponding prefixes $p'_0, p'_1, \ldots, p'_i$, which are the disjoint positions of the $i+1$ occurrences of $\mathcal{FN}(t)$ we are considering, this is equal to

$$@(\ldots @(\mathcal{FN}(t),\ \bot_{\tau_1}(\mathcal{FN}(u_1),\ldots,\mathcal{FN}(t),\ldots,\mathcal{FN}(u_m))),\ldots \bot_{\tau_i}(\mathcal{FN}(u_1),\ldots,\mathcal{FN}(t),\ldots,\mathcal{FN}(u_m))))[\mathcal{FN}(w)]_{r_1\ldots r_o}$$

where $r_1 \ldots r_o$ is the set of disjoint positions
$p'_0 \cdot q_1 \ldots p'_0 \cdot q_n, p'_1 \cdot q_1 \ldots p'_1 \cdot q_n, \ldots, p'_i \cdot q_1 \ldots p'_i \cdot q_n$. □

LEMMA 5.13. *Let t, w be higher-order terms. For every position p in t there are disjoint positions $\{q_1, \ldots, q_n\}_{n>0}$ in $\mathcal{FN}(t)$ such that $\mathcal{FN}(t[w]_p) = \mathcal{FN}(t)[\mathcal{FN}(w)]_{q_1\ldots q_n}$.*

PROOF. We proceed by induction on $|p|$. If p is Λ, it trivially holds. Otherwise, there are three cases:

(1) $t = \lambda x.u$. Then $p = 1 \cdot p'$ and $\mathcal{FN}(\lambda x.u) = \lambda x.\mathcal{FN}(u)$. By induction hypothesis, there are disjoint positions $\{q'_1 \ldots q'_n\}_{n>0}$ such that $\mathcal{FN}(u[w]_{p'}) = \mathcal{FN}(u)[\mathcal{FN}(w)]_{q'_1\ldots q'_n}$, and therefore, taking $q_i = 1 \cdot q'_i$ for all $i \in \{1 \ldots n\}$, we have $\mathcal{FN}(\lambda x.u[w]_p) = \lambda x.\mathcal{FN}(u[w]_{p'}) = \lambda x.\mathcal{FN}(u)[\mathcal{FN}(w)]_{q'_1\ldots q'_n} = \mathcal{FN}(\lambda x.u)[\mathcal{FN}(w)]_{q_1\ldots q_n}$.

(2) $t = @(t_1, t_2)$. Let $p = 1 \cdot p'$ (the case $p = 2 \cdot p'$ is analogous). Then $\mathcal{FN}(@(t_1, t_2)) = @(\mathcal{FN}(t_1), \mathcal{FN}(t_2))$. By induction hypothesis, there are disjoint positions $\{q'_1 \ldots q'_n\}_{n>0}$ such that $\mathcal{FN}(t_1[w]_{p'}) = \mathcal{FN}(t_1)[\mathcal{FN}(w)]_{q'_1\ldots q'_n}$, and therefore, taking $q_i = 1 \cdot q'_i$ for all $i \in \{1 \ldots n\}$, we have $\mathcal{FN}(@(t_1, t_2)[w]_p) = \mathcal{FN}(@(t_1[w]_{p'}, t_2)) = @(\mathcal{FN}(t_1[w]_{p'}), \mathcal{FN}(t_2)) = @(\mathcal{FN}(t_1)[\mathcal{FN}(w)]_{q'_1\ldots q'_n}, \mathcal{FN}(t_2)) = @(\mathcal{FN}(t_1), \mathcal{FN}(t_2))[\mathcal{FN}(w)]_{q_1\ldots q_n} = \mathcal{FN}(@(t_1, t_2))[\mathcal{FN}(w)]_{q_1\ldots q_n}$.

(3) $t = f(t_1, \ldots, t_m)$ with $f \in \mathcal{F}$. Let $p = i \cdot p'$ for some $i \in \{1 \ldots m\}$. Then $\mathcal{FN}(f(t_1, \ldots, t_m)) = f_{new}(\mathcal{N}_{nl_f^1}(\mathcal{FN}(t_1), \mathcal{FN}(\bar{t}_f^{\,1})), \ldots, \mathcal{N}_{nl_f^m}(\mathcal{FN}(t_m), \mathcal{FN}(\bar{t}_f^{\,m})))$. By induction hypothesis, there are disjoint positions $\{q'_1 \ldots q'_n\}_{n>0}$ such that $\mathcal{FN}(t_i[w]_{p'}) = \mathcal{FN}(t_i)[\mathcal{FN}(w)]_{q'_1\ldots q'_{n'}}$.
Then by Lemma 5.12, we have that for every t_j with $j \in \{1 \ldots m\}$ there are disjoint positions $q_1^j \ldots q_{n_j}^j$ such that
— if $i = j$ and $i \in \mathcal{A}_f^j$ and i_j is the position of t_i in $\bar{t}_f^{\,j}$ then $\mathcal{N}_{nl_f^j}(\mathcal{FN}(t_j[w]_p), \mathcal{FN}(\bar{t}_f^{\,j}[w]_{i_j \cdot p})) = \mathcal{N}_{nl_f^j}(\mathcal{FN}(t_j), \mathcal{FN}(\bar{t}_f^{\,j}))[\mathcal{FN}(w)]_{q_1^j\ldots q_{n_j}^j}$
— if $i \neq j$ and $i \in \mathcal{A}_f^j$ and i_j is the position of t_i in $\bar{t}_f^{\,j}$ then $\mathcal{N}_{nl_f^j}(\mathcal{FN}(t_j), \mathcal{FN}(\bar{t}_f^{\,j}[w]_{i_j \cdot p})) = \mathcal{N}_{nl_f^j}(\mathcal{FN}(t_j), \mathcal{FN}(\bar{t}_f^{\,j}))[\mathcal{FN}(w)]_{q_1^j\ldots q_{n_j}^j}$

— if $i = j$ and $i \notin \mathcal{A}_f^j$ and i_j is the position of t_i in $\bar{t}_f^j$ then $\mathcal{N}_{nl_f^j}(\mathcal{FN}(t_j), \mathcal{FN}(\bar{t}_f^j[w]_{i_j \cdot p})) = \mathcal{N}_{nl_f^j}(\mathcal{FN}(t_j), \mathcal{FN}(\bar{t}_f^j))[\mathcal{FN}(w)]_{q_1^j \ldots q_{n_j}^j}$
Therefore $\mathcal{FN}(f(t_1, \ldots, t_i[w]_{p'}, \ldots, t_m)) =$
$\mathcal{FN}(f(t_1, \ldots, t_i, \ldots, t_m))[\mathcal{FN}(w)]_{1 \cdot q_1^1 \ldots 1 \cdot q_{n_1}^1 \ldots m \cdot q_1^m \ldots m \cdot q_{nm}^m}$ □

Now we prove η- and β-compatibility of $\mathcal{FN}$.

LEMMA 5.14. *Let s, t be higher-order terms. If $s =_\eta t$, then $\mathcal{FN}(s) =_\eta \mathcal{FN}(t)$.*

PROOF. First, we prove that if $x \notin \mathcal{V}ar(u)$ then $\mathcal{FN}(\lambda x.@(u, x)) =_\eta \mathcal{FN}(\lambda x.@(u, x))$. By definition of $\mathcal{FN}$, we have that $\mathcal{FN}(\lambda x.@(u, x)) = \lambda x.@(\mathcal{FN}(u), x)$. Since $\mathcal{FN}$ preserves the set of free variables, we have that $x \notin \mathcal{V}ar(\mathcal{FN}(u))$, and hence $\lambda x.@(\mathcal{FN}(u), x) =_\eta \mathcal{FN}(u)$.

Now, assume we have $s[\lambda x.@(u, x)]_p =_\eta s[u]_p$ for some position p, then by Lemma 5.13, there are disjoint positions $q_1, \ldots, q_n$ in $\mathcal{FN}(s)$ such that we have that $\mathcal{FN}(s[\lambda x.@(u, x)]_p) = \mathcal{FN}(t)[\mathcal{FN}(\lambda x.@(u, x))]_{q_1 \ldots q_n}$ and $\mathcal{FN}(s[u]_p) = \mathcal{FN}(t)[\mathcal{FN}(u)]_{q_1 \ldots q_n}$. Therefore, since, $\mathcal{FN}(t)[\mathcal{FN}(\lambda x.@(u, x))]_{q_1 \ldots q_n} =_\eta \mathcal{FN}(t)[\mathcal{FN}(u)]_{q_1 \ldots q_n}$, we conclude that $\mathcal{FN}(s[\lambda x.@(u, x)]_p) =_\eta \mathcal{FN}(s[u]_p)$.

We conclude by induction on the number of η-equality steps applied in $s =_\eta t$. □

LEMMA 5.15. *Let s, t be higher-order terms. If $s =_\beta t$, then $\mathcal{FN}(s) =_\beta \mathcal{FN}(t)$.*

PROOF. First, we prove that $\mathcal{FN}(@(\lambda x.u, v)) =_\beta \mathcal{FN}(u\{x \mapsto v\})$. By definition of $\mathcal{FN}$, we have that $\mathcal{FN}(@(\lambda x.u, v)) = @(\lambda x.\mathcal{FN}(u), \mathcal{FN}(v))$, which is β-equivalent to $\mathcal{FN}(u)\{x \mapsto \mathcal{FN}(v)\}$, and by Lemma 5.11, it is equal to $\mathcal{FN}(u\{x \mapsto v\})$. We then conclude that $\mathcal{FN}(s) =_\beta \mathcal{FN}(t)$, proceeding as in the proof of the previous lemma, using Lemma 5.13 and induction on the number of β-equality steps afterward. □

The coming two lemmas follow easily:

LEMMA 5.16. *Let $t : \tau$ be a higher-order term. Then $\mathcal{FN}(t) : \tau$.*

LEMMA 5.17. *Let t be a higher-order term. Then $\mathcal{FN}(t)$ is headed by an algebraic symbol whenever t is.*

5.3. Neutralization as an interpretation

Definition 5.18. We define the interpretation $\mathcal{N}$ as $\mathcal{N}(s) = \mathcal{FN}(s)\downarrow_\beta$.

THEOREM 5.19. *$\mathcal{N}$ is a normal higher-order interpretation.*

PROOF. By definition and Lemma 5.16, $\mathcal{N}$ is a higher-order interpretation. Since β-normalization preserves both η- and β-equivalence, by Lemma 5.15 we obtain β-compatibility and by Lemma 5.14 we obtain η-compatibility. Preservation of β-normalization trivially holds by definition. Lemma 5.17 implies preservation of

algebraic heads. Finally, Lemma 5.11 implies stability and Lemma 5.13 monotonicity. Therefore $\mathcal{N}$ is a normal higher-order interpretation. □

Full normalization combined with neutralization can be used in conjunction with any normal higher-order reduction ordering. Of course, we then need to define the precedence on function symbols for the extended signature. In practice, we shall make $\bot_\sigma$-functions symbols small, as done now to prove our starting example.

Example 5.20 (*differentiation end*). We consider again the differentiation example, using now neutralization. Let $\mathsf{diff}_{\mathsf{new}} >_{\mathcal{F}} \{\times_{\mathsf{new}}, +_{\mathsf{new}}, \mathsf{cos}\}$ and $\mathsf{diff}_{\mathsf{new}} \in \mathsf{Mul}$.
First rule:
$\mathsf{diff}_{\mathsf{new}}(\mathsf{sin}(@(\mathsf{F},\bot))) > \mathsf{cos}(@(\mathsf{F},\bot)) \times_{\mathsf{new}} @(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot)),\bot)$
Applying first case 2, we recursively obtain two subgoals:
(i) $\mathsf{diff}_{\mathsf{new}}(\mathsf{sin}(@(\mathsf{F},\bot))) > \mathsf{cos}(@(\mathsf{F},\bot))$
(ii) $\mathsf{diff}_{\mathsf{new}}(\mathsf{sin}(@(\mathsf{F},\bot))) > @(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot)),\bot)$.
(i): applying Case 2 yields $\mathsf{diff}_{\mathsf{new}}(\mathsf{sin}(@(\mathsf{F},\bot))) > @(\mathsf{F},\bot)$ shown by Case 1 twice.
(ii): applying Case 5 generates two new subgoals
(iii) $\mathsf{diff}_{\mathsf{new}}(\mathsf{sin}(@(\mathsf{F},\bot))) > \mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot))$,which holds by case 3, then case 1.
(iv) $\mathsf{diff}_{\mathsf{new}}(\mathsf{sin}(@(\mathsf{F},\bot))) > \bot$,which holds by case 1 twice and then case 7.
Second rule:
$\mathsf{diff}_{\mathsf{new}}(@(@(\mathsf{F},\bot) \times_{\mathsf{new}} @(\mathsf{F},\bot),\bot)) > @(@(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot)),\bot) \times_{\mathsf{new}} @(\mathsf{F},\bot),\bot) +_{\mathsf{new}} @(@(\mathsf{F},\bot) \times_{\mathsf{new}} @(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot)),\bot),\bot)$
Case 2 generates two subgoals:
(i) $\mathsf{diff}_{\mathsf{new}}(@(@(\mathsf{F},\bot) \times_{\mathsf{new}} @(\mathsf{F},\bot),\bot)) > @(@(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot)),\bot) \times_{\mathsf{new}} @(\mathsf{F},\bot),\bot)$
(ii) $\mathsf{diff}_{\mathsf{new}}(@(@(\mathsf{F},\bot) \times_{\mathsf{new}} @(\mathsf{F},\bot),\bot)) > @(@(\mathsf{F},\bot) \times_{\mathsf{new}} @(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot)),\bot),\bot)$
We only prove (i), as (ii) is shown exactly in the same way.
By Case 5, (i) generates two new subgoals:
(iii) $\mathsf{diff}_{\mathsf{new}}(@(@(\mathsf{F},\bot) \times_{\mathsf{new}} @(\mathsf{F},\bot),\bot)) > @(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot)),\bot) \times_{\mathsf{new}} @(\mathsf{F},\bot)$
(iv) $\mathsf{diff}_{\mathsf{new}}(@(@(\mathsf{F},\bot) \times_{\mathsf{new}} @(\mathsf{F},\bot),\bot)) > \bot$.
The latter holds by case 1 and then case 7. By Case 2, (iii) yields two subgoals:
(v) $\mathsf{diff}_{\mathsf{new}}(@(@(\mathsf{F},\bot) \times_{\mathsf{new}} @(\mathsf{F},\bot),\bot)) > @(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot)),\bot)$
(vi) $\mathsf{diff}_{\mathsf{new}}(@(@(\mathsf{F},\bot) \times_{\mathsf{new}} @(\mathsf{F},\bot),\bot)) > @(\mathsf{F},\bot)$, solved by applying successively Case 1, 7 and 1. By Case 5, (v) generates
(vii) $\mathsf{diff}_{\mathsf{new}}(@(@(\mathsf{F},\bot) \times_{\mathsf{new}} @(\mathsf{F},\bot),\bot)) > \mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot))$
(viii) $\mathsf{diff}_{\mathsf{new}}(@(@(\mathsf{F},\bot) \times_{\mathsf{new}} @(\mathsf{F},\bot),\bot)) > \bot$ solved by Case 1 first, then 7.
Finally, (vii) is solved by Case 3, 7, and 1 successively.

6. POLYMORPHIC NORMAL REWRITING

We investigate here the case of polymorphic rewriting defined by polymorphic rewrite rules. Polymorphism is introduced via type variables, quantifiers being omitted, hence is *weak* in the usual sense. We shall not mention further this "weakness" which is present throughout this section. Since types are first-order expres-

sions, type instantiation is the usual notion of substitution, acting as a homomorphism of expressions generated by its action on type variables. This applies to terms too via the type of variables in abstractions.

The idea is of course to reduce polymorphic rewriting to monomorphic rewriting by considering all ground type-instances of a given polymorphic rewrite rule. Example 7.5 taken from [Pol 1996] describes an encoding of natural deduction by means of a polymorphic rewriting system whose difficult normal termination proof can be automatically carried out in full.

6.1. Polymorphic algebras

We first recall the basics of polymorphic algebras introduced in [Jouannaud and Rubio 2007]. We are now given a set $\mathcal{S}^\forall$ of distinguished type constants called type variables. We use $\mathcal{T}_{\mathcal{S}^\forall}$ for the set of types, and $\mathcal{T}_\mathcal{S}$, as before, for the set of ground types. Signatures may now contain type variables. Types, declarations and signatures are said to be *polymorphic* if they contain type variables, and *monomorphic* otherwise. As before, we collect all declarations into a single *environment* $\Sigma; \Gamma$, where the signature $\Sigma = \mathcal{S}; \mathcal{S}^\forall; \mathcal{F}$ is the fixed part of the environment, and the variable part Γ collects the type declarations for (term) variables.

The typing judgments are displayed in Figure 2. Since the last two rules introduce a type substitution ξ, we have considered, for uniformity reason, that the first two introduce the identity type substitution ι.

Variables:	**Abstraction:**
$\dfrac{x : \sigma \in \Gamma}{\Gamma \vdash^c_\Sigma x : \sigma\iota}$	$\dfrac{\Gamma \cdot \{x : \sigma\} \vdash^c_\Sigma t : \tau}{\Gamma \vdash^c_\Sigma (\lambda x : \sigma.t) : (\sigma \to \tau)\iota}$
Application:	**Functions:**
$\dfrac{\Gamma \vdash^c_\Sigma s : \sigma \quad \Gamma \vdash^c_\Sigma t : \tau \quad \xi \text{ most general unifier of } \alpha \to \beta = \sigma \wedge \alpha = \tau}{\Gamma\xi \vdash^c_\Sigma @(s,t)\xi : \beta\xi}$	$\dfrac{f : \sigma_1 \times \ldots \times \sigma_n \Rightarrow \sigma \in \mathcal{F} \quad \Gamma \vdash^c_\Sigma t_1 : \tau_1 \ \ldots \ \Gamma \vdash^c_\Sigma t_n : \tau_n \quad \xi \text{ most general unifier of } \sigma_1 = \tau_1 \wedge \ldots \wedge \sigma_n = \tau_n}{\Gamma\xi \vdash^c_\Sigma f(t_1, \ldots, t_n)\xi : \sigma\xi}$

Fig. 2. Typing judgments in polymorphic algebras

Definition 6.1. Given an environment $\Sigma; \Gamma$, a raw term s is *typable* with *principal type* σ, written $\Gamma \vdash_\Sigma s : \sigma$, iff the judgment $\Gamma \vdash^c_\Sigma s : \sigma$ is provable in the system of Figure 2.

Two key properties of this typing system are [Jouannaud and Rubio 2007]:
(i) every typable term has a (unique) principal type;
(ii) if $\Gamma \vdash_\Sigma s : \sigma$, then $\Gamma\xi \vdash_\Sigma s\xi : \sigma\xi$.

6.2. Polymorphic Rewriting

In this section, we consider polymorphic rewriting for terms of a monomorphic type, which is easily extensible to terms of a polymorphic type by considering the type variables in the environment of the term as new constant types.

Definition 6.2. A *polymorphic normal rewrite rule* is a normal rewrite rule $\Gamma \vdash l \to r : \sigma$ such that l and r are higher-order terms in β-normal form having the same principal type σ in the environment Γ. A polymorphic normal rewrite system is a set of such rules.
Given a polymorphic normal rewrite system R, we define the set $R_g = \{R\xi\}_\xi$ of type instances of R by some ground substitution ξ.
Given now a term s of a ground type, we define $s \longrightarrow^p_R t$ iff $s \longrightarrow^p_{R_g} t$.

Given a polymorphic normal rewrite rule $\Gamma \vdash l \to r : \sigma$ and a ground type substitution ξ, then $\Gamma\xi \vdash l\xi \to r\xi : \sigma\xi$ is a monomorphic normal rewrite rule by the property (ii) of the polymorphic type system, justifying our definition of polymorphic rewriting. This would not be true for a definition like Nipkow's which requires l and r to be in η-long form, a property which is not closed under type instantiation.

Let now $s = s[l\gamma]_p$, where $\Gamma \vdash l \to r$ is a polymorphic rule. There are two kinds of redexes $l\gamma$:

— Those for which γ replaces a variable $x : \sigma \in \Gamma$ by a term of type σ. Those substitutions act as in the monomorphic case, they do not change the type of terms. As a consequence, if $\succ$ is a stable ordering such that $l \succ r$, then $l\gamma \succ r\gamma$.
— Those for which γ replaces some variable $x : \sigma \in \Gamma$ by a term of type $\sigma\xi$. Such substitutions instantiate both the term and its type.

Definition 6.3. Let γ be a substitution $\{x_i \mapsto t_i\}_i$ and Γ be an environment such that $(\forall i)\, \Gamma \vdash^c_\Sigma x_i : \sigma_i$. Then γ is *type preserving* in Γ if and only if f $(\forall i)\, \Gamma \vdash^c_\Sigma t_i : \sigma_i$ and γ is a *specialization* by the type instantiation ξ (possibly the identity) in Γ if and only if $(\forall i)\, \Gamma\xi \vdash^c_\Sigma t_i : \sigma_i\xi$.
An ordering $\succ$ of the set of higher-order terms is called *stable* if it is stable under type preserving substitutions, and *polymorphic* if it is stable under specializations.

Note that a stable order is a particular case of a polymorphic order.

A polymorphic higher-order reduction ordering $\succ$ makes it possible to compare all instances of the lefthand and righthand side of a rewrite rule $l \to r$ by doing the single comparison $l \succ r$.

In [Jouannaud and Rubio 2007], we show that HORPO is polymorphic: this is so because types do not influence the choice of rules in an ordering computation, provided the type ordering is extended to type variables in a way that is compatible with substitutions, that is $\sigma \geq_{\mathcal{TS}} \tau$ implies $\sigma\xi \geq_{\mathcal{TS}} \tau\xi$ for any ground type

substitution ξ. Of course, the restricted version $>$ of HORPO introduced here is polymorphic as well. We now define polymorphic interpretations:

Definition 6.4. A higher-order interpretation I is *polymorphic* if it preserves polymorphism, that is, $I(s\xi) =_\eta I(s)\xi$.

THEOREM 6.5. *Let R be a normal higher-order rewrite system whose lefthand sides are all either headed by an algebraic symbol or have a non variable basic type and I a polymorphic normal higher-order interpretation. Then termination of $I(R)$ implies termination of R.*

PROOF. We show that $s \longrightarrow_R t$ implies $I(s) \longrightarrow^+_{I(R)} I(t)$. By definition of polymorphic rewriting, $s \longrightarrow_{\bigcup_\xi R\xi} t$, ξ denoting a monomorphic type substitution. By Theorem 5.5, $I(s) \longrightarrow^+_{I(\bigcup_\xi R\xi)} I(t)$. Now $I(\bigcup_\xi IR\xi) = \bigcup_\xi I(R\xi) =_\eta \bigcup_\xi I(R)\xi$ by preservation of polymorphism. Hence, $I(s) \longrightarrow^+_{I(R)} I(t)$ by definition of polymorphic rewriting. □

We are left showing that neutralization preserves polymorphism. Note that we could take advantage of polymorphism to eliminate (part of) type decorations from coercions and bottom symbols. We will however not do this, so as to facilitate the reading of complex expressions in proofs or examples.

LEMMA 6.6. *Let $t : \sigma$ be a higher-order term and ξ a type substitution. Then $t\xi{\downarrow_\beta} = t{\downarrow_\beta}\,\xi$.*

PROOF. Note first that $t\xi \to_\beta t'$ if and only if there is some t'' with $t \to_\beta t''$ and $t' = t''\xi$. The result follows by induction on the derivation $t\xi \longrightarrow^*_\beta (t\xi){\downarrow_\beta}$. □

Since the neutralization level nl^j_f of an argument of f is smaller than or equal to the arity of its type, we have the following properties:

LEMMA 6.7. *Let $t : \tau$ and $u_1 : \theta_1, \ldots, u_n : \theta_n$ be higher-order terms and ξ a type substitution. Then, $(\forall i \leq ar(\tau))\ \mathcal{N}_i(t\xi, \langle u_1\xi, \ldots, u_n\xi\rangle) = \mathcal{N}_i(t, \langle u_1, \ldots, u_n\rangle)\xi$.*

PROOF. We proceed by induction on i. There are three cases.

(1) If $i < 0$ then $\mathcal{N}_i(t\xi, \langle u_1\xi, \ldots, u_n\xi\rangle) = m_{\tau\xi}(t\xi) = \mathcal{N}_0(t, \langle u_1, \ldots, u_n\rangle)\xi$.
(2) If $i = 0$ then $\mathcal{N}_0(t\xi, \langle u_1\xi, \ldots, u_n\xi\rangle) = t\xi = \mathcal{N}_0(t, \langle u_1, \ldots, u_n\rangle)\xi$.
(3) If $i > 0$ then, by assumption $\tau = \sigma \to \rho$, hence $t\xi : \sigma\xi \to \rho\xi$. Then,
$\mathcal{N}_i(t\xi, \langle u_1\xi, \ldots, u_n\xi\rangle) = \mathcal{N}_{i-1}(@(t\xi, \bot_{\theta_1\xi\to\ldots\to\theta_n\xi\to\sigma\xi}(u_1\xi, \ldots, u_n\xi)))$
$= \mathcal{N}_{i-1}(@(t, \bot_{\theta_1\to\ldots\to\theta_n\to\sigma}(u_1, \ldots, u_n))\xi)$.
Since $i - 1 \leq ar(\rho)$, by induction hypothesis, this is equal to
$\mathcal{N}_{i-1}(@(t, \bot_{\theta_1\to\ldots\to\theta_n\to\sigma}(u_1, \ldots, u_n)))\xi = \mathcal{N}_i(t, \langle u_1, \ldots, u_n\rangle)\xi$. □

THEOREM 6.8. *Full neutralization is polymorphic.*

PROOF. Let $t : \sigma$ be a higher-order term and ξ a type substitution. By Lemma 6.6, it is enough to show that $\mathcal{FN}(t\xi) = \mathcal{FN}(t)\xi$, we we do by induction on $|t|$. There are four cases. If t is a variable x, the result is clear. Otherwise,

(1) $t = \lambda x : \tau.u$. Then $\mathcal{FN}(t\xi) = \mathcal{FN}(\lambda x : \tau\xi.u\xi) = \lambda x : \tau\xi.\mathcal{FN}(u\xi)$.
By induction hypothesis, $\mathcal{FN}(u\xi) = \mathcal{FN}(u)\xi$, which yields the result.
(2) $t = @(t_1, \ldots, t_n)$. Then $\mathcal{FN}(t\xi) = \mathcal{FN}(@(t_1\xi, \ldots, t_n\xi)) = @(\mathcal{FN}(t_1\xi), \ldots, \mathcal{FN}(t_n\xi)) = @(\mathcal{FN}(t_1\xi), \ldots, \mathcal{FN}(t_n\xi))$.
By induction hypothesis, this is equal to
$@(\mathcal{FN}(t_1)\xi, \ldots, \mathcal{FN}(t_n)\xi) = @(\mathcal{FN}(t_1), \ldots, \mathcal{FN}(t_n))\xi = \mathcal{FN}(@(t_1, \ldots, t_n))\xi = \mathcal{FN}(t)\xi$.
(3) $t = f(t_1, \ldots, t_n)$ with $f \in \mathcal{F}$. Then
$\mathcal{FN}(t\xi) = \mathcal{FN}(f(t_1, \ldots, t_n)\xi) = \mathcal{FN}(f(t_1\xi, \ldots, t_n\xi)) = f_{new}(\mathcal{N}_{nl_f^1}(\mathcal{FN}(t_1\xi), \mathcal{FN}(\bar{t}_f^1\xi)), \ldots, \mathcal{N}_{nl_f^n}(\mathcal{FN}(t_n\xi), \mathcal{FN}(\bar{t}_f^n\xi)))$.
By induction hypothesis, this is equal to
$f_{new}(\mathcal{N}_{nl_f^1}(\mathcal{FN}(t_1)\xi, \mathcal{FN}(\bar{t}_f^1)\xi)), \ldots, \mathcal{N}_{nl_f^n}(\mathcal{FN}(t_n)\xi, \mathcal{FN}(\bar{t}_f^n)\xi)))$
$= f_{new}(\mathcal{N}_{nl_f^1}(\mathcal{FN}(t_1), \mathcal{FN}(\bar{t}_f^1))\xi, \ldots, \mathcal{N}_{nl_f^n}(\mathcal{FN}(t_n), \mathcal{FN}(\bar{t}_f^n))\xi)$,
by using lemma 6.7. Extracting the type substitution we get
$f_{new}(\mathcal{N}_{nl_f^1}(\mathcal{FN}(t_1, \mathcal{FN}(\bar{t}_f^1))), \ldots, \mathcal{N}_{nl_f^n}(\mathcal{FN}(t_n, \mathcal{FN}(\bar{t}_f^n))))\xi$
$= \mathcal{FN}(f(t_1, \ldots, t_n))\xi = \mathcal{FN}(t)\xi$. □

7. EXAMPLES

We present here several complex examples whose termination is proved with normal HORPO in conjunction with neutralization. For all of them, we give the necessary ingredients for computing the appropriate neutralizations and comparisons. 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 as type ordering $\geq_{\mathcal{T}_\mathcal{S}}$ an RPO as the one described just above Definition 4.3 where, additionally, we equate all sort symbols in the precedence on sort. The precedence on function symbols and statuses will be given in full.

The rules are first given in a format aiming at an easier reading by writing $F(X)$ for $@(F, X)$. When it comes to the computations, we make the contrary choice, to make clear that $@$ is the top function symbol of the term $F(X)$. Since these computations are complex, all examples have been checked by our implementation, which is available from the web at the following url:
`www.lsi.upc.edu/~albert/normal.html`.
Consequently, we will allow ourselves to skip the computation part in most cases.

Example 7.1. The coming encoding of first-order prenex normal forms is adapted from [Nipkow 1991], 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.

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

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

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

We carry out the proof of the first rule, abbreviating $\bot_{form}$ as $\bot$ and making application symbols explicit. First, we compute the full neutralization of both sides:
$\mathcal{FN}(P \wedge \forall(\lambda x.@(Q,x)))\downarrow_\beta = P \wedge \forall_{new}(@(Q,\bot))$, and
$\mathcal{FN}(\forall(\lambda x.(P \wedge @(Q,x))))\downarrow_\beta = \forall_{new}(P \wedge @(Q,\bot))$.

Second, we show that $P \wedge \forall_{new}(@(Q,\bot)) > \forall_{new}(P \wedge @(Q,\bot))$. By case 2 we need to show $P \wedge \forall_{new}(@(Q,\bot)) > P \wedge @(Q,\bot)$, and by case 3 we need $\{P, \forall_{new}(@(Q,\bot))\}(>)_{mul}\{P, @(Q,\bot)\}$. This requires
$\forall_{new}(@(Q,\bot)) > @(Q,\bot)$ which holds by case 1.

Example 7.2. This example of surjective disjoint union is from [van de Pol and Schwichtenberg 1995]. The monomorphic signature and rules are parametrized by $\alpha \in \mathcal{S} = \{A, B, U\}$ for sake of conciseness.

$$\mathcal{F} = \{inl : A \to U;\ inr : B \to U;\ case_\alpha : U \times (A \to \alpha) \times (B \to \alpha) \to \alpha\}.$$

$$case_\alpha(inl(X), F, G) \to F(X) \quad \| \quad case_\alpha(inr(Y), F, G) \to G(Y)$$
$$case_\alpha(Z, \lambda x.H(inl(x)), \lambda y.H(inr(y))) \to H(Z)$$

Neutralization: $\mathcal{L}^2_{\mathsf{case}} = 1$, $\mathcal{L}^3_{\mathsf{case}} = 1$, $\mathcal{A}^2_{\mathsf{case}} = \{1\}$,$\mathcal{A}^3_{\mathsf{case}} = \{1\}$.
Statuses are multiset and precedence is equality. We sketch the proof of the first rule, computing first the β-normalization of the full neutralizations of both sides:
$s = \mathcal{FN}(case_\alpha(inl(X), F, G))\downarrow_\beta =$
$case_{new}(inl(X), @(F, \bot_{U\to A}(inl(X))), @(G, \bot_{U\to B}(inl(X))))$, and
$t = \mathcal{FN}(@(F,X))\downarrow_\beta = @(F,X)$.

The proof that $s > t$ uses successively cases 1, 8, and then 1 twice.

Example 7.3 (*[van de Pol 1993]*). Let

$$\begin{array}{rl} \mathcal{S} = & \{proc, data\} \\ F = & \{+ : proc \times proc \to proc, \cdot : proc \times proc \to proc, \delta :\to proc, \\ & \Sigma : (data \to proc) \to proc)\} \end{array}$$

Here, $+$ stands for the choice operator, $\cdot$ for sequential composition, δ for deadlock, and Σ for the data dependent choice. The rules are the following:

$$\begin{array}{rcl}
\{x : proc\} & \vdash & x + x \to x \\
\{x, y, z : proc\} & \vdash & (x + y) \cdot z \to (x \cdot z) + (y \cdot z) \\
\{x, y, z : proc\} & \vdash & (x \cdot y) \cdot z \to x \cdot (y \cdot z) \\
\{x : proc\} & \vdash & x + \delta \to x \\
\{x : proc\} & \vdash & \delta \cdot x \to \delta \\
\{x : proc\} & \vdash & \Sigma(\lambda d.x) \to x \\
\{D : data,\ P : data \to proc\} & \vdash & \Sigma(\lambda d.P(d)) + P(D) \to \Sigma(\lambda d.P(d)) \\
\{P, Q : data \to proc\} & \vdash & \left\{ \begin{array}{c} \Sigma(\lambda d.P(d) + Q(d)) \\ \to \\ \Sigma(\lambda d.P(d)) + \Sigma(\lambda d.Q(d)) \end{array} \right. \\
\{x : proc,\ P : data \to proc\} & \vdash & \Sigma(\lambda d.P(d)) \cdot x \to \Sigma(\lambda d.(P(d) \cdot x))
\end{array}$$

Example 7.4. Monomorphic encoding of natural deduction, from [Pol 1996]. Let $\mathcal{S} = \{o,\ c : * \times * \to *\}$. The signature and rules follow, parametrized by three arbitrary types σ, τ and ρ, making them infinite. The coming calculations are not meant to be automated.

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

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

$$\begin{array}{rcl||rcl}
app_{\sigma,\tau}(abs_{\sigma,\tau}(F), X) & \to & F(X) & \exists^-_{\sigma,\tau}(\exists^+_\sigma(Z, X), G) & \to & G(Z, X) \\
\Pi^0_{\sigma,\tau}(\Pi_{\sigma,\tau}(X, Y)) & \to & X & \Pi^1_{\sigma,\tau}(\Pi_{\sigma,\tau}(X, Y)) & \to & Y \\
\multicolumn{6}{c}{app_{\sigma,\tau}(\exists^-_{\rho,\sigma\to\tau}(T, H), X) \to \exists^-_{\rho,\tau}(T, \lambda x : o\ y : \rho.app_{\sigma,\tau}(H(x, y), X))} \\
\multicolumn{6}{c}{\Pi^0_{\sigma,\tau}(\exists^-_{\rho,c(\sigma,\tau)}(T, I)) \to \exists^-_{\rho,\tau}(T, \lambda x : o\ y : \rho.\Pi^0_{\sigma,\tau}(I(x, y)))} \\
\multicolumn{6}{c}{\Pi^1_{\sigma,\tau}(\exists^-_{\rho,c(\sigma,\tau)}(T, I)) \to \exists^-_{\rho,\tau}(T, \lambda x : o\ y : \rho.\Pi^1_{\sigma,\tau}(I(x, y)))} \\
\multicolumn{6}{c}{\exists^-_{\sigma,\tau}(\exists^-_{\rho,c(o,\sigma)}(T, J), G) \to \exists^-_{\rho,\tau}(T, \lambda x : o\ y : \rho.\exists^-_{\sigma,\tau}(J(x, y), G))}
\end{array}$$

Neutralization: $\mathcal{L}^1_{\exists^-_{\sigma,\tau}} = -1, \mathcal{L}^2_{\exists^-_{\sigma,\tau}} = 2$ and $\mathcal{A}^2_{\exists^-_{\sigma,\tau}} = \{1\}$ for all types σ, 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 types ρ, σ, τ;
Precedence: $(\forall \rho, \sigma, \tau)\ \{app_{\sigma,\tau}, \Pi^0_{\sigma,\tau}, \Pi^1_{\sigma,\tau}\} >_{\mathcal{F}} \exists^-_{new\ \rho,\tau}$ and $\exists^-_{new\ \rho,\tau} = \exists^-_{new\ \sigma,\tau}$

We carry out the computation for the most difficult rule, the last. The β-normalization of the full neutralization of both sides is:

$$\begin{array}{l}
\mathcal{FN}(\exists^-_{\sigma,\tau}(\exists^-_{\rho,c(o,\sigma)}(T, J), G))\downarrow_\beta = \\
\exists^-_{new\ \sigma,\tau}(\ m_{c(o,\sigma)}(\exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T), @(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T)))), \\
\qquad @(G,\ \bot_{c(o,\sigma)\to o}(\exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T), @(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T)))), \\
\qquad\qquad \bot_{c(o,\sigma)\to\sigma}(\exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T), @(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))))))
\end{array}$$

and,

$\mathcal{FN}(\exists^-_{\rho,\tau}(T, \lambda x\!:\!o\ y\!:\!\rho.\exists^-_{\sigma,\tau}(@(J,x,y),G)))\downarrow_\beta =$
$\exists^-_{new\ \rho,\tau}(m_{c(o,\rho)}(T), \exists^-_{new\ \sigma,\tau}(\ m_{c(o,\sigma)}(@(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))),$
$@(G,\ \bot_{c(o,\sigma)\to o}(@(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))),$
$\bot_{c(o,\sigma)\to\sigma}(@(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))))))$

We prove $s > t$ by case 4, yielding two subgoals:
(i) $\exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T), @(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))) > m_{c(o,\rho)}(T)$, and
(ii) $s > \exists^-_{new\ \sigma,\tau}(\ m_{c(o,\sigma)}(@(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))),$
$@(G,\ \bot_{c(o,\sigma)\to o}(@(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))),$
$\bot_{c(o,\sigma)\to\sigma}(@(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T)))))$

For (i) we apply case 1 twice (note that $m_{c(o,\rho)}(T)$ has minimal type).
For (ii) we apply case 4 again which leads us to two new subgoals:
(iii) $m_{c(o,\sigma)}(\exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T), @(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T)))) >$
$m_{c(o,\sigma)}(@(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T)))$, and
(iv) $@(G,\ \bot_{c(o,\sigma)\to o}(\exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T), @(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T)))),$
$\bot_{c(o,\sigma)\to\sigma}(\exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T), @(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))))) \geq$
$@(G,\ \bot_{c(o,\sigma)\to o}(@(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))),$
$\bot_{c(o,\sigma)\to\sigma}(@(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))))$

For (iii) we apply case 3 first and then case 1, and for (iv) we apply case 8, which leads us to prove:
$\bot_{c(o,\sigma)\to o}(\exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T), @(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T)))) \geq$
$\bot_{c(o,\sigma)\to o}(@(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T)))$.
which holds by case 3 and 1.

Example 7.5. The previous encoding of natural deduction is now made polymorphic. As a result, it becomes finite, hence amenable to automation.
Let $\mathcal{S} = \{o,\ c : * \times * \to *\}$, and $\mathcal{S}^\forall = \{\sigma, \tau, \rho\}$. Signature and rules follow.

$$\mathcal{F} = \{\ app : (\sigma\to\tau)\times\sigma\to\tau;\ \ abs : (\sigma\to\tau)\to(\sigma\to\tau);$$
$$\Pi : \sigma\times\tau\to c(\sigma,\tau);\ \Pi^0 : c(\sigma,\tau)\to\sigma;\ \Pi^1 : c(\sigma,\tau)\to\tau;$$
$$\exists^+ : o\times\sigma\to c(o,\sigma);\ \ \exists^- : c(o,\sigma)\times(o\to\sigma\to\tau)\to\tau\ \ \}$$

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

$$\begin{array}{rcl} app(abs(F),X) & \to & F(X) \\ \Pi^0(\Pi(X,Y)) & \to & X \end{array} \quad\Big\|\quad \begin{array}{rcl} \exists^-(\exists^+(Z,X),G) & \to & G(Z,X) \\ \Pi^1(\Pi(X,Y)) & \to & Y \end{array}$$

$$\begin{array}{rcl} app(\exists^-(T,H),X) & \to & \exists^-(T,\lambda x:o\ y:\rho.app(H(x,y),X)) \\ \Pi^0(\exists^-(T,I)) & \to & \exists^-(T,\lambda x:o\ y:\rho.\Pi^0(I(x,y))) \\ \Pi^1(\exists^-(T,I)) & \to & \exists^-(T,\lambda x:o\ y:\rho.\Pi^1(I(x,y))) \\ \exists^-(\exists^-(T,J),G) & \to & \exists^-(T,\lambda x:o\ y:\rho.\exists^-(J(x,y),G)) \end{array}$$

Statuses: $\exists^-_{\mathsf{new}} \in \mathsf{Lex}$ and $\mathsf{app}, \Pi^0, \Pi^1 \in \mathsf{Mul}$. Precedence: $\{app, \Pi^0, \Pi^1\} >_{\mathcal{F}} \exists^-_{new}$.

Neutralization: $\mathcal{L}^1_{\exists\_=} = -1, \mathcal{L}^2_{\exists\_=} = 2$ and $\mathcal{A}^2_{\exists\_=} = \{1\}$.

The computations are now exactly the same as for the monomorphic encoding of natural deduction done in Example 7.4.

8. REDUCING NORMAL TO PLAIN HIGHER-ORDER TERMINATION

In this section, we introduce a transformation that reduces the problem of normal higher-order termination to that of plain higher-order termination. Such a transformation is of course most useful since it allows one to prove normal termination by using already existing tools that can prove plain higher-order termination.

We proceed in two steps. First, we reduce normal termination of R to termination of the union of β-reduction and (S) modulo η, where S *simulates* R in a way that we are going to describe next. Here rewriting modulo η means rewriting on η-equivalence classes. Second, we show that termination of the union of β-reduction and (S) modulo η can itself be reduced to termination of $S' \cup \beta\eta$, where S' is obtained from S, although in most cases S' will be S itself. Moreover, we also show that, under some restrictions, termination of (S) modulo η union β-reduction can also be reduced to termination of $S' \cup \beta$, where S' is an η-expansion of S that operates on η-long forms. Let us give a precise definition of rewriting modulo η:

Definition 8.1. Given a set of higher-order rewrite rules R, we denote by $(R)_\eta$ the rewrite relation defined as $s' \longrightarrow_{(R)_\eta} t'$ if $s' =_\eta s \longrightarrow_R t =_\eta t'$.

8.1. Rigid simulation

We know that normal termination differs from plain termination because of the existence of versatile terms (or subterms). Our transformation will therefore eliminate these versatile subterms when possible.

Definition 8.2. A term l in β-normal form is a *rigid pattern* if (i) l is not a variable nor an abstraction; (ii) no subterm of l is a versatile application. A higher-order rewrite rule is *rigid* if its lefthand side is a rigid pattern. A higher-order rewrite system is *rigid* if so are its rewrite rules.

Being a rigid pattern is different from being a higher-order pattern [Miller 1991], for which variables *must* be applied to different bound variables, resulting in versatile applications. This superficial difference will be understood later.

LEMMA 8.3. *If l is a rigid pattern and t an arbitrary β-normal term s.t. $p \in \mathcal{P}os(t)$, then $t[l\gamma]_p$ is β-normal for every β-normal substitution γ.*

PROOF. The absence of a β-redex in $l\gamma$ follows from the absence of a versatile subterm in l. Since l is not a variable nor an abstraction, $t[l\gamma]_p$ has no β-redex. □

Term rewriting systems whose lefthand sides are not rigid patterns will be simulated by rigid systems:

Definition 8.4. A normal higher-order rewrite system R is *simulated* by a rigid system R_A iff, for any rule $l \to r \in R$, there is a pair made of a rigid rule $u \to v$ in R_A and an *abstracting substitution* γ, such that $u\gamma =_\eta l$ and $v\gamma \longrightarrow^*_{R_A \cup \beta} r' =_\eta r$.

Abstraction by fresh free variables is the main tool to obtain a rigid system from one which is not, therefore explaining our notation R_A.

Example 8.5. Consider the system $R = \{f(\lambda x.h(@(H, c(x)))) \to g(H)\}$, in which $H : \alpha \to \beta$ is the only free variable. R can be abstracted by

$$R_A = \{f(X) \to g(X), \quad h(Y) \to Y, \quad c(z) \to z\}$$

The rule in R is abstracted by the rule $f(X) \to g(X)$ added to R_A with $\{X \mapsto \lambda z.h(@(H, c(z)))\}$ as abstracting substitution, requiring that $\lambda z.h(@(H, c(z))) \longrightarrow^*_{R_A} \lambda z.@(H, z) =_\eta H$. Considering this property as a new rule, it is abstracted in turn by the rule $h(Y) \to Y$ in R_A with $\{Y \mapsto @(H, c(z))\}$ as abstracting substitution, requiring that $@(H, c(z)) \longrightarrow^*_{R_A} @(H, z)$. This new requirement is satisfied with the versatile subterm free rule $c(z) \to z$ added to R_A.

LEMMA 8.6. *Let R be a rigid system and s, t terms s.t. $s \xrightarrow[R]{p} t$ for some $p \in \mathcal{P}os(s)$. Then, $s{\downarrow_\beta} \longrightarrow^*_{R\cup\beta} t{\downarrow_\beta}$ (with $s{\downarrow_\beta} \longrightarrow^+_{R\cup\beta} t{\downarrow_\beta}$ in case $p = \wedge$).*

PROOF. Since $s \xrightarrow[R]{p} t$, there is a rule $u \to v$ in R and a substitution γ, such that $s|_p = u\gamma$ and $t = s[v\gamma]_p$. Since u is a rigid pattern, then $(u\gamma){\downarrow_\beta} = u(\gamma{\downarrow_\beta})$ by Lemma 8.3, and $u(\gamma{\downarrow_\beta}) \longrightarrow^+_R v(\gamma{\downarrow_\beta}) \longrightarrow^*_\beta v(\gamma{\downarrow_\beta}){\downarrow_\beta} = (v\gamma){\downarrow_\beta}$.

If $p = \wedge$ we are done. Otherwise, u is not an abstraction since R is a rigid system. Moreover, $u\gamma$ is not an abstraction either since u is not a variable, hence $s{\downarrow_\beta} = (s[u\gamma]_p){\downarrow_\beta} = s{\downarrow_\beta}\ [u\gamma{\downarrow_\beta}]_P$ where P is the (possibly empty) set of positions of z in $s[z]_p{\downarrow_\beta}$. Since $(u\gamma){\downarrow_\beta} \longrightarrow^+_{R\cup\beta} (v\gamma){\downarrow_\beta}$, then $s{\downarrow_\beta}\ [u\gamma{\downarrow_\beta}]_P \longrightarrow^*_{R\cup\beta} s{\downarrow_\beta}\ [v\gamma{\downarrow_\beta}]_P \longrightarrow^*_\beta (s{\downarrow_\beta}\ [v\gamma{\downarrow_\beta}]_P){\downarrow_\beta} = (s[v\gamma]){\downarrow_\beta}$. □

THEOREM 8.7. *Let R be a normal higher-order rewrite system simulated by a rigid system S. Then $\longrightarrow_{R_{\beta\eta}}$ is terminating if $\longrightarrow_{(S)_\eta \cup \beta}$ is terminating.*

PROOF. Assume that $S \cup \beta$ is terminating modulo η. Let then $\succ$ be the higher-order reduction ordering $\longrightarrow^+_{(S)_\eta\cup\beta}$ and $>$ be defined as $s > t$ iff there exist $u \to v \in S$ and γ such that $s = u\gamma$ and $v\gamma \longrightarrow^*_{S\cup\beta} t$. Note that this implies that $> \subseteq \succ^+$. By Theorem 3.8, we need to show that $(\succ, >)$ is a normal higher-order ordering, and for all $l \to r \in R$ then $l' > r'$ for some $l' =_\eta l$ and $r' =_\eta r$.

Since S simulates R, then for every rule $l \to r \in R$ there exists $u \to v \in S$ and a substitution γ s.t. $l' = u\gamma =_\eta l$ and $v\gamma \longrightarrow^*_{S\cup\beta} r'$ for some $r' =_\eta r$, which implies $l' > r'$ by definition of $>$.

We now show that $\succ$ is normal η-compatible, which means that for all β-normal typed terms s, s', t such that $s' =_\eta s \longrightarrow^+_{(S)_\eta\cup\beta} t$ there exists some β-

normal term $t' =_\eta t$ such that $s' \longrightarrow^+_{(S)_\eta \cup \beta} t'$. Since s is β-normal, we have that $s' =_\eta s \longrightarrow_{(S)_\eta} w \longrightarrow^*_{(S)_\eta \cup \beta} t$ for some term w and, by definition of $\longrightarrow_{(S)_\eta}$, we have $s' \longrightarrow_{(S)_\eta} w$, which directly implies that $s' \longrightarrow^+_{(S)_\eta \cup \beta} t$, hence we take t as t'.

We are left showing that it is normal stable when equipped with the relation $\succ$. Assume that $s > t$. By assumption, there is $l \to r$ and γ s.t. $s = l\gamma \longrightarrow^\Lambda_S r\gamma \longrightarrow^*_{S\cup\beta} t$. Given an arbitrary normal substitution θ, $s\theta \longrightarrow^\Lambda_S r\gamma\theta \longrightarrow^*_{S\cup\beta} t\theta$. By Lemma 8.6, $s\theta{\downarrow_\beta} \longrightarrow^+_{S\cup\beta} r\gamma\theta{\downarrow_\beta}$ and $r\gamma\theta{\downarrow_\beta} \longrightarrow^*_{S\cup\beta} t\theta{\downarrow_\beta}$, therefore $s\theta{\downarrow_\beta} \longrightarrow^+_{S\cup\beta} r\gamma\theta{\downarrow_\beta} \longrightarrow^*_{S\cup\beta} t\theta{\downarrow_\beta}$, hence, by definition, since $\longrightarrow_{S\cup\beta}$ is included in $\longrightarrow_{(S)_\eta\cup\beta}$, we have $s\theta{\downarrow_\beta} \succ t\theta{\downarrow_\beta}$. □

Example 8.8. Consider the rules of Example 7.4 (we could take the polymorphic version of Example 7.5 as well). By Theorem 5.19, we can apply first neutralization as an interpretation and then Theorem 8.7 to transform the resulting system to a new one whose plain termination implies that of normal rewriting with the original one.

Using the same neutralization as in Example 7.4, and new variables $V_\tau : \tau$ for all types τ, the rules resulting from abstraction are the following:

$$\begin{array}{rcl||rcl}
app_{\sigma,\tau}(abs_{\sigma,\tau}(F), X) & \to & F(X) & \exists^-_{new\ \sigma,\tau}(m_{c(o,\sigma)}(\exists^+_\sigma(Z,X)), V_\tau) & \to & V_\tau \\
\Pi^0_{\sigma,\tau}(\Pi_{\sigma,\tau}(X,Y)) & \to & X & \Pi^1_{\sigma,\tau}(\Pi_{\sigma,\tau}(X,Y)) & \to & Y \\
\bot_{c(o,\sigma)\to o}(\exists^+_\sigma(Z,X)) & \to & Z & \bot_{c(o,\sigma)\to\sigma}(\exists^+_\sigma(Z,X)) & \to & X
\end{array}$$

$$\begin{array}{rcl}
app_{\sigma,\tau}(\exists^-_{new\ \rho,\sigma\to\tau}(m_{c(o,\rho)}(T), V_{\sigma\to\tau}), X) & \to & \exists^-_{new\ \rho,\tau}(m_{c(o,\rho)}(T), app_{\sigma,\tau}(V_{\sigma\to\tau}, X)) \\
\Pi^0_{\sigma,\tau}(\exists^-_{new\ \rho,c(\sigma,\tau)}(m_{c(o,\rho)}(T), V_{c(\sigma,\tau)})) & \to & \exists^-_{new\ \rho,\tau}(m_{c(o,\rho)}(T), \Pi^0_{\sigma,\tau}(V_{c(\sigma,\tau)})) \\
\Pi^1_{\sigma,\tau}(\exists^-_{new\ \rho,c(\sigma,\tau)}(m_{c(o,\rho)}(T), V_{c(\sigma,\tau)})) & \to & \exists^-_{new\ \rho,\tau}(m_{c(o,\rho)}(T), \Pi^1_{\sigma,\tau}(V_{c(\sigma,\tau)})) \\
\exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T), V_{c(o,\sigma)})) & \to & V_{c(o,\sigma)} \\
\exists^-_{new\ \sigma,\tau}(m_{c(o,\sigma)}(\exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T), V_{c(o,\sigma)})), V_\tau) & \to & \\
\multicolumn{3}{r}{\exists^-_{new\ \rho,\tau}(m_{c(o,\rho)}(T), \exists^-_{new\ \sigma,\tau}(m_{c(o,\sigma)}(V_{c(o,\sigma)}), V_\tau))}
\end{array}$$

This system can be easily proved terminating by most of the existing tools for proving termination of higher-order rewriting.

Let us show in detail how the last rule of the original example is transformed into the last two given above. The original rule is

$$\exists^-_{\sigma,\tau}(\exists^-_{\rho,c(o,\sigma)}(T, J), G) \to \exists^-_{\rho,\tau}(T, \lambda x : o\ y : \rho.\exists^-_{\sigma,\tau}(J(x,y), G))$$

and after applying the same neutralization as in Example 7.4 to both sides of the rule, we obtain

$$\begin{array}{l}
\exists^-_{new\ \sigma,\tau}(\ m_{c(o,\sigma)}(\exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T), @(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T)))), \\
\qquad @(G,\ \bot_{c(o,\sigma)\to o}(\exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T), @(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T)))), \\
\qquad\qquad \bot_{c(o,\sigma)\to\sigma}(\exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T), @(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))))))) \to \\
\exists^-_{new\ \rho,\tau}(m_{c(o,\rho)}(T), \exists^-_{new\ \sigma,\tau}(\ m_{c(o,\sigma)}(@(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))), \\
\qquad\qquad @(G,\ \bot_{c(o,\sigma)\to o}(@(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))), \\
\qquad\qquad\qquad \bot_{c(o,\sigma)\to\sigma}(@(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))))))
\end{array}$$

Then the rule is abstracted by

$$\boxed{\begin{array}{l}\exists^-_{new\ \sigma,\tau}(m_{c(o,\sigma)}(\exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T),V_{c(o,\sigma)})),V_\tau) \to \\ \qquad \exists^-_{new\ \rho,\tau}(m_{c(o,\rho)}(T),\exists^-_{new\ \sigma,\tau}(m_{c(o,\sigma)}(V_{c(o,\sigma)}),V_\tau))\end{array}}$$

with the abstracting substitution γ

$$\left\{\begin{array}{lll} V_{c(o,\sigma)} & \mapsto & @(J,\bot_{c(o,\rho)\to o}(T),\bot_{c(o,\rho)\to\rho}(T)) \\ V_\tau & \mapsto & @(G,\bot_{c(o,\sigma)\to o}(\exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T),@(J,\bot_{c(o,\rho)\to o}(T),\bot_{c(o,\rho)\to\rho}(T)))), \\ & & \qquad \bot_{c(o,\sigma)\to\sigma}(\exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T),@(J,\bot_{c(o,\rho)\to o}(T),\bot_{c(o,\rho)\to\rho}(T))))) \end{array}\right\}$$

Following Definition 8.4, we need to add rules ensuring that the new righthand side instantiated by γ, namely $\exists^-_{new\ \rho,\tau}(m_{c(o,\rho)}(T),\exists^-_{new\ \sigma,\tau}(m_{c(o,\sigma)}(V_{c(o,\sigma)}),V_\tau))\gamma$, rewrites to a term η-equivalent to the original righthand side. No middle η-equivalent term is needed. Adding rules to the simulating system S so that

$$\begin{array}{l}\exists^-_{new\ \rho,\tau}(\ m_{c(o,\rho)}(T),\exists^-_{new\ \sigma,\tau}(m_{c(o,\sigma)}(@(J,\bot_{c(o,\rho)\to o}(T),\bot_{c(o,\rho)\to\rho}(T)), \\ \qquad @(G,\ \bot_{c(o,\sigma)\to o}(\ \exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T),@(J,\bot_{c(o,\rho)\to o}(T),\bot_{c(o,\rho)\to\rho}(T)))\ ), \\ \qquad\qquad \bot_{c(o,\sigma)\to\sigma}(\ \exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T),@(J,\bot_{c(o,\rho)\to o}(T),\bot_{c(o,\rho)\to\rho}(T)))\ )))))\end{array}$$

rewrites to

$$\begin{array}{l}\exists^-_{new\ \rho,\tau}(\ m_{c(o,\rho)}(T),\exists^-_{new\ \sigma,\tau}(m_{c(o,\sigma)}(@(J,\bot_{c(o,\rho)\to o}(T),\bot_{c(o,\rho)\to\rho}(T))), \\ \qquad @(G,\ \bot_{c(o,\sigma)\to o}(\ @(J,\bot_{c(o,\rho)\to o}(T),\bot_{c(o,\rho)\to\rho}(T))\ ), \\ \qquad\qquad \bot_{c(o,\sigma)\to\sigma}(\ @(J,\bot_{c(o,\rho)\to o}(T),\bot_{c(o,\rho)\to\rho}(T))\ ))))\end{array}$$

is enough, which can be achieved with the rule

$\exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T),@(J,\bot_{c(o,\rho)\to o}(T),\bot_{c(o,\rho)\to\rho}(T))) \to$
$@(J,\bot_{c(o,\rho)\to o}(T),\bot_{c(o,\rho)\to\rho}(T))$

Since its lefthand side is not a rigid pattern, it must be abstracted again by the rule

$$\boxed{\exists^-_{new\ \rho,c(o,\sigma)}(m_{c(o,\rho)}(T),V_{c(o,\sigma)})) \to V_{c(o,\sigma)}}$$

with the abstracting substitution $\{V_{c(o,\sigma)} \mapsto @(J,\bot_{c(o,\rho)\to o}(T),\bot_{c(o,\rho)\to\rho}(T))\}$, and we are done since the new righthand side with the substitution applied coincides with the original one. Hence framed rules a part of the simulating system. □

Our result allows one to show normal termination of R by checking for higher-order termination of the rules of the rigid system S modulo η.

Computing a rigid system S simulating a given non-rigid system R is not always possible, but succeeds often in practice. The way used in the previous examples is the following: abstract by a fresh free variable the smallest subterm of l containing an outermost versatile subterm, and whose free variables are included in that of l. This defines the abstracting substitution γ. Then, the righthand side is constructed so as to ensure the (stronger than needed) property $v\gamma \longrightarrow_{R_A} r' =_\eta r$ in Definition 8.4. Here is one more example using this transformation strategy:

Example 8.9. Let $H : \alpha \to \beta \to \rho$, $x : \alpha$ and $y : \beta$ be variables, and

$$R = \{f(h_1(g(x,y)),@(@(H,h_2(g(x,y))),h_3(g(x,y)))) \to @(@(H,x),y)\}$$

which can be abstracted by the three rules

$$\{f(h_1(g(x,y)),Z_1)\to Z_1,\quad h_2(g(x,y)))\to x,\quad h_3(g(x,y))))\to y\}$$

Let us now consider a new system with a much bigger rule:

$$\begin{array}{l} f_1(m_1(f_2(m_2(T),@(J,h_1(T),h_2(T)))),\\ \quad @(G,h_3(f_2(m_2(T),@(J,h_1(T),h_2(T)))),\\ \quad\quad h_4(f_2(m_2(T),@(J,h_1(T),h_2(T))))))\to\\ f_3(m_3(T),f_1(m_1(@(J,h_1(T),h_2(T))),\\ \quad @(G,h_3(@(J,h_1(T),h_2(T))),\\ \quad\quad h_4(@(J,h_1(T),h_2(T))))))\end{array}$$

which contains the free variables $J:\alpha\to\beta\to\rho$ and $T:o$. The abstraction includes the following additional three rigid rules:

$$\begin{array}{c}\{f_1(m_1(f_2(m_2(T),X_J)),X_G)\to f_3(m_3(T),f_1(m_1(X_J),X_G)),\\ h_3(f_2(m_2(T),X_J))\to h_3(X_J),\quad h_4(f_2(m_2(T),X_J))\to h_4(X_J)\}\end{array}$$

8.2. From termination modulo η to termination union η-reduction

We now turn to η-compatibility. Regarding termination, η-compatibility can easily be replaced by one-side η-compatibility, that is, either $s' =_\eta s \succ t$ implies $s'\succ t$ or $s\succ t=_\eta t'$ implies $s\succ t'$. This fact is used to show that termination modulo η can be guaranteed by showing termination union η-reduction (after some minor transformation). Therefore, we can apply any technique that ensures termination union η-reduction, which is the case of orderings like HORPO or CPO, in order to show termination modulo η of the original system. As shown below, since the result is proved by working on η-normalized terms, we only have to η-normalize both sides of the rules and add a few rules needed to catch η-reductions that may involve part of the rule and part of the context. For instance if we have a rule $@(a,x)\to b$ in our rewrite system, then $\lambda x.@(a,x)$ can be rewritten into $\lambda x.b$, but its η-normalization a cannot be rewritten by the η-normalization of $@(a,x)\to b$ which is the same rule. We therefore need to extend the set of rules by adding the η-normalization of the rules with all possible contexts which create η-redexes on top, and then η-normalize these rules. Apart from the extra η-normalization phase, these extensions are similar in spirit to Peterson and Stickel's notion of associative commutative extension [Peterson and Stickel 1981], and its generalization by Jouannaud and Kirchner [Jouannaud and Kirchner 1986].

Definition 8.10. Let R be a higher-order term rewrite system. Let

$$\begin{array}{rcl} R{\downarrow_\eta} &=& \{l{\downarrow_\eta}\to r{\downarrow_\eta}|\ l\to r\in R\}\\ R^\eta &=& R{\downarrow_\eta}\cup\{\ (\lambda x_i\dots x_n.l){\downarrow_\eta}\to(\lambda x_i\dots x_n.r){\downarrow_\eta}|\ l\to r\in R,\ i\in[1..n],\\ && \qquad l=@(l',x_1,\dots,x_n),\ (\forall j\neq k)\,x_j\neq x_k,\text{ and }(\forall j)\,x_j\notin\mathcal{V}ar(l')\}\end{array}$$

For example, if R contains a rule $@(a,x,y) \to b$ then R^η contains the rules

$$\{@(a,x,y) \ \to \ b, \quad @(a,x) \ \to \ \lambda y.b, \quad a \ \to \ \lambda xy.b\}$$

THEOREM 8.11. *Let R be a higher-order term rewrite system. Then $\longrightarrow_{(R)_\eta \cup \beta}$ is terminating if $\longrightarrow_{R^\eta \cup \beta \cup \eta}$ is terminating.*

PROOF. We show that $s' =_\eta s \longrightarrow_R t =_\eta t'$ implies $s'{\downarrow_\eta} \longrightarrow^+_{R^\eta \cup \eta} t'{\downarrow_\eta}$ and $s' =_\eta s \longrightarrow_\beta t =_\eta t'$ implies $s'{\downarrow_\eta} \longrightarrow^*_{\beta \cup \eta} t'{\downarrow_\eta}$. By confluence of η, $s'{\downarrow_\eta} = s{\downarrow_\eta}$ and $t{\downarrow_\eta} = t'{\downarrow_\eta}$. We therefore need to show that $s \longrightarrow_R t$ and $s \longrightarrow_\beta t$ imply $s{\downarrow_\eta} \longrightarrow^+_{R^\eta \cup \eta} t{\downarrow_\eta}$ and $s{\downarrow_\eta} \longrightarrow^*_{\beta \cup \eta} t{\downarrow_\eta}$ respectively.

— Suppose $s = s[l\sigma]_p$ and $t = s[r\sigma]_p$ for some rule $l \to r$ in R, substitution σ and position p in s. We proceed by induction on the size of s.
  — If $p = \Lambda$ then it holds since, by Property 8 of λ-calculus, we have $l\sigma{\downarrow_\eta} = l{\downarrow_\eta}$ $\sigma{\downarrow_\eta}$ and $r\sigma{\downarrow_\eta} = r{\downarrow_\eta}$ $\sigma{\downarrow_\eta}$ and $l{\downarrow_\eta} \to r{\downarrow_\eta}$ in R^η.
  — If $s = \lambda x_i \dots x_n.l\sigma$ and $l = @(l', x_1, \dots, x_n)$, where $n > 0$, all x_j are different, $x_j \notin \mathcal{V}ar(l')$ and $x_j \notin \mathcal{D}om(\sigma)$, there is a rule $@(l', x_1, \dots, x_{i-1}){\downarrow_\eta} \to \lambda x_i \dots x_n.r$ ${\downarrow_\eta}$ in R^η. It follows that s ${\downarrow_\eta} = @(l', x_1, \dots, x_{i-1})\sigma$ ${\downarrow_\eta} = @(l', x_1, \dots, x_{i-1}){\downarrow_\eta}\, \sigma{\downarrow_\eta} \to_{R^\eta} \lambda x_i \dots x_n.r{\downarrow_\eta}\, \sigma{\downarrow_\eta} = s[r\sigma]{\downarrow_\eta}$.
  — If $s = \lambda x.@(s'[l\sigma]_{p'}, x)$ with $x \notin \mathcal{V}ar(s'[l\sigma]_{p'})$ then $s{\downarrow_\eta} = s'[l\sigma]_{p'}{\downarrow_\eta}$. Further, since rewriting cannot introduce new variables, $t = \lambda x.@(s'[r\sigma]_{p'}, x)$ with $x \notin \mathcal{V}ar(s'[r\sigma]_{p'})$, hence $t{\downarrow_\eta} = s'[r\sigma]_{p'}{\downarrow_\eta}$. Induction hypothesis concludes.
  — Otherwise, if $s = \lambda x.s'[l\sigma]_{p'}$ then $s{\downarrow_\eta} = \lambda x.(s'[l\sigma]_{p'}{\downarrow_\eta})$, and $t = \lambda x.s'[r\sigma]_{p'}$ with $t{\downarrow_\eta} = \lambda x.(s'[r\sigma]_{p'}{\downarrow_\eta}){\downarrow_\eta}$, by confluence of η-reduction. Now, by induction hypothesis, we have $s'[l\sigma]_{p'}{\downarrow_\eta} \longrightarrow^+_{R^\eta \cup \eta} s'[r\sigma]_{p'}{\downarrow_\eta}$, and hence $s{\downarrow_\eta} = \lambda x.(s'[l\sigma]_{p'}{\downarrow_\eta}) \longrightarrow^+_{R^\eta \cup \eta} \lambda x.(s'[r\sigma]_{p'}{\downarrow_\eta}) \longrightarrow^*_\eta t{\downarrow_\eta}$.
  — If $s = f(s_1 \dots s'[l\sigma]_{p'} \dots s_n)$ then $s{\downarrow_\eta} = f(s_1{\downarrow_\eta} \dots s'[l\sigma]_{p'}{\downarrow_\eta} \dots s_n{\downarrow_\eta})$, and $t = f(s_1 \dots s'[r\sigma]_{p'} \dots s_n)$ with $t{\downarrow_\eta} = f(s_1{\downarrow_\eta} \dots s'[r\sigma]_{p'}{\downarrow_\eta} \dots s_n{\downarrow_\eta})$. Now, by induction hypothesis, we have $s'[l\sigma]_{p'}{\downarrow_\eta} \longrightarrow^+_{R^\eta \cup \eta} s'[r\sigma]_{p'}{\downarrow_\eta}$, and hence $s{\downarrow_\eta} = f(s_1{\downarrow_\eta} \dots s'[l\sigma]_{p'}{\downarrow_\eta} \dots s_n{\downarrow_\eta}) \longrightarrow^+_{R^\eta \cup \eta} f(s_1{\downarrow_\eta} \dots s'[r\sigma]_{p'}{\downarrow_\eta} \dots s_n{\downarrow_\eta}) = t{\downarrow_\eta}$.
  — If $s = @(s'[l\sigma]_{p'}, s_2)$ or $s = @(s_1, s'[l\sigma]_{p'})$ it holds as in the previous case.
— Suppose $s = s[@(\lambda x.u, v)]_p$ and $t = s[u\{x \mapsto v\}]_p$ for some position p in s. There are two cases.
  — If $u = @(u', x)$ with $x \notin \mathcal{V}ar(u')$ then, by confluence of η-reduction, $s{\downarrow_\eta} = s[@(u', v)]_p{\downarrow_\eta}$, and since $t = s[@(u', x)\{x \mapsto v\}]_p = s[@(u', v)]_p$, we have that $s{\downarrow_\eta} = t{\downarrow_\eta}$ and thus trivially, $s{\downarrow_\eta} \longrightarrow^*_{\beta \cup \eta} t{\downarrow_\eta}$.
  — Otherwise $s{\downarrow_\eta} = s[z]_p{\downarrow_\eta}$ $\{z \mapsto @(\lambda x.u{\downarrow_\eta}, v{\downarrow_\eta})\}$, for a fresh variable $z \in \mathcal{V}ar(s[z]_p{\downarrow_\eta})$ by property of η. It follows that $s{\downarrow_\eta} \longrightarrow_\beta s[z]_p{\downarrow_\eta}$ $\{z \mapsto u{\downarrow_\eta}$ $\{x \mapsto v{\downarrow_\eta}\}\}$. Therefore, by confluence of η-reductions, $s{\downarrow_\eta} \longrightarrow^+_{\beta \cup \eta} t{\downarrow_\eta}$.

To conclude, we show that if $\longrightarrow_{R^\eta \cup \beta \cup \eta}$ is terminating then there cannot be an infinite sequence with $\longrightarrow_{(R)_\eta \cup \beta}$. Assume there is an infinite sequence

$s_0 \longrightarrow_{(R)_\eta \cup \beta} s_1 \longrightarrow_{(R)_\eta \cup \beta} \ldots$. By the previous property, we get an infinite sequence $s_0{\downarrow_\eta} \longrightarrow^*_{R^{\eta \cup \beta \cup \eta}} s_1{\downarrow_\eta} \longrightarrow^*_{R^{\eta \cup \beta \cup \eta}} \ldots$. Since β-reduction is terminating there must be infinitely many R^η-steps, hence infinitely many $(R)_\eta$-steps in the original sequence, contradicting our assumption. □

8.3. From termination modulo η to plain termination on η-long forms

We reduce here termination modulo η to plain termination again, but using this time η-expanded forms. The question was first considered in Kop's PhD thesis [Kop 2012], Sections 2.2 and 2.3, where it is shown that termination of R modulo η can be ensured by proving termination of some η-expansion of R, provided the lefthand sides of rules in R are higher-order patterns.

Once R is η-expanded, any termination technique for plain higher-order systems can be used. But since η-expansion relies on types, and type instantiation does not preserve η-long forms, these results do not accomodate polymorphism.

Kop considers a restricted form of η-expansion where a subterm u is expanded inside its superterm s, denoted $s[u] \hookrightarrow_\eta s[\lambda x.@(u, x)]$ if

— $x : \sigma$ is a fresh variable;
— $u : \sigma \to \tau$ is not
(i) a free variable; (ii) an abstraction; (iii) the left argument of an application; (iv) an application of the form $@(y, t_1, \ldots, t_n)$ where y is free in s and $n > 0$; (v) any of the t_i's in $@(y, t_1, \ldots, t_n)$ whenever y is free in s.

The η-long form of s, denoted by $s{\uparrow^\eta}$, is the normal form of s with respect to restricted η-expansion. The η-long form of a rewrite system R is the set

$$R{\uparrow^\eta} = \{\ @(l, z_1, \ldots, z_n){\uparrow^\eta} \to @(r, z_1, \ldots, z_n){\uparrow^\eta} \ :\ l \to r \in R \\ l : \tau_1 \to \ldots \tau_n \to o,\ o \text{ is a base type and} (\forall i \leq n)\ z_i : \tau_i \text{ is fresh}\}$$

Kop proves the following result (2.15(7) in [Kop 2012]):

LEMMA 8.12. *Let R be a set of rules in η-long form, whose lefthand sides are patterns not headed by λ. Then $s \longrightarrow_{(R \cup \beta)} t$ implies $s{\uparrow^\eta} \longrightarrow^+_{R{\uparrow^\eta} \cup \beta} t{\uparrow^\eta}$.*

We shall now reduce termination modulo η to plain termination via the computation of η-long forms in Kop's sense, hence allowing us to use Lemma 8.12. A non-trivial obstacle is that different η-equivalent terms may have different η-long forms. The reason originates in the syntactic restrictions in the definition of η-expansion that ensure termination, when the subterm of arrow type to be expanded is a free variable or the first argument of an application. For an example, $@(\lambda x.@(a, x), b)$ and $@(a, b)$ are different η-equivalent η-long forms (in her sense).

In order to avoid the problem with free variables we show termination by proving the absence of infinite sequences of closed terms, as already done in Theorem 3.8.

The second problem is more delicate, and that is where we need to introduce a new tool. Let $\beta\cap\eta$ be a new rewrite system encoding the intersection of β and η reductions via their two critical pairs:

$$\begin{array}{rcll} @(\lambda x.@(u,x),v) & \to & @(u,v) & \textbf{if } x \notin \mathcal{V}ar(u) \\ \lambda x.@(\lambda y.u,x) & \to & \lambda y.u & \textbf{if } x \notin \mathcal{V}ar(\lambda y.u) \end{array}$$

.

Clearly, $\beta\cap\eta$-reductions is included in β- and η-reductions. We now investigate the properties of this rewrite system.

LEMMA 8.13 (DIAMOND). *The rewrite system $\beta\cap\eta$ is strongly confluent.*

PROOF. Using Huet's characterization of strong confluence for linear systems [Huet 1980], we need to prove that the 4 critical pairs are strongly confluent.

(1) First rule with itself. The superposition term is $s = @(\lambda x.@(\lambda y.@(u,y),x),v)$, with $x \notin \mathcal{V}ar(\lambda y.@(u,y))$ and $y \notin \mathcal{V}ar(u)$. Then $s \longrightarrow @(\lambda x.@(u,x),v)$ and $s \longrightarrow @(\lambda y.@(u,y),v)$, both terms rewriting in one step to $@(u,v)$.
(2) Second rule with itself. The superposition term is $s = \lambda x.@(\lambda y.@(\lambda z.u,y),x)$, with $x,y \notin \mathcal{V}ar(\lambda z.u)$. Then $s \longrightarrow \lambda x.@(\lambda z.u,x)$ and $s \longrightarrow \lambda y.@(\lambda z.u,y)$, both terms rewriting in one step to $\lambda z.u$.
(3) Second rule inside the first. The superposition term is $s = @(\lambda x.@(\lambda y.u,x),v)$, with $x \notin \mathcal{V}ar(\lambda y.u)$. Then s rewrites to $@(\lambda y.u,v)$ in both cases.
(4) First rule inside the second. The superposition term is $s = \lambda x.@(\lambda y.@(u,y),x)$, with $x \notin \mathcal{V}ar(\lambda y.@(u,y))$ and $y \notin \mathcal{V}ar(u)$. Then s rewrites to $\lambda x.@(u,x)$ with the first rule and $\lambda y.@(u,y)$ with the second, both terms being equal up to α-renaming. □

Being confluent and terminating, the rewrite system $\beta\cap\eta$ enjoys the unique normal form property. We denote by $u{\downarrow_{\beta\cap\eta}}$ the normal form of u with respect to $\beta\cap\eta$.

LEMMA 8.14. *Let u,v be two terms such that $u \longrightarrow_R v$, where R is a higher-order rewrite system whose lefthand sides are patterns. Then, $u{\downarrow_{\beta\cap\eta}} \longrightarrow^+_{R\cup\beta} v{\downarrow_{\beta\cap\eta}}$.*

PROOF. The proof is by induction on u compared with $\longrightarrow^+_{\beta\cap\eta}$. Let $u \xrightarrow[R]{p} v$. If $u = u{\downarrow_{\beta\cap\eta}}$, we are done. Otherwise, $u \xrightarrow[\beta\cap\eta]{q} u'$. We show by cases on p,q that $u' \longrightarrow_R v'$ and $v \longrightarrow_{\beta\cap\eta} v'$ for some v' and conclude y induction hypothesis on u' and confluence of $\beta\cap\eta$ which ensures that $u{\downarrow_{\beta\cap\eta}} = u'{\downarrow_{\beta\cap\eta}}$ and $v{\downarrow_{\beta\cap\eta}} = v'{\downarrow_{\beta\cap\eta}}$.

(1) If p and q are disjoint, both steps commute as needed.
(2) $p \geq q$. Since lefthand sides of rules in R are rigid patterns, their lefthand side is not an abstraction nor a β-redex, hence no overlap with $\beta\cap\eta$ is possible. Therefore, the R-redex is inside the substitution of the other redex, and since $\beta\cap\eta$ is linear, both steps must commute and the result holds as above.

(3) $q > p$, hence $p = q \cdot m$ for some position m. This is the difficult case.
Assume first that $m \in \mathcal{P}os(l)$ and $l|_m$ is an application, hence there is an overlap between the R-rule and the first rule in $\beta \cap \eta$. Then $l|_m\sigma = @(\lambda x.@(w_1, x), w_2)$. Since lefthand sides of rules in R are β-normal, $l|_m$ cannot have an abstraction as first argument of the application, hence it can only be a variable. But this is impossible too by our assumption that l is a rigid pattern.
Assume now that $m \in \mathcal{P}os(l)$ and $l|_m$ is an abstraction, implying an overlap between the R-rule and the second rule in $\beta \cap \eta$. Then $l|_m\sigma = \lambda x.@(\lambda y.w, x)$ with $x \notin \mathcal{V}ar(\lambda y.u)$. Higher-order substitutions do not capture variables, hence $l|_m = \lambda x.@(w', x)$, and w' must be an abstraction or a free variable. Impossible.
We are left with the case where no overlap occurs, which holds as before. □

LEMMA 8.15. *Given two terms u, v such that $u \longrightarrow_\beta v$ then $u{\downarrow_{\beta\cap\eta}} \longrightarrow^*_\beta v{\downarrow_{\beta\cap\eta}}$.*

PROOF. Let $u \xrightarrow[\beta]{p} v$, hence $u|_p = @(\lambda x.s', t')$. The proof is by induction on u. If $u = u{\downarrow_{\beta\cap\eta}}$, we are done. Otherwise, $u \xrightarrow[\beta\cap\eta]{q} u'$ for some term u' and position q. We show that $u' \longrightarrow_\beta v''$ and $v \longrightarrow^*_{\beta\cap\eta} v''$ and conclude by induction hypothesis on u'.

(1) If the position q is disjoint from p, or at a strict prefix of p without overlapping p, or inside s', or inside t', then the same $\beta\cap\eta$-rewrite takes place in v (possibly many times in the latter case), yielding the result.
(2) If $u|_q = @(\lambda y.@(\lambda x.s', y), t)$, with $t' = y \notin \mathcal{V}ar(\lambda x.s')$. Then u' and v are equal up to variable renaming.
(3) If $u|_q = \lambda y.@(\lambda x.s', y)$, that is, $t' = y \notin \mathcal{V}ar(\lambda x.s')$. Then u' and v are equal up to variable renaming again. □

LEMMA 8.16. *Let u, v be η-equivalent, $\beta\cap\eta$-normal closed terms. Then $u{\uparrow^\eta} = v{\uparrow^\eta}$.*

PROOF. We first prove that if u is closed and normal with respect to $\beta \cap \eta$ and $u \longrightarrow_\eta u'$ then $u{\uparrow^\eta} = u'{\uparrow^\eta}$.

By assumption we have that there is a position p in u such that $u|_p = \lambda x.@(w, x)$ with $x \notin \mathcal{V}ar(w)$ and $u' = u[w]_p$. Moreover, u' is also closed and normal with respect to $\beta \cap \eta$. Now, we show that, $u' = u[w] \hookrightarrow_\eta u[\lambda x.@(w, x)] = u$, which implies that $u{\uparrow^\eta} = u'{\uparrow^\eta}$.

Assume that the step $u[w] \hookrightarrow_\eta u[\lambda x.@(w, x)]$ cannot be applied, then, since u' is closed there are only two possible reasons.

— If w is an abstraction of the form $\lambda y.s$ then, we have that $u|_p = \lambda x.@(\lambda y.s, x)$, which contradicts the fact that u is normal with respect to $\beta \cap \eta$.
— If w is the left argument of an application then, with q the position just above p, we have that $u_q = @(\lambda x.@(w, x), w')$ for some w', which again contradicts the fact that u is normal with respect to $\beta \cap \eta$.

To conclude, from $u \longrightarrow_\eta u'$ implies $u\uparrow^\eta = u'\uparrow^\eta$, since u' is also closed and normal with respect to $\beta \cap \eta$, we have by easy induction that $u\uparrow^\eta = u{\downarrow_\eta}\uparrow^\eta$, for every closed term and normal with respect to $\beta \cap \eta$. Therefore, we have $u\uparrow^\eta = u{\downarrow_\eta}\uparrow^\eta$ and $v\uparrow^\eta = v{\downarrow_\eta}\uparrow^\eta$, and $s{\downarrow_\eta} = v{\downarrow_\eta}$, since $u =_\eta v$, and we conclude. □

We are now ready for the main result of this section:

THEOREM 8.17. *Let R be a higher-order term rewrite system, where all left-hand sides are rigid patterns. Then $\longrightarrow_{(R)_\eta \cup \beta}$ is terminating if $\longrightarrow_{R\uparrow^\eta \cup \beta}$ is terminating on terms in η-long form.*

PROOF. Let s and t be two closed terms. First we show that if $s \longrightarrow_{(R)_\eta} t$ then $s{\downarrow_{\beta\cap\eta}}\uparrow^\eta \longrightarrow^+_{R\uparrow^\eta\cup\beta} t{\downarrow_{\beta\cap\eta}}\uparrow^\eta$. By definition, $s =_\eta s' \longrightarrow_R t' =_\eta t$ and, by Lemma 8.14, $s'{\downarrow_{\beta\cap\eta}} \longrightarrow^+_{R\cup\beta} t'{\downarrow_{\beta\cap\eta}}$. This implies that $s{\downarrow_{\beta\cap\eta}} =_\eta s'{\downarrow_{\beta\cap\eta}} \longrightarrow^+_{R\cup\beta} t'{\downarrow_{\beta\cap\eta}} =_\eta t{\downarrow_{\beta\cap\eta}}$, and by Lemma 8.12 and 8.16, $s{\downarrow_{\beta\cap\eta}}\uparrow^\eta = s'{\downarrow_{\beta\cap\eta}}\uparrow^\eta \longrightarrow^+_{(R\uparrow^\eta\cup\beta)} t'{\downarrow_{\beta\cap\eta}}\uparrow^\eta = t{\downarrow_{\beta\cap\eta}}\uparrow^\eta$.

Similarly, by Lemma 8.15 first and then Lemma 8.12, we conclude that $s \longrightarrow_\beta t$ implies $s{\downarrow_{\beta\cap\eta}}\uparrow^\eta \longrightarrow^*_{R\uparrow^\eta\cup\beta} t{\downarrow_{\beta\cap\eta}}\uparrow^\eta$.

We now show the absence of infinite sequences of closed terms. Let $s_0 \longrightarrow_{(R)_\eta\cup\beta} s_1 \longrightarrow_{(R)_\eta\cup\beta} \dots$ be one. Since β alone is terminating, there must be infinitely many R-steps. By the previous two properties, we get an infinite sequence $s_{i_0}{\downarrow_{\beta\cap\eta}}\uparrow^\eta \longrightarrow^+_{(R\uparrow^\eta\cup\beta)} s_{i_1}{\downarrow_{\beta\cap\eta}}\uparrow^\eta \longrightarrow^+_{(R\uparrow^\eta\cup\beta)} \dots$ with $j \le i_j < i_{j+1}$ for all j, contradicting the termination assumption of $\longrightarrow_{R\uparrow^\eta\cup\beta}$ on terms in η-long form. □

The method based on η-expansions has indeed several drawbacks: termination proof methods are better suited to η-reductions than η-expansions, which introduces abstractions; it requires lefthand sides of rules to be rigid patterns, a property which may not be compatible with other transformation methods and, moreover, it is not compatible with polymorphism. Despite the fact that η-long normal forms are often used in the implementations we are aware of, we believe that this method has less potential than the one based on η-reductions.

9. RELATED WORK

Proving termination properties of Nipkow's rewriting was considered first in [van de Pol and Schwichtenberg 1995], then in [Jouannaud and Rubio 2006], and later in [Blanqui 2007; Kusakari et al. 2009; Blanqui 2006]. Related investigations are carried out in [van Raamsdonk 2001].

Van de Pol was the first to look for and propose a solution to the problem of normal higher-order termination [van de Pol and Schwichtenberg 1995]. His solution, however, is a *methodology* needing important user-interaction to build an interpretation-based ordering first, and then prove that the ordering constructed has the required compatibility, monotonicity, and stability properties. As a methodology, it is quite powerful: van de Pol was able to prove many difficult terminating examples, some of which are listed in this paper. Although there is no hope to

succeed automating this technique, a partial, powerful approximation is presented in [Fuhs and Kop 2012] for the case of simple types, where automatable higher-order polynomial interpretations are considered. How to extend this technique to polymorphic types, which are needed in example 7.5, is however unclear.

Inspired by van de Pol's work, we then developed a dedicated version of HORPO, which was indeed the first automatable, implemented method [Jouannaud and Rubio 2006]. Most ingredients of the present work were introduced there, in a form that was strongly tied to HORPO.

A third method is based on the notion of general schema as formulated in [Blanqui et al. 1999; Blanqui et al. 2002] where the notion of computability closure was introduced. The computability closure of a term l is a set $C(l)$ such that a set of higher-order rules R including β is terminating provided $r \in C(l)$ for each rule $l \to r \in R \setminus \beta$. In [Blanqui 2007], Blanqui shows that the computability closure makes it possible to define a well-founded order that contains the very first version of HORPO, where the order on types was just a congruence [Jouannaud and Rubio 1999], and can also be adapted to consider normal termination of higher-order rules whose lefthand sides are patterns in Miller's sense [Miller 1991], an assumption which is fundamental in his work. On the other hand, it is shown in [Blanqui et al. 2013] that HORPO enhanced with the computability closure [Jouannaud and Rubio 2007], an order named CHORPO, is included in CPO, the computability path ordering [Blanqui et al. 2008; Blanqui et al. 2013]. Since these orderings are much easier to use and implement than the computability closure whose implementation involves search, we believe that the methods presented here, which can be easily adapted to CPO, overrun those based on the direct use of the computability closure.

Finally, higher-order dependency pairs have also been used for normal termination in [Sakai et al. 2001; Sakai and Kusakari 2005; Kusakari et al. 2009; Suzuki et al. 2011; Blanqui 2006; Kop and van Raamsdonk 2010]. Patterns play an important role there too. A good survey of all these techniques is given in [Kop 2012], where the author also presents a general framework based on transforming different higher-order formalisms, including pattern higher-order systems, into a single one, namely *algebraic functional systems with meta-variables*. However, as is the case of the computability closure, methods based on dependency pairs require lefthand sides of rules to be headed by a function symbol, a restriction that we do not have and believe can be too restrictive for practice.

We do not really know how the dependency pairs method compares with ours. Comparing NHORPO with dependency pairs is not really fair as NHORPO inherits all known weaknesses of RPO that are far improved by the dependency pairs method. Still, as far as we know, the monomorphic version of Example 7.5 cannot be proved with the existing dependency pairs techniques. For the most part, this is due to the notion of neutralization which allowed us to carry out many quite difficult normal termination proofs. Neutralization is a general technique, as we have shown

here, that is applied to transform the rewrite system before proving termination and hence other methods can benefit from its use. It could indeed be applied before using the dependency pair method or even try to include it as a *processor* to enhance the strength of dependency pairs framework in the higher-order setting. However, neutralization may transform a pattern into a non-pattern, as we have seen with Example 7.1, and hence it cannot be used with its full strength by methods which require lefthand sides of rules to be patterns, as is the case of all methods described above but ours. Note however that in all such cases, the bound variable argument of a free higher-order variable becomes a constant $\perp$ of the same type after neutralization. It is quite possible that the definition of a higher-order pattern can be extended to handle such a simple case without loosing its numerous virtues. In particular, note that the most general unifier of the equation $X(\perp) = v$ is $X = \lambda x.v$, where $x \notin \mathcal{V}ar(v)$, which also is the most general unifier of the equation $X(x) = v$, with $x \in \mathcal{V}ar(v)$, found in pattern unification.

Note finally that the results given in Section 8 provide a way to combine neutralization with higher-order dependency pairs techniques, since the transformed systems fulfill the pattern condition. Generally speaking, an important part of our work can be seen as develop ping preprocessors for proving termination of higher-order rewriting. Neutralization and the transformation techniques described in Section 8 are such preprocessors which allow the user to use existing techniques aiming at proving termination of higher-order rewriting as a back-end. They are also compatible with weak polymorphism, a key, novel feature of our work. We have shown how powerful these preprocessors are.

10. CONCLUSION

The methods we have presented here for normal termination of a set of higher-order rewrite rules are very general and provide easily implementable normal higher-order orderings. The user has to provide a precedence and statuses as is usual with the recursive path ordering. He or she may also provide neutralization levels together with filters selecting appropriate arguments for each function symbol. This requires some expertise, but can be automated by searching non-deterministically for appropriate neutralization levels and filters, as done by many implementations for the precedence and statuses required by the recursive path ordering. The list of examples given in the paper shows the strength of our approach. Some are indeed complex enough so that their termination is by no means obvious even to the expert.

Some of our methods are based on reduction orderings but others are just transformations techniques applied to the rewrite system before using any other termination technique such as those based on the computational closure or on higher-order dependency pairs. Implementating these techniques will show which are most useful.

We believe that there is room for carrying out this research further. Much can be done to improve the techniques for showing higher-order termination, which will

then benefit to normal higher-order termination. Indeed the higher-order recursive path ordering itself has been already generalized, to semantic precedences [Borralleras and Rubio 2001], and to the calculus of constructions [Walukiewicz-Chrzaszcz 2003]. Generalization of all existing techniques to handle richer type disciplines should be a next step. Search for effective transformations reducing normal termination to plain higher-order termination should also be carried on.

Acknowledgments: The referees pointed out a few inaccuracies, imprecise definitions and proofs. We thank them warmly. We also thank Cynthia Kop for scrutinizing the paper as nobody else would have done.

REFERENCES

1991. *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*. IEEE Computer Society.

Hendrick Pieter Barendregt. 1993. Lambda Calculus with Types. In *Handbook of Logic in Computer Science, Volume 2*. Oxford Univ. Press, 117–309. eds. Abramsky et al.

Frédéric Blanqui. 2006. Higher-order dependency pairs. In *8th International Workshop on Termination - WST 2006*. Seattle, United States. http://hal.inria.fr/inria-00084821

Frédéric Blanqui. 2007. Computability Closure: Ten Years Later. In *Rewriting, Computation and Proof (Lecture Notes in Computer Science)*, Hubert Comon-Lundh, Claude Kirchner, and Hélène Kirchner (Eds.), Vol. 4600. Springer, 68–88.

Frédéric Blanqui, Jean-Pierre Jouannaud, and Mitsuhiro Okada. 1999. The Calculus of algebraic Constructions. In *RTA (Lecture Notes in Computer Science)*, Paliath Narendran and Michaël Rusinowitch (Eds.), Vol. 1631. Springer, 301–316.

Frédéric Blanqui, Jean-Pierre Jouannaud, and Mitsuhiro Okada. 2002. Inductive-data-type systems. *Theor. Comput. Sci.* 272, 1-2 (2002), 41–68.

Frédéric Blanqui, Jean-Pierre Jouannaud, and Albert Rubio. 2008. The Computability Path Ordering: The End of a Quest. In *CSL (Lecture Notes in Computer Science)*, Michael Kaminski and Simone Martini (Eds.), Vol. 5213. Springer, 1–14.

Frédéric Blanqui, Jean-Pierre Jouannaud, and Albert Rubio. 2013. The Computability Path Ordering. (2013). in preparation.

Miquel Bofill, Cristina Borralleras, Enric Rodrguez-Carbonell, and Albert Rubio. 2012. The recursive path and polynomial ordering for first-order and higher-order terms. *Journal of Logic and Computation* (2012). DOI: http://dx.doi.org/10.1093/logcom/exs027

Cristina Borralleras and Albert Rubio. 2001. A Monotonic Higher-Order Semantic Path Ordering. In *LPAR (Lecture Notes in Computer Science)*, Robert Nieuwenhuis and Andrei Voronkov (Eds.), Vol. 2250. Springer, 531–547.

Roberto Di Cosmo and Delia Kesner. 1994. Simulating Expansions without Expansions. *Mathematical Structures in Computer Science* 4, 3 (1994), 315–362.

Nachum Dershowitz. 1982. Orderings for Term-Rewriting Systems. *Theor. Comput. Sci.* 17 (1982), 279–301.

Nachum Dershowitz and Jean-Pierre Jouannaud. 1990. Rewrite Systems. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*. Elsevier, 243–320.

Carsten Fuhs and Cynthia Kop. 2012. Polynomial Interpretations for Higher-Order Rewriting. In *RTA (LIPIcs)*, Ashish Tiwari (Ed.), Vol. 15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 176–192.

Gérard P. Huet. 1980. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems: Abstract Properties and Applications to Term Rewriting Systems. *J. ACM* 27, 4 (1980), 797–821. DOI: http://dx.doi.org/10.1145/322217.322230

Jean-Pierre Jouannaud. 2005. Higher-Order Rewriting: Framework, Confluence and Termination, See Middeldorp et al. [2005], 224–250.

Jean-Pierre Jouannaud and Hélène Kirchner. 1986. Completion of a Set of Rules Modulo a Set of Equations. *SIAM J. Comput.* 15, 4 (1986), 1155–1194.

Jean-Pierre Jouannaud and Jianqi Li. 2012. Church-Rosser Properties of Normal Rewriting. In *CSL (LIPIcs)*, Patrick Cégielski and Arnaud Durand (Eds.), Vol. 16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 350–365.

Jean-Pierre Jouannaud and Mitsuhiro Okada. 1991. A Computation Model for Executable Higher-Order Algebraic Specification Languages, See DBL [1991], 350–361.

Jean-Pierre Jouannaud and Albert Rubio. 1999. The Higher-Order Recursive Path Ordering. In *LICS*. IEEE Computer Society, 402–411.

Jean-Pierre Jouannaud and Albert Rubio. 2006. Higher-Order Orderings for Normal Rewriting. In *RTA (Lecture Notes in Computer Science)*, F. Pfenning (Ed.), Vol. 4098. Springer, 387–399.

Jean-Pierre Jouannaud and Albert Rubio. 2007. Polymorphic higher-order recursive path orderings. *J. ACM* 54, 1 (2007).

Cynthia Kop. 2012. *Higher Order Termination*. Ph.D. Dissertation. VU University Amsterdam, Amsterdam, The Netherlands.

Cynthia Kop and Femke van Raamsdonk. 2010. Higher-order dependency pairs with argument filterings. In *11th International Workshop on Termination - WST 2010*. Edinburgh, UK.

Keiichirou Kusakari, Yasuo Isogai, Masahiko Sakai, and Frédéric Blanqui. 2009. Static Dependency Pair Method based on Strong Computability for Higher-Order Rewrite Systems. *IEICE Transactions on Information and Systems* E92-D, 10 (2009), 2007–2015.

Richard Mayr and Tobias Nipkow. 1998. Higher-Order Rewrite Systems and Their Confluence. *Theor. Comput. Sci.* 192, 1 (1998), 3–29.

Aart Middeldorp, Vincent van Oostrom, Femke van Raamsdonk, and Roel C. de Vrijer (Eds.). 2005. *Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday*. Lecture Notes in Computer Science, Vol. 3838. Springer.

Dale Miller. 1991. A Logic Programming Language with Lambda-Abstraction, Function Variables, and Simple Unification. *J. Log. Comput.* 1, 4 (1991), 497–536.

Tobias Nipkow. 1991. Higher-Order Critical Pairs, See DBL [1991], 342–349.

Tobias Nipkow and Lawrence C. Paulson. 1992. Isabelle-91. In *CADE (Lecture Notes in Computer Science)*, Deepak Kapur (Ed.), Vol. 607. Springer, 673–676.

Gerald E. Peterson and Mark E. Stickel. 1981. Complete Sets of Reductions for Some Equational Theories. *J. ACM* 28, 2 (1981), 233–264.

Jaco van de Pol. 1996. *Termination of Higher-Order Rewrite Systems*. Ph.D. Dissertation. Utrecht Universiteit, Utrecht, The Netherlands.

Masahiko Sakai and Keiichirou Kusakari. 2005. On Dependency Pair Method for Proving Termination of Higher-Order Rewrite Systems. *IEICE Transactions on Information and Systems* E88-D, 3 (2005), 583–593.

Masahiko Sakai, Yoshitsugu Watanabe, and Toshiki Sakabe. 2001. An Extension of Dependency Pair Method for Proving Termination of Higher-Order Rewrite Systems. *IEICE Transactions on Information and Systems* E84-D, 8 (2001), 1025–1032.

Sho Suzuki, Keiichirou Kusakari, and Frédéric Blanqui. 2011. Argument filterings and usable rules in higher-order rewrite systems. *IPSJ Transactions on Programming* 4, 2 (2011), 1–12.

Terese. 2003. Term Rewriting Systems. In *Cambridge Tracts in Theoretical Computer Science, Marc Bezem, Jan Willem Klop, and Roel de Vrijer editors*. Cambridge University Press.

Jaco van de Pol. 1993. Termination Proofs for Higher-order Rewrite Systems. In *HOA (Lecture Notes in Computer Science)*, Jan Heering, Karl Meinke, Bernhard Möller, and Tobias Nipkow (Eds.), Vol. 816. Springer, 305–325.

Jaco van de Pol and Helmut Schwichtenberg. 1995. Strict Functionals for Termination Proofs. In *TLCA (Lecture Notes in Computer Science)*, Mariangiola Dezani-Ciancaglini and Gordon D. Plotkin (Eds.), Vol. 902. Springer, 350–364.

Femke van Raamsdonk. 2001. On Termination of Higher-Order Rewriting. In *RTA (Lecture Notes in Computer Science)*, Aart Middeldorp (Ed.), Vol. 2051. Springer, 261–275.

Daria Walukiewicz-Chrzaszcz. 2003. Termination of rewriting in the Calculus of Constructions. *J. Funct. Program.* 13, 2 (2003), 339–414.