# Higher-Order Orderings for Normal Rewriting[⋆]

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

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

**Abstract.** We extend the termination proof methods based on reduction orderings to higher-order rewriting systems using higher-order pattern matching, and accomodate for any use of eta, as a reduction, as an expansion or as an equation.

## 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 pionneered in this context by Nipkow [15] and is available in Isabelle [18]. Its main feature is the use of higher-order pattern matching for firing rules. A recent generalization of Nipkow's setting allows one for rewrite rules of polymorphic, higher-order type [12], see also [8]. Besides, it is shown that using the eta rule as an expansion [17] or as a reduction [12] yields very similar confluence checks based on higher-order critical pairs.

A first contribution of this paper is a general setting for adressing termination of all variants of higher-order rewriting à la Nipkow, thanks to the notion of a *normal higher-order reduction ordering*. While higher-order reduction orderings actually *include* $\beta\eta$-reductions, normal higher-order reduction orderings must be compatible with $\beta\eta$-equality since higher-order rewriting operates on $\beta\eta$-equivalence classes of terms. This is done by computing with $\beta\eta$-normal forms as inputs. We show however that monotonicity, stability, compatibility and well-foundedness cannot be satisfied at the same time. It becomes necessary to use higher-order reduction orderings enjoying a stronger stability property, and at the same time, a slightly weaker monotonicity property for terms in $\beta\eta$-normal forms. Restricting the higher-order recursive path ordering [10] to achieve these properties is our second contribution. Finally, the obtained ordering is used inside a powerful schema transforming an arbitrary higher-order

⋆ This work was partly supported by the RNTL project AVERROES and France Telecom.

⋆⋆ Project LogiCal, Pôle Commun de Recherche en Informatique du Plateau de Saclay, CNRS, École Polytechnique, INRIA, Université Paris-Sud.

reduction ordering satisfying these monotonicity and stability properties into a normal higher-order reduction ordering. This is our third contribution. The obtained ordering allows us to prove all standard examples of higher-order rules processing abstract syntax. In contrast with the higher-order recursive path ordering, there is no need here for the closure mecanism developped in [10].

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

Readers are assumed familiar with the basics of term rewriting [7, 13] and typed lambda calculi [4, 5]. Most ideas and results presented here originate from [11], an unpublished preliminary draft.

## 2 Polymorphic Higher-Order Algebras

This section recalls our framework of polymorphic algebras [10].

### 2.1 Types

Given a set $\mathcal{S}$ of *sort symbols* of a fixed arity, denoted by $s : *^n \to *$, and a set $\mathcal{S}^\forall$ of *type variables*, the set $\mathcal{T}_{\mathcal{S}^\forall}$ of *polymorphic types* is generated from these sets by the constructor $\to$ for *functional types*:

$$\begin{array}{c}\mathcal{T}_{\mathcal{S}^\forall} := \alpha \mid s(\mathcal{T}^n_{\mathcal{S}^\forall}) \mid (\mathcal{T}_{\mathcal{S}^\forall} \to \mathcal{T}_{\mathcal{S}^\forall}) \\ \text{for } \alpha \in \mathcal{S}^\forall \text{ and } s : *^n \to * \ \in \mathcal{S}\end{array}$$

$\mathcal{V}ar(\sigma)$ denotes the set of (type) variables of the type $\sigma \in \mathcal{T}_{\mathcal{S}^\forall}$. Types are *functional* when headed by the $\to$ symbol, and *data types* when they are headed by a sort symbol. $\to$ associates to the right.

A *type substitution* is a mapping from $\mathcal{S}^\forall$ to $\mathcal{T}_{\mathcal{S}^\forall}$ extended to an endomorphism of $\mathcal{T}_{\mathcal{S}^\forall}$. We write $\sigma\xi$ for the application of the type substitution $\xi$ to the type $\sigma$. We denote by $\mathcal{D}om(\sigma) = \{\alpha \in \mathcal{S}^\forall \mid \alpha\sigma \neq \alpha\}$ the domain of $\sigma \in \mathcal{T}_{\mathcal{S}^\forall}$, by $\sigma|_{\mathcal{V}}$ its restriction to the domain $\mathcal{D}om(\sigma) \cap \mathcal{V}$, by $\mathcal{R}an(\sigma) = \bigcup_{\alpha \in \mathcal{D}om(\sigma)} \mathcal{V}ar(\alpha\sigma)$ its *range*. By a renaming of the type $\sigma$ apart from $V \subset \mathcal{X}$, we mean a type $\sigma\xi$ where $\xi$ is a type renaming such that $\mathcal{D}om(\xi) = \mathcal{R}an(\sigma)$ and $\mathcal{R}an(\xi) \cap \mathcal{V} = \emptyset$.

We shall use $\alpha, \beta$ for type variables, $\sigma, \tau, \rho, \theta$ for arbitrary types, and $\xi, \zeta$ to denote type substitutions.

### 2.2 Signatures

We are given a set of function symbols denoted by the letters $f, g, h$, which are meant to be algebraic operators equiped with a fixed number $n$ of arguments (called the *arity*) of respective types $\sigma_1 \in \mathcal{T}_{\mathcal{S}^\forall}, \ldots, \sigma_n \in \mathcal{T}_{\mathcal{S}^\forall}$, and an *output type* $\sigma \in \mathcal{T}_{\mathcal{S}^\forall}$ such that $\mathcal{V}ar(\sigma) \subseteq \bigcup_i \mathcal{V}ar(\sigma_i)$. Let

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

be the set of all function symbols. The membership of a given function symbol $f$ to a set $\mathcal{F}_{\sigma_1 \times \ldots \times \sigma_n \to \sigma}$ is called a *type declaration* and written $f : \sigma_1 \times \ldots \times \sigma_n \to \sigma$. We assume that there is a unique type declaration for each function symbol in the signature. If $n = 0$, the declaration $f :\to \sigma$ is written $f : \sigma$ when $\sigma$ is not a functional type. Type declarations are not types, but $\sigma_1 \to \ldots \to \sigma_n \to \sigma$ is a type if $f : \sigma_1 \times \ldots \times \sigma_n \to \sigma$ is a type declaration. A type declaration is *first-order* if it uses only sorts, and higher-order otherwise. It is *polymorphic* if it uses some polymorphic type, otherwise, it is *monomorphic*. Polymorphic type declarations are implicitely universally quantified: they can be renamed arbitrarily. Note that type instantiation does not change the arity of a function symbol.

### 2.3 Terms

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

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

Raw terms of the form $\lambda x : \sigma.u$ are called *abstractions*. $@(u, v)$ denotes the application of $u$ to $v$. We may omit the type $\sigma$ in $\lambda x : \sigma.u$ as well as the application operator, writing $u(v)$ for $@(u, v)$, in particular when $u$ is a higher-order variable. We also write $u(v_1, \ldots, v_n)$, or $@(u, v_1, \ldots, v_n)$ for $u(v_1) \ldots (v_n)$, assuming $n \geq 1$. The raw term $@(u, \overline{v})$ is called a (partial) *left-flattening* of $s = u(v_1) \ldots (v_n)$, $u$ being possibly an application itself. $\mathcal{V}ar(t)$ is the set of free variables of $t$, while $\mathcal{BV}ar(t)$ is its set of bound variables. $\overline{s}$ shall be ambiguously used to denote the list $\langle s_1 \ldots s_n \rangle$, or the multiset or the set $\{s_1 \ldots s_n\}$ of raw terms $s_1, \ldots, s_n$. We will use the convention that the list $\langle s_k \ldots s_l \rangle$ is empty if $l < k$.

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

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

### 2.4 Typing rules

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

Our typing judgements are written as $\Gamma \vdash_{\mathcal{F}} s : \sigma$. A raw term $s$ has type $\sigma$ in the environment $\Gamma$ if the judgement $\Gamma \vdash_{\mathcal{F}} s : \sigma$ is provable in our inference system. Given an environment $\Gamma$, a raw term $s$ is *typable* if there exists a type $\sigma$ such that $\Gamma \vdash_{\mathcal{F}} s : \sigma$, in which case it is called a *term*.

Types can be seen as terms of type $*$. We omit the straightforward type system for typing types, aiming at verifying arities of sort symbols.

Some properties of our type system are instrumental in developing a theory of higher-order rewriting:

**Lemma 1.** *Given an environment $\Gamma$ and a typable term $s$, there exists a unique type $\sigma$ such that $\Gamma \vdash_{\mathcal{F}} s : \sigma$.*

**Variables:**
$$\frac{x:\sigma \in \Gamma}{\Gamma \vdash_{\mathcal{F}} x:\sigma}$$

**Functions:**
$$\frac{\begin{array}{c} f:\sigma_1 \times \ldots \times \sigma_n \to \sigma \in \mathcal{F} \\ \xi \text{ some type substitution of domain} \subseteq \bigcup_i \mathcal{V}ar(\sigma_i) \\ \Gamma \vdash_{\mathcal{F}} t_1:\sigma_1\xi \ \ldots\ \Gamma \vdash_{\mathcal{F}} t_n:\sigma_n\xi \end{array}}{\Gamma \vdash_{\mathcal{F}} f(t_1,\ldots,t_n):\sigma\xi}$$

**Abstraction:**
$$\frac{\Gamma \cdot \{x:\sigma\} \vdash_{\mathcal{F}} t:\tau}{\Gamma \vdash_{\mathcal{F}} (\lambda x:\sigma.t):\sigma \to \tau}$$

**Application:**
$$\frac{\Gamma \vdash_{\mathcal{F}} s:\sigma \to \tau \quad \Gamma \vdash_{\mathcal{F}} t:\sigma}{\Gamma \vdash_{\mathcal{F}} @(s,t):\tau}$$

**Fig. 1.** The type system for polymorphic higher-order algebras

Note that type substitutions apply to types in terms: $x\xi = x$, $(\lambda x : \sigma.s)\xi = \lambda x : \sigma\xi.s\xi$, $(u,v)\xi = (u\xi, v\xi)$, and $f(\overline{u})\xi = f(\overline{u}\xi)$.

**Lemma 2.** *$\Gamma \vdash_{\mathcal{F}} s : \sigma$ implies $\Gamma\xi \vdash_{\mathcal{F}} s\xi : \sigma\xi$ for any $\xi$.*

**Lemma 3.** *Given a signature $\mathcal{F}$, environment $\Gamma$, term $s$ and type $\sigma$ such that $\Gamma \vdash_{\mathcal{F}} s : \sigma$, then $\Gamma \cdot \Gamma' \vdash_{\mathcal{F}} s : \sigma$ for all $\Gamma'$ compatible with $\Gamma$.*

**Lemma 4.** *Given a signature $\mathcal{F}$, environment $\Gamma$, term $s$ and type $\sigma$ such that $\Gamma \vdash_{\mathcal{F}} s : \sigma$, then for all $p \in \mathcal{D}om(s)$, there exists a canonical environment $\Gamma_{s|_p}$ and a type $\tau$ such that $\Gamma_{s|_p} \vdash_{\mathcal{F}} s|_p : \tau$ is a subproof of the proof of $\Gamma \vdash_{\mathcal{F}} s : \sigma$. Moreover, $\Gamma_{s|_{(p\cdot q)}} = (\Gamma_{s|_p})_{(s|_p)|_q}$.*

**Lemma 5.** *Given a signature $\mathcal{F}$, an environment $\Gamma$, two terms $s$ and $v$, two types $\sigma$ and $\tau$, and a position $p \in \mathcal{P}os(s)$ such that $\Gamma \vdash_{\mathcal{F}} s : \sigma$, $\Gamma_{s|_p} \vdash_{\mathcal{F}} s|_p : \tau$ and $\Gamma_{s|_p} \vdash_{\mathcal{F}} v : \tau$, then $\Gamma \vdash_{\mathcal{F}} s[v]_p : \sigma$.*

**Definition 2.** *A* substitution *$\gamma = \{(x_1 : \sigma_1) \mapsto (\Gamma_1, t_1), \ldots, (x_n : \sigma_n) \mapsto (\Gamma_n, t_n)\}$, is a finite set of quadruples made of a variable symbol, a type, an environment and a term, such that*
*(i) $\forall i \in [1..n]$, $t_i \neq x_i$ and $\Gamma_i \vdash_{\mathcal{F}} t_i : \sigma_i$,*
*(ii) $\forall i \neq j \in [1..n]$, $x_i \neq x_j$, and*
*(iii) $\forall i \neq j \in [1..n]$, $\Gamma_i$ and $\Gamma_j$ are compatible environments.*
*We may omit the type $\sigma_i$ and environment $\Gamma_i$ in $(x_i : \sigma_i) \mapsto (\Gamma_i, t_i)$.*

*The set of (input) variables of the substitution $\gamma$ is $\mathcal{V}ar(\gamma) = \{x_1, \ldots, x_n\}$, its* domain *is the environment $\mathcal{D}om(\gamma) = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ while its* range *is the environment $\mathcal{R}an(\gamma) = \bigcup_{i\in[1..n]} \Gamma_i$.*

*We denote by $\gamma_{|V}$ the restriction of the substitution $\gamma$ to the domain $V \cap \mathcal{V}ar(\gamma)$, and by $\gamma_{\setminus V}$ the substitution $\gamma_{|(\mathcal{X}\setminus V)}$.*

Note that $\mathcal{R}an(\gamma)$ is indeed an environment by assumption (iii).

**Lemma 6.** *Given* $\gamma = \{(x_1 : \sigma_1) \mapsto (\Gamma_1, t_1), \ldots, (x_n : \sigma_n) \mapsto (\Gamma_n, t_n)\}$*, then* $\mathcal{R}an(\gamma) \vdash_{\mathcal{F}} t_i : \sigma_i$*.*

**Definition 3.** *A substitution* $\gamma$ *is* compatible *with an environment* $\Gamma$ *if*
*(i)* $\mathcal{D}om(\gamma)$ *is compatible with* $\Gamma$*,*
*(ii)* $\mathcal{R}an(\gamma)$ *is compatible with* $\Gamma \setminus \mathcal{D}om(\gamma)$*.*
*We will also say that* $\gamma$ *is compatible with the judgement* $\Gamma \vdash_{\mathcal{F}} s : \sigma$*.*

**Definition 4.** *A substitution* $\gamma$ *compatible with a judgement* $\Gamma \vdash_{\mathcal{F}} s : \sigma$ *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* $(x : \sigma) \mapsto (\Gamma, t) \in \Gamma$ | *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* $s\gamma = \lambda z : \tau.u(\{x \mapsto z\} \cup \gamma_{\setminus\{x\}})$*,* $z$ *fresh.* |

**Lemma 7.** *Given a signature* $\mathcal{F}$ *and a substitution* $\gamma$ *compatible with the judgement* $\Gamma \vdash_{\mathcal{F}} s : \sigma$*, then* $\Gamma \cdot \mathcal{R}an(\gamma) \vdash_{\mathcal{F}} s\gamma : \sigma$*.*

When writing $s\gamma$, we make the assumption that the domain of $\gamma$ is compatible with the judgement $\Gamma \vdash_{\mathcal{F}} s : \sigma$. We use the letter $\gamma$ for substitutions and postfix notation for their application.

### 2.5 Conversions

The following three equations originate from the $\lambda$-calculus, and are called $\alpha$-, $\beta$- and $\eta$-equality respectively:

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

In the above equations, $u, v$ and $w$ stand for arbitrary terms to which substitutions $\{x \to y\}$ and $\{x \to u\}$ apply. We consider $\alpha$-convertible terms as identical, and therefore omit $\alpha$-conversions in the sequel. The congruence generated by the $\beta$- and $\eta$-equalities is written $\longrightarrow^*_{\beta\eta}$ or $=_{\beta\eta}$. An important property, *subject reduction*, is that typable terms $u, v$ such that $u =_{\beta\eta} v$ have the same type. Both equalities can be oriented as rewrite rules. There are two possible choices for rewriting with $\eta$, either as a reduction (from left to ritht) 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 confluence properties one may need, with respect to: $\beta\eta$-reductions; $\beta$-reductions and $\eta$-expansions; $\beta$-reductions modulo $\eta$-equality. Using the notations $u \longrightarrow_\beta v$ for one $\beta$-rewrite step, $u \longrightarrow^*_\beta v$ for its transitive closure, $u\downarrow_\beta$ for the $\beta$-normal form of $u$, and $\longleftrightarrow^*_\eta$ or $=_\eta$ for $\eta$-equality, the Church-Rosser property of $\beta$-reductions modulo $\eta$-equality for typable terms can be phrased as

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

## 3 Normal Higher-Order Rewriting of Higher Type

Normal higher-order rewriting [17, 15] allows defining computations over $\lambda$-terms used as a suitable abstract syntax for encoding functional objects like programs or specifications. Nipkow's framework assumes that rules are of basic type, and that lefthand sides of rules are patterns in the sense of Miller [16], two assumptions which are useless for termination purposes. We therefore do not assume them.

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

**Definition 5.** *A* normal rewrite rule *is a rewrite rule $\Gamma \vdash l \rightarrow r : \sigma$ such that $l$ and $r$ are higher-order terms in $\beta$-normal form satisfying $\Gamma \vdash_\mathcal{F} l : \sigma$ for some type $\sigma$ iff $\Gamma \vdash_\mathcal{F} r : \sigma$. A* normal term rewriting system *is a set of normal rewrite rules.*

*Given a normal term rewriting system $R$, an environment $\Gamma$, two $\beta$-normal terms $s$ and $t$, and a type $\sigma$ such that $\Gamma \vdash_\mathcal{F} s : \sigma$, we say that $s$ rewrites to $t$ at position $p$ with the normal rule $\Gamma_i \vdash l_i \rightarrow r_i : \sigma_i$, the type substitution $\xi$ and the term substitution $\gamma$, written $\Gamma \vdash s \longrightarrow^p_{R_{\beta\eta}} t$, or $s \longrightarrow^p_{R_{\beta\eta}} t$ assuming the environment $\Gamma$, if the following conditions hold:*

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

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

**Lemma 8.** *Let $s$ be a term such that $\Gamma \vdash_{\mathcal{F}} s : \sigma$ and $\Gamma \vdash s \longrightarrow_{R^{\eta}_{\beta}} t$. Then $\Gamma \vdash_{\mathcal{F}} t : \sigma$.*

*Proof.* By Lemma 2, $\Gamma_i\xi \vdash_{\mathcal{F}} l_i\xi : \sigma_i\xi$. By conditions (i) and (ii) in Definition 5, the substitution $\gamma$ is compatible with the environment $\Gamma_i\xi$, and therefore, by lemma 7, $\Gamma_i\xi \cdot \mathcal{R}an(\gamma) \vdash_{\mathcal{F}} l_i\xi\gamma : \sigma_i\xi$. By condition (ii) and lemma 3, $\Gamma_{s|_p} \vdash_{\mathcal{F}} l_i\xi\gamma : \sigma_i\xi$, hence $\Gamma_{s|_p} \vdash_{\mathcal{F}} s|_p : \sigma_i\xi$ by subject reduction. Note that this tells us how to compute $\xi$ in practice. Similarly, $\Gamma_{s|_p} \vdash_{\mathcal{F}} r_i\xi\gamma : \sigma_i\xi$. By lemma 5, we deduce that $\Gamma \vdash_{\mathcal{F}} s[r_i\xi\gamma]_p : \sigma$. Using now condition (iv) and the subject-reduction property, we finally conclude that $\Gamma \vdash_{\mathcal{F}} t : \sigma$. $\square$

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

*Example 1.* We present here an encoding of symbolic derivation in which functions are represented by $\lambda$-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 to compute the derivative of all expressions: rewriting the expression $\mathsf{diff}(\lambda \mathsf{x}.\mathsf{sin}(\mathsf{x})) =_{\beta} \mathsf{diff}(\lambda \mathsf{x}.\mathsf{sin}(\lambda \mathsf{y}.\mathsf{y}\ \mathsf{x}))$ does require higher-order pattern matching. We shall give a mechanical termination proof of both rules in Section 5.

### 3.1 Normal Higher-Order Reduction Orderings

We shall use well-founded relations for proving strong normalization properties. For our purpose, these relations may not be transitive, but their transitive closures will be well-founded orderings, justifying some abuse of terminology. Reduction orderings operating on judgements turn out to be an adequate tool for showing termination of normal rewriting. We consider two classes of reduction orderings called *higher-order reduction ordering* when they include $\beta\eta$-reductions and *normal higher-order reduction ordering* when they are compatible with $=_{\beta\eta}$.

**Definition 6.** *A binary relation $\succ$ on the set of judgements is*

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

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

*A* normal higher-order reduction ordering $\succ_{\beta}^{\eta}$ *is a well-founded ordering of the set of judgements satisfying coherence, polymorphism, stability, normal-monotonicity and compatibility.*

One may argue whether $\eta$-reductions should or should not be included into a higher-order reduction ordering, since there are two possibilities of orientation for the $\eta$-equality. By including it, we indicate our preference for using $\eta$-reductions instead of $\eta$-expansions. This preference does not have any impact on the rest of this paper.

Let us show now that no ordering $\succ$ can satisfy monotonicity, stability, compatibility and well-foundedness, therefore explaining the need for the weaker notion of normal-monotonicity.

Assume $s : \sigma \succ t : \sigma$ (omitting judgements), where $s : \sigma$ is in $\beta$-normal form. Given a variable $X : \sigma \to \tau$, $@(X, s)$ is in $\beta$-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. This problem does not happen with normal-monotonicity since $@(\lambda y.a, s)$ is not in $\beta$-normal form.

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

*Proof.* Let $s$ be a ground normal term such that $\Gamma \vdash_{\mathcal{F}} s \xrightarrow[\Gamma_i\xi \vdash l_i\xi \to r_i\xi : \sigma_i\xi]{p} t$. By definition of normal rewriting, $t$ is a ground normal term. It therefore suffices to show that $\Gamma \vdash_{\mathcal{F}} s \succ t$, which we proceed to do now.
By assumption, $\Gamma_i \vdash_{\mathcal{F}} l_i \succ r_i$ and by polymorphism, $\Gamma_i\xi \vdash_{\mathcal{F}} l_i\xi \succ r_i\xi$. By stability, $\Gamma_i\xi \cdot \mathcal{R}an(\gamma) \vdash_{\mathcal{F}} l_i\xi\gamma \succ r_i\xi\gamma$, therefore, by coherence, $\Gamma_{s|_p} \vdash_{\mathcal{F}} l_i\xi\gamma \succ r_i\xi\gamma$. By definition, $s|_p =_{\beta\eta} l_i\xi\gamma$, hence, by compatibility, $\Gamma_{s|_p} \vdash_{\mathcal{F}} s_{|}p \succ r_i\xi\gamma$. By monotonicity of $\succ$ for normal ground terms (of equal type), $\Gamma_{s|_p} \cdot \Gamma \vdash_{\mathcal{F}} s \succ s[r_i\xi\gamma]_p$. By coherence $\Gamma \vdash_{\mathcal{F}} s \succ s[r_i\xi\gamma]_p$, hence $\Gamma \vdash_{\mathcal{F}} s \succ t$ by compatibility. $\square$

## 4 Building Normal Higher-Order Reduction Orderings

In this section, we explain how to systematically build normal higher-order reduction orderings from higher-order reduction orderings satisfying a stronger stability property. To this end, we introduce a specific treatment of abstractions, called *neutralization*, which transforms a term built from the signature $\mathcal{F}$ into a term built from an enlarged signature $\mathcal{F}_{new}$, obtained from $\mathcal{F}$ by adding a function symbol $\bot_\sigma$ for every type $\sigma$ and a function symbol $f_{new}$ for some of the function symbols in $\mathcal{F}$. We write $\bot_\sigma$ for $\bot_\sigma()$. The higher-order rules we want to prove terminating are of course built from terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, not in $\mathcal{T}(\mathcal{F}_{new}, \mathcal{X})$.

### 4.1 Neutralization and Normalization

**Definition 7.** *The* neutralization of level $i$ *(*$i$-neutralization *in short) of a typable term* $t \in \mathcal{T}(\mathcal{F}_{new}, \mathcal{X})$ *with respect to the list of (typable) terms* $\langle u_1 : \theta_1, \ldots, u_n : \theta_n \rangle$ *in* $\mathcal{T}(\mathcal{F}_{new}, \mathcal{X})$*, is the term* $\mathcal{N}_i(t, \langle u_1, \ldots, u_n \rangle)$ *defined as follows:*

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

Terms of a functional type are neutralized by applying them to the $\perp$-expression of the appropriate type. For each function symbol, we will actually control which arguments of a functional type can be neutralized:

**Definition 8.** *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:*

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

We shall now neutralize terms recursively. To this end, we need adding to the signature a new function symbol $f_{new} : \sigma'_1 \times \ldots \times \sigma'_n \to \sigma$ for every declaration $f : \sigma_1 \times \ldots \times \sigma_n \to \sigma$, such that $\sigma'_i = \tau_{q+1} \to \ldots \to \tau_k \to \tau$ if $\sigma_i = \tau_1 \to \ldots \to \tau_k \to \tau$ and $nl_f^i = q \leq k$.

**Definition 9.** *The* full neutralization *of a term* $t$ *is the term* $\mathcal{FN}(t)$ *s.t.*

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

*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 using Case 3 repeatedly yields $\mathcal{FN}(@(t_1, \ldots, t_n)) = @(\mathcal{FN}(t_1), \ldots, \mathcal{FN}(t_n))$ for flattened applications. Note also that the filtered list of arguments $\bar{t}_f^i$ is itself recursively neutralized before using it to neutralize $t_i$.

*Example 1 (continued).* We illustrate here the full neutralization of the lefthand and righthand sides of the rules of Example 1. 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 $\bot_{\mathsf{real}}$ abbreviated as $\bot$ is a constant:

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

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

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

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

and of the second rule:

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

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

### 4.2 Properties of neutralization

The following lemmas investigate the interactions between neutralization, instantiation, type instantiation and normalization.

**Lemma 9.** *Let $t$ and $u_1, \ldots, u_n$ be higher-order terms. Then* $\mathcal{N}_i(t, \langle u_1, \ldots, u_n\rangle){\downarrow} = \mathcal{N}_i(t{\downarrow}, \langle u_1{\downarrow}, \ldots, u_n{\downarrow}\rangle){\downarrow}$.

*Proof.* Let $t : \tau$. We proceed by induction on $i$. There are two cases.

1. If $i = 0$ or $\tau$ is a data-type, then, by definition, $\mathcal{N}_i(t, \langle u_1, \ldots, u_n\rangle){\downarrow} = t{\downarrow} = \mathcal{N}_i(t{\downarrow}, \langle u_1{\downarrow}, \ldots, u_n{\downarrow}\rangle)$.
2. Otherwise $i > 0$ and $\tau = \sigma \to \rho$. Then, $\mathcal{N}_i(t, \langle u_1, \ldots, u_n\rangle){\downarrow} = \mathcal{N}_{i-1}(@(t, \bot_\theta(u_1, \ldots, u_n)), \langle u_1, \ldots, u_n\rangle){\downarrow}$. By induction hypothesis, this is equal to $\mathcal{N}_{i-1}(@(t, \bot_\theta(u_1, \ldots, u_n)){\downarrow}, \langle u_1{\downarrow}, \ldots, u_n{\downarrow}\rangle){\downarrow}$ $= \mathcal{N}_{i-1}(@(t{\downarrow}, \bot_\theta(u_1{\downarrow}, \ldots, u_n{\downarrow})){\downarrow}, \langle u_1, \ldots, u_n\rangle){\downarrow}$. By induction hypothesis again, this is equal to $\mathcal{N}_{i-1}(@(t{\downarrow}, \bot_\theta(u_1{\downarrow}, \ldots, u_n{\downarrow})), \langle u_1{\downarrow}, \ldots, u_n{\downarrow}\rangle){\downarrow}$ $= \mathcal{N}_i(t{\downarrow}, \langle u_1{\downarrow}, \ldots, u_n{\downarrow}\rangle){\downarrow}$. $\square$

**Lemma 10.** *Let $t = \lambda x_1 \dots x_j.u : \tau$ be a $\beta\eta$-normalized term such that $0 \leq j \leq ar(\tau)$ and $u$ is not an abstraction. Then for all $0 \leq i \leq ar(\tau)$ and all $\beta\eta$-normalized terms $u_1, \dots, u_n$,*

*(i)* *if $i \leq j$ then $N_i(t, \langle u_1, \dots, u_n\rangle)\downarrow=$*
$\lambda x_{i+1} \dots x_j.u\{x_1 \mapsto \perp_{\theta_1}(u_1, \dots, u_n), \dots, x_i \mapsto \perp_{\theta_i}(u_1, \dots, u_n)\}$*;*

*(ii)* *if $i > j$ then $N_i(t, \langle u_1, \dots, u_n\rangle)\downarrow=$*
$@(u\{x_1 \mapsto \perp_{\theta_1}(u_1, \dots, u_n), \dots, x_j \mapsto \perp_{\theta_j}(u_1, \dots, u_n)\},$
$\perp_{\theta_{j+1}}(u_1, \dots, u_n), \dots, \perp_{\theta_i}(u_1, \dots, u_n))$.

Note that $t$ is not an abstraction if $j = 0$ and has an arrow type if $i > 0$.

*Proof.* We proceed by induction on $i$. Let $\tau = \sigma_1 \to \dots \to \sigma_j \to \rho$.

1. $i = 0$. Then $N_i(t, \langle u_1, \dots, u_n\rangle)\downarrow= t\downarrow= t$.
2. $i > 0$. Then $N_i(\lambda x_1 \dots x_j.u, \langle u_1, \dots, u_n\rangle)\downarrow=$
   $N_{i-1}(@(\lambda x_1 \dots x_j.u, \perp_{\theta_1}(u_1, \dots, u_n)), \langle u_1, \dots, u_n\rangle)\downarrow$.
   By Lemma 9, since all $u_1, \dots, u_n$ are $\beta\eta$-normalized, this is equal to
   $N_{i-1}(@(\lambda x_1 \dots x_j.u, \perp_{\theta_1}(u_1, \dots, u_n))\downarrow, \langle u_1, \dots, u_n\rangle)\downarrow=$
   $N_{i-1}(\lambda x_2 \dots x_j.u\{x_1 \mapsto \perp_{\theta_1}(u_1, \dots, u_n)\}\downarrow, \langle u_1, \dots, u_n\rangle)\downarrow=$
   $N_{i-1}(\lambda x_2 \dots x_j.u\{x_1 \mapsto \perp_{\theta_1}(u_1, \dots, u_n)\}, \langle u_1, \dots, u_n\rangle)\downarrow$.
   By induction hypothesis, since there are $j - 1$ variables in $x_2 \dots x_j$,
   - if $i - 1 \leq j - 1$, that is, $i \leq j$, then this is equal to
     $\lambda x_{i+1} \dots x_j.u\{x_1 \mapsto \perp_{\theta_1}(u_1, \dots, u_n), \dots, x_i \mapsto \perp_{\theta_i}(u_1, \dots, u_n)\}$;
   - if $i - 1 > j - 1$, that is, $i > j$, then this is equal to
     $@(u\{x_1 \mapsto \perp_{\theta_1}(u_1, \dots, u_n), \dots, x_j \mapsto \perp_{\theta_j}(u_1, \dots, u_n)\},$
     $\perp_{\theta_{j+1}}(u_1, \dots, u_n), \dots, \perp_{\theta_i}(u_1, \dots, u_n))$;

   and we are done. □

**Lemma 11.** *Let $t$ be a $\beta\eta$-normalized term. $\mathcal{V}ar(\mathcal{FN}(t)\downarrow) = \mathcal{V}ar(t)$ and*

1. *$\mathcal{FN}(x)\downarrow= x$ for $x \in \mathcal{X}$;*
2. *$\mathcal{FN}(\lambda x.u)\downarrow= \lambda x.\mathcal{FN}(u)\downarrow$;*
3. *$\mathcal{FN}(@(t_1, t_2))\downarrow= @(\mathcal{FN}(t_1)\downarrow, \mathcal{FN}(t_2)\downarrow)$;*
4. *$\mathcal{FN}(f(t_1, \dots, t_n))\downarrow=$*
   $f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1)\downarrow, \mathcal{FN}(\bar{t}^1_f)\downarrow)\downarrow, \dots, \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n)\downarrow, \mathcal{FN}(\bar{t}^n_f)\downarrow)\downarrow)$.

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

1. Let $t \in \mathcal{X}$. Then $\mathcal{FN}(t)\downarrow= t\downarrow= t$.
2. Let $t = \lambda x.u$.
   Then $\mathcal{FN}(t)\downarrow= (\lambda x.\mathcal{FN}(u))\downarrow= \lambda x.\mathcal{FN}(u)\downarrow$. By induction hypothesis, $\mathcal{V}ar(\mathcal{FN}(u)\downarrow) = \mathcal{V}ar(u)$, hence $\mathcal{V}ar(\mathcal{FN}(t)\downarrow) = \mathcal{V}ar(t)$.

3. Let $t = @(t_1, t_2)$.
Then $\mathcal{FN}(t)\downarrow = @(\mathcal{FN}(t_1), \mathcal{FN}(t_2))\downarrow = @(\mathcal{FN}(t_1\downarrow), \mathcal{FN}(t_2\downarrow))\downarrow$. Since $t$ is normal, $t_1$ cannot be an abstraction, hence, by induction hypothesis, $\mathcal{FN}(t_1)\downarrow$ is not an abstraction either. Therefore, $\mathcal{FN}(t)\downarrow = @(\mathcal{FN}(t_1\downarrow), \mathcal{FN}(t_2\downarrow))$. Finally, since$\mathcal{V}ar(t_1) = \mathcal{V}ar(\mathcal{FN}(t_1\downarrow))$ and $\mathcal{V}ar(t_2) = \mathcal{V}ar(\mathcal{FN}(t_2\downarrow))$ by induction hypothesis, we easily conclude that $\mathcal{V}ar(\mathcal{FN}(t)\downarrow) = \mathcal{V}ar(t)$.
4. Let $t = f(t_1, \ldots, t_n)$ with $f \in \mathcal{F}$. Then $\mathcal{FN}(t)\downarrow =$
$f_{new}(\mathcal{N}_{nl_f^1}(\mathcal{FN}(t_1), \mathcal{FN}(\bar{t}_f^1)), \ldots, \mathcal{N}_{nl_f^n}(\mathcal{FN}(t_n), \mathcal{FN}(\bar{t}_f^n)))\downarrow$
$= f_{new}(\mathcal{N}_{nl_f^1}(\mathcal{FN}(t_1), \mathcal{FN}(\bar{t}_f^1))\downarrow, \ldots, \mathcal{N}_{nl_f^n}(\mathcal{FN}(t_n), \mathcal{FN}(\bar{t}_f^n))\downarrow)$,
and by Lemma 9, $\mathcal{FN}(t)\downarrow =$
$f_{new}(\mathcal{N}_{nl_f^1}(\mathcal{FN}(t_1)\downarrow, \mathcal{FN}(\bar{t}_f^1)\downarrow)\downarrow, \ldots, \mathcal{N}_{nl_f^n}(\mathcal{FN}(t_n)\downarrow, \mathcal{FN}(\bar{t}_f^n)\downarrow)\downarrow)$.
On the other hand, by induction hypothesis, $\mathcal{V}ar(\mathcal{FN}(t_i)\downarrow) = \mathcal{V}ar(t_i)$ for all $i \in [1..n]$, and, by Lemma 10, $\mathcal{V}ar(\mathcal{N}_{nl_f^1}(\mathcal{FN}(t_i)\downarrow, \mathcal{FN}(\bar{t}_f^i)\downarrow)\downarrow) =$ $\mathcal{V}ar(\mathcal{FN}(t_i)\downarrow) \cup \mathcal{V}ar(\mathcal{FN}(\bar{t}_f^i)\downarrow) = \mathcal{V}ar(t_i) \cup \mathcal{V}ar(\mathcal{FN}(\bar{t}_f^i)\downarrow)$. Since $\mathcal{V}ar(\mathcal{FN}(\bar{t}_f^i\downarrow) \subseteq \mathcal{V}ar(t)$ for all $i \in [1..n]$, it follows that $\mathcal{V}ar(\mathcal{FN}(t)\downarrow) =$ $\mathcal{V}ar(t)$. □

Let us now move to type instantiations:

**Lemma 12.** *Let $t : \sigma$ be a higher-order term. Then $t\xi\downarrow = t\downarrow \xi$.*

*Proof.* The proof uses the fact that $t\xi \longrightarrow_{\beta\eta} t'$ if and only if there is some $t''$ with $t \longrightarrow_{\eta} t''$ and $t' = t''\xi$. The result follows by induction. □

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

**Lemma 13.** *Let $t : \tau$ and $u_1 : \theta_1, \ldots, u_n : \theta_n$ be higher-order terms and $\xi$ a type substitution. Then, for all $i \leq ar(\tau)$,*
$\mathcal{N}_i(t\xi, \langle u_1\xi, \ldots, u_n\xi\rangle) = \mathcal{N}_i(t, \langle u_1, \ldots, u_n\rangle)\xi$.

*Proof.* We proceed by induction on $i$. The are two cases.

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

**Lemma 14.** *Let $t : \sigma$ be a higher-order term and $\xi$ a type substitution. Then $\mathcal{FN}(t\xi) = \mathcal{FN}(t)\xi$.*

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

1. $t$ is a variable $x$. Then $\mathcal{FN}(x\xi) = x\xi = \mathcal{FN}(x)\xi$.
2. $t = \lambda x.u$. Then $\mathcal{FN}(t\xi) = \mathcal{FN}(\lambda x\xi.u\xi) = \lambda x\xi.\mathcal{FN}(u\xi)$. By induction hypothesis, $\mathcal{FN}(u\xi) = \mathcal{FN}(u)\xi$, and therefore $\lambda x\xi.\mathcal{FN}(u)\xi = (\lambda x.\mathcal{FN}(u))\xi = \lambda x.\mathcal{FN}(u)\xi = \mathcal{FN}(\lambda x.u)\xi = \mathcal{FN}(t)\xi$.
3. $t = @(t_1, \ldots, t_n) : \sigma$. Then $\mathcal{FN}(t\xi) = \mathcal{FN}(@(t_1\xi, \ldots, t_n\xi) : \sigma\xi) = @(\mathcal{FN}(t_1\xi), \ldots, \mathcal{FN}(t_n\xi)) : \sigma\xi = @(\mathcal{FN}(t_1\xi), \ldots, \mathcal{FN}(t_n\xi)) : \sigma\xi$. By induction hypothesis, this is equal to $@(\mathcal{FN}(t_1)\xi, \ldots, \mathcal{FN}(t_n)\xi) = @(\mathcal{FN}(t_1), \ldots, \mathcal{FN}(t_n))\xi = \mathcal{FN}(@(t_1, \ldots, t_n))\xi = \mathcal{FN}(t)\xi$.
4. $t = f(t_1, \ldots, t_n)$ with $f \in \mathcal{F}$. Then $\mathcal{FN}(t\xi) = \mathcal{FN}(f(t_1, \ldots, t_n)\xi) = \mathcal{FN}(f(t_1\xi, \ldots, t_n\xi)) = f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1\xi), \mathcal{FN}(\bar{t}^1_f\xi)), \ldots, \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n\xi), \mathcal{FN}(\bar{t}^n_f\xi)))$.
   By induction hypothesis, this is equal to
   $f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1)\xi, \mathcal{FN}(\bar{t}^1_f)\xi)), \ldots, \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n)\xi, \mathcal{FN}(\bar{t}^n_f)\xi)))$
   $= f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1), \mathcal{FN}(\bar{t}^1_f))\xi, \ldots, \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n), \mathcal{FN}(\bar{t}^n_f))\xi)$,
   by using lemma 13. Extracting the type substitution we get
   $f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1, \mathcal{FN}(\bar{t}^1_f))), \ldots, \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n, \mathcal{FN}(\bar{t}^n_f))))\xi$
   $= \mathcal{FN}(f(t_1, \ldots, t_n))\xi = \mathcal{FN}(t)\xi$. $\square$

We now move to properties involving instantiations:

**Lemma 15.** *Let $t : \tau$ and $u_1, \ldots, u_n$ be higher-order terms and $\gamma$ 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 two cases.

1. Assume $i = 0$ or $\tau$ is a data type. Then, $t\gamma : \tau$, hence $\mathcal{N}_i(t\gamma, \langle u_1\gamma, \ldots, u_n\gamma\rangle) = t\gamma = \mathcal{N}_i(t, \langle u_1, \ldots, u_n\rangle)\gamma$.
2. Otherwise $i > 0$ and $\tau = \sigma \to \rho$. Then, $\mathcal{N}_i(t\gamma, \langle u_1\gamma, \ldots, u_n\gamma\rangle) = \mathcal{N}_{i-1}(@(t\gamma, \perp_\theta(u_1\gamma, \ldots, u_n\gamma))) = \mathcal{N}_{i-1}(@(t, \perp_\theta(u_1, \ldots, u_n))\gamma)$.
   By induction hypothesis, this is equal to
   $\mathcal{N}_{i-1}(@(t, \perp_\theta(u_1, \ldots, u_n)))\gamma = \mathcal{N}_i(t, \langle u_1, \ldots, u_n\rangle)\gamma$. $\square$

**Lemma 16.** *Let $t$ be a higher-order term and $\gamma$ a substitution. Then $\mathcal{FN}(t\gamma){\downarrow} = (\mathcal{FN}(t)\mathcal{FN}(\gamma)){\downarrow}$.*

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

1. $t$ is a variable $x$. Then $\mathcal{FN}(x\gamma) = x\mathcal{FN}(\gamma)$.
2. $t = \lambda x.u$. Then $\mathcal{FN}(t\gamma)\downarrow = \mathcal{FN}(\lambda x.u\gamma)\downarrow = \lambda x.\mathcal{FN}(u\gamma)\downarrow$. By induction hypothesis, $\mathcal{FN}(u\gamma)\downarrow = \mathcal{FN}(u)\mathcal{FN}(\gamma)\downarrow$, and therefore $\lambda x.\mathcal{FN}(u\gamma)\downarrow = \lambda x.\mathcal{FN}(u)\mathcal{FN}(\gamma)\downarrow = \lambda x.\mathcal{FN}(u)\mathcal{FN}(\gamma)\downarrow =$ $(\lambda x.\mathcal{FN}(u))\mathcal{FN}(\gamma)\downarrow = \mathcal{FN}(\lambda x.u)\mathcal{FN}(\gamma)\downarrow = \mathcal{FN}(t)\mathcal{FN}(\gamma)\downarrow$.
3. $t = @(t_1, \ldots, t_n)$. Then $\mathcal{FN}(t\gamma)\downarrow = \mathcal{FN}(@(t_1\gamma, \ldots, t_n\gamma))\downarrow =$ $@(\mathcal{FN}(t_1\gamma), \ldots, \mathcal{FN}(t_n\gamma))\downarrow = @(\mathcal{FN}(t_1\gamma)\downarrow, \ldots, \mathcal{FN}(t_n\gamma)\downarrow)\downarrow$.
   By induction hypothesis,
   $@(\mathcal{FN}(t_1)\mathcal{FN}(\gamma)\downarrow, \ldots, \mathcal{FN}(t_n)\mathcal{FN}(\gamma)\downarrow)\downarrow =$
   $@(\mathcal{FN}(t_1)\mathcal{FN}(\gamma), \ldots, \mathcal{FN}(t_n)\mathcal{FN}(\gamma))\downarrow =$
   $@(\mathcal{FN}(t_1), \ldots, \mathcal{FN}(t_n))\mathcal{FN}(\gamma)\downarrow =$
   $\mathcal{FN}(@(t_1, \ldots, t_n)\mathcal{FN}(\gamma))\downarrow = \mathcal{FN}(t)\mathcal{FN}(\gamma)\downarrow$.
4. $t = f(t_1, \ldots, t_n)$ with $f \in \mathcal{F}$. Then $\mathcal{FN}(t\gamma)\downarrow =$
   $\mathcal{FN}(f(t_1, \ldots, t_n)\gamma)\downarrow = \mathcal{FN}(f(t_1\gamma, \ldots, t_n\gamma))\downarrow =$
   $f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1\gamma), \mathcal{FN}(\bar{t}^1_f\gamma)), \ldots, \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n\gamma), \mathcal{FN}(\bar{t}^n_f\gamma)))\downarrow =$
   $f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1\gamma), \mathcal{FN}(\bar{t}^1_f\gamma))\downarrow, \ldots, \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n\gamma), \mathcal{FN}(\bar{t}^n_f\gamma))\downarrow)$.
   By lemma 9, this is equal to
   $f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1\gamma)\downarrow, \mathcal{FN}(\bar{t}^1_f\gamma)\downarrow)\downarrow, \ldots, \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n\gamma)\downarrow, \mathcal{FN}(\bar{t}^n_f\gamma)\downarrow)\downarrow)$
   and by induction hypothesis, to
   $f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1)\mathcal{FN}(\gamma)\downarrow, \mathcal{FN}(\bar{t}^1_f)\mathcal{FN}(\gamma)\downarrow)\downarrow, \ldots,$
   $\quad \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n)\mathcal{FN}(\gamma)\downarrow, \mathcal{FN}(\bar{t}^n_f)\mathcal{FN}(\gamma)\downarrow)\downarrow)$.
   By Lemma 9 used once again, this is equal to
   $f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1)\mathcal{FN}(\gamma), \mathcal{FN}(\bar{t}^1_f)\mathcal{FN}(\gamma)\downarrow)\downarrow, \ldots,$
   $\quad \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n)\mathcal{FN}(\gamma), \mathcal{FN}(\bar{t}^n_f)\mathcal{FN}(\gamma)\downarrow)\downarrow)$
   $= f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1)\mathcal{FN}(\gamma), \mathcal{FN}(\bar{t}^1_f)\mathcal{FN}(\gamma)), \ldots,$
   $\quad \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n)\mathcal{FN}(\gamma), \mathcal{FN}(\bar{t}^n_f)\mathcal{FN}(\gamma)))\downarrow =$
   $f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1), \mathcal{FN}(\bar{t}^1_f))\mathcal{FN}(\gamma), \ldots,$
   $\quad \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n), \mathcal{FN}(\bar{t}^n_f))\mathcal{FN}(\gamma))\downarrow$,
   by lemma 15. Extracting the substitution, we get
   $f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1), \mathcal{FN}(\bar{t}^1_f)), \ldots, \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n), \mathcal{FN}(\bar{t}^n_f)))\mathcal{FN}(\gamma)\downarrow$
   $= \mathcal{FN}(f(t_1, \ldots, t_n))\mathcal{FN}(\gamma)\downarrow = \mathcal{FN}(t)\mathcal{FN}(\gamma)\downarrow$. □

**Lemma 17.** *Let $t$ be a higher-order term. Then $\mathcal{FN}(t)\downarrow = \mathcal{FN}(t\downarrow)\downarrow$.*

*Proof.* We proceed by induction on $\longrightarrow_{\beta\eta} \cup \rhd$ which is well-founded [7]. There are four cases:

1. $t$ is a variable $x$. It follows from the fact that $x{\downarrow} = x$.
2. $t = \lambda x.u$. There are two cases.
   If $u = @(v, x)$, with $x \notin \mathcal{V}ar(v)$ then $\mathcal{FN}(t){\downarrow} = \mathcal{FN}(\lambda x.@(v, x)){\downarrow} = \lambda x.\mathcal{FN}(@(v, x)){\downarrow} = \lambda x.@(\mathcal{FN}(v), \mathcal{FN}(x)){\downarrow} = \lambda x.@(\mathcal{FN}(v), x){\downarrow} = \mathcal{FN}(v){\downarrow}$ since $x \notin \mathcal{V}ar(v) = \mathcal{V}ar(\mathcal{FN}(v){\downarrow})$. By induction hypothesis, we get $\mathcal{FN}(v){\downarrow} = \mathcal{FN}(v{\downarrow}){\downarrow}$. Since $\mathcal{FN}(v{\downarrow}){\downarrow} = \mathcal{FN}(\lambda x.@(v, x){\downarrow}){\downarrow}$, we are done.
   Otherwise, $(\lambda x.u){\downarrow} = \lambda x.u{\downarrow}$ and therefore $\mathcal{FN}(t){\downarrow} = \mathcal{FN}(\lambda x.u){\downarrow} = (\lambda x.\mathcal{FN}(u)){\downarrow} = (\lambda x.\mathcal{FN}(u){\downarrow}){\downarrow}$. By induction hypothesis, this is equal to $(\lambda x.\mathcal{FN}(u{\downarrow}){\downarrow}){\downarrow} = (\lambda x.\mathcal{FN}(u{\downarrow}){\downarrow}){\downarrow} = (\lambda x.\mathcal{FN}(u{\downarrow})){\downarrow} = \mathcal{FN}(\lambda x.u{\downarrow}){\downarrow} = \mathcal{FN}((\lambda x.u){\downarrow}){\downarrow}$ by our assumption and we are done.
3. $t = @(u, v)$. Then $\mathcal{FN}(t){\downarrow} = \mathcal{FN}(@(u, v)){\downarrow} = @(\mathcal{FN}(u), \mathcal{FN}(v)){\downarrow} = @(\mathcal{FN}(u){\downarrow}, \mathcal{FN}(v){\downarrow}){\downarrow}$. By induction hypothesis, this is equal to $@(\mathcal{FN}(u{\downarrow}){\downarrow}, \mathcal{FN}(v{\downarrow}){\downarrow}){\downarrow} = \mathcal{FN}(@(u{\downarrow}, v{\downarrow})){\downarrow}$.
   If $@(u{\downarrow}, v{\downarrow})$ is normalized then $\mathcal{FN}(@(u{\downarrow}, v{\downarrow})){\downarrow} = \mathcal{FN}(@(u, v){\downarrow}){\downarrow}$ and we are done.
   Otherwise, $u{\downarrow} = \lambda x.w$, hence $\mathcal{FN}(@(u{\downarrow}, v{\downarrow})){\downarrow} = \mathcal{FN}(@(\lambda x.w, v{\downarrow})){\downarrow} = @(\lambda x.\mathcal{FN}(w), \mathcal{FN}(v{\downarrow})){\downarrow} = \mathcal{FN}(w)\{x \mapsto \mathcal{FN}(v{\downarrow})\}{\downarrow}$. By Lemma 16, this is equal to $\mathcal{FN}(w\{x \mapsto v{\downarrow}\}){\downarrow}$. Since $@(u, v) \longrightarrow^{+}_{\beta\eta} w\{x \mapsto v{\downarrow}\}$, by induction hypothesis, this is equal to $\mathcal{FN}(w\{x \mapsto v{\downarrow}\}{\downarrow}){\downarrow} = \mathcal{FN}(@(u, v){\downarrow}){\downarrow}$.
4. $t = f(t_1, \ldots, t_n)$ with $f \in \mathcal{F}$. Then $\mathcal{FN}(t){\downarrow} = \mathcal{FN}(f(t_1, \ldots, t_n)){\downarrow} =$
   $f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1), \mathcal{FN}(\bar{t}^1_f)), \ldots, \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n), \mathcal{FN}(\bar{t}^n_f))){\downarrow} =$
   $f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1), \mathcal{FN}(\bar{t}^1_f){\downarrow}, \ldots, \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n), \mathcal{FN}(\bar{t}^n_f)){\downarrow})$.
   By Lemma 9, this is equal to
   $f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1){\downarrow}, \mathcal{FN}(\bar{t}^1_f){\downarrow}){\downarrow}, \ldots, \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n){\downarrow}, \mathcal{FN}(\bar{t}^n_f){\downarrow}){\downarrow})$
   hence, by induction hypothesis, to
   $f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1{\downarrow}){\downarrow}, \mathcal{FN}(\bar{t}^1_f{\downarrow}){\downarrow}){\downarrow}, \ldots, \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n{\downarrow}){\downarrow}, \mathcal{FN}(\bar{t}^n_f{\downarrow}){\downarrow}){\downarrow})$.
   By Lemma 9 again, this is equal to
   $= f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1{\downarrow}), \mathcal{FN}(\bar{t}^1_f{\downarrow})){\downarrow}, \ldots, \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n{\downarrow}), \mathcal{FN}(\bar{t}^n_f{\downarrow})){\downarrow})$
   $= f_{new}(\mathcal{N}_{nl^1_f}(\mathcal{FN}(t_1{\downarrow}), \mathcal{FN}(\bar{t}^1_f{\downarrow})), \ldots, \mathcal{N}_{nl^n_f}(\mathcal{FN}(t_n{\downarrow}), \mathcal{FN}(\bar{t}^n_f{\downarrow}))){\downarrow} =$
   $\mathcal{FN}(f(t_1{\downarrow}, \ldots, t_n{\downarrow})){\downarrow} = \mathcal{FN}(f(t_1, \ldots, t_n){\downarrow}){\downarrow}$. □

### 4.3 The Neutralized Ordering Schema

**Definition 10.** *Given two terms $s, t$ and a higher-order ordering $\succ$, we define the neutralized ordering $\succ_n$ as follows:*

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

We now need to show that $\succ_n$ is a normal, higher-order reduction ordering. Well-foundedness follows from well-foundedness of $\succ$, while compatibility follows from the definition. Stability and normal-monotonicity depend upon the particular ordering $\succ$ used in the construction.

**Definition 11.** *An ordering $\succ$ on higher-order terms satisfies*

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

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

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

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

The result follows from Lemmas 18, 19, 21, 23 and 24.

**Lemma 18.** *If $\succ$ is coherent then $\succ_n$ is coherent.*

For the proof to make sense, we need here to explicit the typing judgements.

*Proof.* We prove that for all terms $s, t$ such that $(\Gamma \vdash_{\mathcal{F}} s : \sigma) \succ_n (\Gamma \vdash_{\mathcal{F}} t : \sigma)$, and for all environment $\Gamma'$ such that $\Gamma$ and $\Gamma'$ are compatible, $\Gamma' \vdash_{\mathcal{F}} s : \sigma$ and $\Gamma' \vdash_{\mathcal{F}} t : \sigma$, then $(\Gamma' \vdash_{\mathcal{F}} s : \sigma) \succ_n (\Gamma' \vdash_{\mathcal{F}} t : \sigma)$. By coherence of $\succ$, $(\Gamma \vdash_{\mathcal{F}} \mathcal{FN}(s){\downarrow} : \sigma) \succ (\Gamma \vdash_{\mathcal{F}} \mathcal{FN}(t){\downarrow} : \sigma)$ implies $(\Gamma' \vdash_{\mathcal{F}} \mathcal{FN}(s){\downarrow} : \sigma) \succ (\Gamma' \vdash_{\mathcal{F}} \mathcal{FN}(t){\downarrow} : \sigma)$, and therefore $(\Gamma' \vdash_{\mathcal{F}} s : \sigma) \succ_n (\Gamma' \vdash_{\mathcal{F}} t : \sigma)$. □

**Lemma 19.** *If $\succ$ is polymorphic then $\succ_n$ is polymorphic.*

*Proof.* Let $s, t$ be higher-order terms and $\xi$ a type substitution. We prove that $s \succ_n t$ implies $s\xi \succ_n t\xi$. By definition, $\mathcal{FN}(s){\downarrow} \succ \mathcal{FN}(t){\downarrow}$ and, by polymorphism of $\succ$, $\mathcal{FN}(s){\downarrow}\, \xi \succ \mathcal{FN}(t){\downarrow}\, \xi$. By lemma 12 $\mathcal{FN}(s){\downarrow}\, \xi = \mathcal{FN}(s)\xi{\downarrow}$ and $\mathcal{FN}(t){\downarrow}\, \xi = \mathcal{FN}(t)\xi{\downarrow}$. By lemma 14, $\mathcal{FN}(s)\xi = \mathcal{FN}(s\xi)$ and $\mathcal{FN}(t)\xi = \mathcal{FN}(t\xi)$. Altogether, we get $\mathcal{FN}(s\xi){\downarrow} = \mathcal{FN}(s)\xi{\downarrow} = \mathcal{FN}(s){\downarrow}\, \xi \succ \mathcal{FN}(t){\downarrow}\, \xi = \mathcal{FN}(t)\xi{\downarrow} = \mathcal{FN}(t\xi){\downarrow}$ which implies $s\xi \succ_n t\xi$. □

**Lemma 20.** *Let $t$ be a higher-order $\beta\eta$-normal term and $\gamma$ be a $\beta\eta$-normal substitution. Then $t\gamma{\downarrow} = t\gamma{\downarrow}_\beta$.*

**Lemma 21.** *Assume $\succ$ satisfies schema-stability and functionality. Then $\succ_n$ is stable.*

*Proof.* Let $s, t$ be higher-order terms and $\gamma$ a substitution. We prove that $s \succ_n t$ implies $s\gamma \succ_n t\gamma$, that is, $\mathcal{FN}(s)\downarrow \succ \mathcal{FN}(t)\downarrow$ implies $\mathcal{FN}(s\gamma)\downarrow \succ \mathcal{FN}(t\gamma)\downarrow$. By Lemma 16 and the confluence prop3erty of $\beta\eta$-reduction, $\mathcal{FN}(s\gamma)\downarrow = (\mathcal{FN}(s)\downarrow \mathcal{FN}(\gamma)\downarrow)\downarrow =$ and $\mathcal{FN}(t\gamma)\downarrow = (\mathcal{FN}(t)\downarrow \mathcal{FN}(\gamma)\downarrow)\downarrow$. Let $s_1$ and $t_1$ be the $\beta\eta$-normal terms $\mathcal{FN}(s)\downarrow$ and $\mathcal{FN}(t)\downarrow$ respectively, and let $\gamma'$ be the $\beta\eta$-normal substitution $\mathcal{FN}(\gamma)\downarrow$. By Lemma 20, $s_1\gamma'\downarrow = s_1\gamma'\downarrow_\beta$ and $t_1\gamma'\downarrow = t_1\gamma'\downarrow_\beta$.

Since $s_1 \succ t_1$, by schema-stability of $\succ$ we get $s_1\gamma'\downarrow_\beta \succ t'\gamma'$ for some term $t'$ such that $t_1\gamma' \longrightarrow^*_\beta t'\gamma'$. By confluence, $t'\gamma' \longrightarrow^*_\beta t_1\gamma'\downarrow_\beta$. By functionality of $\succ$, it follows that $t'\gamma' \succ t_1\gamma'\downarrow_\beta$, and therefore, by transitivity, $s_1\gamma'\downarrow = s_1\gamma'\downarrow_\beta \succ t_1\gamma'\downarrow_\beta = t_1\gamma'\downarrow$. □

**Lemma 22.** *Assume that $\succ$ is monotonic and satisfies the schema-replacement property. Assume further that $s : \tau, t : \tau$ and $u_1, \ldots u_n, v_1, \ldots v_n$, are $\beta\eta$-normal terms satisfying $s \succ t$ and $u_i \succeq v_i$ for all $i \in \{1 \ldots n\}$. Then,*

$$N_i(s, \langle u_1, \ldots u_n \rangle)\downarrow \succ N_i(t, \langle v_1, \ldots v_n \rangle)\downarrow$$

*Proof.* We proceed by induction on $i$. There are two cases:

1. If $i = 0$ or $\tau$ is a data type, then $N_i(s)\downarrow = s\downarrow = s \succ t = t\downarrow = N_i(t)\downarrow$.
2. Otherwise, $i > 0$ and $\tau = \sigma \rightarrow \rho$. Then
   $N_i(s, \langle u_1, \ldots u_n \rangle)\downarrow =$
   $N_{i-1}(@(s, \bot_\theta(u_1, \ldots u_n)))\downarrow$ and
   $N_i(t, \langle v_1, \ldots v_n \rangle)\downarrow = N_{i-1}(@(t, \bot_\theta(v_1, \ldots v_n)))\downarrow$. By Lemma 9
   $N_i(s)\downarrow = N_{i-1}(@(s, \bot_\theta(u_1, \ldots u_n))\downarrow)\downarrow$ and
   $N_i(t)\downarrow = N_{i-1}(@(t, \bot_\theta(v_1, \ldots v_n))\downarrow)\downarrow$.
   We are left showing that $s : \tau \succ t : \tau$ implies $@(s, \bot_\theta(u_1, \ldots u_n))\downarrow \succ @(t, \bot_\theta(v_1, \ldots v_n))\downarrow$. There are four cases:
   (a) If $s = \lambda x.u$ and $t = \lambda x.v$ then $@(s, \bot_\sigma)\downarrow = u\{x \mapsto \bot_\theta(u_1, \ldots u_n)\}$ and $@(t, \bot_\sigma)\downarrow = v\{x \mapsto \bot_\theta(v_1, \ldots v_n)\}$. By monotonicity and transitivity of $\succ$, $\bot_\theta(u_1, \ldots u_n) \succeq \bot_\theta(v_1, \ldots v_n)$. Hence, by schema-replacement property, $@(s, \bot_\theta(u_1, \ldots u_n))\downarrow \succ @(t, \bot_\theta(v_1, \ldots v_n))\downarrow$,
   (b) If $s = \lambda x.u$ and $t$ is not an abstraction, then
   $@(s, \bot_\theta(u_1, \ldots u_n))\downarrow = u\{x \mapsto \bot_\theta(u_1, \ldots u_n)\}$ and
   $@(t, \bot_\theta(v_1, \ldots v_n))\downarrow = @(t, \bot_\theta(v_1, \ldots v_n))$.
   By the schema replacement property as before, we get
   $@(s, \bot_\theta(u_1, \ldots u_n))\downarrow \succ @(t, \bot_\theta(v_1, \ldots v_n))\downarrow$.

(c) If $s, t$ are not abstractions, then
$@(s, \bot_\theta(u_1, \dots u_n))\!\downarrow = @(s, \bot_\theta(u_1, \dots u_n))$ and
$@(t, \bot_\theta(v_1, \dots v_n))\!\downarrow = @(t, \bot_\theta(v_1, \dots v_n))$.
By monotonicity and transitivity we get
$@(s, \bot_\theta(u_1, \dots u_n))\!\downarrow \succ @(t, \bot_\theta(v_1, \dots v_n))\!\downarrow$.

(d) If $s$ is not an abstraction and $t = \lambda x.v$ then,
$@(s, \bot_\theta(u_1, \dots u_n))\!\downarrow = @(s, \bot_\theta(u_1, \dots u_n))$ and
$@(t, \bot_\theta(v_1, \dots v_n))\!\downarrow = v\{x \mapsto \bot_\theta(v_1, \dots v_n)\}$.
Now, by monotonicity, transitivity and functionality we get
$@(s, \bot_\theta(u_1, \dots u_n))\!\downarrow = @(s, \bot_\theta(u_1, \dots u_n)) \succ$
$v\{x \mapsto \bot_\theta(v_1, \dots v_n)\} = @(t, \bot_\theta(v_1, \dots v_n))\!\downarrow$.

Finally, since $@(s, \bot_\theta(u_1, \dots u_n))\!\downarrow \succ @(t, \bot_\theta(v_1, \dots v_n))\!\downarrow$, we can conclude by induction hypothesis: $N_i(s)\!\downarrow = N_{i-1}(@(s, \bot_\theta(u_1, \dots u_n))\!\downarrow)\!\downarrow \succ$
$N_{i-1}(@(t, \bot_\theta(v_1, \dots v_n))\!\downarrow)\!\downarrow = N_i(t)\!\downarrow$. □

**Lemma 23.** *Assume that $\succ$ satisfies the monotonicity, functionality and schema replacement properties. Then $\succ_n$ is normal-monotonic.*

*Proof.* Let $s, t$ be higher-order terms and $u[]$ a ground context such that $u[s]$ is in $\beta\eta$-normal form. We prove that $s \succ_n t$ implies $u[s] \succ_n u[t]$, that is, $\mathcal{FN}(s)\!\downarrow \succ \mathcal{FN}(t)\!\downarrow$ implies $\mathcal{FN}(u[s])\!\downarrow \succ \mathcal{FN}(u[t])\!\downarrow$.

We proceed by induction on the $|u|$. If $u$ is empty we are done. Otherwise, there are four cases:

1. $u[] = \lambda x.u'[]$. By Lemma 11, $\mathcal{FN}(u[s])\!\downarrow = \lambda x.\mathcal{FN}(u'[s])\!\downarrow$ and $\mathcal{FN}(u[t])\!\downarrow = \lambda x.\mathcal{FN}(u'[t])\!\downarrow$. By induction hypothesis, $\mathcal{FN}(u'[s])\!\downarrow \succ \mathcal{FN}(u'[t])\!\downarrow$, and, by monotonicity of $\succ$, we conclude that $\mathcal{FN}(u[s])\!\downarrow = \lambda x.\mathcal{FN}(u'[s])\!\downarrow \succ \lambda x.\mathcal{FN}(u'[t])\!\downarrow = \mathcal{FN}(u[t])\!\downarrow$.
2. $u[] = @(v, u'[])$. By Lemma 11, $\mathcal{FN}(u[s])\!\downarrow = @(\mathcal{FN}(v)\!\downarrow, \mathcal{FN}(u'[s])\!\downarrow)$ and $\mathcal{FN}(u[t])\!\downarrow = @(\mathcal{FN}(v)\!\downarrow, \mathcal{FN}(u'[t])\!\downarrow)$. By induction hypothesis, $\mathcal{FN}(u'[s])\!\downarrow \succ \mathcal{FN}(u'[t])\!\downarrow$, and, by monotonicity of $\succ$, we conclude that $@(\mathcal{FN}(v)\!\downarrow, \mathcal{FN}(u'[s])\!\downarrow) \succ @(\mathcal{FN}(v)\!\downarrow, \mathcal{FN}(u'[t])\!\downarrow)$.
3. $u[] = @(u'[], v)$. By Lemma 11, $\mathcal{FN}(u[s])\!\downarrow = @(\mathcal{FN}(u'[s])\!\downarrow, \mathcal{FN}(v)\!\downarrow)$ and $\mathcal{FN}(u[t])\!\downarrow = @(\mathcal{FN}(u'[t])\!\downarrow, \mathcal{FN}(v)\!\downarrow)$. By induction hypothesis, $\mathcal{FN}(u'[s])\!\downarrow \succ \mathcal{FN}(u'[t])\!\downarrow$, and, by monotonicity of $\succ$, we conclude as before.
4. $u[] = f(\dots u'[] \dots)$. By Lemma 11,
$\mathcal{FN}(u[s])\!\downarrow = f_{new}(\dots \mathcal{N}_{nl_f^k}(\mathcal{FN}(u'[s])\!\downarrow, \mathcal{FN}(\overline{u[s]}_f^k)\!\downarrow)\!\downarrow \dots)$ and
$\mathcal{FN}(u[t])\!\downarrow = f_{new}(\dots \mathcal{N}_{nl_f^k}(\mathcal{FN}(u'[t])\!\downarrow, \mathcal{FN}(\overline{u[t]}_f^k)\!\downarrow)\!\downarrow \dots)$.

By induction hypothesis, $\mathcal{FN}(u'[s])\downarrow\succ \mathcal{FN}(u'[t])\downarrow$, and by Lemma 22 $\mathcal{N}_{nl_f^k}(\mathcal{FN}(u'[s])\downarrow, \mathcal{FN}(\overline{u[s]}_f^k)\downarrow)\downarrow\succ \mathcal{N}_{nl_f^k}(\mathcal{FN}(u'[t])\downarrow, \mathcal{FN}(\overline{u[t]}_f^k)\downarrow)\downarrow$ (note that, since $k$ may be in $\mathcal{A}_f^k$ the arguments used for neutralization in $\mathcal{N}_{nl_f^k}(\mathcal{FN}(u'[s])\downarrow, \mathcal{FN}(\overline{u[s]}_f^k)\downarrow)\downarrow$ are, by induction, greater than or equal to the arguments used in $\mathcal{N}_{nl_f^k}(\mathcal{FN}(u'[t])\downarrow, \mathcal{FN}(\overline{u[t]}_f^k)\downarrow)\downarrow$).
Finally, by monotonicity of $\succ$, we conclude that
$f_{new}(\ldots \mathcal{N}_{nl_f^k}(\mathcal{FN}(u'[s])\downarrow, \mathcal{FN}(\overline{u[s]}_f^k)\downarrow)\ldots) \succ$
$f_{new}(\ldots \mathcal{N}_{nl_f^k}(\mathcal{FN}(u'[t])\downarrow, \mathcal{FN}(\overline{u[t]}_f^k)\downarrow)\ldots)$. □

**Lemma 24.** *$\succ_n$ is compatible.*

*Proof.* Let $s', s, t, t'$ be higher-order terms. Then we show that $s' =_{\beta\eta} s \succ_n t =_{\beta\eta} t'$ implies $s' \succ_n t'$. By assumption, $s\downarrow = s'\downarrow$ and $t\downarrow = t'\downarrow$, and by lemma 17
$\mathcal{FN}(s)\downarrow = \mathcal{FN}(s\downarrow)\downarrow = \mathcal{FN}(s'\downarrow)\downarrow = \mathcal{FN}(s')\downarrow$ and
$\mathcal{FN}(t)\downarrow = \mathcal{FN}(t\downarrow)\downarrow = \mathcal{FN}(t'\downarrow)\downarrow = \mathcal{FN}(t')\downarrow$.
Therefore, $s \succ_n t$ implies $\mathcal{FN}(s')\downarrow = \mathcal{FN}(s)\downarrow\succ \mathcal{FN}(t)\downarrow = \mathcal{FN}(t')\downarrow$.
Hence $\mathcal{FN}(s')\downarrow\succ \mathcal{FN}(t')\downarrow$ and therefore $s' \succ_n t'$. □

## 5 The Normal Higher-Order Recursive Path Ordering

Our purpose here is to define a higher-order ordering satisfying schema-stability and schema-replacement. The higher-order recursive path ordering $\succ_{horpo}$ [10] is not an appropriate answer, because it does not satisfy schema-stability. Fortunately, a simple restriction suffices in case of an application on left (Cases 5 and 9 of our coming definition) resulting in the *restricted higher-order recursive path ordering*. We assume given:

1. a partition $Mul \uplus Lex$ of $\mathcal{F}$;
2. a quasi-ordering $\geq_{\mathcal{F}}$ on $\mathcal{F}$, called the *precedence*, such that $>_{\mathcal{F}}$ is well-founded;
3. a well-founded quasi-ordering $\geq_{\mathcal{T_S}}$ on types such that for all ground types $\tau, \sigma, \alpha$,

$$\tau \to \sigma =_{\mathcal{T_S}} \alpha \text{ iff } \alpha = \tau' \to \sigma',\ \tau' =_{\mathcal{T_S}} \tau \text{ and } \sigma =_{\mathcal{T_S}} \sigma'$$
and
$$\tau \to \sigma >_{\mathcal{T_S}} \alpha \text{ implies } \sigma \geq_{\mathcal{T_S}} \alpha \text{ or } \alpha = \tau' \to \sigma',\ \tau' =_{\mathcal{T_S}} \tau \text{ and } \sigma >_{\mathcal{T_S}} \sigma'$$

**Definition 12.** *Given $s : \sigma$ and $t : \tau$, $s \underset{rhorpo}{\succ} t$ iff $\sigma \geq_{\mathcal{T_S}} \tau$ and*

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

*where $A = \forall v \in \overline{t}\ \ s \underset{rhorpo}{\succ} v$ or $u \underset{horpo}{\succeq} v$ for some $u \in \overline{(}s)$*

Of course, making bound variables fit may need renaming in Case 10. Note that we could add the definition $@(X, \overline{s}) \succeq_{rhorpo} @(X, \overline{t})$ if $\overline{s}(\succeq_{rhorpo})_{mon}\overline{t}$ which would yield a strictly richer relation with the same properties. We do not since this would force us defining explicitely the equivalence relation to be added to the strict ordering.

**Lemma 25.** *$(\succ_{rhorpo})^*$ is a higher-order reduction ordering satisfying schema-stability and schema-replacement.*

*Proof.* Containement of $\succ_{rhorpo}$ into $\succ_{horpo}$ is clear, which implies well-foundedness of $\succ_{rhorpo}$. So are monotonicity, since contexts are assumed ground, and polymorphism, since type instantiation does not introduce redexes of any kind in terms. Coherence and functionality are both straithforward. Schema-replacement forces the use of Rule 10, for typing reason. It then follows by an easy induction. We are left with schema-stability, which implies stability. Schema-stability, i.e., $s \succ_{rhorpo} t$ where $s, t$ are normal terms, implies $t\gamma \longrightarrow^* t'\gamma$ for some $t'$ such that $s\gamma\downarrow \succ_{rhorpo} t'\gamma$, is proved by induction on the definition of the ordering.

1. Assume $s \succ_{rhorpo} t$ by Case 1, hence $u \succ_{rhorpo} t$ for some $u \in \mathcal{CC}(s)$. By induction hypothesis, there exists some $t'$ such that $t\gamma \longrightarrow^* t'\gamma$ and $u\gamma{\downarrow} \succ_{rhorpo} t'\gamma$. Since $s$ is headed by a symbol in $\mathcal{F} \cup \mathcal{S}$, $u\gamma{\downarrow} \in \mathcal{CC}(s\gamma{\downarrow})$, hence $s\gamma{\downarrow} \succ_{rhorpo} t'\gamma$ by Case 1.
2. Assume $s \succ_{rhorpo} t$ by Case 2, hence $s = f(\overline{s}), t = g(\overline{t}), f >_{\mathcal{FS}} g$ and, for every $v \in \overline{t}$, $s \succ_{rhorpo} v$, hence, by induction hypothesis, there exists $v'$ such that $v\gamma \longrightarrow^* v'\gamma$ and $s\gamma{\downarrow} \succ_{rhorpo} v'\gamma$. Let $t' = g(\overline{v'})$, hence $t\gamma \longrightarrow^* t'\gamma$. Since $s\gamma{\downarrow}$ is headed by $f$, by Case 2, $s\gamma{\downarrow} \succ_{rhorpo} t'\gamma$.
5. Assume $s \succ_{rhorpo} t$ by Case 5. By induction hypothesis, $t\gamma \longrightarrow^* t'\gamma$ for some $t'$ and $u\gamma{\downarrow} \succ_{rhorpo} t'\gamma$ for some $u \in \{s_1, s_2\}$. Our assumption on $s_1$ ensures that $s\gamma{\downarrow} = @(s_1\gamma{\downarrow}, s_2\gamma{\downarrow})$, allowing us to conclude by Case 5.
9. Assume $s \succ_{rhorpo} t$ by Case 9. This case is similar to case 5.
11. Assume $s \succ_{rhorpo} t$ by Case 11 or 12. These cases cannot happen since $s$ is assumed in normal form.
12. All other cases are either similar to the above ones or straightforward. $\square$

As a consequence,

**Theorem 3.** *$(\succ_{rhorpo})^*_n$ is a normal higher-order reduction ordering.*

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

*Example 1 (end).* Let $\mathsf{diff_{new}} >_{\mathcal{F}} \{\times_{\mathsf{new}}, +_{\mathsf{new}}, \mathsf{cos}\}$ and $\mathsf{diff_{new}} \in \mathsf{Mul}$.
**First rule:** $\mathsf{diff_{new}}(\mathsf{sin}(@(\mathsf{F}, \perp))) \succ_{\mathsf{rhorpo}} \mathsf{cos}(@(\mathsf{F}, \perp)) \times_{\mathsf{new}} @(\mathsf{diff_{new}}(@(\mathsf{F}, \perp)), \perp)$
Applying first case 2, we recursively obtain two subgoals:
(i) $\mathsf{diff_{new}}(\mathsf{sin}(@(\mathsf{F}, \perp))) \succ_{\mathsf{rhorpo}} \mathsf{cos}(@(\mathsf{F}, \perp))$
(ii) $\mathsf{diff_{new}}(\mathsf{sin}(@(\mathsf{F}, \perp))) \succ_{\mathsf{rhorpo}} @(\mathsf{diff_{new}}(@(\mathsf{F}, \perp)), \perp)$.
(i): applying Case 2 yields $\mathsf{diff_{new}}(\mathsf{sin}(@(\mathsf{F}, \perp))) \succ_{\mathsf{rhorpo}} @(\mathsf{F}, \perp)$ shown by Case 1 twice.
(ii): applying Case 7 generates two new subgoals
(iii) $\mathsf{diff_{new}}(\mathsf{sin}(@(\mathsf{F}, \perp))) \succ_{\mathsf{rhorpo}} \mathsf{diff_{new}}(@(\mathsf{F}, \perp))$ , which holds by case 3, then case 1.
(iv) $\mathsf{diff_{new}}(\mathsf{sin}(@(\mathsf{F}, \perp))) \succ_{\mathsf{rhorpo}} \perp$ , which holds by case 1 twice and then case 5.
**Second rule:** $\mathsf{diff_{new}}(@(@(\mathsf{F}, \perp) \times_{\mathsf{new}} @(\mathsf{F}, \perp), \perp)) \succ_{\mathsf{rhorpo}}$
$@(@(\mathsf{diff_{new}}(@(\mathsf{F}, \perp)), \perp) \times_{\mathsf{new}} @(\mathsf{F}, \perp), \perp) +_{\mathsf{new}} @(@(\mathsf{F}, \perp) \times_{\mathsf{new}} @(\mathsf{diff_{new}}(@(\mathsf{F}, \perp)), \perp), \perp)$
Case 2 generates two subgoals:
(i) $\mathsf{diff_{new}}(@(@(\mathsf{F}, \perp) \times_{\mathsf{new}} @(\mathsf{F}, \perp), \perp)) \succ_{\mathsf{rhorpo}} @(@(\mathsf{diff_{new}}(@(\mathsf{F}, \perp)), \perp) \times_{\mathsf{new}} @(\mathsf{F}, \perp), \perp)$
(ii) $\mathsf{diff_{new}}(@(@(\mathsf{F}, \perp) \times_{\mathsf{new}} @(\mathsf{F}, \perp), \perp)) \succ_{\mathsf{rhorpo}} @(@(\mathsf{F}, \perp) \times_{\mathsf{new}} @(\mathsf{diff_{new}}(@(\mathsf{F}, \perp)), \perp), \perp)$.
By Case 7, (i) generates two new subgoals:

(iii) $\mathsf{diff}_{\mathsf{new}}(@(@(\mathsf{F},\bot)\times_{\mathsf{new}} @(\mathsf{F},\bot),\bot)) \succ_{\mathsf{rhorpo}} @(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot)),\bot)\times_{\mathsf{new}} @(\mathsf{F},\bot)$
(iv) $\mathsf{diff}_{\mathsf{new}}(@(@(\mathsf{F},\bot)\times_{\mathsf{new}} @(\mathsf{F},\bot),\bot)) \succ_{\mathsf{rhorpo}} \bot$.
The latter holds by case 1 and then case 5. By Case 2, (iv) yields two subgoals:
(v) $\mathsf{diff}_{\mathsf{new}}(@(@(\mathsf{F},\bot)\times_{\mathsf{new}} @(\mathsf{F},\bot),\bot)) \succ_{\mathsf{rhorpo}} @(\mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot)),\bot)$
(vi) $\mathsf{diff}_{\mathsf{new}}(@(@(\mathsf{F},\bot)\times_{\mathsf{new}} @(\mathsf{F},\bot),\bot)) \succ_{\mathsf{rhorpo}} @(\mathsf{F},\bot)$.
By Case 7, (v) generates
(vii) $\mathsf{diff}_{\mathsf{new}}(@(@(\mathsf{F},\bot)\times_{\mathsf{new}} @(\mathsf{F},\bot),\bot)) \succ_{\mathsf{rhorpo}} \mathsf{diff}_{\mathsf{new}}(@(\mathsf{F},\bot))$
(viii) $\mathsf{diff}_{\mathsf{new}}(@(@(\mathsf{F},\bot)\times_{\mathsf{new}} @(\mathsf{F},\bot),\bot)) \succ_{\mathsf{rhorpo}} \bot$ solved by Case 1 first, then 5.
Finally, (vii) is solved by Case 3, 5, and 1 successively.

## 6 Examples

We present here several complex examples proven terminating with the normal higher-order vrecursive path ordering. For all of them, we give the necessary ingredients for computing the appropriate neutralizations and comparisons. 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 reverse choice, to make clear that $@$ is the top function symbol of the term $F(X)$.

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

*Example 2.* The coming encoding of first-order prenex normal forms is adapted from [17], where its local confluence is proved via the computation of its (higher-order) critical pairs. Formulas are represented as $\lambda$-terms with sort $form$. The idea is that quantifiers are higher-order constructors binding a variable via the use of a functional argument.

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

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

Ingredients for neutralization: $\mathcal{L}^1_\forall=1$, $\mathcal{L}^1_\exists=1$, $\mathcal{A}^1_\forall=\{\}$, $\mathcal{A}^1_\exists=\{\}$.

Statuses: $\forall_{\mathsf{new}}, \exists_{\mathsf{new}} \in \mathsf{Mul}$
Precedence: $\wedge >_{\mathcal{F}} \{\forall_{\mathsf{new}}, \exists_{\mathsf{new}}\}$, $\vee >_{\mathcal{F}} \{\forall_{\mathsf{new}}, \exists_{\mathsf{new}}\}$, $\neg >_{\mathcal{F}} \{\forall_{\mathsf{new}}, \exists_{\mathsf{new}}\}$. $\Box$

We now carry out the proof of the first and the last rule. All others follow the same pattern.

$$P \wedge \forall(\lambda x.@(Q,x)) \to \forall(\lambda x.(P \wedge @(Q,x))) \quad (1)$$
$$\neg(\exists(\lambda x.@(Q,x))) \to \forall(\lambda x.\neg(@(Q,x))) \quad (8)$$

The ingredients for the proof are the following.
For neutralization we have: $\mathcal{L}^1_\forall = 1$, $\mathcal{L}^1_\exists = 1$, $\mathcal{A}^1_\forall = \{\}$, $\mathcal{A}^1_\exists = \{\}$.
For the ordering we have $\forall_{\mathsf{new}}, \exists_{\mathsf{new}}, \neg \in \mathsf{Mul}$ and as precedence:
$\wedge >_{\mathcal{F}} \{\forall_{\mathsf{new}}, \exists_{\mathsf{new}}\}$, $\vee >_{\mathcal{F}} \{\forall_{\mathsf{new}}, \exists_{\mathsf{new}}\}$, $\neg >_{\mathcal{F}} \{\forall_{\mathsf{new}}, \exists_{\mathsf{new}}\}$.
In the following proof we abbreviate $\bot_{form}$ as $\bot$.

We start with the proof of rule (1). Let us first compute the full neutralization of both sides:
$\mathcal{FN}(P \wedge \forall(\lambda x.@(Q,x))) = P \wedge \forall_{new}(@(Q,\bot))$, and
$\mathcal{FN}(\forall(\lambda x.(P \wedge @(Q,x)))) = \forall_{new}(P \wedge @(Q,\bot))$.
We show that $P \wedge \forall_{new}(@(Q,\bot)) \succ_{rhorpo} \forall_{new}(P \wedge @(Q,\bot))$. By case 2 we need to show $P \wedge \forall_{new}(@(Q,\bot)) \succ_{rhorpo} P \wedge @(Q,\bot)$, and by case 3 we need $\{P, \forall_{new}(@(Q,\bot))\}(\succ_{rhorpo})_{mul}\{P, @(Q,\bot)\}$. This requires $\forall_{new}(@(Q,\bot)) \succ_{rhorpo} @(Q,\bot)$ which holds by case 1.

Let us prove now rule (8). The full neutralization of both sides are:
$\mathcal{FN}(\neg(\exists(\lambda x.@(Q,x)))) = \neg(\exists_{new}(@(Q,\bot)))$, and
$\mathcal{FN}(\forall(\lambda x.\neg(@(Q,x))) = \forall_{new}(\neg(@(Q,\bot)))$.
Now we show that $\neg(\exists_{new}(@(Q,\bot))) \succ_{rhorpo} \forall_{new}(\neg(@(Q,\bot)))$. By case 2 we need to show $\neg(\exists_{new}(@(Q,\bot))) \succ_{rhorpo} \neg(@(Q,\bot))$. Applying case 3 we are left with $\{\exists_{new}(@(Q,\bot))\}(\succ_{rhorpo})_{mul}\{@(Q,\bot)\}$, which holds by case 1. $\Box$

*Example 3.* This example of surjective disjoint union is taken from [21]. Signature and rules are parameterized by $\alpha \in \mathcal{S} = \{A, B, U\}$:

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

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

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 and Precedence: any choice will do.

Let us rewrite the rules with explicit applications.

$$\begin{aligned} case_\alpha(inl(X), F, G) &\to @(F, X) \ (1) \\ case_\alpha(inr(Y), F, G) &\to @(G, Y) \ (2) \\ case_\alpha(Z, \lambda x.@(H, inl(x)), \lambda y.@(H, inr(y))) &\to @(H, Z) \ (3) \end{aligned}$$

For neutralization we have: $\mathcal{L}^2_{\mathsf{case}_\alpha}$=1, $\mathcal{L}^3_{\mathsf{case}_\alpha}$=1, $\mathcal{A}^2_{\mathsf{case}_\alpha}$=$\{1\}$, $\mathcal{A}^3_{\mathsf{case}_\alpha}$=$\{1\}$. For this example we can use the empty precedence.

We start with the proof of rule (1). Let us first compute the full neutralization of both sides:
$s = \mathcal{FN}(case_\alpha(inl(X), F, G)) =$
$case_{new}(inl(X), @(F, \bot_{U\to A}(inl(X))), @(G, \bot_{U\to B}(inl(X))))$, and
$t = \mathcal{FN}(@(F, X)) = @(F, X)$.

Now we prove $s \succ_{rhorpo} t$:
$case_{new}(inl(X), @(F, \bot_{U\to A}(inl(X))), @(G, \bot_{U\to B}(inl(X)))) \succ_{rhorpo}$ $@(F, X)$, by case 1, requiring $@(F, \bot_{U\to A}(inl(X))) \succ_{rhorpo} @(F, X)$. This, by case 9, needs $\{F, \bot_{U\to A}(inl(X))\}(\succ_{rhorpo})_{mul}\{F, X\}$, and thus $\bot_{U\to A}(inl(X)) \succ_{rhorpo} X$, which holds by applying case 1 twice.

For rule (2), similarly the full neutralization of both sides is:
$s = \mathcal{FN}(case_\alpha(inr(Y), F, G)) =$
$case_{new}(inr(Y), @(F, \bot_{U\to A}(inr(Y))), @(G, \bot_{U\to B}(inr(Y))))$, and
$t = \mathcal{FN}(@(G, Y)) = @(G, Y)$.

Then the proof of $s \succ_{rhorpo} t$ is exactly as the one for rule (1).

Finally we prove rule (3). The full neutralization of both sides is:
$s = \mathcal{FN}(case_\alpha(Z, \lambda x.@(H, inl(x)), \lambda y.@(H, inr(y)))) =$
$case_{new}(Z, @(H, inl(\bot_{U\to A}(Z))), @(H, inr(\bot_{U\to B}(Z))))$, and
$t = \mathcal{FN}(@(H, Z)) = @(H, Z)$.

Let us show that $s \succ_{rhorpo} t$:
$case_{new}(Z, @(H, inl(\bot_{U\to A}(Z))), @(H, inr(\bot_{U\to B}(Z)))) \succ_{rhorpo}$ $@(H, Z)$ by case 1 requiring $@(H, inl(\bot_{U\to A}(Z))) \succ_{rhorpo} @(H, Z)$. This, by case 9, needs $\{H, inl(\bot_{U\to A}(Z))\}(\succ_{rhorpo})_{mul}\{H, Z\}$, and thus $inl(\bot_{U\to A}(Z)) \succ_{rhorpo} Z$, which holds by applying case 1 twice. □

*Example 4.* Encoding of natural deduction, taken from [20].

Let $\mathcal{S} = \{o,\ c : * \times * \to *\}$, and $\mathcal{S}^{\forall} = \sigma, \tau, \rho$. The signature follows:

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

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

Let us now give the rules:

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

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

We now give the ingredients for neutralization:
Neutralization: $\mathcal{L}^2_{\exists^-_{\sigma,\tau}} = 2$ and $\mathcal{A}^2_{\exists^-_{\sigma,\tau}} = \{1\}$ for all possible types $\sigma$ and $\tau$.
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 possible types $\rho$ $\sigma$ and $\tau$;
Precedence: $\{app_{\sigma,\tau}, \Pi^0_{\sigma,\tau}, \Pi^1_{\sigma,\tau}\} >_{\mathcal{F}} \exists^-_{new\ \rho,\tau}$ and $\exists^-_{new\ \rho,\tau} = \exists^-_{new\ \sigma,\tau}$ for all possible types $\rho$, $\sigma$ and $\tau$.

We start with the proof of rule (1). The full neutralization do not change any of both sides. Then we have to prove that
$app_{\sigma,\tau}(abs_{\sigma,\tau}(F), X) \succ_{rhorpo} @(F, X)$, which holds by case 7 and case 1 for showing $abs_{\sigma,\tau}(F) \succ_{rhorpo} F$.

For rule (2). The full neutralization do not change any of both sides. Then we have to prove that
$\Pi^0_{\sigma,\tau}(\Pi_{\sigma,\tau}(X,Y)) \succ_{rhorpo} X$, which holds by case 1 twice.
For rule (3) we have the same proof as for rules (2).

For rule (4). The full neutralization of both sides is:
$s = \mathcal{FN}(\exists^-_{\sigma,\tau}(\exists^+_{\sigma}(Z,X), G)) =$
$\exists^-_{new\ \sigma,\tau}(\exists^+_{\sigma}(Z,X), @(G, \perp_{c(o,\sigma)\to o}(\exists^+_{\sigma}(Z,X)), \perp_{c(o,\sigma)\to\sigma}(\exists^+_{\sigma}(Z,X))))$,
and $t = \mathcal{FN}(@(G,Z,X)) = @(G,Z,X)$.
Let us prove $s \succ_{rhorpo} t$. Applying case 1 we need
$@(G, \perp_{c(o,\sigma)\to o}(\exists^+_{\sigma}(Z,X)), \perp_{c(o,\sigma)\to\sigma}(\exists^+_{\sigma}(Z,X))) \succ_{rhorpo} @(G,Z,X)$, and,

by case 9, it requires $\{G, \bot_{c(o,\sigma)\to o}(\exists^+_\sigma(Z,X)), \bot_{c(o,\sigma)\to\sigma}(\exists^+_\sigma(Z,X))\}(\succ_{rhorpo})_{mul}\{G,Z,X\}$, and thus, $\bot_{c(o,\sigma)\to o}(\exists^+_\sigma(Z,X)) \succ_{rhorpo} Z$ and $\bot_{c(o,\sigma)\to\sigma}(\exists^+_\sigma(Z,X)) \succ_{rhorpo} X$ which both hold by case 1.

For rule (5). The full neutralization of both sides is:
$s = \mathcal{FN}(app_{\sigma,\tau}(\exists^-_{\rho,\sigma\to\tau}(T,H),X)) =$
$app_{\sigma,\tau}(\exists^-_{new\ \rho,\sigma\to\tau}(T, @(H, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))), X)$, and
$t = \mathcal{FN}(\exists^-_{\rho,\tau}(T, \lambda x\!:\!o\, y\!:\!\rho. app_{\sigma,\tau}(@(H,x,y),X))) =$
$\exists^-_{new\ \rho,\tau}(T, app_{\sigma,\tau}(@(H, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T)), X))$.
For the proof of $s \succ_{rhorpo} t$ we apply case 2, with yields the subgoals
(i) $app_{\sigma,\tau}(\exists^-_{new\ \rho,\sigma\to\tau}(T, @(H, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))), X) \succ_{rhorpo} T$,
and (ii) $app_{\sigma,\tau}(\exists^-_{new\ \rho,\sigma\to\tau}(T, @(H, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))), X) \succ_{rhorpo}$ $app_{\sigma,\tau}(@(H, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T)), X)$. Subgoal (i) holds by case 1 twice. For (ii), we apply case 3, which requires
$\{\exists^-_{new\ \rho,\sigma\to\tau}(T, @(H, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))), X\}(\succ_{rhorpo})_{mul}$
$\{@(H, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T)), X\}$, and thus
$\exists^-_{new\ \rho,\sigma\to\tau}(T, @(H, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))) \succ_{rhorpo} @(H, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))$, which holds by case 1.

For rule (6). The full neutralization of both sides is:
$s = \mathcal{FN}(\Pi^0_{\sigma,\tau}(\exists^-_{\rho,c(\sigma,\tau)}(T,I))) =$
$\Pi^0_{\sigma,\tau}(\exists^-_{new\ \rho,c(\sigma,\tau)}(T, @(I, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))))$, and
$t = \mathcal{FN}(\exists^-_{\rho,\tau}(T, \lambda x\!:\!o\, y\!:\!\rho. \Pi^0_{\sigma,\tau}(@(I,x,y)))) =$
$\exists^-_{new\ \rho,\tau}(T, \Pi^0_{\sigma,\tau}(@(I, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))))$.
Proving $s \succ_{rhorpo} t$ follows the same pattern as in the rule (5). For rule (7) the proof is the same as for rule (6).

For rule (8). The full neutralization of both sides is:
$$\begin{aligned}
&\mathcal{FN}(\exists^-_{\sigma,\tau}(\exists^-_{\rho,c(o,\sigma)}(T,J),G)) = \\
&\exists^-_{new\ \sigma,\tau}(\ \exists^-_{new\ \rho,c(o,\sigma)}(T, @(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))), \\
&\qquad @(G, \bot_{c(o,\sigma)\to o}(\exists^-_{new\ \rho,c(o,\sigma)}(T, @(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T)))), \\
&\qquad\quad \bot_{c(o,\sigma)\to\sigma}(\exists^-_{new\ \rho,c(o,\sigma)}(T, @(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))))))
\end{aligned}$$
and, $\mathcal{FN}(\exists^-_{\rho,\tau}(T, \lambda x\!:\!o\, y\!:\!\rho. \exists^-_{\sigma,\tau}(@(J,x,y),G))) =$
$$\begin{aligned}
&\exists^-_{new\ \rho,\tau}(T, \exists^-_{new\ \sigma,\tau}(\ @(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T)), \\
&\qquad @(G, \bot_{c(o,\sigma)\to o}(@(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))), \\
&\qquad\quad \bot_{c(o,\sigma)\to\sigma}(@(J, \bot_{c(o,\rho)\to o}(T), \bot_{c(o,\rho)\to\rho}(T))))))
\end{aligned}$$
In this particular rule it can be seen that in the definition of full neutralization the arguments are first fully neutralized and moreover the arguments used for $i$-neutalization are also first fully neutralized.

For proving $s \succ_{rhorpo} t$ we first apply case 4 this has two subgoals:
(i) $\exists^{-}_{new\ \rho,c(o,\sigma)}(T, @(J, \perp_{c(o,\rho)\to o}(T), \perp_{c(o,\rho)\to\rho}(T))) \succ_{rhorpo} T$, and
(ii) $s \succ_{rhorpo} \exists^{-}_{new\ \sigma,\tau}( @(J, \perp_{c(o,\rho)\to o}(T), \perp_{c(o,\rho)\to\rho}(T)),$
$@(G, \perp_{c(o,\sigma)\to o}(@(J, \perp_{c(o,\rho)\to o}(T), \perp_{c(o,\rho)\to\rho}(T))),$
$\perp_{c(o,\sigma)\to\sigma}(@(J, \perp_{c(o,\rho)\to o}(T), \perp_{c(o,\rho)\to\rho}(T)))))$
For (i) we apply case 1 twice (note that all data types are equal in the type ordering $\geq_{\mathcal{T}_S}$).
For (ii) we apply case 4 again which leads us to two new subgoals:
(iii) $\exists^{-}_{new\ \rho,c(o,\sigma)}(T, @(J, \perp_{c(o,\rho)\to o}(T), \perp_{c(o,\rho)\to\rho}(T))) \succ_{rhorpo}$
$@(J, \perp_{c(o,\rho)\to o}(T), \perp_{c(o,\rho)\to\rho}(T))$, and
(iv) $@(G, \perp_{c(o,\sigma)\to o}(\exists^{-}_{new\ \rho,c(o,\sigma)}(T, @(J, \perp_{c(o,\rho)\to o}(T), \perp_{c(o,\rho)\to\rho}(T)))),$
$\perp_{c(o,\sigma)\to\sigma}(\exists^{-}_{new\ \rho,c(o,\sigma)}(T, @(J, \perp_{c(o,\rho)\to o}(T), \perp_{c(o,\rho)\to\rho}(T))))) \succ_{rhorpo}$
$@(G, \perp_{c(o,\sigma)\to o}(@(J, \perp_{c(o,\rho)\to o}(T), \perp_{c(o,\rho)\to\rho}(T))),$
$\perp_{c(o,\sigma)\to\sigma}(@(J, \perp_{c(o,\rho)\to o}(T), \perp_{c(o,\rho)\to\rho}(T))))$
For (iii) we apply case 1, and for (iv) we apply case 9, which leads us to prove:
$\perp_{c(o,\sigma)\to o}(\exists^{-}_{new\ \rho,c(o,\sigma)}(T, @(J, \perp_{c(o,\rho)\to o}(T), \perp_{c(o,\rho)\to\rho}(T)))) \succ_{rhorpo}$
$\perp_{c(o,\sigma)\to o}(@(J, \perp_{c(o,\rho)\to o}(T), \perp_{c(o,\rho)\to\rho}(T)))$.
which holds by case 3 and 1. □

*Example 5.* (taken from [**?**]) Let

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

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

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

We can now give the various ingredients for neutralization and carry out the comparisons:

## 7 Conclusion

Proving termination properties of Nipkow's rewriting was considered in [21] and [3].

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

The latter generalizes the notion of general schema as formulated in [2] where the notion of computability closure was introduced. However, what can be done with the schema can be done with the higher-order recursive path ordering when using the computability closure of $f(\bar{t})$ in the subterm case, instead of simply the set of subterms $\bar{t}$ itself. The general definition of the normal higher-order recursive path ordering with closure is given in [**?**]. It is however interesting to notice that the neutralization mecanism is powerful enough so as to dispense us with using the closure for all these complex examples taken from the literature that we have considered here. It remains to be seen whether the closure plays in the context of normal higher-order rewriting, a role as important as for proving termination of recursor rules for inductive types for which plain pattern matching is used instead of higher-order pattern matching.

We believe that there is ample room for generalization, and indeed the higher-order recursive path ordering itself has been already generalized, to associative-commutative terms [**?**], and to the calculus of constructions [22].

## References

1. F. Blanqui, J.-P. Jouannaud, and M. Okada. The Calculus of Algebraic Constructions. In Narendran and Rusinowitch, Proc. RTA'99, 1999.

2. F. Blanqui, J.-P. Jouannaud, and M. Okada. Inductive Data Types. *Theoretical Computer Science*, 277:. 2001.
3. F. Blanqui. Termination and Confluence of Higher-Order Rewriting Systems. In Proc. RTA'00, 2000.
4. Henk Barendregt. Functional Programming and Lambda Calculus. In [19], pages 321–364.
5. Henk Barendregt. *Handbook of Logic in Computer Science*, chapter Typed lambda calculi. Oxford Univ. Press, 1993. eds. Abramsky et al.
6. Nachum Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17(3):279–301, March 1982.
7. Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In [19], pages 321–364.
8. Jean-Pierre Jouannaud. Higher-Order rewriting: Framework, Confluence and termination. In Art Middeldorp, Vincent van Oostrom, Femke van Raamsdonk and Roel de Vrijer editors, *Processes, Terms and Cycles: Steps on the road to infinity*. Essays Dedicated to Jan Willem Klop on the occasion of his 60th Birthday. Springer Verlag, 2005.
9. Jean-Pierre Jouannaud and Albert Rubio. The higher-order recursive path ordering. In Giuseppe Longo, editor, *Fourteenth Annual IEEE Symposium on Logic in Computer Science*, Trento, Italy, July 1999. IEEE Comp. Soc. Press.
10. Jean-Pierre Jouannaud and Albert Rubio. Polymorphic Higher-Order Recursive Path Orderings. submitted to JACM. http://www.lix.polytechnique.fr/Labo/jouannaud.
11. Jean-Pierre Jouannaud and Albert Rubio. Higher-Order Recursive Path Orderings à la carte. 2004. http://www.lix.polytechnique.fr/Labo/jouannaud.
12. Jean-Pierre Jouannaud, Femke van Raamsdonk and Albert Rubio Higher-order rewriting with types and arities. 2005. see http://www.lix.polytechnique.fr/Labo/jouannaud.
13. Jan Wilhelm Klop. Combinatory Reduction Relations. Mathematical Centre Tracts 127. Mathematisch Centrum, Amsterdam, 1980.
14. Jan Wilhelm Klop. Term Rewriting Systems. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2:2–116. Oxford University Press, 1992.
15. Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(1):3–29, February 1998.
16. Dale Miller. Lambda PROLOG. environ 90.
17. Tobias Nipkow. Higher-order critical pairs. In *6th IEEE Symp. on Logic in Computer Science*, pages 342–349. IEEE Computer Society Press, 1991.
18. Lawrence C. Paulson. Isabelle: the next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*. Academic Press, 1990.
19. J. van Leeuwen, editor. Handbook of Theoretical Computer Science, volume B. North-Holland, 1990.
20. J. van de Pol. Strict functional for termination proofs. In *Typed Lambda Calculi and Applications, Edinburgh*. Springer-Verlag, 1995.
21. J. Van de Pol and H. Schwichtenberg. Strict functional for termination proofs. In *Typed Lambda Calculi and Applications, Edinburgh*. Springer-Verlag, 1995.
22. Daria Walukiewicz-Chrzaszcz. Termination of rewriting in the Calculus of Constructions. In *Proceedings of the Workshop on Logical Frameworks and Meta-languages, Santa Barbara, California*, 2000.