Higher-Order Recursive Path Orderings à la carte*

Jean-Pierre Jouannaud
LIX, École Polytechnique
91400 Palaiseau, FRANCE
Email: jouannaud@lix.polytechnique.fr
http://www.lix.polytechnique.fr/~jouannaud

Albert Rubio
Technical University of Catalonia
Pau Gargallo 5
08028 Barcelona, SPAIN
Email: rubio@lsi.upc.es

Abstract

This paper extends the termination proof techniques based on reduction orderings to a higher-order setting, by defining a family of recursive path orderings for terms of a typed lambda-calculus generated by a signature of polymorphic higher-order function symbols. These relations can be generated from two given well-founded orderings, on the function symbols and on the type constructors. The obtained orderings on terms are well-founded, include β-reductions, are monotonic for terms of equal type, and stable under substitution. They can be used to prove the strong normalization property of the various existing kinds of higher-order calculi in which constants can be defined by higher-order rewrite rules, regardless of whether these calculi use first-order or higher-order pattern matching. For example, the polymorphic version of Gödel's recursor for the natural numbers is easily oriented. And indeed, our ordering is polymorphic, in the sense that a single comparison allows to prove the termination property of all monomorphic instances of a polymorphic rewrite rule. Many non-trivial examples are given which examplify the expressive power of these orderings.

This paper is an extended and improved version of [22]. Polymorphic algebras have been made more expressive than in our previous framework. The notions of polymorphic higher-order ordering and polymorphic normalized higher-order ordering are new. The ordering itself has been made much more powerful by replacing the congruence on types used there by an ordering on types. This yields a very elegant presentation of the whole machinery by integrating both orderings into a single one operating on terms and types as well. This presentation should in turn be considered as the key missing stone on the way towards the definition of a recursive path ordering for dependent type calculi. Finally, the normalized higher-order recursive path ordering is new as well.

*This work was partly supported by the RNRT project CALIFE and the CICYT project HEMOSS, ref. TIC98-0949-C02-01.

1 Introduction

Rewrite rules are increasingly used in programming languages and logical systems, with two main goals: defining functions by pattern matching; describing rule-based decision procedures. ML, Alf [8] and Isabelle [32] examplify the first use. A future version of Coq [18] will examplify the second use [14]. In Isabelle, rules operate on terms in β-normal, η-expanded form. In ML and Alf, they operate on arbitrary terms. In a future version of Coq, both kinds should coexist.

The use of rules in logical systems is subjected to three main meta-theoretic properties : subject reduction, local confluence, and strong normalization. The first two are usually easy. The last one is difficult, requiring the use of sophisticated proof techniques based, for example, on Tait and Girard's computability predicate technique. Our ambition is to remedy to this situation by developping for the higher-order case the kind of semi-automated termination proof techniques that are available for the first-order case, of which the most popular one is the recursive path ordering [12].

Our contribution to this program is a reduction ordering for typed higher-order terms following a typing discipline including polymorphic sort constructors, which conservatively extends β-reductions for higher-order terms on the one hand, and on the other hand Dershowitz's recursive path ordering for first-order unisorted terms. In the latter, the precedence rule allows to decrease from the term $s = f(s_1, \ldots, s_n)$ to the term $g(t_1, \ldots, t_n)$, provided that (i) f is bigger than g in the given precedence on function symbols, and (ii) s is bigger than every t_i. For typing reasons, in our ordering the latter condition becomes: (ii) for every t_i, either s is bigger than t_i or some s_j is bigger than or equal to t_i. Indeed, we can instead allow t_i to be obtained from the subterms of s by computability preserving operations. Here, computability refers to Tait and Girard's strong normalization proof technique which we have used to show that our ordering is well-founded. And indeed the restriction of our proof to the first-order sublanguage yields a new, simple proof of well-foundedness of Dershowitz's recursive path ordering. This proof does not use Kruskal's tree theorem, and of course, does not show that the recursive path ordering is a well-ordering. It appears therefore that proving the property of well-foundedness of the recursive path ordering is quite easy -to a point that one wonders why this proof was not discovered before- while proving the slightly stronger propertry of well-orderedness becomes quite difficult.

In a preliminary version of this work presented at the Federated Logic Conference in Trento, our ordering could only compare terms of equal types (after identifying sorts such as Nat or List). In the present version, our ordering is capable of ordering terms of *decreasing types*, the ordering on types being simply a slightly weakened form of Dershowitz's recursive path ordering. Several other improvements have been made, which allow to prove a great variety of practical examples. To hint at the strength of our ordering, let us mention that the polymorphic version of Gödel's recursor for the natural numbers is easily oriented. And indeed, our ordering can prove at once the termination property of all monomorphic instances of a polymorphic rewrite rule. Many other examples are given which examplify the expressive power of the ordering.

In the litterature, one can find several attempts at designing methods for proving strong normalization of higher-order rewrite rules based on ordering comparisons. These orderings are either quite weak [26, 21], or need an important user interaction [10]. Besides, they operate on terms in η-long β-normal form, hence apply only to the higher-order rewriting "à la Nipkow" [27], based on higher-order pattern matching modulo $\beta\eta$. To our knowledge, our ordering is the first to operate on arbitrary higher-order terms, therefore applying to the other kind of rewriting, based on plain pattern matching. And indeed we want to stress several important features of our approach. Firstly, it can be seen as a way to lift an ordinal notation operating on a first-order language (here, the set of labelled trees ordered by the recursive path ordering) to an ordinal notation of higher type operating on a set of well-typed λ-expressions built over

the first-order language. Secondly, the analysis of our ordering, based on Tait and Girard's computability predicate proof technique, leads to hiding this technique away, by allowing one to carry out future meta-theoretical investigations based on ordering comparisons rather than by a direct use of the computability predicate technique. Thirdly, a very elegant presentation of the whole ordering machinery obtained by integrating both orderings on terms and types into a single one operating on both kinds shows that this presentation can in turn be the basis for generalizing the ordering to dependent type calculi. Last but not least, a very simple modification of the ordering can be used to prove strong normalization of higher-order rewrite rules operating on terms in η-long β-normal form, and this is indeed true of any higher-order rewrite ordering containing β-reductions.

The framework we use is described in Section 2. This framework includes several novel aspects, among which two important notions of polymorphic higher-order rewriting and of polymorphic higher-order rewrite orderings (both allowing on demand for higher-order pattern matching). The basic version of our ordering is defined and studied in Section 3, where several examples are also given. The notion of computable closure used to boost the expressivity of the ordering is introduced and studied in Section 4. Section 6 introduces another enhencement, whose role is to improve the treatment of bound variables. The latter ordering is then adapted to rewriting on terms in η-long β-normal form in Section 5. We discuss several other improvements of our method in Section 7. Finally, further potential extensions of our work as well as some related work are discussed in Section 8. All examples which can be found in [30, 27, 10] are solved with our method but one which resisted all attacks.

The reader is expected to be familiar with the basics of term rewriting systems [13, 24] and typed lambda calculi [1, 2].

2 Polymorphic Higher-Order Algebras

This section describes our framework of polymorphic higher-order algebras, together with higher-order rewriting and rewrite orderings. Polymorphic higher-order algebras enjoy the property that typable terms have a unique type in a given environment, which eases the following developments. Several definitions of (polymorphic) higher-order rewriting are considered, and the associated (polymorphic) rewrite orderings are then introduced. Types will therefore play a central role in this paper, but the reader should be aware that the paper is by no means about polymorphic typing : it is about polymorphic higher-order orderings.

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*:

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

We denote by $\mathcal{V}ar(\sigma)$ the set of type variables of the type $\sigma \in \mathcal{T}_{\mathcal{S}^\forall}$, and by $\mathcal{T}_{\mathcal{S}}$ the set of *monomorphic* or *ground* types, whose set of type variables is empty.

Types are *functional* when they are headed by the $\to$ symbol, and *data types* when they are headed by a sort symbol. As usual, $\to$ associates to the right. We will often explicit the functional structure of an arbitrary type τ by writing it in the *canonical form* $\sigma_1 \to \ldots \to \sigma_n \to \sigma$, with $n \geq 0$, where σ is a data type or a type variable called *canonical output type* of τ denoted by $\vec{\tau}^0$, and n is the *functional level* (or *arity*) of τ denoted by $ar(\tau)$.

Following [3], we could have also considered inductive types in our framework, without having to face new difficulties. We choosed not to do so in the present framework, which, we think, is already quite powerful and complex. We will come back on this question in Section 7.

A *type substitution* is a mapping from $\mathcal{S}^\forall$ to $\mathcal{T}_{\mathcal{S}^\forall}$ extended to an endomorphism of $\mathcal{T}_{\mathcal{S}^\forall}$. *Type variable renamings* are bijective type substitutions. We write $\sigma\xi$ for the application of the type substitution ξ to the type σ. 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*, and by $\simeq$ the equivalence of types under type renaming. By a renaming of the type σ apart from $V \subset \mathcal{X}$, we mean a type $\sigma\xi$ where ξ is a type renaming such that $\mathcal{D}om(\xi) = \mathcal{R}an(\sigma)$ and $\mathcal{R}an(\xi) \cap \mathcal{V} = \emptyset$.

In the following, we use α, β for type variables, $\sigma, \tau, \rho, \theta$ for arbitrary types, and ξ, ζ 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 equipped 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 *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 $f : \sigma_1 \times \ldots \times \sigma_n \to \sigma$. In case $n = 0$, the declaration becomes $f :\to \sigma$ or simply $f : \sigma$ when σ is not a functional type. Type declarations are not types, although they are used for typing purposes. Note however that $\sigma_1 \to \ldots \to \sigma_n \to \sigma$ 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*. The intended meaning of a polymorphic type declaration is the set of all its type instantiations, that is, polymorphic type declarations are implicitely universally quantified, or, equivalently, can be renamed by an arbitrary type renaming.

$\mathcal{F}$ is called a *first-order signature* if all its type declarations are first-order, and a higher-order signature otherwise. It is a *polymorphic signature* if some type declaration is polymorphic, and a monomorphic signature otherwise. Type instantiation does not change the arity of a function symbol. Polymorphic signatures capture infinitely many monomorphic (and polymorphic) ones via type instantiation.

A function symbol having several type declarations in a given signature is said to be *overloaded*. Overloading is quite useful, but it complicates typing unless the following property is satisfied :

Definition 2.1 *A (possibly polymorphic) signature $\mathcal{F}$ is* regular *if for any two type declarations $f : \sigma_1 \to \ldots \to \sigma_n \to \sigma$ and $f : \sigma'_1 \to \ldots \to \sigma'_n \to \sigma'$ where $\bigcup_i \mathcal{V}ar(\sigma_i) \cap \bigcup_i \mathcal{V}ar(\sigma'_i) = \emptyset$, the equational problem $(\bigwedge_{i=1}^{i=n} \sigma_i = \sigma'_i) \wedge \sigma \neq \sigma'$ has no solution.*

Regular signatures are important because they provide deterministic typing, as we will see in Section 2.4. A non-overloaded signature is of course regular. Note that regularity is decidable, since the existence of a ground solution (for the type variables) to the above formula is decidable [6].

It is worth noting that regularity implies the condition that in a given type declaration, all type variables occuring in an output type already occur in one of the input types. As a consequence, polymorphic constants are not allowed. Although regularity should be relaxed for constants in order to ease specifications, as we will see later in several examples, we will assume it throughout the paper, unless explicitely mentionned otherwise.

2.3 Terms

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

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

Terms of the form $\lambda x : \sigma.u$ are called *abstractions*, while the other terms are said to be *neutral*. $@(u, v)$ denotes the application of u to v. We may sometimes omit the type σ 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 (which do not have arity in the present framework). As a matter of convenience, we may write $u(v_1, \ldots, v_n)$, or $@(u, v_1, \ldots, v_n)$ for $u(v_1) \ldots (v_n)$, assuming $n \geq 1$. The term $@(u, \overline{v})$ is called a (partial) *left-flattening* of $s = u(v_1) \ldots (v_n)$, u being possibly an application itself (hence the word "partial").

Terms are identified with finite labeled trees by considering $\lambda x : \sigma._-$, for each variable x and type σ, as a unary function symbol taking a term u as argument to construct the term $\lambda x : \sigma.u$. We denote the set of free variables of the term t by $\mathcal{V}ar(t)$, its set of bound variables by $\mathcal{BV}ar(t)$, its size (the number of symbols occurring in t) by $|t|$.

Positions are strings of positive integers. $\wedge$ and $\cdot$ denote respectively the empty string (root position) and the concatenation of strings. We use $\mathcal{P}os(t)$ for the set of positions in t. The *subterm* of t at position p is denoted by $t|_p$, and we write $t \trianglerighteq t|_p$ for the subterm relationship. The result of replacing $t|_p$ at position p in t by u is denoted by $t[u]_p$. We sometimes use $t[x : \sigma]_p$ for a term with a (unique) hole of type σ at position p, also called a context.

The notation $\overline{s}$ will be ambiguously used to denote a list, or a multiset, or a set of terms $s_1, \ldots, s_n$.

2.4 Typing Rules

Typing rules restrict the set of terms by constraining them to follow a precise discipline.

Definition 2.2 *An* environment Γ *is a finite set of pairs written as $\{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$, where x_i is a variable, σ_i is a type, and $x_i \neq x_j$ for $i \neq j$. $\mathcal{V}ar(\Gamma) = \{x_1, \ldots, x_n\}$ is the set of variables of Γ. The size $|\Gamma|$ of the environment Γ is the sum of the sizes of its constituants. Given two environments Γ and Γ', their* composition *is the environment $\Gamma \cdot \Gamma' = \Gamma' \cup \{x : \sigma \in \Gamma \mid x \notin \mathcal{V}ar(\Gamma')\}$. Two environments Γ and Γ' are* compatible *if $\Gamma \cdot \Gamma' = \Gamma \cup \Gamma'$.*

Our typing judgements are written as $\Gamma \vdash_{\mathcal{F}} s : \sigma$, and read "$s$ has type σ in the environment Γ". They are displayed at Figure 1.

A term s has type σ in the environment Γ if the judgement $\Gamma \vdash_{\mathcal{F}} s : \sigma$ is provable in our inference system. Given an environment Γ, a term s is *typable* if there exists a type σ such that $\Gamma \vdash_{\mathcal{F}} s : \sigma$.

An example of signature together with typable terms is given at example 2.

We now prove the main properties of our type system which are instrumental, in our view, to develop a theory of higher-order rewriting.

Lemma 2.3 *Assume $\mathcal{F}$ is regular. Then, the problem, given an environment Γ and a term s, wether there exists a type σ such that $\Gamma \vdash_{\mathcal{F}} s : \sigma$ is decidable in linear time in the sum of the sizes of Γ and s. Moreover, σ is unique whenever it exists.*

Proof: We prove unicity and linear time complexity together by induction on the size of (Γ, s).

If s is a variable, both properties follow from set membership.

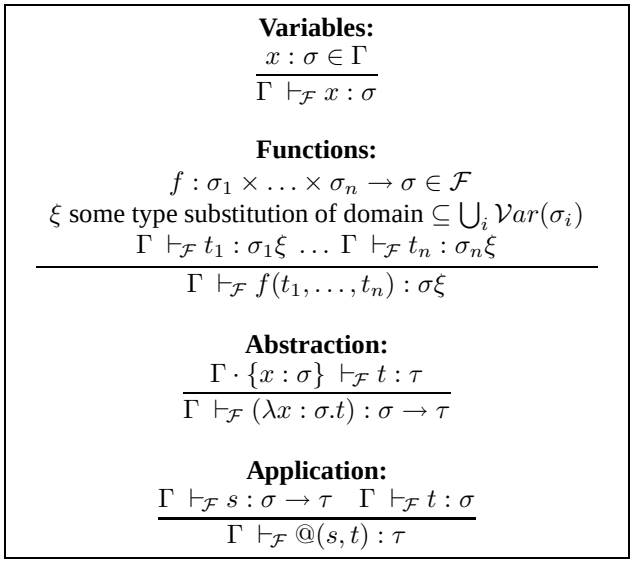

Figure 1. The type system for polymorphic higher-order algebras

If $s = @(u, v)$, we apply the induction hypothesis to both (Γ, u) and (Γ, v) and check the inference step in constant time, yielding unicity and linear time complexity.

If $s = \lambda x : \tau.u$, we reason by induction on $(\Gamma \cdot \{x : \tau\}, u)$, since $|\Gamma| + |\lambda x : \tau.u| > |\Gamma \cdot \{x : \tau\}| + |u|$. Both properties follow directly.

If $s = f(s_1, \ldots, s_n)$, we first apply the induction hypothesis to $(\Gamma, s_1), \ldots, (\Gamma, s_n)$. By regularity assumption, for each declaration $f : \sigma_1 \times \ldots \times \sigma_n \to \sigma \in \mathcal{F}$, if any, such that the types obtained for $s_1, \ldots, s_n$ match the types $\sigma_1, \ldots, \sigma_n$, the output type obtained for s in case of success is the same. The inference step is again done in constant time, therefore proving our second claim. □

The existence of a unique type for a term in a given environment will be important in the sequel. Polymorphic constants violate our regularity assumption, but unique typing is nevertheless maintained for them by our typing rules (the type variables occuring in the type declaration of a polymorphic constant a cannot be renamed, though, unless a occurs as a subterm in a bigger term). This will be used in section 6, where possibly polymorphic constants are introduced to replace some bound variables. And indeed, we could have given a more powerful type system in two different ways in order to cope with function declarations for which the output type has extra variables. In both cases, the rule **Functions** is modified. In the first case, by allowing for α-conversion of these variables, yielding type uniqueness up to α-conversion. In the second case, by replacing pattern matching (for computing ξ) by unification. Type uniqueness would be lost, but this type system would instead enjoy the principal typing property, which would be enough for our purpose.

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 $f(\overline{u})\xi = f(\overline{u\xi})$. A whole judgement can indeed be instantiated:

Lemma 2.4 *Assume that the signature $\mathcal{F}$ is regular. Then, $\Gamma \vdash_{\mathcal{F}} s : \sigma$ implies $\Gamma\xi \vdash_{\mathcal{F}} s\xi : \sigma\xi$ for any ξ.*

Proof: By induction on the type derivation of s and by case upon the last rule applied.

If s is a variable or an application, the result is clear. If $s = \lambda x : \tau.u$, then, the last rule applied in the typing derivation is **Abstraction**, and therefore $\sigma = \tau \to \rho$ and $\Gamma \cdot \{x : \tau\} \vdash_{\mathcal{F}} u : \rho$. By induction hypothesis, $\Gamma\xi \cdot \{x : \tau\xi\} \vdash_{\mathcal{F}} u\xi : \rho\xi$, and by using **Abstaction**, $\Gamma\xi \vdash_{\mathcal{F}} \lambda x : \tau\xi.u\xi : \tau\xi \to \rho\xi$, that

is, $\Gamma\xi \vdash_{\mathcal{F}} s\xi : \sigma\xi$. If $s = f(s_1, \ldots, s_n)$ with $f : \sigma_1 \times \ldots \times \sigma_n \to \sigma$, then, the last rule applied in the typing derivation is **Functions**, and therefore $\sigma = \tau\xi$ for some type substitution ξ and $\Gamma \vdash_{\mathcal{F}} s_i : \sigma_i\xi$ for $i \in [1..n]$. By induction hypothesis, $\Gamma\zeta \vdash_{\mathcal{F}} s_i : (\sigma_i\xi)\zeta = \sigma_i(\xi\zeta)$, hence $\Gamma\zeta \vdash_{\mathcal{F}} f(s_1, \ldots, s_n) : \tau(\xi\zeta) = (\tau\xi)\zeta = \sigma\zeta$, and we are done. □

Lemma 2.5 *Assume given a regular signature $\mathcal{F}$, and environment Γ, a term s and a type σ such that $\Gamma \vdash_{\mathcal{F}} s : \sigma$ holds. Then, $\Gamma \cdot \Gamma' \vdash_{\mathcal{F}} s : \sigma$ for all Γ' compatible with Γ.*

Proof: By induction on the proof of the judgement $\Gamma \vdash_{\mathcal{F}} s : \sigma$, and by case upon the last rule applied.

There is subtlety with the case **Abstraction**, for which there is a type τ such that $\Gamma \cdot \{x : \sigma\} \vdash_{\mathcal{F}} u : \tau$. We then apply our induction hypothesis to the judgement $\Gamma \cdot \{x : \sigma\} \vdash_{\mathcal{F}} u : \tau$ and the environment $\Gamma' \cdot \{x : \sigma\}$ which is easily shown compatible with $\Gamma \cdot \{x : \sigma\}$, since so are Γ and Γ', yielding $(\Gamma \cdot \{x : \sigma\}) \cdot (\Gamma' \cdot \{x : \sigma\}) \vdash_{\mathcal{F}} u : \tau$. But $(\Gamma \cdot \{x : \sigma\}) \cdot (\Gamma' \cdot \{x : \sigma\}) = (\Gamma \cdot \Gamma') \cdot \{x : \sigma\}$, and therefore we can apply **Abstraction** to get the result.

Other cases are straightforward. □

Lemma 2.6 *Assume given a regular signature $\mathcal{F}$, an environment Γ, a term s and a type σ such that $\Gamma \vdash_{\mathcal{F}} s : \sigma$ holds. Then, for all $p \in \mathcal{D}om(s)$, there exists a canonical environment $\Gamma_{s|_p}$ and a type τ such that $\Gamma_{s|_p} \vdash_{\mathcal{F}} s|_p : \tau$ is a subproof of the proof of the judgement $\Gamma \vdash_{\mathcal{F}} s : \sigma$. Moreover, $\Gamma_{s|_{(p \cdot q)}} = (\Gamma_{s|_p})_{(s|_p)|_q}$.*

Proof: By induction on the length of p. If $p = \Lambda$, the result is trivial, with $s|_\Lambda = s$, $\Gamma_{s|_\Lambda} = \Gamma$ and $\tau = \sigma$. Otherwise, we discuss by case according to the top function symbol of s.

If $s = @(s_1, s_2)$ and $p = i \cdot q$ (with $i = 1, 2$), or $s = f(s_1, \ldots, s_n)$ and $p = i \cdot q$ (with $i \in [1..n]$), then, by **Application** or **Functions**, there is a type σ_i such that $\Gamma \vdash_{\mathcal{F}} s|_i : \sigma_i$. We apply the induction hypothesis to the judgement $\Gamma \vdash_{\mathcal{F}} s|_i : \sigma_i$ and the subterm $(s|_i)|_q = s|_{(i \cdot q)}$ of $s|_i$ at position q, yelding a canonical environment $\Gamma_{s|_{(i \cdot q)}}$ such that the proof of the judgement $\Gamma_{s|_{(i \cdot q)}} \vdash_{\mathcal{F}} s|_{(i \cdot q)} : \tau$ for some τ is a subproof of the proof of the judgement $\Gamma \vdash_{\mathcal{F}} s|_i : \sigma_i$, hence of $\Gamma \vdash_{\mathcal{F}} s : \sigma$, showing this case.

If $s = \lambda x : \rho.u$ and $p = 1 \cdot q$, by the **Abstraction** rule, $\Gamma \cdot \{x : \rho\} \vdash_{\mathcal{F}} u : \theta$ for some θ, a judgement whose proof is a subproof of the proof of $\Gamma \vdash_{\mathcal{F}} s : \sigma$. Now, $s|_p = u|_q$, and by induction hypothesis, there exists a canonical environment $(\Gamma \cdot \{x : \rho\})_{u|_q}$ such that $(\Gamma \cdot \{x : \rho\})_{u|_q} \vdash_{\mathcal{F}} u|_q : \tau$ is a subproof of the proof of the judgement $\Gamma \cdot \{x : \rho\} \vdash_{\mathcal{F}} u : \theta$, hence of the proof of the judgement $\Gamma \vdash_{\mathcal{F}} s : \sigma$. We take $\Gamma_{s|_p} = (\Gamma \cdot \{x : \tau\})_{u|_q}$ and we are done.

Finaly, the property $\Gamma_{s|_{(p \cdot q)}} = (\Gamma_{s|_p})_{(s|_p)|_q}$ follows directly from our construction. □

Lemma 2.7 *Assume given a regular signature $\mathcal{F}$, an environment Γ, two terms s and v, two types σ and τ, 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, $\Gamma \vdash_{\mathcal{F}} s[v]_p : \sigma$.*

Proof: By induction on the length of p. The basic case is trivial, with $\Gamma_{s|_\Lambda} = \Gamma$, $\tau = \sigma$, and $s[v]_\Lambda = v$.

If $s = @(s_1, s_2)$ and $p = i \cdot q$ (with $i = 1, 2$), or $s = f(s_1, \ldots, s_n)$ and $p = i \cdot q$ (with $i \in [1..n]$), then, by **Application** or **Functions**, there is a type σ_i such that $\Gamma_{s|_i} = \Gamma \vdash_{\mathcal{F}} s|_i : \sigma_i$. Noting that, by lemma 2.6, $\Gamma_{s|_p} = (\Gamma_{s|_i})_{(s|_i)|_q} \vdash_{\mathcal{F}} s|_p = (s|_i)|_q : \tau$ and $\Gamma_{s|_p} = (\Gamma_{s|_i})_{(s|_i)|_q} \vdash_{\mathcal{F}} v : \tau$, we can apply the induction hypothesis to the judgement $\Gamma \vdash_{\mathcal{F}} s|_i : \sigma_i$ and the position q of $s|_i$. The result then follows by using the rule **Application** or **Functions**.

If $s = \lambda x : \rho.u$ and $p = 1 \cdot q$, by the **Abstraction** rule, $\Gamma_{s|_1} = \Gamma \cdot \{x : \rho\} \vdash_{\mathcal{F}} u : \theta$ for some θ. By lemma 2.6, $\Gamma_{s|_p} = (\Gamma \cdot \{x : \rho\})_{u|_q}$, and therefore $(\Gamma \cdot \{x : \rho\})_{u|_q} \vdash_{\mathcal{F}} v : \tau$. By induction hypothesis, $\Gamma \cdot \{x : \rho\} \vdash_{\mathcal{F}} u[v]_q : \theta$, and the result follows by applying **Abstraction**. □

We now introduce substitutions:

Definition 2.8 *A* term substitution*, or simply* substitution *is a finite set* $\gamma = \{(x_1 : \sigma_1) \mapsto (\Gamma_1, t_1), \ldots, (x_n : \sigma_n) \mapsto (\Gamma_n, t_n)\}$*, whose elements are quadruples made of a variable symbol, a type, an environment and a term, such that*

(i) $\forall i \in [1..n]$, $t_i \neq x_i$ *and* $\Gamma_i \vdash_{\mathcal{F}} t_i : \sigma_i$,

(ii) $\forall i \neq j \in [1..n]$, $x_i \neq x_j$, *and*

(iii) $\forall i \neq j \in [1..n]$, Γ_i *and* Γ_j *are compatible environments.*

We sometimes omit the parentheses, the type σ_i *and the environment* Γ_i *in* $(x_i : \sigma_i) \mapsto (\Gamma_i, t_i)$.

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

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

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

Lemma 2.9 *Assume given a substitution* $\gamma = \{(x_1 : \sigma_1) \mapsto (\Gamma_1, t_1), \ldots, (x_n : \sigma_n) \mapsto (\Gamma_n, t_n)\}$*. Then,* $\mathcal{R}an(\gamma) \vdash_{\mathcal{F}} t_i : \sigma_i$.

Proof: By condition (iii) of Definition 2.8, $\Gamma_i \subseteq \mathcal{R}an(\gamma) = \bigcup_{i \in [1..n]} \Gamma_i$, which implies by Lemma 2.5 that $\mathcal{R}an(\gamma) \vdash_{\mathcal{F}} t_i : \sigma_i$.

Definition 2.10 *A substitution* γ *is said to be* compatible *with an environment* Γ *if*

(i) $\mathcal{D}om(\gamma)$ *is compatible with* Γ,

(ii) $\mathcal{R}an(\gamma)$ *is compatible with* $\Gamma \setminus \mathcal{D}om(\gamma)$.

We will also say that γ *is compatible with the judgement* $\Gamma \vdash_{\mathcal{F}} s : \sigma$.

Definition 2.11 *A substitution* γ *compatible with a judgement* $\Gamma \vdash_{\mathcal{F}} s : \sigma$ *operates as an endomorphism on* s *(keeping its bound variables unchanged) and yields a term* $s\gamma$ *defined as follows:*

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*	$s\gamma = \lambda x : \tau.u\gamma_{\setminus \{x\}}$

Lemma 2.12 *Assume given a regular signature* $\mathcal{F}$ *and a substitution* γ *compatible with the judgement* $\Gamma \vdash_{\mathcal{F}} s : \sigma$*. Then,* $\Gamma \cdot \mathcal{R}an(\gamma) \vdash_{\mathcal{F}} s\gamma : \sigma$.

Proof: By induction on the derivation of the judgement $\Gamma \vdash_{\mathcal{F}} s : \sigma$.

Assume that $s = x \notin \mathcal{V}ar(\gamma)$. By Definition 2.11, it follows that $x\gamma = x$. Therefore, $\Gamma \cdot \mathcal{R}an(\gamma) \vdash_{\mathcal{F}} x : \sigma$ by compatibility assumption of $\mathcal{R}an(\gamma)$ with $\Gamma \subseteq \mathcal{D}om(\gamma)$ and Lemma 2.5.

Assume that $s = x \in \mathcal{V}ar(\gamma)$. Then, by compatibility assumption of $\mathcal{D}om(\gamma)$ with Γ, there exist some Γ' and t such that $(x : \sigma) \mapsto (\Gamma', t) \in \gamma$. Then, $s\gamma = t$ by Definition 2.11, $\mathcal{R}an(\gamma) \vdash_{\mathcal{F}} t : \sigma$ by Lemma 2.9, and $\Gamma \cdot \mathcal{R}an(\gamma) \vdash_{\mathcal{F}} t : \sigma$ by Lemma 2.5.

Assume that $s = @(u, v)$. Then, $s\gamma = @(u\gamma, v\gamma)$ by Definition 2.11, and $\Gamma \vdash_{\mathcal{F}} u : \tau \to \sigma$ and $\Gamma \vdash_{\mathcal{F}} v : \tau$ for some type τ by the rule **Application**. By induction hypothesis, $\Gamma \vdash_{\mathcal{F}} u\gamma : \tau \to \sigma$ and $\Gamma \vdash_{\mathcal{F}} v\gamma : \tau$, and therefore, $\Gamma \vdash_{\mathcal{F}} s\gamma : \sigma$ by the rule **Application**.

The case $s = f(s_1, \ldots, s_n)$ is similar.

Assume finally that $s = \lambda x : \tau.u$. By the rule **Abstraction**, $\Gamma \cdot \{x : \tau\} \vdash_{\mathcal{F}} u : \tau \to \sigma$. Since γ compatible with the environment Γ, $\mathcal{D}om(\gamma)$ and Γ are compatible, and therefore, $\mathcal{D}om(\gamma_{\setminus \{x\}})$ and

$\Gamma \cdot \{x : \tau\}$ are compatible. Similarly, $\mathcal{R}an(\gamma)$ and $\Gamma \setminus \mathcal{D}om(\gamma)$ are compatible, and therefore $\mathcal{R}an(\gamma_{\setminus\{x\}})$ and $\Gamma \cdot \{x : \tau\} \setminus \mathcal{D}om(\gamma_{\setminus\{x\}})$ are compatible, showing that the substitution $\gamma_{\setminus\{x\}}$ is compatible with the environment $\Gamma \cdot \{x : \tau\}$. Therefore, by induction hypothesis, $\Gamma \cdot \{x : \tau\} \vdash_{\mathcal{F}} u\gamma_{\setminus\{x\}} : \tau \to \sigma$, and by the rule **Abstraction**, it follows that $\Gamma \vdash_{\mathcal{F}} \lambda x.\tau : (u\gamma_{\setminus\{x\}}) : \tau \to \sigma$. We conclude with Definition 2.11. □

When writing $s\gamma$, we will always make the assumption that the domain of γ is compatible with the judgement $\Gamma \vdash_{\mathcal{F}} s : \sigma$. We will use the letter γ for substitutions and postfix notation for their application. We will sometimes use the notation $A\gamma$ where A is a set of terms, and γ a substitution, for the set of the instantiated terms of A.

2.5 Plain higher-order rewriting [20]

We now come to the definition of higher-order rewriting. Among the three possible variations by Klop [24], Jouannaud and Okada [20], and Nipkow [30], we consider here the last two which differ significantly, while the first one can be easily encoded via the other two. Based on using plain pattern matching for firing rules, plain higher-order rewriting is the first of these two.

Definition 2.13 *Given a regular signature $\mathcal{F}$, a* rewrite rule *is a quadruple written $\Gamma \vdash l \to r : \sigma$, where l and r are higher-order terms such that*
(i) $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$,
(ii) $\Gamma \vdash_{\mathcal{F}} l : \sigma$ and $\Gamma \vdash_{\mathcal{F}} r : \sigma$.
The rewrite rule is said to be polymorphic *if σ is a polymorphic type. A* plain term rewriting system*, or simply* term rewriting system *is a set of rewrite rules.*

We will often omit the type σ or even the environment Γ in the rule $\Gamma \vdash_{\mathcal{F}} l \to r : \sigma$.

Examples of polymorphic rules are given later in this section. Here is an example of a triple which is not a rewrite rule because its lefthand and righthand sides do not have the same type: let $\mathcal{F} = \{f : \alpha \to \alpha, g : \beta \to \beta, 0 : \mathbb{N}\}$, $\mathcal{T}_{\mathcal{S}^\forall} = \{\alpha, \beta\}$, and consider the triple $\{x : \alpha\} \vdash f(x) \to g(0)$. We have $\{x : \alpha\} \vdash_{\mathcal{F}} f(x) : \alpha$, $\{x : \alpha\} \vdash_{\mathcal{F}} g(0) : \mathbb{N}$, and $\alpha \neq \mathbb{N}$. On the other hand, the type instance $\{x : \mathbb{N}\} \vdash f(x) \to g(0) : \mathbb{N}$ is a rule, as well as the term instance $\{x : \alpha\} \vdash f(0) \to g(0) : \mathbb{N}$.

The following property follows easily from the definition, Lemma 2.4 and Lemma 2.12:

Lemma 2.14 *Assume that $\Gamma \vdash l \to r : \sigma$ is a rewrite rule. Then, for every type substitution ξ and every term substitution γ compatible with $\Gamma\xi$, the quadruple $\Gamma\xi \cdot \mathcal{R}an(\gamma) \vdash l\xi\gamma \to r\xi\gamma : \sigma\xi$ is a rewrite rule.*

We are now ready for defining the rewrite relation :

Definition 2.15 *Given a plain higher-order rewriting system R and an environment Γ, a term s such that $\Gamma \vdash_{\mathcal{F}} s : \sigma$ rewrites to a term t at position p with the rule $\Gamma_i \vdash l_i \to r_i : \sigma_i$, the type substitution ξ and the term substitution γ, written $\Gamma \vdash s \xrightarrow[\Gamma_i \vdash l_i \to r_i]{p} t$, or simply $\Gamma \vdash s \to_R t$, or even $s \to_R t$ assuming the environment Γ, if the following conditions are satisfied :*
(i) $\mathcal{D}om(\gamma) \subseteq \Gamma_i\xi$,
(ii) $\Gamma_i\xi \cdot \mathcal{R}an(\gamma) \subseteq \Gamma_{s|_p}$,
(iii) $s|_p = l_i\xi\gamma$,
(iv) $t = s[r_i\xi\gamma]_p$.

These conditions mean that the rule used for rewriting s at position p is the instance $\Gamma_i\xi \vdash l_i\xi \to r_i\xi : \sigma_i\xi$ of the polymorphic rule $\Gamma_i \vdash l_i \to r_i : \sigma_i$ by the type substitution ξ, which maps the types of the variables in l_i with the type of the terms by which these variables are replaced. Note that condition (iii)

implies that variables which are bound in the rule do not occur free in γ, therefore avoiding capturing variables when rewriting.

Example 1 Let $\mathcal{S} = \{o_1,\ o_2,\ o_3,\ o_4\}$, $\mathcal{S}^\forall = \{\alpha : *, \beta : *\}$, and $\mathcal{F} = \{f : \alpha \times \beta \to \alpha, g : \alpha \times \beta \to \beta\}$.
Let $\Gamma = \{x_1 : o_1,\ x_2 : o_2,\ x_3 : o_3,\ x_4 : o_4\}$, and $s = g(f(x_1, x_2) + f(x_3, x_4))$. We have $\Gamma \vdash_\mathcal{F} s : o_3$.
Let $\gamma = \{x_1 : o_1 \mapsto (\{x_1 : o_2,\ x_6 : o_1\}, g(x_1, x_6)), x_3 : o_3 \mapsto (\{x_2 : o_2,\ x_5 : o_3\}, g(x_2, x_5)), x_6 : o_2 \mapsto (\{x_1 : o_2,\ x_5 : o_3\}, f(x_1, x_5))\}$.
$\mathcal{D}om(\gamma) = \{x_1 : o_1,\ x_3 : o_3,\ x_6 : o_2\}$, and $\mathcal{R}an(\gamma) = \{x_1 : o_2, x_2 : o_2, x_5 : o_3,\ x_6 : o_1\}$.
$s\gamma = g(f(g(x_1, x_6), x_2), f(g(x_2, x_5), x_4))$.
$\Gamma \cdot \mathcal{R}an(\gamma) = \{x_1 : o_2,\ x_2 : o_2,\ x_3 : o_3,\ x_4 : o_4,\ x_5 : o_3,\ x_6 : o_1\}$.
$\Gamma \cdot \mathcal{R}an(\gamma) \vdash_\mathcal{F} s\gamma : o_3$.

Typechecking the rewritten term is not necessary, thanks to the so-called *Subject reduction* property:

Lemma 2.16 *Assume that* $\Gamma \vdash_\mathcal{F} s : \sigma$ *and* $\Gamma \vdash s \to_R t$. *Then* $\Gamma \vdash_\mathcal{F} t : \sigma$.

Proof: By Lemma 2.4, $\Gamma_i\xi \vdash_\mathcal{F} l_i\xi : \sigma_i\xi$. By conditions (i) and (ii) in Definition 2.15, the substitution γ is compatible with the environment $\Gamma_i\xi$, and therefore, by lemma 2.12, $\Gamma_i\xi \cdot \mathcal{R}an(\gamma) \vdash_\mathcal{F} l_i\xi\gamma : \sigma_i\xi$. By condition (ii) and lemma 2.5, $\Gamma_{s|_p} \vdash_\mathcal{F} l_i\xi\gamma : \sigma_i\xi$, and therefore, by condition (iii), $\Gamma_{s|_p} \vdash_\mathcal{F} s|_p : \sigma_i\xi$. Note that this tells us how to compute ξ in practice. Similarly, $\Gamma_{s|_p} \vdash_\mathcal{F} r_i\xi\gamma : \sigma_i\xi$. By lemma 2.7), we deduce that $\Gamma \vdash_\mathcal{F} s[r_i\xi\gamma]_p : \sigma_i\xi$. Using now condition (iv), we finally conclude that $\Gamma \vdash_\mathcal{F} t : \sigma$. □

Plain rewriting uses plain pattern matching: given s, σ, p and l_i, r_i, σ_i, the property $\Gamma_{s|_p} \vdash_\mathcal{F} s|_p : \sigma_i\xi$ allows to compute the type substitution ξ in linear time. Now, computing the substitution γ, if any, such that $l|_p = l_i\xi\gamma$ can be done in linear time as well. Overall, plain pattern matching in our formalism is linear. Note that it follows from condition (ii) that $\mathcal{R}an(\gamma) \subseteq \Gamma_{s|_p}$.

Example 2 We give here the specification for Gödel's system T. Let $\mathcal{S} = \{\mathbb{N}\}$, $\mathcal{S}^\forall = \{\alpha\}$, $\mathcal{F} = \{0 :\to \mathbb{N},\ s : \mathbb{N} \to \mathbb{N},\ + : \mathbb{N} \times \mathbb{N} \to \mathbb{N},\ rec : \mathbb{N} \times \alpha \times (\mathbb{N} \to \alpha \to \alpha) \to \alpha\}$. Gödel's recursor for natural numbers is defined by the following rewrite rules:

$$\begin{array}{rcrcl} \{U : \alpha,\ X : \mathbb{N} \to \alpha \to \alpha\} & \vdash & rec(0, U, X) & \to & U \\ \{x : \mathbb{N},\ U : \alpha,\ X : \mathbb{N} \to \alpha \to \alpha\} & \vdash & rec(s(x), U, X) & \to & @(X, x, rec(x, U, X)) \end{array}$$

Let $s = rec(S(0), 0, rec(0, \lambda\, x : \mathbb{N}\ y : \mathbb{N}.\ +(x, y), \lambda\, x : \mathbb{N}\ y : \mathbb{N} \to \mathbb{N} \to \mathbb{N}\ z : \mathbb{N}.y(+(x, z))))$, which typechecks by instantiating the type declaration of rec with the type substitutions $\{\alpha \mapsto \mathbb{N}\}$ and $\{\alpha \mapsto \mathbb{N} \to \mathbb{N} \to \mathbb{N}\}$. We can rewrite this term with the (corresponding type instances of the) rule for recursors. Using a call-by-value stategy, we obtain:

$$\begin{array}{c} rec(S(0), 0, rec(0, \lambda\, x : \mathbb{N}\ y : \mathbb{N}.\ +(x, y), \lambda\, x : \mathbb{N}\ y : \mathbb{N} \to \mathbb{N} \to \mathbb{N}\ z : \mathbb{N}.y(+(x, z)))) \\ \to^3_{\{U:\alpha,\ X:\mathbb{N}\to\alpha\to\alpha\}\ \vdash\ rec(0,U,X)\to U} rec(S(0), 0, \lambda\, x : \mathbb{N}\ y : \mathbb{N}.\ +(x, y)) \\ \to^\epsilon_{\{x:\mathbb{N},\ U:\alpha,\ X:\mathbb{N}\to\alpha\to\alpha\}\ \vdash\ rec(S(x),U,X)\to U} @(\lambda\, x : \mathbb{N}\ y : \mathbb{N}.\ +(x, y), 0, rec(0, 0, \lambda\, x : \mathbb{N}\ y : \mathbb{N}.\ +(x, y))) \\ \to^\epsilon_\beta @(\lambda\, y : \mathbb{N}.\ +(0, y), rec(0, 0, \lambda\, x : \mathbb{N}\ y : \mathbb{N}.\ +(x, y))) \to^\epsilon_\beta +(0, rec(0, 0, \lambda\, x : \mathbb{N}\ y : \mathbb{N}.\ +(x, y))) \\ \to^2_{\{U:\alpha,\ X:\mathbb{N}\to\alpha\to\alpha\}\ \vdash\ rec(0,U,X)\to U} +(0, 0) \\ \to^\epsilon_{\{x:\mathbb{N}\}\ \vdash\ +(x,0)\to x} 0 \end{array}$$

As a general benefit, the use of polymorphic signatures allows us to have only one recursor rule, instead of inifinitely many rules described by one rule schema as in Gödel's original presentation. □

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

Because plain higher-order rewriting is type preserving, we may often omit the environment Γ in which a term s is typechecked as well as its type, and consider the sequence of terms originating from s by reduction from a given set R of higher-order rules. In other words, we will often consider rewriting as a relation operating directly on terms. This will allow us to sometimes simplify our notations in the rest of the paper.

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

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

2.6 Conversion rules

Equations are rewrite rules which can be used in both directions. These three particular equations originate from the λ-calculus, and are called α-, β- and η-equality:

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

The above equations are indeed equation schemas : all occurrences of u and v stand for arbitrary terms to which the substitutions $\{x \to u\}$ and $\{x \to y\}$ apply. Of course, it must be shown that the lefthand and righthand sides of rules are typable with the same type in their environment, thanks to Lemma 2.12. As already said, this justifies an abuse of notation, by forgetting the environment of the equations. As usual, we also do not distinguish α-convertible terms.

We use $\xrightarrow[\beta]{*}$ for the congruence generated by the β-equality, and $\longrightarrow_\beta$ for the β-reduction rule:

$$\{u:\alpha,\ v:\beta\} \vdash_{\mathcal{F}} @(\lambda x:\alpha.v, u) \quad \longrightarrow_\beta \quad v\{x \mapsto u\}$$

Since η-expansion may not terminate, its use is restricted by spelling out in which context it applies:

$$\{u:\sigma_1\to\ldots\to\sigma_n\to\sigma\} \vdash_{\mathcal{F}} s[u]_p \xrightarrow[\eta]{p} s[\lambda x_1:\sigma_1,\ldots,x_n:\sigma_n.@(u,x_1,\ldots,x_n)]_p$$

$$\textbf{if} \begin{cases} \sigma \text{ is a canonical output type} \\ x_1,\ldots,x_n \notin \mathcal{V}ar(u) \\ u \text{ is not an abstraction} \\ s|_q \text{ is not an application in case } p = q\cdot 1 \end{cases}$$

The last condition means that the first argument of an application cannot be recursively expanded on top, therefore preventing non-termination. We could of course allow any finite η-expansion of these terms.

It is well-known that the simply typed λ-calculus is confluent (modulo α-conversions) and terminating with respect to β-reductions and either the above notion of η-expansions, or the more usual notion of η-reduction. We write $s\downarrow_\beta$, ($s\downarrow$ for short) for the unique *β-normal form* of the term s, $s\uparrow^\eta$ ($s\uparrow$ for

short), for the unique (up to α-equivalence) η-*long form* of s wrt. η-expansion, $s\downarrow_\eta$ for the unique (up to α-equivalence) η-normal form of s wrt. η-reduction, and $u\updownarrow^\eta_\beta$, ($u\updownarrow$ for short) for its unique (up to α-equivalence) normal form with respect to β-reductions and η-expansions, also called η-*long normal form* (terms in η-long normal form are called *normalized*). We may sometimes annotate the arrow with the set of positions at which rewriting is allowed or disallowed, as in $\xrightarrow[\eta]{\Lambda\downarrow}$, $\uparrow^\Lambda$, or $\xrightarrow[\eta]{(\neq\Lambda)\uparrow}$. These definitions extend to substitutions in a natural way. We allow ourselves to normalize sets of terms, writing $A\downarrow$, for the set of normalized terms of the set A. These normal forms satisfy the following well-known properties:

Lemma 2.17 $u\updownarrow = u\downarrow\uparrow = u\uparrow\downarrow$.

Lemma 2.18 *Normalized terms are of the following two forms:*
(i) $\lambda\overline{x}:\overline{\rho}.@(X,v_1,\ldots,v_p)$*, for some* $\overline{x}:\overline{\rho}$*,* $X:\tau_1\to\ldots\to\tau_p\to\tau\in\mathcal{X}$ *where* $p>0$ *and* τ *is a data type or a type variable, and normalized terms* $v_1,\ldots,v_p$*, omitting* $@()$ *when* $p=0$*;*
(ii) $\lambda\overline{x}:\overline{\rho}.@(F(u_1,\ldots,u_n),v_1,\ldots,v_p)$*, for some* $\overline{x}:\overline{\rho}$*,* $F\in\mathcal{F}_{\sigma_1\times\ldots\times\sigma_n\to(\tau_1\to\ldots\to\tau_p\to\tau)}$ *where* τ *is a data type or a type variable, and normalized terms* $u_1,\ldots,u_n,v_1,\ldots,v_p$*, omitting* $@()$ *when* $p=0$ *and the other two parentheses when* $n=0$*.*

In normalized terms, the first argument of an application cannot be in η-long form. The following definition aims at characterizing classes of possibly β-normal terms which are not fully η-expanded.

Definition 2.19 *A term* t *is* tail expanded *(resp.* tail normal*) if it is of the following form:*
(i) $t\in\mathcal{X}$*,*
(ii) $t=f(u_1,\ldots,u_n)$*, and* $u_1,\ldots,u_n$ *are in* η*-long form (resp. normalized),*
(iii) $t=@(u_1,\ldots,u_n)$*,* u_1 *is tail expanded (resp. tail normal and is not an abstraction) and* $u_2,\ldots,u_n$ *are in* η*-long normal form (resp. normalized),*
(iv) $t=\lambda x:\sigma.u$*,* u *is tail expanded (resp. tail normal) and not of the form* $@(v,x)$ *with* $x\notin\mathcal{V}ar(v)$*.*

Every normalized term s of type σ contains a tail normal term of the same type σ as a subterm, which is a proper subterm if σ is functional, and the term t itself otherwise.

We denote by $t\uparrow^{\neq\Lambda}$ (resp. $t\updownarrow^{\neq\Lambda}$) the unique tail expanded (resp. tail normal) term s η-equivalent ($\beta\eta$-equivalent) to t.

Expansion and tail expansion are indeed mutually recursive, as examplified by the following properties rephrasing definition 2.19, that will be used without mentionning when needed:

$$\begin{array}{rcl}
(\lambda x.u)\uparrow^{\neq\Lambda} & = & \lambda x.(u\uparrow^{\neq\Lambda}) \quad \text{provided } u\neq @(v,x)\\
(\lambda x.u)\uparrow & = & \lambda x.(u\uparrow)\\
f(\overline{u})\uparrow^{\neq\Lambda} & = & f(\overline{u}\uparrow)\\
f(\overline{u})\uparrow & = & (f(\overline{u})\uparrow^{\neq\Lambda})\uparrow^\Lambda\\
@(u_1,u_2,\ldots,u_n)\uparrow^{\neq\Lambda} & = & @(u_1\uparrow^{\neq\Lambda},u_2\uparrow,\ldots,u_n\uparrow)\\
@(\overline{u})\uparrow & = & (@(\overline{u})\uparrow^{\neq\Lambda})\uparrow^\Lambda
\end{array}$$

These easy properties can be regarded as an algorithm for computing the η-expansion and tail η-expansion of a term.

Important properties of normal terms carry over to tail normal terms :

Lemma 2.20 $u\updownarrow^{\neq\Lambda} = u\uparrow^{\neq\Lambda}\downarrow$.

Lemma 2.21 *Let s be tail normal, and ξ is a type substitution. Then $s\xi\uparrow^{\neq\Lambda} = s\xi\uparrow^{\neq\Lambda}$.*

Lemma 2.22 *Let s be a tail normal term, t be a tail normal neutral term, and $p \in \mathcal{P}os(s)$ such that $s|_p$ and t have the same type. Then $s[t]_p$ is tail normal.*

Lemma 2.23 *Let $s \longleftrightarrow_{\beta\eta} t$. Then s and t have the same tail normal form.*

We can see that tail normal terms enjoy properties similar to those of normalized terms. Being simpler is an argument for their use.

2.7 Normalized higher-order rewriting [30, 27]

The idea of normalized higher-order rewriting is to define computations over λ-terms used as a suitable abstract syntax for encoding functional objects like programs or specifications. It uses higher-order pattern matching for firing rules. We do not assume here that output types of function declarations or variables are data types, a quite restrictive assumption always made in the litterature : output types may be functionnal or even polymorphic. Because our rules will be defined as pairs of tail normal terms, they will not be η-expanded at the top as one might have expected, and will rewrite tail normal subterms.

Definition 2.24 *A* normalized rewrite rule *is a rewrite rule $\Gamma \ \vdash \ l \to r : \sigma$ such that l and r are tail normal terms. A* normalized term rewriting system *is a set of normalized rewrite rules.*

Given a normalized term rewriting system R and an environment Γ, a tail normal term s such that $\Gamma \ \vdash_{\mathcal{F}} s : \sigma$ rewrites to a term t at position p with the tail normal rule $\Gamma_i \ \vdash \ l_i \to r_i : \sigma_i$, the type substitution ξ and the term substitution γ, written $\Gamma \ \vdash \ s \underset{\Gamma_i\xi \ \vdash \ l_i\xi \to r_i\xi : \sigma_i\xi}{\xrightarrow{p}} t$, or simply $\Gamma \ \vdash \ s \to_R t$, or even $s \to_R t$ assuming the environment Γ, if the following conditions are satisfied :
(i) $\mathcal{D}om(\gamma) \subseteq \Gamma_i\xi$,
(ii) $\Gamma_i\xi \cdot \mathcal{R}an(\gamma) \subseteq \Gamma_{s|_p}$,
(iii) $s|_p$ *is tail normal and* $s|_p \longleftrightarrow^*_{\beta\eta} l_i\xi \uparrow^{\neq\eta} \gamma$,
(iv) $t = (s[r_i\xi\uparrow^{\neq\Lambda} \gamma \downarrow_\beta]_p) \downarrow_\beta$.

Note that it is not a restriction to assume that $s|_p$ is tail normal, since we can always chose p fulfilling this property.

As previously, we can show that $\Gamma_{s|_p} \vdash_{\mathcal{F}} s|_p : \sigma_i\xi$, allowing to compute the type substitution ξ in linear time by first-order pattern matching. Then, normalized rewriting uses higher-order pattern matching to compute the substitution γ, if any, such that $s|_p \longleftrightarrow^*_{\beta\eta} l_i\xi \uparrow^\eta \gamma$. Higher-order pattern matching is an open problem for order strictly bigger than 4, but is decidable, with a linear complexity again, when the lefthand sides of rules are patterns in the sense of Miller [28]. A key observation is the following:

Lemma 2.25 *Let $\Gamma \vdash_{\mathcal{F}} s : \sigma$ for some tail normal term s and $\Gamma \ \vdash \ s \to_R t$. Then $\Gamma \vdash_{\mathcal{F}} t : \sigma$ and t is tail normal.*

Proof: Normalization of t is by definition and Lemma 2.22. For the typing judgement, the proof is similar to that of Lemma 2.14, by using in addition confluence of $\beta\eta$ reductions in order to show that $s|_p$ and $l_i\gamma$ have the same type, since β-reductions and η-reductions are type preserving. □

We can adopt a more efficient definition of $t = s[r_i\xi \uparrow^\eta \gamma \downarrow_\beta]_p$ in case the output type of a function symbol is a data type, an assumption frequently met in practice but not required here. In the general case, it is enough to climb up the term from the position p to the root as long as the symbol on the path is an application operator. This makes it easy to implement with an appropriate data structure for terms.

Example 3 This encoding of first-order prenex normal forms is adapted from [30], where its local confluence is proved via the computation of its (higher-order) critical pairs. Formulas are represented as λ-terms with sort $form$. The idea is that quantifiers bind variables via the use of a functional argument.

Let $\mathcal{S} = \{form\}$, $\mathcal{F} = \{\wedge, \vee : form \times form \to form; \neg : form \to form; \forall, \exists : (form \to form) \to form\}$. The rules are the following:

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

We give later a termination proof of these rules based on the ordering we develop in the paper. □

This example makes sense in the context of normalized higher-order rewriting, that is, when rewriting modulo $\beta\eta$, simply because using plain pattern matching instead of higher-order pattern matching would not allow to pull out quantifiers for all terms. For example, the formula $\phi \wedge \forall(\lambda z.z \vee z)$ does not match with the lefthand side of the first rule if plain pattern matching is used while it does with higher-order pattern matching.

2.8 Higher-Order Reduction Orderings

We will make intensive use of well-founded relations for proving strong normalization properties, using the vocabulary of rewrite systems for these relations. For our purpose, these relations may not be transitive, hence are not necessarily orderings, although their transitive closures will be well-founded orderings, which justifies to sometimes call them orderings by abuse of terminology.

For our purpose, a *strict ordering* is an irreflexive and transitive relation, and an *ordering* is the union of its strict part with α-conversion. The following results will play a key role, see [13]:

Assume $\succ, \succ_1, \ldots, \succ_n$ are relations on sets $S, S_1, \ldots, S_n$. Let

- $(\succ_1, \ldots, \succ_n)_{mon}$ be the relation on $S_1 \times \ldots \times S_n$ defined as $(s_1, \ldots, s_n)(\succ_1, \ldots, \succ_n)_{mon}(t_1, \ldots, t_n)$ iff $s_1 \succ_1 t_1, \ldots, s_n \succ_n t_n$;

- $(\succ_1, \ldots, \succ_n)_{lex}$ be the relation on $S_1 \times \ldots \times S_n$ defined as $(s_1, \ldots, s_n)(\succ_1, \ldots, \succ_n)_{lex}(t_1, \ldots, t_n)$ iff $s_1 = t_1, \ldots, s_{i-1} = t_{i-1}$ and $s_i \succ_i t_i$ for some $i \in [1..n]$;

- $\succ_{mul}$ be the relation on the set of multisets of elements of S defined as $M \cup \{x\} \succ_{mul} N \cup \{y_1, \ldots, y_n\}$ iff $M \succ_{mul} N$ and $\forall i \in [1..n]\ x \succ y_i$.

It is well known that these operations preserve the well-foundedness of the relations $\succ, \succ_1, \ldots, \succ_n$.

As already stressed, we are going to define non-transitive, well-founded relations on higher-order terms. In addition, these relations will be monotonic for terms of equivalent type, for some given quasi-ordering $>_{\mathcal{T}_\mathcal{S}}$ on types. Therefore, the union of these relations with strict-subterm may not be well-founded in general. However, it will be for an adequate restriction of the subterm relation:

Definition 2.26 *Let $\geq_{\mathcal{T}_\mathcal{S}}$ be a type ordering. The (strict)* type-decreasing subterm relation*, denoted by $\rhd_{>\mathcal{T}_\mathcal{S}}$, is defined as $s : \sigma \rhd_{>\mathcal{T}_\mathcal{S}} t : \tau$ iff $s \rhd t$, $\sigma \geq_{\mathcal{T}_\mathcal{S}} \tau$ and $\mathcal{V}ar(s) \subseteq \mathcal{V}ar(t)$.*

Lemma 2.27 *Let $>$ be a well-founded relation of a set S of terms which is monotonic for terms of equivalent types (in $=_{\mathcal{T}_S}$). Then, $> \cup \rhd_{>_{\mathcal{T}_S}}$ is well-founded.*

Proof: Since $\geq_{\mathcal{T}_S}$ is well-founded, it is enough to show the property for terms of equivalent type. This follows from the fact that $\rhd_{>_{\mathcal{T}_S}}$ and $>$ commute for such terms by the monotonicity property of $>$. □

We end up this section by defining the notion of reduction orderings operating on higher-order terms, which allow one's to show that the rewrite relations $\longrightarrow_R \cup \longrightarrow_\beta$ or $\longrightarrow_{R^\eta_\beta}$ are well-founded by simply comparing the lefthand and righthand sides of the (polymorphic) rules in R:

Definition 2.28 *A* higher-order reduction ordering $\succ$ *is a well-founded ordering of the set of judgements such that:*

(i) $\succ$ *is* monotonic, *i.e.,* $(\Gamma \vdash_{\mathcal{F}} s : \sigma) \succ (\Gamma \vdash_{\mathcal{F}} t : \sigma)$ *implies* $\forall \Gamma' \vdash_{\mathcal{F}} u[x : \sigma] : \tau$ *such that* Γ *and* Γ' *are compatible, then* $(\Gamma \cdot \Gamma' \vdash_{\mathcal{F}} u[s] : \tau) \succ (\Gamma \cdot \Gamma' \vdash_{\mathcal{F}} u[t] : \tau)$*;*

(ii) $\succ$ *is* stable, *i.e.,* $(\Gamma \vdash_{\mathcal{F}} s : \sigma) \succ (\Gamma \vdash_{\mathcal{F}} t : \sigma)$ *implies that for all substitution* γ *whose domain is compatible with* Γ*, then* $(\Gamma \cdot \mathcal{R}an(\gamma) \vdash_{\mathcal{F}} s\gamma : \sigma) \succ (\Gamma \cdot \mathcal{R}an(\gamma) \vdash_{\mathcal{F}} t\gamma : \sigma)$*;*

(iii) $\succ$ *is* compatible, *i.e.,* $(\Gamma \vdash_{\mathcal{F}} s : \sigma) \succ (\Gamma \vdash_{\mathcal{F}} t : \sigma)$ *implies* $(\Gamma' \vdash_{\mathcal{F}} s : \sigma) \succ (\Gamma' \vdash_{\mathcal{F}} t : \sigma)$ *for every environment* Γ' *such that* Γ *and* Γ' *are compatible,* $\Gamma' \vdash_{\mathcal{F}} s : \sigma$ *and* $\Gamma' \vdash_{\mathcal{F}} t : \sigma$*;*

(iv) $\succ$ *is* functional, *i.e.,* $(\Gamma \vdash_{\mathcal{F}} s : \sigma \longrightarrow_\beta t : \sigma)$ *implies* $(\Gamma \vdash_{\mathcal{F}} s : \sigma) \succ (\Gamma \vdash_{\mathcal{F}} t : \sigma)$.

Besides, $\succeq$ *is said to be* polymorphic *if* $(\Gamma \vdash_{\mathcal{F}} s : \sigma) \succ (\Gamma \vdash_{\mathcal{F}} t : \sigma)$ *implies* $(\Gamma\xi \vdash_{\mathcal{F}} s\xi : \sigma\xi) \succ (\Gamma\xi \vdash_{\mathcal{F}} t\xi : \sigma\xi)$ *for all type substitutions* ξ.

Polymorphism can be seen as a monotonicity property of System F's application operator of a term to a type : the need for stating polymorphism arises from the lack of type quantification and type application in our formalism.

Since rewrite rules of a higher-order term rewriting system R are type-preserving in any environment where they can be used, we will often abuse notations by writing $(\Gamma \vdash_{\mathcal{F}} s : \sigma \succ t : \sigma)$ instead of $(\Gamma \vdash_{\mathcal{F}} s : \sigma) \succ (\Gamma \vdash_{\mathcal{F}} t : \sigma)$ when comparing two terms s and t such that $\Gamma \vdash_{\mathcal{F}} s \longrightarrow_R t$. This amounts to view the ordering as a relation on typed terms instead of a relation on judgements. We may even sometimes omit types and/or environments, considering the ordering as operating directly on terms, and write $\Gamma \vdash s \succ t$, $s : \sigma \succ t : \tau$, or $s \succ t$.

Theorem 2.29 *Let $\succeq$ be a polymorphic, higher-order reduction ordering and $R = \{\Gamma_i \vdash_{\mathcal{F}} l_i \to r_i\}_{i \in I}$ be a higher-order rewrite system such that $\Gamma_i \vdash_{\mathcal{F}} l_i \succ r_i$ for every $i \in I$. Then the relation $\longrightarrow_R \cup \longrightarrow_\beta$ is strongly normalizing.*

Proof: Assume that $\Gamma \vdash_{\mathcal{F}} s : \sigma$ and $\Gamma \vdash_{\mathcal{F}} s \xrightarrow[\Gamma_i\xi \vdash l_i\xi \to r_i\xi : \sigma_i\xi]{p} t$. By definition of higher-order rewriting, $\Gamma_{s|_p} \vdash_{\mathcal{F}} s|_p : \sigma_i\xi$, $\mathcal{D}om(\gamma) \subseteq \Gamma_i\xi$, $\Gamma_i\xi \cdot \mathcal{R}an(\gamma) \subseteq \Gamma_{s|_p}$, $s|_p = l_i\xi\gamma$, and $t = s[r_i\xi\gamma]_p$. By assumption, $\Gamma_i \vdash_{\mathcal{F}} l_i \succ r_i : \sigma_i$. By polymorphism, $\Gamma_i\xi \vdash_{\mathcal{F}} l_i\xi \succ r_i\xi : \sigma_i\xi$. By stability, $\Gamma_i\xi \cdot \mathcal{R}an(\gamma) \vdash_{\mathcal{F}} l_i\xi\gamma \succ r_i\xi\gamma : \sigma_i\xi$. By compatibility, $\Gamma_{s|_p} \vdash_{\mathcal{F}} l_i\xi\gamma \succ r_i\xi\gamma : \sigma_i\xi$. By monotonicity of $\succ$ for terms of equal type, $\Gamma_{s|_p} \cdot \Gamma \vdash_{\mathcal{F}} s[l_i\xi\gamma] = s \succ s[r_i\xi\gamma] = t : \sigma$. By compatibility again, $\Gamma \vdash_{\mathcal{F}} s \succ t$. Finally, the case of a β-reduction is similar. □

The polymorphic property of higher-order reduction orderings allows to show termination in all monomorphic instances of the signature by means of a single comparison for each polymorphic rewrite rule. Polymorphic higher-order reduction orderings are therefore an appropriate tool in order to make termination proofs of polymorphic plain higher-order rewrite systems. In case of a monomorphic rewrite system, there is of course no need for a polymorphic ordering, even if the signature itself is polymorphic.

2.9 Normalized Higher-Order Reduction Orderings

Higher-order reduction orderings turn out to be adequate to show termination of normalized rewriting.

Definition 2.30 *A* higher-order normalized reduction ordering $\succ$ *is a well-founded ordering of the set of judgements such that:*

(i) $\succ$ *is* tail monotonic *for tail expanded terms : for all tail-expanded terms* s *and* t *such that* $(\Gamma \vdash_{\mathcal{F}} s : \sigma) \succ (\Gamma \vdash_{\mathcal{F}} t : \sigma)$*, then* $\forall \Gamma' \vdash_{\mathcal{F}} u[x : \sigma] : \tau$ *such that* Γ *and* Γ' *are compatible, and* $u[s]$ *and* $u[t]$ *are tail expanded,* $(\Gamma \cdot \Gamma' \vdash_{\mathcal{F}} u[s] : \tau) \succ (\Gamma \cdot \Gamma' \vdash_{\mathcal{F}} u[t] : \tau)$*;*

(ii) $\succ$ *is* stable*;*

(iii) $\succ$ *is* compatible *for tail expanded terms;*

(iv) $\succ$ *is* tail functional *: for all tail expanded terms* s *and* t *such that* $(\Gamma \vdash_{\mathcal{F}} s : \sigma \longrightarrow_{\beta} t : \sigma)$*, then* $(\Gamma \vdash_{\mathcal{F}} s : \sigma) \succ (\Gamma \vdash_{\mathcal{F}} t : \sigma)$.

Restricting functionality and monotonicity to tail expanded terms will be important in Section 6 where such a normalized ordering is exhibited which does not satisfy functionality and monotonicity for arbitrary terms.

Definition 2.31 *A subrelation* $\succ_{\beta}^{\eta}$ *of a higher-order normalized reduction ordering* $\succ$ *is said to be*

(i) β-stable*, in which case* $\succ_{\beta}^{\eta}$ *is said to be a* normalized higher-order reduction ordering*, if* $(\Gamma \vdash_{\mathcal{F}} s) \succ_{\beta}^{\eta} (\Gamma \vdash_{\mathcal{F}} t)$ *implies* $(\Gamma \cdot \mathcal{R}an(\gamma) \vdash_{\mathcal{F}} s\gamma\downarrow_{\beta}) \succ (\Gamma \cdot \mathcal{R}an(\gamma) \vdash_{\mathcal{F}} t\gamma\downarrow_{\beta})$ *for all tail normal terms* s, t *and tail normal substitution* γ *compatible with* Γ*;*

(ii) η-polymorphic *(or simply* polymorphic*) if* $(\Gamma \vdash_{\mathcal{F}} s) \succ_{\beta}^{\eta} (\Gamma \vdash_{\mathcal{F}} t)$ *implies* $(\Gamma\xi \vdash_{\mathcal{F}} s\xi\uparrow^{\neq\Lambda}) \succ_{\beta}^{\eta} (\Gamma\xi \vdash_{\mathcal{F}} t\xi\uparrow^{\neq\Lambda})$ *for all tail normal terms* s, t *and all type substitution* ξ.

Theorem 2.32 *Assume that* $\succ$ *is a higher-order normalized reduction ordering and that* $\succ_{\beta}^{\eta}$ *is a* β*-stable and* η*-polymorphic subrelation of* $\succ$*. Let* $R = \{\Gamma_i \vdash l_i \to r_i : \sigma_i\}_{i \in I}$ *be a higher-order rewrite system such that* $(\Gamma_i \vdash_{\mathcal{F}} l_i) \succ_{\beta}^{\eta} (\Gamma_i \vdash_{\mathcal{F}} r_i)$ *for every* $i \in I$*. Then the relation* $\longrightarrow_{R_{\beta}^{\eta}}$ *is strongly normalizing.*

Proof: Let s be a tail normal term such that $\Gamma \vdash_{\mathcal{F}} s \xrightarrow[\Gamma_i\xi \vdash l_i\xi \to r_i\xi : \sigma_i\xi]{p} t$. Since $s|_p$ is tail normal, and $s|_p \longleftrightarrow_{\beta\eta}^{*} l_i\xi\uparrow^{\neq\Lambda} \gamma$ by definition of rewriting, we get $s|_p = l_i\xi\uparrow^{\neq\Lambda} \gamma \downarrow_{\beta}$ by the confluence property of β-reductions together with tail expansions. Since s itself is tail normal, it follows that $s = s[l_i\xi\uparrow^{\neq\Lambda} \gamma \downarrow_{\beta}]_p$. By assumption, $\Gamma_i \vdash_{\mathcal{F}} l_i \succ_{\beta}^{\eta} r_i$. By η-polymorphism, $\Gamma_i\xi \vdash_{\mathcal{F}} l_i\xi\uparrow^{\neq\Lambda} \succ_{\beta}^{\eta} r_i\xi\uparrow^{\neq\Lambda}$. By β-stability, $\Gamma_i\xi \cdot \mathcal{R}an(\gamma) \vdash_{\mathcal{F}} (l_i\xi\uparrow^{\neq\Lambda})\gamma \downarrow_{\beta} \succ \vdash_{\mathcal{F}} (r_i\xi\uparrow^{\neq\Lambda})\gamma \downarrow_{\beta}$. By compatibility, we get $\Gamma_{s|_p} \vdash_{\mathcal{F}} (l_i\xi\uparrow^{\neq\Lambda} \gamma) \downarrow_{\beta} \succ (r_i\xi\uparrow^{\neq\Lambda})\gamma \downarrow_{\beta}$. By monotonicity of $\succ$ for tail normal terms of equal type, $\Gamma_{s|_p} \cdot \Gamma \vdash_{\mathcal{F}} s[l_i\xi\uparrow^{\neq\Lambda} \gamma \downarrow_{\beta}]_p = s \succ s[r_i\xi\uparrow^{\neq\Lambda} \gamma \downarrow_{\beta}]_p$. By compatibility again, $\Gamma \vdash_{\mathcal{F}} s[l_i\xi\uparrow^{\neq\Lambda} \gamma \downarrow_{\beta}]_p = s \succ s[r_i\xi\uparrow^{\neq\Lambda} \gamma \downarrow_{\beta}]_p$. By tail functionnality, $\Gamma \vdash_{\mathcal{F}} s[r_i\xi\uparrow^{\neq\Lambda} \gamma \downarrow_{\beta}]_p \succ t$. Finally, by transitivity, $\Gamma \vdash_{\mathcal{F}} s \succ t$, implying strong normalization as claimed. □

We are left constructing polymorphic (normalized) higher-order reduction orderings.

3 The Higher-Order Recursive Path Ordering

In this section, we present a first version of our higher-order recursive path ordering, together with examples of its use. We will give more elaborated versions in the subsequent sections. Some proofs will be omitted in this section, to avoid duplications.

3.1 The HORPO ordering

The higher-order recursive path ordering on typed higher-order terms is generated from three basic ingredients, presented first. The relation itself is defined in the fourth subsection.

3.1.1 Type ordering

We assume given two (intimately related) quasi-orderings on types $\geq_{\mathcal{T}_\mathcal{S}}$ and $\geq^{\rightarrow}_{\mathcal{T}_\mathcal{S}}$ such that the following properties are satisfied :

1. *containement* : $>_{\mathcal{T}_\mathcal{S}} \subseteq >^{\rightarrow}_{\mathcal{T}_\mathcal{S}}$ and $=_{\mathcal{T}_\mathcal{S}} = =^{\rightarrow}_{\mathcal{T}_\mathcal{S}}$;
2. *Well-foundedness*: $>^{\rightarrow}_{\mathcal{T}_\mathcal{S}}$ is well-founded (this is no restriction for finite signatures);
3. *Arrow subterm property* : $\tau \rightarrow \sigma >^{\rightarrow}_{\mathcal{T}_\mathcal{S}} \tau$ and $\tau \rightarrow \sigma >^{\rightarrow}_{\mathcal{T}_\mathcal{S}} \sigma$;
4. *Functional preservation* (i), we also say that $=_{\mathcal{T}_\mathcal{S}}$ is *arrow preserving*:
$$\tau \rightarrow \sigma =_{\mathcal{T}_\mathcal{S}} \alpha \text{ iff } \alpha = \tau' \rightarrow \sigma',\ \tau' =_{\mathcal{T}_\mathcal{S}} \tau \text{ and } \sigma =_{\mathcal{T}_\mathcal{S}} \sigma';$$
5. *Functional preservation* (ii), we also say that $>_{\mathcal{T}_\mathcal{S}}$ is *arrow decreasing*:
$$\tau \rightarrow \sigma >_{\mathcal{T}_\mathcal{S}} \alpha \text{ implies } \sigma \geq_{\mathcal{T}_\mathcal{S}} \alpha \text{ or } \alpha = \tau' \rightarrow \sigma',\ \tau' =_{\mathcal{T}_\mathcal{S}} \tau \text{ and } \sigma >_{\mathcal{T}_\mathcal{S}} \sigma';$$

The ordering $\geq_{\mathcal{T}_\mathcal{S}}$ will be used in the definition of the higher-order recursive path ordering, while the ordering $>^{\rightarrow}_{\mathcal{T}_\mathcal{S}}$ will be used to build up induction arguments for which the subterm property is necessary.
The next two properties have to do with polymorphic signatures:

6. *Stability* (under substitution): If $\sigma >_{\mathcal{T}_\mathcal{S}} \tau$, ($\sigma =_{\mathcal{T}_\mathcal{S}} \tau$, respectively) then $\sigma\xi >_{\mathcal{T}_\mathcal{S}} \tau\xi$ for every ground type substitution ξ ($\sigma\xi =_{\mathcal{T}_\mathcal{S}} \tau\xi$, respectively);
7. *Compatibility* of signature declarations with type instantiations: for any function symbol $f : \sigma_1 \times \ldots \times \sigma_n \rightarrow \sigma$ and type instantiations $\xi',\ \xi''$ such that $\forall i \in [1..n]\ \sigma_i\xi' =_{\mathcal{T}_\mathcal{S}} \sigma_i\xi''$, then $\sigma\xi' =_{\mathcal{T}_\mathcal{S}} \sigma\xi''$.

Operating on type expressions, this ordering is an ordering on first-order unisorted terms. We will see later how the recursive path ordering can be restricted so as to meet the above properties.

Lemma 3.1 *Assume that $\sigma =_{\mathcal{T}_\mathcal{S}} \alpha$ and σ is a data type, then α is a data type as well.*

Preservation of data types follows from the preservation of arrows and stability (which is needed to avoid having a type variable equal to a data type). Preservation of functional and data types will play a key role in our proofs.
We denote by $\mathcal{T}_\mathcal{S}^{min}$ the set of ground types which are minimal with respect to $>^{\rightarrow}_{\mathcal{T}_\mathcal{S}}$.

Lemma 3.2 *Assuming that $\mathcal{S} \neq \emptyset$, then $\mathcal{T}_\mathcal{S}^{min}$ is a non-empty set of data types.*

Proof: Because $>^{\rightarrow}_{\mathcal{T}_\mathcal{S}}$ is well-founded and arrow types cannot be minimal in $>^{\rightarrow}_{\mathcal{T}_\mathcal{S}}$. □

3.1.2 Statuses

We assume given a partition $Mul \uplus Lex$ of $\mathcal{F}$ and we say that $f \in Mul$ (resp. $f \in Lex$) has a *multiset* (resp. *lexicographic*) status.

3.1.3 Precedence

We assume given a quasi-ordering $\geq_{\mathcal{F}}$ on $\mathcal{F}$, called the *precedence*, such that

1. if $f : \overline{\sigma} \to \sigma =_{\mathcal{F}} g : \overline{\tau} \to \tau$, then $\sigma =_{\mathcal{T_S}} \tau$ and $f \in Lex$ iff ($g \in Lex$ and $\overline{\sigma}(=_{\mathcal{T_S}})_{mon}\overline{\tau}$);
2. $>_{\mathcal{F}}$ is well-founded;
3. free variables are considered as constants incomparable in $>_{\mathcal{F}}$ among themselves and with other function symbols, coming therefore together with a type declaration.

3.1.4 Definition of the higher-order recursive path ordering

We can now give our definition of the higher-order recursive path ordering (HORPO), which builds upon Dershowitz's recursive path ordering for first-order terms [12], and improves quite significantly over [22].

In contrast to the usual first-order case, it is not possible to consider the terms to be compared as ground, since the property is not closed by taking subterms under an abstraction. We will therefore define HORPO for non-ground terms, by considering, as in the first-order case, the free variables of the typing environement as new constant symbols which are incomparable among themselves and with other function symbols. It therefore makes sense to compare typing judgements $\Gamma \vdash s : \sigma$ and $\Sigma \vdash t : \tau$ rather than terms.

Following the tradition, we implicitely consider equivalence classes of terms modulo α-conversion, using the syntactic equality symbol $=$ for the equivalence $=_\alpha$, and define our ordering $\succ_{horpo}$ by means of a set of rules, writing $\succeq_{horpo}$ for $\succ_{horpo} \cup =$. In contrast, the recursive path ordering [13] contains a non-trivial equivalence allowing to freely permute subterms below multiset function symbols. Although one may argue about the practical relevance of this feature, we will adress the question in Section 7.

The ordering definition starts with 4 rules reproducing Dershowitz's recursive path ordering for first-order terms, with one main difference when higher-order terms are compared: the rules can also take care of higher-order terms in the arguments of the smaller side by having a corresponding bigger higher-order term in the arguments of the bigger side (this is redundant for first-order terms because of the subterm property). This idea is captured in the following proposition:

$$A = \forall v \in \overline{t}\ \ (\Gamma \vdash_{\mathcal{F}} s : \sigma) \underset{horpo}{\succ} (\Sigma \vdash_{\mathcal{F}} v : \rho) \text{ or } (\Gamma \vdash_{\mathcal{F}} u : \theta) \underset{horpo}{\succ} (\Sigma \vdash_{\mathcal{F}} v : \rho) \text{ for some } u \in \overline{s}$$

The definition we give below operates on judgements. However, we will quickly forget the judgement version and adopt a more easily readable term version.

Definition 3.3 *Given two judements* $\Gamma \vdash_{\mathcal{F}} s : \sigma$ *and* $\Sigma \vdash_{\mathcal{F}} t : \tau$,

$$(\Gamma \vdash_{\mathcal{F}} s : \sigma) \underset{horpo}{\succ} (\Sigma \vdash_{\mathcal{F}} t : \tau) \textit{ iff } \sigma \geq_{\mathcal{T_S}} \tau \textit{ and}$$

1. $s = f(\overline{s})$ *with* $f \in \mathcal{F}$, *and* $(\Gamma \vdash_{\mathcal{F}} u : \theta) \underset{horpo}{\succeq} (\Sigma \vdash_{\mathcal{F}} t : \tau)$ *for some* $u \in \overline{s}$

2. $s = f(\overline{s})$ *with* $f \in \mathcal{F}$, *and* $t = g(\overline{t})$ *with* $f >_{\mathcal{F}} g$, *and* A

3. $s = f(\overline{s})$ *and* $t = g(\overline{t})$ *with* $f =_{\mathcal{F}} g \in Mul$, *and* $(\Gamma \vdash_{\mathcal{F}} \overline{s} : \overline{\sigma})\ (\underset{horpo}{\succ})_{mul}\ (\Sigma \vdash_{\mathcal{F}} \overline{t} : \overline{\tau})$

4. $s = f(\overline{s})$ *and* $t = g(\overline{t})$ *with* $f =_{\mathcal{F}} g \in Lex$, *and* $(\Gamma \vdash_{\mathcal{F}} \overline{s} : \overline{\sigma})\ (\underset{horpo}{\succ})_{lex}\ (\Sigma \vdash_{\mathcal{F}} \overline{t} : \overline{\tau})$ *and* A

5. $s = @(s_1, s_2)$, *and* $(\Gamma \vdash_{\mathcal{F}} s_1 : \rho \to \sigma) \underset{horpo}{\succeq} (\Sigma \vdash_{\mathcal{F}} t : \tau)$ *or* $(\Gamma \vdash_{\mathcal{F}} s_2 : \rho) \underset{horpo}{\succeq} (\Sigma \vdash_{\mathcal{F}} t : \tau)$

6. $s = \lambda x : \alpha.u$ *with* $x \notin \mathcal{V}ar(t)$, *and* $(\Gamma \cdot \{x : \alpha\} \vdash_{\mathcal{F}} u : \theta) \underset{horpo}{\succeq} (\Sigma \vdash_{\mathcal{F}} t : \tau)$

7. $s = f(\overline{s})$ *with* $f \in \mathcal{F}$, $t = @(\overline{t})$ *is a partial 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* $(\Gamma \vdash_{\mathcal{F}} s : \sigma) \underset{horpo}{\succ} (\Sigma \vdash_{\mathcal{F}} v : \rho)$

9. $\sigma = \alpha \to \theta$, $t = \lambda x : \beta.v$, $\alpha =_{\mathcal{T}_S} \beta$ *and* $(\Gamma \vdash_{\mathcal{F}} @(s, x)) : \theta \underset{horpo}{\succ} (\Sigma \vdash_{\mathcal{F}} v : \rho)$

10. $s = @(s_1, s_2)$, $t = @(\overline{t})$ *is a partial left-flattening of* t, *and*
$\{(\Gamma \vdash_{\mathcal{F}} s_1 : \theta \to \sigma), (\Gamma \vdash_{\mathcal{F}} s_2 : \theta)\}(\underset{horpo}{\succ})_{mul}(\Sigma \vdash_{\mathcal{F}} \overline{t} : \overline{\tau})$

11. $s = \lambda x : \alpha.u,\ t = \lambda x : \beta.v$, $\alpha =_{\mathcal{T}_S} \beta$, *and* $(\Gamma \cdot \{x : \alpha\} \vdash_{\mathcal{F}} u : \theta) \underset{horpo}{\succ} (\Sigma \cdot \{x : \beta\} \vdash_{\mathcal{F}} v : \rho)$

12. $s = @(\lambda x : \alpha.u, v)$ *and* $(\Gamma \vdash_{\mathcal{F}} u\{x \mapsto v\} : \sigma) \underset{horpo}{\succeq} (\Sigma \vdash_{\mathcal{F}} t : \tau)$

The definition is recursive, and, apart from case 12, recursive calls operate on judgements whose terms which are subterms of the term in the starting judgement. This ensures the well-foundedness of the definition, since the union of β-reduction with subterm is well-founded, by comparing pairs of argument terms in the well-founded compatible relation $(\longrightarrow_\beta \cup \rhd, \rhd)_{lex}$. This will actually be used as an inductive argument in many proofs to come[1].

From now on, we will forget about judgements when comparing terms, leaving implicit the environment and the type used in their typing judgements.

Example 4 (example 1 continued) We use here the ordering on types generated by the properties of the type ordering introduced in Section 3.1.1, and assume a multiset status for rec. The first rule succeeds immediately by case 1. For the second rule, we apply case 7, and need to show recursively that (i) $X \succeq_{horpo} X$, (ii) $s(x) \succ_{horpo} x$, and (iii) $rec(s(x), u, X) \succ_{horpo} rec(x, u, X)$. (i) is trivial. (ii) is by case 1. (iii) is by case 3, calling again recursively for $s(x) \succ_{horpo} x$.

Note that we have proved Gödel's polymorphic recursor, for which the output type of rec is any given type. This is because we do not care about types in our comparisons, provided two compared terms are typable with related types. This example was already proved in [22].

We can of course now add some defining rules for sum and product (omitting environments):

$$\begin{array}{rcl} x + 0 & \to & 0 \\ x + s(y) & \to & s(x + y) \\ x * y & \to & rec(y, 0, \lambda z_1 z_2.x + z_2) \end{array}$$

[1] In [36] and related articles, it is argued that the recursive path ordering is not an inductive definition, but a fixpoint definition, making then very complicated everything that had been considered very simple before. To remedy this misunderstanding, it must be stressed that multiset and lexicographic extensions are defined for arbitrary relations and preserve well-foundedness of arbitrary (well-founded) relations (see Section 2.8). They also preserve orderings, of course, but defining multiset and lexicographic extensions for orderings only is simply not appropriate.

The first two rules are easy work. For the third, we use the precedence $* >_{\mathcal{F}} rec$ to eliminate the rec operator. But the computation fails, since there is no subterm of $x * y$ to take care of the righthand side subterm $\lambda z_1 z_2.x + z_2$. We will come back to this example later, after having boosted the ordering. □

Example 5 This example is partly taken from [22]. Let $\mathcal{S} = \{List, \mathbb{N}\}$ and $\mathcal{F} = \{nil :\rightarrow List,\ cons : \mathbb{N} \times List \rightarrow List,\ map : List \times (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow List\}$. The rules for map are:

$$\begin{array}{rcrcl} \{X : \alpha \rightarrow \alpha\} & \vdash & map(nil, X) & \rightarrow & nil \\ \{x : \alpha,\ l : List,\ X : \alpha \rightarrow \alpha\} & \vdash & map(cons(x, l), X) & \rightarrow & cons(@(X, x), map(l, X)) \end{array}$$

Using the ordering on types generated as previously by the properties of the type ordering and the equality of the two sort constants $\mathbb{N}$ and $List$, the first rule is trivially taken care of by case 1. For the second, let $map \in Mul$ and $map >_{\mathcal{F}} cons$. Since $map >_{\mathcal{F}} cons$, applying case 2, we need to show that $map(cons(x, l), X) \succ_{horpo} @(X, x)$ and $map(cons(x, l), X) \succ_{horpo} map(l, X)$. The latter is true by case 3, since $cons(x, l) \succ_{horpo} l$ by case 1. The first is by case 7, as X is an argument of the first term and $cons(x, l) \succ_{horpo} x$ by case 1.

Let us now consider a polymorphic version of the same example, with $\mathcal{S} = \{List\}$, $\mathcal{S}^{\forall} = \{\alpha\}$, and $\mathcal{F} = \{nil :\rightarrow List,\ cons : \alpha \times List \rightarrow List,\ map : List \times (\alpha \rightarrow \alpha) \rightarrow List\}$, then the computation fails since $@(X, x)$ has now type α which cannot be compared to the sort $List$.

We now reformulate the specification of polymorphic lists by taking advantage of our richer type discipline with overloading. Let $\mathcal{S} = \{Nlist : *,\ List : * \rightarrow *\}$, $\mathcal{S}^{\forall} = \{\alpha\}$, and $\mathcal{F} = \{nil :\rightarrow NList,\ cons : \alpha \times NList \rightarrow List(\alpha),\ cons : \alpha \times List(\alpha) \rightarrow List(\alpha),\ map : NList \times (\alpha \rightarrow \alpha) \rightarrow NList,\ map : List(\alpha) \times (\alpha \rightarrow \alpha) \rightarrow List(\alpha)\}$. The rules for map become:

$$\begin{array}{rcrcl} \{X : \alpha \rightarrow \alpha\} & \vdash & map(nil, X) & \rightarrow & nil \\ \{x : \alpha,\ l : NList,\ X : \alpha \rightarrow \alpha\} & \vdash & map(cons(x, l), X) & \rightarrow & cons(@(X, x), map(l, X)) \\ \{x : \alpha,\ l : List(\alpha),\ X : \alpha \rightarrow \alpha\} & \vdash & map(cons(x, l), X) & \rightarrow & cons(@(X, x), map(l, X)) \end{array}$$

The need for duplicating the second map rule comes from the regularity assumption which should be weakened for constants, allowing to declare $nil : List(\alpha)$. Note the two uses of the first map declaration in the first rule and in the righthand side of the second rule, and the three uses of the second map declaration for the other occurrences of map. Letting again $map \in Mul$ and $map >_{\mathcal{F}} cons$, the first rule is taken care of by case 1 as previously. For the second, we need to set $List(\alpha) >_{\mathcal{T}_{\mathcal{S}}} NList$ for the recursive comparison $map(cons(x, l), X) \succ_{horpo} map(l, X)$, and $List(\alpha) >_{\mathcal{T}_{\mathcal{S}}} \alpha$ for the other comparison $map(cons(x, l), X) \succ_{horpo} @(X, x)$. Both are possible (see section 3.5). The computation goes then as previously, and is similar for the third rule. □

Important observations are the following:

Comparing two terms s and t requires comparing their types, the type ordering being used in the construction of interpretations for the strong normalization proof. Note that the present ordering compares terms of related types, improving over [22], where terms of equivalent types only could be compared (the equivalence on types being generated there by the equality on sorts). Without type comparisons based on a type ordering, we could never allow a subterm case for terms headed by an application or by an abstraction. Note that the type of the lefthand side of a recursive comparison may have increased in cases 1 and 5.

When the signature is first-order, Cases 1, 2, 3 and 4 of the definition of $\succ_{horpo}$ together reduce to the usual recursive path ordering for first-order terms, whose complexity is known to be in $O(n^2)$ for an input of size n. Accordingly, we conjecture that a polynomial complexity of low degree is again obtained when dropping Case 12 (we have not investigate this point yet).

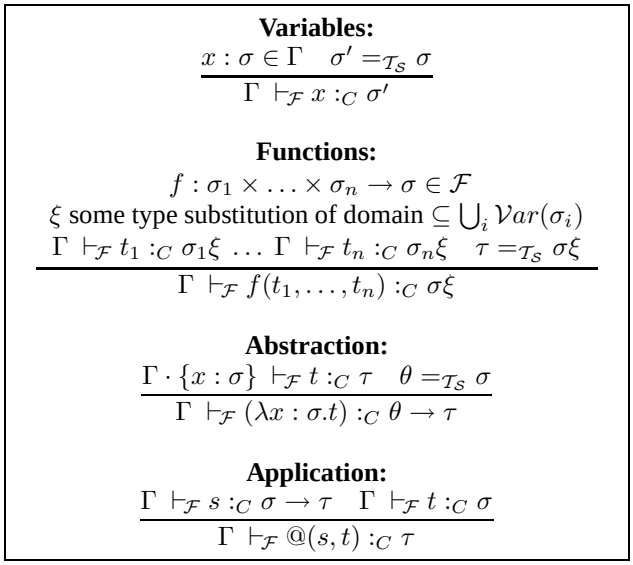

Figure 2. Candidate judgements

All other cases deal with applications and abstractions. Note the following points: there is no precedence case for applications against abstractions; left-flattening is used in the righthand sides only, that is, in Cases 7 and 10, and indeed, our strong normalization proof does not go through if it is also used in the lefthand sides.

Note the tradeoff between the increase of types and the decrease of size when using left-flattening in Case 10: moving from $@(@(a, b), c))$ to $@(a, b, c)$ replaces the subterm $@(a, b)$ by the two smaller subterms a, b, but a has a bigger type than $@(a, b)$. Type considerations may therefore be used to drive the choice of the amount of flattening. We think it is also possible to use a lexicographic comparison, although we did not check this alternative yet.

Another remark to ease the implementation of the ordering is that the non-deterministic or comparison of proposition A used in cases 2, 4, and 7, can be replaced by the equivalent deterministic one:

$$\forall v : \rho \in \overline{t} \text{ if } \rho \leq_{\mathcal{T}_S} \tau \ \ then\ s \underset{horpo}{\succ} v \text{ otherwise } u \underset{horpo}{\succeq} v \text{ for some } u : \theta \in \overline{s} \text{ such that } \theta \geq_{\mathcal{T}_S} \rho$$

We now state the main result of this section, whose proof is the subject of the coming two subsections:

Theorem 3.4 *$\succ_{horpo}$ is decidable, and included into a polymorphic, higher-order reduction ordering.*

3.1.5 Candidate Terms

The use of equivalent types leads naturally to the definition of an extended set of typable terms. These terms will be the basis for using Tait and Girard's reducibility technique. In fact, we will not use the smallest possible set of terms for which our proof technique applies, but the largest one that we were able to characterize via the extended type system described in Figure 2.

Definition 3.5 *Terms typable in the type system of Figure 2 are called* candidate terms.

We first show that every candidate term has a unique type (up to equivalence) in a given environment:

Lemma 3.6 *Assume* $\Gamma \vdash_{\mathcal{F}} s :_C \sigma$. *Then* $\Gamma \vdash_{\mathcal{F}} s :_C \tau$ *iff* $\sigma =_{\mathcal{I}_S} \tau$.

Proof: By induction on the type derivation, and by case on the last judgement applied, using functional preservation for arrow types. **Variables** is trivial. For **Functions**, the if direction results easily from the transitivity of $=_{\mathcal{I}_S}$, but the only if one is a bit more delicate : let $\sigma_1, \ldots, \sigma_n$ be the types of $t_1, \ldots, t_n$ with ξ the associated type substitution in the first type derivation, and $\sigma'_1, \ldots, \sigma'_n$ be the types of $t_1, \ldots, t_n$ with ξ' the associated type substitution in the second type derivation. By induction hypothesis, $\sigma'_i =_{\mathcal{I}_S} \sigma_i$, hence $\sigma_i\xi' =_{\mathcal{I}_S} \sigma_i\xi$ and therefore $\sigma\xi' =_{\mathcal{I}_S} \sigma\xi$ by compatibility of signature declarations with type instantiations. We conclude then easily by transitivity of $=_{\mathcal{I}_S}$. For **Abstraction**, since a type equivalent to $\sigma \to \tau$ is of the form $\sigma' \to \tau'$ with $\sigma' =_{\mathcal{I}_S} \sigma$ and $\tau' =_{\mathcal{I}_S} \tau$, we use the fact that, by induction hypothesis, $t :_C \tau'$ iff $\tau' =_{\mathcal{I}_S} \tau$. For **Application**, we use the induction hypothesis on t to have $\tau' =_{\mathcal{I}_S} \tau$ and the induction hypothesis on s to ajust the type for the application rule to make sense. □

This will allow us to talk about *the type* (up to equivalence) of a candidate term.

Note finally three easy, but important properties of the set of candidate terms:

1. Because β-reduction is type preserving, the set of candidate terms is closed under β-reductions;
2. Because β-reduction is strongly normalizing for a typed λ-calculus with arbitrary constants, the set of candidate terms is well-founded with respect to β-reductions (consider a signature in which $f : \sigma'_1 \times \ldots \times \sigma'_n \to \sigma' \in \mathcal{F}$ provided $f : \sigma_1 \times \ldots \times \sigma_n \to \sigma \in \mathcal{F}$ with $\sigma'_i =_{\mathcal{I}_S} \sigma_i$ for every $i \in [1..n]$ and $\sigma' =_{\mathcal{I}_S} \sigma$;
3. HORPO applies to candidate terms as well, by keeping the same definition.

Candidate terms are used throughout Subsections 3.2 and 3.3.

3.2 Ordering properties of HORPO

From now on, we will mostly compare terms, forgetting about the whole judgements which can be in general inferred from the context. This is made easier by the fact that the typing environment and the type itself do not change along a derivation.

A weakness of our definition is that the relation $\succ_{horpo}$ does not satisfy transitivity. This is not a theoretical problem, however, because we will show that it is well-founded, hence its transitive closure is a well-founded ordering. On the practical side, the use of the transitive closure of $\succ_{horpo}$ may sometimes be necessary, as we will see later.

All properties studied in this section refer to candidate terms.

Lemma 3.7 (Monotonicity) $\succ_{horpo}$ *is monotonic for candidate terms of equivalent types.*

Proof: Assume that $s :_C \sigma \succ_{horpo} t :_C \tau$ with $\sigma =_{\mathcal{I}_S} \tau$. We have to consider three possible cases.

- Let $f : \ldots \times \alpha \times \ldots \to \theta$ be a function symbol such that $\alpha\xi =_{\mathcal{I}_S} \sigma =_{\mathcal{I}_S} \tau$, for some type instantiation ξ. If $f \in Mul$, then $f(\ldots s \ldots) :_C \theta\xi \succ_{horpo} f(\ldots t \ldots) :_C \theta\xi$ follows by case 3. Otherwise, if $f \in Lex$, then $\{\ldots s \ldots\}(\succ_{horpo})_{lex}\{\ldots t \ldots\}$ and, for every $v \in \{\ldots t \ldots\}$, there exists $u \in \{\ldots s \ldots\}$ such that $u \succeq_{horpo} v$. Therefore, $f(\ldots s \ldots) \succ_{horpo} f(\ldots t \ldots)$ by case 4.
- If s, t and u have appropriate types, then $@(s, u) \succ_{horpo} @(t, u)$ and $@(u, s) \succ_{horpo} @(u, t)$ follow by case 10.
- Similarly, $\lambda x.s \succ_{horpo} \lambda x.t$ follows by case 11.

Finally, we can conclude $u[s] :_C \rho \succ_{horpo} u[t] :_C \rho$ for all terms $u :_C \rho$, by induction on the size of u. $\Box$

This of course implies the monotonicity of the quasi-ordering $\succeq_{horpo}$ for candidate terms of equivalent types.

A key usual consequence of monotonicity is that every subterm of a strongly normalizing term is itself strongly normalizing. This is true for type preserving derivations, but false in a context like here where ordering derivations may decrease the type of terms, since a (possibly type decreasing) rewriting of a subterm cannot be lifted to a rewriting of the whole term. This difficulty, however, cannot happen if the type of the subterm is minimal. So, the strong normalization property of a term implies the strong normalization property of its subterms of minimal type. This property will be used later.

Lemma 3.8 (Stability) *$\succ_{horpo}$ is stable.*

Proof: We prove that $\Gamma \vdash_{\mathcal{F}} s :_C \sigma \succ_{horpo} t :_C \tau$ implies $\Gamma \cdot \mathcal{R}an(\gamma) \vdash_{\mathcal{F}} s\gamma :_C \sigma \succ_{horpo} t\gamma :_C \tau$ for all substitution γ, by induction on $(\longrightarrow_\beta \cup \rhd, \rhd)_{lex}$. Since term substitutions do not affect the typing judgments, in order to simplify the proof we will only consider the term comparisons of the ordering and will omit any reference to types and judgements. There are several cases according to the definition.

1. If $s \succ_{horpo} t$ by case 1, then $s_i \succeq_{horpo} t$, and by induction hypothesis $s_i\gamma \succeq_{horpo} t\gamma$, and therefore, $s\gamma \succ_{horpo} t\gamma$ by case 1.
2. If $s \succ_{horpo} t$ by case 2, then $s = f(\overline{s})$, $t = g(\overline{t})$, $f >_{\mathcal{F}} g$ and for all $t_i :_C \rho \in \overline{t}$ either $s \succ_{horpo} t_i$ or $s_j \succeq_{horpo} t_i$ for some $s_j :_C \rho' \in \overline{s}$. By induction hypothesis, for all $t_i\gamma :_C \rho \in \overline{t\gamma}$ either $s\gamma \succ_{horpo} t_i\gamma$ or $s_j\gamma \succeq_{horpo} t_i\gamma$ for some $s_j\gamma :_C \rho' \in \overline{s\gamma}$. Therefore, $s\gamma = f(\overline{s\gamma}) \succ_{horpo} g(\overline{t\gamma}) = t\gamma$ by case 2.
3. If $s \succ_{horpo} t$ by case 3, then $s = f(\overline{s})$, $t = g(\overline{t})$, $f =_{\mathcal{F}} g$, $f, g \in Mul$ and $\overline{s}(\succ_{horpo})_{mul}\overline{t}$. By induction hypothesis $\overline{s\gamma}(\succ_{horpo})_{mul}\overline{t\gamma}$, and hence $s\gamma \succ_{horpo} t\gamma$ by case 3.
4. If $s \succ_{horpo} t$ by case 4, then $s = f(\overline{s})$, $t = g(\overline{t})$, $f =_{\mathcal{F}} g$, $f, g \in Lex$, $\overline{s}(\succ_{horpo})_{lex}\overline{t}$, and for all $t_i :_C \rho \in \overline{t}$ either $s \succ_{horpo} t_i$ or $s_j \succeq_{horpo} t_i$ for some $s_j :_C \rho' \in \overline{s}$. By induction hypothesis $\overline{s\gamma}(\succ_{horpo})_{lex}\overline{t\gamma}$, and as in the precedence case, by induction hypothesis, for all $t_i\gamma :_C \rho \in \overline{t\gamma}$ either $s\gamma \succ_{horpo} t_i\gamma$ or $s_j\gamma \succeq_{horpo} t_i\gamma$ for some $s_j\gamma :_C \rho' \in \overline{s\gamma}$. Therefore $s\gamma \succ_{horpo} t\gamma$ by case 4.
5. If $s \succ_{horpo} t$ by case 5, the reasonning is similar to Case 1.
6. If $s \succ_{horpo} t$ by case 6, then $s = \lambda x.u$ and $u \succeq_{horpo} t$. Let γ a substitution of domain $\mathcal{V}ar(s) \cup \mathcal{V}ar(t)$, hence $x \notin \mathcal{D}om(\gamma)$ by assumption on x. By induction hypothesis, $u\gamma \succeq_{horpo} t\gamma$, hence $s\gamma = \lambda x.u\gamma \succ_{horpo} t\gamma$ by case 6.
7. If $s \succ_{horpo} t = @(\overline{t})$ by case 7, then for every $t_i :_C \rho \in \overline{t}$, either $s \succ_{horpo} t_i$, and $s\gamma \succ_{horpo} t_i\gamma$ by induction hypothesis, or $s_j \succeq_{horpo} t_i$ for some $s_j :_C \rho' \in \overline{s}$, and $s_j\gamma \succ_{horpo} t_i\gamma$ by induction hypothesis. Therefore, since $@(\overline{t\gamma})$ is a partial left-flattening of $t\gamma$, $s\gamma \succ_{horpo} t\gamma$ by case 7.
8. If $s \succ_{horpo} t$ by Case 8, then $t = \lambda x.v$ with $x \notin \mathcal{V}ar(v)$ and $s \succ_{horpo} v$. By induction hypothesis, $s\gamma \succ_{horpo} v\gamma$ for evey substitution γ of domain $\mathcal{V}ar(v) \setminus \{x\}$ such that $x \notin \mathcal{V}ar(v\gamma)$, hence $s\gamma \succ_{horpo} t\gamma = \lambda x.v\gamma$ by Case 8.
9. If $s = @(s_1, s_2) \succ_{horpo} t = @(\overline{t})$ by case 10, then the proof goes as in case 2, allowing us to conclude by case 10.
10. If $s \succ_{horpo} t$ by case 11, then $s = \lambda x.u$ and $t = \lambda x.v$ and $u \succ_{horpo} v$. Therefore, by induction hypothesis $u\gamma \succ_{horpo} v\gamma$. Assuming that $x \notin \mathcal{D}om(\gamma)$, then $s\gamma = \lambda x.u\gamma \succ_{horpo} \lambda x.v\gamma = t\gamma$ by case 11.

11. If $s = \succ_{horpo} t$ by case 12, then the property holds by stability of β-reduction. □

Lemma 3.9 *$\succ_{horpo}$ is compatible.*

The proof is left to the reader. Now we prove that $\succ_{horpo}$ is polymorphic.

Lemma 3.10 (Polymorphism) *$\succ_{horpo}$ is polymorphic.*

Proof: Here, it is enough to consider terms instead of candidate terms. Let $\mathcal{F}$ be a polymorphic signature, $s : \sigma$ and $t : \tau$ be two terms such that $\Gamma \vdash_{\mathcal{F}} s : \sigma \succ_{horpo} t : \tau$. We need to show that $\Gamma\xi \vdash_{\mathcal{F}} s : \sigma\xi \succ_{horpo} t : \sigma\xi$ for all type substitutions ξ, and proceed by induction on $(\longrightarrow_\beta \cup \rhd, \rhd)_{lex}$.

First, by stability under substitutions of the type ordering, $\sigma \geq_{\mathcal{TS}} \tau$ implies $\sigma\xi \geq_{\mathcal{TS}} \tau\xi$. Now we consider several cases according to the definition. As previously, we omit the environment, showing that $s : \sigma \succ_{horpo} t : \tau$ implies $s : \sigma\xi \succ_{horpo} t : \tau\xi$.

1. If $s : \sigma \succ_{horpo} t : \tau$ by case 1, then $s = f(\overline{s})$ with $f \in \mathcal{F}$ and $u : \rho \succeq_{horpo} t : \tau$, for some $u \in \overline{s}$ and type ρ. Then by induction hypothesis $u : \rho\xi \succeq_{horpo} t : \tau\xi$, and therefore, $s : \sigma\xi \succ_{horpo} t : \tau\xi$ by case 1.

2. If $s : \sigma \succ_{horpo} t : \tau$ by case 2, then $s = f(\overline{s})$, $t = g(\overline{t})$, $f >_{\mathcal{F}} g$ and for all $t_i : \rho \in \overline{t}$ either $s : \sigma \succ_{horpo} t_i : \rho$ or $s_j : \rho' \succeq_{horpo} t_i : \rho$ for some $s_j : \rho' \in \overline{s}$. By induction hypothesis, for all $t_i : \rho\xi \in \overline{t}$ either $s : \sigma\xi \succ_{horpo} t_i : \rho\xi$ or $s_j : \rho'\xi \succeq_{horpo} t_i : \rho\xi$ for some $s_j : \rho'\xi \in \overline{s}$. Therefore, $s : \sigma\xi = f(\overline{s}) \succ_{horpo} g(\overline{t}) = t : \tau\xi$ by case 2.

3. If $s : \sigma \succ_{horpo} t : \tau$ by case 3, then $s = f(\overline{s})$, $t = g(\overline{t})$, $f =_{\mathcal{F}} g$, $f, g \in Mul$ and $\overline{s} : \overline{\sigma}(\succ_{horpo})_{mul}\overline{t} : \overline{\tau}$. By induction hypothesis $\overline{s} : \overline{\sigma}\xi(\succ_{horpo})_{mul}\overline{t} : \overline{\tau}\xi$, and hence $s : \sigma\xi \succ_{horpo} t : \tau\xi$ by case 3.

4. If $s : \sigma \succ_{horpo} t : \tau$ by case 4, then $s = f(\overline{s})$, $t = g(\overline{t})$, $f =_{\mathcal{F}} g$, $f, g \in Lex$, $\overline{s} : \overline{\sigma}(\succ_{horpo})_{lex}\overline{t} : \overline{\tau}$, and for all $t_i : \rho \in \overline{t}$ either $s : \sigma \succ_{horpo} t_i : \rho$ or $s_j : \rho' \succeq_{horpo} t_i : \rho$ for some $s_j : \rho' \in \overline{s}$. By induction hypothesis $\overline{s} : \overline{\sigma}\xi(\succ_{horpo})_{lex}\overline{t} : \overline{\tau}\xi$, and as in the precedence case, by induction hypothesis, for all $t_i : \rho\xi \in \overline{t}$ either $s : \sigma\xi \succ_{horpo} t_i : \rho\xi$ or $s_j : \rho'\xi \succeq_{horpo} t_i : \rho\xi$ for some $s_j : \rho'\xi \in \overline{s}$. Therefore $s : \sigma\xi \succ_{horpo} t : \tau\xi$ by case 4.

5. If $s : \sigma \succ_{horpo} t : \tau$ by case 5, the reasonning is similar as in case 1.

6. If $s : \sigma \succ_{horpo} t : \tau$ by case 6, then $s = \lambda x.u$ and $u : \rho \succeq_{horpo} t : \tau$. By induction hypothesis, $u : \rho\xi \succeq_{horpo} t : \tau\xi$, hence $s : \sigma\xi = \lambda x.u : \sigma\xi \succ_{horpo} t : \tau\xi$ by case 6.

7. If $s : \sigma \succ_{horpo} t = @(\overline{t}) : \tau$ by case 7, then for every $t_i : \rho \in \overline{t}$, either $s : \sigma \succ_{horpo} t_i : \rho$, and $s : \sigma\xi \succ_{horpo} t_i : \rho\xi$ by induction hypothesis, or $s_j : \rho' \succeq_{horpo} t_i : \rho$ for some $s_j : \rho' \in \overline{s}$, and $s_j : \rho'\xi \succ_{horpo} t_i : \rho\xi$ by induction hypothesis. Therefore, since $@(\overline{t}) : \tau\xi$ is a partial left-flattening of $t : \tau\xi$, $s : \sigma\xi \succ_{horpo} t : \tau\xi$ by case 7.

8. If $s : \sigma \succ_{horpo} t : \tau$ by Case 8, then $t = \lambda x.v$ with $x \notin \mathcal{V}ar(v)$ and $s : \sigma \succ_{horpo} v : \rho$. By induction hypothesis, $s : \sigma\xi \succ_{horpo} v : \rho\xi$, and hence $s : \sigma\xi \succ_{horpo} t : \tau\xi = \lambda x.v : \tau\xi$ by Case 8.

9. If $s = @(s_1, s_2) : \sigma \succ_{horpo} t = @(\overline{t}) : \tau$ by case 10, then by induction we can conclude again by case 10.

10. If $s : \sigma \succ_{horpo} t : \tau$ by case 11, then $s = \lambda x.u$ and $t = \lambda x.v$ and $u : \rho \succ_{horpo} v : \rho'$. Therefore, by induction hypothesis $u : \rho\xi \succ_{horpo} v : \rho'\xi$. Then $s : \sigma\xi = \lambda x.u\gamma \succ_{horpo} \lambda x.v = t : \tau\xi$ by case 11.

11. If $s = \succ_{horpo} t$ by case 12, then the property holds by polymorphism of β-reduction. □

Although the above proofs are slightly more difficult technically than the usual proofs for the recursive path ordering, they follow the same kind of pattern (polymorphism was actually never considered before). This contrasts with the proof of strong normalization to come.

3.3 Strong Normalization

For the recursive path ordering, strong normalization follows from the fact that it contains the embedding relation which is a well-order by Kruskal's tree theorem. Since we do not know of any non-trivial extension of Kruskal's tree theorem for higher-order terms that includes β-reductions (and wasted much of our time in looking for an appropriate one), we will adopt a completely different method, the computability predicate proof method due to Tait and Girard. This proof method will also suggest an important improvement of our ordering, discussed in Section 4.

Note that, as a bi-product, we obtain a proof of well-foundedness for the recursive path ordering on first-order terms which does not rely on Kruskal's theorem. It should be pointed out here that, in the first-order case, a very simple part of Tait's method is used, since there is no need of the computability predicate. What is used is the technique based on the fact that, given an arbitrary strongly normalizing substitution γ, one can then build an induction argument to prove that, for an arbitrary term t, the term $t\gamma$ is strongly normalizing. The identity substitution being trivially strongly normalizable, we then conclude for t. This proof technique had previously been used in a first order context in [15].

Since there is no type application in our calculus, our (weak) notion of polymorphism does not interfer with rewriting, hence with strong normalization. It is therefore sufficient to prove the strong normalization property of the calculus for a monomorphic signature (obtained by decorating each monomorphic instance of a function symbol by its type declaration, yielding a possibly infinite signature) which we start doing now.

We give no more than the necessary details for the understanding of the reader, assuming some familiarity with the computability (also called reducibility) candidates method of Tait and Girard [16].

3.3.1 Candidate interpretation of ground types

Again, this section refers to candidate terms, rather than to candidate judgements. Our definition of computability for candidate terms of a ground type is standard, but we make sure that computability is compatible with the equivalence $=_{\mathcal{T}_S}$ on types. Technically, we denote by $[\![\sigma]\!]$ the computability predicate of the type σ, which, by construction, will be equal to the predicate $[\![\alpha]\!]$ for any type $\alpha =_{\mathcal{T}_S} \sigma$.

Without loss of generality, we assume a global environment for variables.

Definition 3.11 *The family of* candidate interpretations $\{[\![\sigma]\!]\}_{\sigma \in \mathcal{T}_S}$ *is the family of subsets of the set of candidates whose elements are the least sets satisfying the following properties:*

(i) If σ is a data type, then $s :_C \sigma \in [\![\sigma]\!]$ iff $t \in [\![\tau]\!]$ $\forall t :_C \tau$ such that $s \succ_{horpo} t$

(ii) If $s :_C \sigma = \tau \to \rho$ then $s \in [\![\sigma]\!]$ if $@(s,t) \in [\![\rho]\!]$ for every $t \in [\![\tau]\!]$;

A candidate term s of ground type σ is said to be computable *if $s \in [\![\sigma]\!]$. A vector $\overline{s}$ of terms is computable iff so are all its components.*

The above definition is based on a lexicographic combination of an induction on the well-founded type ordering $>^{\rightarrow}_{\mathcal{T}_S}$, and a fixpoint computation for data types. The existence of a least fixpoint is ensured by the monotonicity of the underlying family (indexed by the set of data types) of functionals with

respect to set inclusion. By an "induction on the definition of the candidate interpretations", we mean an outer induction on the type ordering $>_{\mathcal{T}_S}^{\rightarrow}$ followed by an inner induction on the fixpoint computation. The following important property is a straightforward consequence of the definition, Lemma 3.6 and functional preservation:

Lemma 3.12 *Assume $\sigma =_{\mathcal{T}_S} \tau$. Then $[\![\sigma]\!] = [\![\tau]\!]$.*

The next lemma follows easily as well, by induction on n:

Lemma 3.13 *Let $\sigma = \sigma_1 \to \ldots \to \sigma_n \to \tau$, where $n > 0$. Then $s \in [\![\sigma]\!]$ iff $@(s, t_1, \ldots, t_n) \in [\![\tau]\!]$ for all $t_1 \in [\![\sigma_1]\!], \ldots, t_n \in [\![\sigma_n]\!]$.*

In the sequel, we will always assume that functional types are in canonical form and that $n > 0$ in $\sigma = \sigma_1 \to \ldots \to \sigma_n \to \tau$.

We first recall the properties of the interpretations.

Property 3.14 (Computability Properties)
(i) Every computable term is strongly normalizable;
(ii) Assuming that s is computable and $s \succeq_{horpo} t$, then t is computable;
(iii) A neutral term s is computable iff t is computable for every t such that $s \succ_{horpo} t$;
(iv) If $\overline{t}$ be a vector of computable terms such that $@(\overline{t})$ is a candidate term, then $@(\overline{t})$ is computable;
(v) $\lambda x : \sigma.u$ is computable iff $u\{x \mapsto w\}$ is computable for every computable term $w :_C \sigma$;
(vi) Let $s :_C \sigma \in \mathcal{T}_{\mathcal{S}}^{min}$. Then s is computable iff it is strongly normalizable.

Note that variables are not asumed to be computable. Variables of a data type are of course computable by definition since they have no reduct. But variables of a functional type will be computable by Property 3.14 (iii). This will actually forbid us to prove the computability properties (i), (ii) and (iii) separately. A possible alternative would be modify the definition of the computability predicates by adding the property that variables of a functional type are computable. This would make the properties (i), (ii) and (iii) independant to the price of other complications.

Proof:

- Property (iv). Straigtforward induction on the length of $\overline{t}$.
- Properties (i), (ii), (iii).

  Note first that the only if part of property (iii) is property (ii). We are left with (i), (ii) and the if part of (iii) which we now spell out as follows:

  Given a type σ, a term $s :_C \sigma \in [\![\sigma]\!]$, a term $t :_C \tau$ such that $s \succ_{horpo} t$, and a neutral term $u :_C \sigma$ such that $w :_C \theta \in [\![\theta]\!]$ for every w such that $u \succ_{horpo} w$, we prove by induction on the definition of $[\![\sigma]\!]$ that (i) s is strongly normalizable, (ii) t is computable, and (iii) u is computable.

  1. Assume first that σ is a data type.
     (i) All reducts of s are computable by definition of the interpretations, hence strongly normalizable by induction hypothesis, and therefore, so is s.
     (ii) By definition of the candidate interpretations.
     (iii) By definition of the candidate interpretations.
  2. Assume that σ is a functional type.

(i) Let $s :_C \sigma = \theta_0 \succ_{horpo} s_1 :_C \theta_1 \ldots \succ_{horpo} s_n :_C \theta_n \succ_{horpo} \ldots$ be a derivation issuing from s. Note that $s_n \in [\![\theta_n]\!]$ by assumption for $n = 0$ and repeated applications of induction property (ii) otherwise. Such derivations are of the following two kinds:

(a) $\sigma >_{\mathcal{T}_S} \theta_n$ for some n, in which case s_n is strongly normalizable by induction hypothesis, hence the derivation issuing from s is finite;

(b) $\theta_n =_{\mathcal{T}_S} \sigma$ for all n, in which case, the sequence of terms $@(s_n, y :_C \sigma_1) :_C \sigma_2 \to \ldots \sigma_n \to \tau$ is well-typed, and strictly decreasing by monotonicity Lemma 3.7. Since $\sigma >^{\to}_{\mathcal{T}_S} \sigma_1$, $y :_C \sigma_1$ is computable by induction hypothesis (iii), hence, by definition, $@(s_n, y)$ is computable. By induction hypothesis (i), the above sequence is again finite.

(ii) Let $\sigma = \theta \to \rho$. By arrow preservation and arrow decreasing properties, there are two cases:

(a) $\rho \geq_{\mathcal{T}_S} \tau$. Since s is computable, $@(s, u)$ is computable for every $u \in [\![\theta]\!]$. Let $y :_C \theta$. By induction hypothesis (iii), $y \in [\![\theta]\!]$, hence $@(u, y)$ is computable. Since $@(s, y) :_C \rho \succ_{horpo} t :_C \tau$ by case 5 of the definition, we conclude by induction hypothesis (ii) that t is computable.

(b) $\tau = \theta' \to \rho'$, with $\theta =_{\mathcal{T}_S} \theta'$ and $\rho \geq_{\mathcal{T}_S} \rho'$. Since s is computable, given $u \in [\![\theta]\!]$, then $@(s, u) \in [\![\rho]\!]$, hence, by induction hypothesis (ii) $@(t, u) \in [\![\rho']\!]$. Since $[\![\theta]\!] = [\![\theta']\!]$ by Lemma 3.12, $t \in [\![\tau]\!]$ by definition of the interpretations.

(iii) Let $\sigma = \sigma_1 \to \ldots \to \sigma_n \to \tau$, where $n > 0$ and τ is a data type. By Lemma 3.13, t is computable iff $@(t, u_1, \ldots, u_n)$ is computable for arbitrary terms $u_1 \in [\![\sigma_1]\!], \ldots, u_n \in [\![\sigma_n]\!]$ which are strongly nor malizable by Property 3.14 (i). By definition, as τ is a data type, $@(t, u_1, \ldots, u_n)$ is computable iff so are all its reducts.

We prove by induction on the multiset of computable terms $\{u_1, \ldots, u_n\}$ ordered by $(\succeq_{horpo})_{mul}$ the property (H) stating that terms w strictly smaller than $@(t, u_1, \ldots, u_i)$ in $\succ_{horpo}$ are computable. Taking $i = n$ yields the desired property. If $i = 0$, terms strictly smaller than t are computable by assumption. For the general case $(i + 1) \leq n$, we need to consider all terms w strictly smaller than $@(@(t, u_1, \ldots, u_i), u_{i+1})$. Since t is neutral, hence is not an abstraction, there are two possible cases:

(a) $@(@(t, u_1, \ldots, u_i), u_{i+1}) \succ_{horpo} w$ by Case 5. There are again two possibilities:

– $@(t, u_1, \ldots, u_i) \succeq_{horpo} w$, and therefore $@(t, u_1, \ldots, u_i) \succ_{horpo} w$ for type reason since w is also a reduct of $@(t, u_1, \ldots, u_{i+1})$. We then conclude by induction hypothesis (H).

– $u_{i+1} \succeq_{horpo} w$. We conclude by assumption and induction property (ii).

(b) $@(@(t, u_1, \ldots, u_i), u_{i+1}) \succ_{horpo} w$ by Case 10, hence $w = @(\overline{w})$. By definition of the multiset extension and for type reasons, there are the following two possibilities:

– for all $v \in \overline{w}$, either $@(t, u_1, \ldots, u_i) \succ_{horpo} v$, and v is computable by induction hypothesis (H), or $u_{i+1} \succeq_{horpo} v$, in which case v is computable by assumption and induction property (ii). It follows that w is computable by Property 3.14 (iv).

– $w_1 = @(t, u_1, \ldots, u_i)$ and $u_{i+1} \succ_{horpo} w_2$, implying that w_2 is computable by assumption and induction property (ii). By induction property (H), all reducts of w are computable. Since w and t have the same (data) type, w is therefore computable by induction property (iii).

As a consequence, all reducts of $@(t, u_1, \ldots, u_n)$ are computable and we are done. □

- Property (v).

The only if part is property (ii) together with the definition of computability. For the if part, assuming that $u\{x \mapsto s\}$ is computable for an arbitrary computable s, we prove that $@(\lambda x.u, w)$ is computable for an arbitrary computable w.

Since variables are computable by property (iii), $u = u\{x \mapsto x\}$ is computable by assumption. By property (i), u and w are strongly normalizable. Since $\lambda x.u \succeq_{horpo} v$ implies $u \succeq_{horpo} v$ or $v = \lambda x.u'$ and $u \succeq_{horpo} u'$, an easy induction on u shows that $\lambda x.u$ is strongly normalizable as well. We will prove that $@(\lambda x.u, w)$ is computable by induction on the pair $\{\lambda x.u, w\}$ ordered by $(\succ_{horpo})_{lex}$. By property (iii), the neutral term $@(\lambda x.u, w)$ is computable iff v is computable for all v such that $@(\lambda x.u, w) \succ_{horpo} v$. There are several cases to be considered.

1. If the comparison is by case 5, there are two cases:
   - if $w \succeq_{horpo} v$, we conclude by property (ii).
   - if $\lambda x.u \succ_{horpo} v$, there are two cases. If $u \succeq_{horpo} v$ by case 6, we conclude by property (ii) again. Otherwise, $v = \lambda x.u'$ and $u \succ_{horpo} u'$, hence $u\{x \mapsto w\} \succ_{horpo} u'\{x \mapsto w\}$ by Lemma 3.8. By assumption and property (ii), $u'\{x \mapsto w\}$ is therefore computable. Hence, $@(v, w)$ is computable by induction hypothesis applied to the pair $(v = \lambda x.u', w)$. We then conclude by definition of the interpretations that v is computable.
2. If the comparison is by case 10, then $v = @(\overline{v})$ and all terms in $\{\overline{v}\}$ are smaller than w or $\lambda x.u$. There are two cases:
   - $v_1 = \lambda x.u$ and $w \succ_{horpo} v_i$ for $i > 1$. Then v_i is computable by property (ii) and, since $u\{x \mapsto v_2\}$ is computable by the main assumption, $@(v_1, v_2)$ is computable by induction hypothesis. If $v = @(v_1, v_2)$, we are done, and we conclude by Lemma 3.13 otherwise.
   - For all other cases, terms in $\overline{v}$ are reducts of $\lambda x.u$ and w. Reducts of w and reducts of $\lambda x.u$ which are themselves reducts of u are computable by property (ii). If all terms in $\overline{v}$ are such reducts, v is computable by Lemma 3.13.

   Otherwise, for typing reason, v_1 is a reduct of $\lambda x.u$ of the form $\lambda x.u'$ with $u \succ_{horpo} u'$, and all other terms in $\overline{v}$ are reducts of the previous kind. By the main assumption, $u\{x \mapsto v''\}$ is computable. Besides, $u\{x \mapsto v''\} \succ_{horpo} u'\{x \mapsto v''\}$ by stability property of the ordering. Therefore $u'\{x \mapsto v''\}$ is computable by Property (ii). By induction hypothesis, $@(v_1, v_2)$ is again computable. If $v = @(v_1, v_2)$, we are done, otherwise v is computable by Lemma 3.13.
3. Otherwise, $@(\lambda x.u, w) \succ_{horpo} v$ by case 12, then $u\{x \mapsto w\} \succeq_{horpo} v$. By assumption, $u\{x \mapsto w\}$ is computable, and hence v is computable by property (ii).

- Property (vi).

  The only if direction is property (i). For the if direction, let s be a strongly normalizable term of type $\sigma \in \mathcal{T}_{\mathcal{S}}^{min}$. We prove that s is computable by induction on $\succ_{horpo}$. Since σ is a data type, s must be neutral. Let now $s \succ_{horpo} t :_C \tau$, hence $\sigma \geq_{\mathcal{T}_{\mathcal{S}}} \tau$. By definition of $\mathcal{T}_{\mathcal{S}}^{min}$, $\tau =_{\mathcal{T}_{\mathcal{S}}} \sigma$, hence, by Lemma 3.1, τ is a data type, and since σ is minimal, so is τ, hence $\tau \in \mathcal{T}_{\mathcal{S}}^{min}$. By assumption on s, t must be strongly normalizable, and by induction hypothesis, it is therefore computable. Since this is true of all reducts of s, by definition s is computable. □

The following lemma and proof are both essential:

Lemma 3.15 *Let $f : \overline{\sigma} \to \tau \in \mathcal{F}$ and $\overline{s} :_C \overline{\sigma}$ be a vector of computable terms. Then $f(\overline{s})$ is computable.*

Proof: Since terms in $\overline{s}$ are computable, by Property 3.14 (i), they are strongly normalizable. We use this remark to build our induction argument: we prove that $f(\overline{s})$ is computable by induction on the pair $(f, \overline{s})$ ordered lexicographically by $(>_{\mathcal{F}}, (\succ_{horpo})_{stat_f})_{lex}$.

Since $f(\overline{s})$ is neutral, by Property 3.14 (iii), it is computable iff every t such that $f(\overline{s}) \succ_{horpo} t$ is computable, which we prove by an inner induction on the size of t. We discuss according to the possible cases of the definition of $\succ_{horpo}$.

1. Let $f(\overline{s}) \succ_{horpo} t$ by case 1, hence $s_i \succeq_{horpo} t$ for some $s_i \in \overline{s}$. Since s_i is computable, t is computable by Property 3.14 (ii).

2. Let $s = f(\overline{s}) \succ_{horpo} t$ by case 2. Then $t = g(\overline{t})$, $f >_{\mathcal{F}} g$ and for every $v \in \overline{t}$ either $s \succ_{horpo} v$, in which case v is computable by the inner induction hypothesis, or $u \succeq_{horpo} v$ for some $u \in \overline{s}$ and v is computable by Property 3.14 (ii). Therefore, $\overline{t}$ is computable, and since $f >_{\mathcal{F}} g$, t is computable by the outer induction hypothesis.

3. If $f(\overline{s}) \succ_{horpo} t$ by case 3, then $t = g(\overline{t})$, $f =_{\mathcal{F}} g$, and $\overline{s}(\succ_{horpo})_{mul}\overline{t}$. By definition of the multiset comparison, for every $t_i \in \overline{t}$ there is some $s_j \in \overline{s}$, s.t. $s_j \succeq_{horpo} t_i$, hence, by Property 3.14 (ii), t_i is computable. This allows us to conclude by the outer induction hypothesis that t is computable.

4. If $f(\overline{s}) \succ_{horpo} t$ by case 4, then $t = g(\overline{t})$, $f =_{\mathcal{F}} g$, $\overline{s}(\succ_{horpo})_{lex}\overline{t}$ and for every $v \in \overline{t}$ either $f(\overline{s}) \succ_{horpo} v$ or $u \succeq_{horpo} v$ for some $u \in \overline{s}$. As in the precedence case, this implies that $\overline{t}$ is computable. Then, since $\overline{s}(\succ_{horpo})_{lex}\overline{t}$, t is computable by the outer induction hypothesis.

5. If $f(\overline{s}) \succ_{horpo} t$ by case 7, let $@(t_1, \ldots, t_n)$ be the partial left-flattening of t used in that proof. By the same token as in case 2, every term in $\overline{t}$ is computable, hence t is computable by Property 3.14 (iv).

6. If $f(\overline{s}) \succ_{horpo} t$ by case 8, then $t = \lambda x.u$ with $x \notin \mathcal{V}ar(u)$, and $f(\overline{s}) \succ_{horpo} u$. By the inner induction hypothesis, u is computable. Hence, $u\{x \mapsto w\} = u$ is computable for any computable w, and therefore, $t = \lambda x.u$ is computable by Property 3.14 (v). □

3.3.2 Strong normalization proof

Lemma 3.16 *Let γ be a computable substitution and t be an algebraic λ-term. Then $t\gamma$ is computable.*

Proof: The proof proceeds by induction on the size of t.

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

2. t is an abstraction $\lambda x.u$. By Property 3.14 (v), $t\gamma$ is computable if $u\gamma\{x \mapsto w\}$ is computable for every well-typed computable candidate term w. Taking $\delta = \gamma \cup \{x \mapsto w\}$, we have $u\gamma\{x \mapsto w\} = u(\gamma \cup \{x \mapsto w\})$ since x may not occur in γ. Since δ is computable and $|t| > |u|$, by induction hypothesis, $u\delta$ is computable.

3. $t = @(t_1, t_2)$. Then $t_1\gamma$ and $t_2\gamma$ are computable by induction hypothesis, hence t is computable by Property 3.14 (iv).

4. $t = f(t_1, \ldots, t_n)$. Then $t_i\gamma$ is computable by induction hypothesis, hence $t\gamma$ is computable by Lemma 3.15. □

We can now easily conclude the proof of well-foundedness needed for our main theorem, by showing that every term is strongly normalizable with respect to $\succ_{horpo}$.

Proof: Given an arbitrary term t, let γ be the identity substitution. Since γ is computable, $t = t\gamma$ is computable by Lemma 3.16, and strongly normalizable by Property 3.14 (i). □

3.4 Examples

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

Example 6 Insertion Sort. Let $\mathcal{S} = \{\mathbb{N}, List\}$ and
$\mathcal{F} = \{nil :\rightarrow List; cons : \mathbb{N} \times List \rightarrow List; max, min : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}; insert : \mathbb{N} \times List \times (\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}) \times (\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}) \rightarrow List; sort : List \times (\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}) \times (\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}) \rightarrow List; ascending\_sort, descending\_sort : List \rightarrow List\}$.

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

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

$$\begin{array}{rcl} insert(n,nil,X,Y) & \rightarrow & cons(n,nil) \\ insert(n,cons(m,l),X,Y) & \rightarrow & \\ \multicolumn{3}{c}{cons(X(n,m),insert(Y(n,m),l,X,Y))} \end{array}$$

The first $insert$ rule is easily taken care of by applying case 2 with the precedence $insert >_{\mathcal{F}} cons$, and then case 1. For the second $insert$ rule, we apply first case 2, and we recursively need to show: firstly, that $insert(n,cons(m,l),X,Y) \succ_{horpo} @(X,n,m)$, which follows by applying rule 2, and then case 1 recursively; and secondly that $insert(n,cons(m,l),X,Y) \succ_{horpo} insert(Y(n,m),l,X,Y)$, which succeeds as well by case 4, with a right-to-left lexicographic status for $insert$, and calling recursively with $insert(n,cons(m,l),X,Y) \succ_{horpo} @(Y,n,m)$, which is solved by case 10.

$$\begin{array}{rcl} sort(nil,X,Y) & \rightarrow & nil \\ sort(cons(n,l),X,Y) & \rightarrow & \\ \multicolumn{3}{c}{insert(n,sort(l,X,Y),X,Y)} \end{array}$$

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

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

$$\begin{array}{rcl} ascending\_sort(l) & \rightarrow & \\ \multicolumn{3}{c}{sort(l,\lambda xy.min(x,y),\lambda xy.max(x,y))} \\ descending\_sort(l) & \rightarrow & \\ \multicolumn{3}{c}{sort(l,\lambda xy.max(x,y),\lambda xy.min(x,y))} \end{array}$$

This is so, because the term $\lambda xy.min(x,y)$ occuring in the lefthand side has type $\mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$, which is not comparable to any lefthand side type. We will come back to this example in Section 7. □

We now come to a more tricky example, where we will need the transitive closure of the ordering to show termination of a rule, that is, we will need to invent a *middle term* s such that $l \succ_{horpo} s \succ_{horpo} r$ for some rule $l \rightarrow r$.

Example 7 (Surjective Disjoint Union, taken from [10]) Let $\mathcal{S} = \{A, B, U\}$, $\alpha \in \{A, B, U\}$, $\mathcal{F} = \{inl : A \to U; inr : B \to U; case_\alpha : U \times (A \to \alpha) \times (B \to \alpha) \to \alpha\}$. The rules are the following:

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

The first two rules are taken care of by Case 7. For the last, we show that
$case_\alpha(Z, \lambda x.H(inl(x)), \lambda y.H(inr(y))) \succ_{horpo} @(\lambda x.H(x), Z) \succ_{horpo} @(H, Z)$.
The first comparison is again done by Case 7, generating the proof obligation $\lambda x.H(inl(x)) \succ_{horpo} \lambda x.H(x)$, which suceeds by applying successively Cases 11 and 1. The second comparison is by Case 12. □

3.5 The ordering on types

Given a partition $Mul \uplus Lex$ of $\mathcal{S}$ called a *status*, and a partial quasi-ordering $>_\mathcal{S}$ on sort constructors called a *precedence* such that equal constructors of Lex status have the same arity, we define the following rpo-like quasi ordering on types:

$$\text{Let } A = \forall v \in \overline{\tau}\ \sigma \underset{horpo}{\succ} v$$

Definition 3.17 *$\sigma \geq_{\mathcal{T}_\mathcal{S}} \tau$ iff*

1. *$\sigma = c(\overline{\sigma})$ and $\sigma_i \geq_{\mathcal{T}_\mathcal{S}} \tau$ for some $\sigma_i \in \overline{\sigma}$*
2. *$\sigma = \alpha \to \beta$, and $\beta \geq_{\mathcal{T}_\mathcal{S}} \tau$*
3. *$\sigma = c(\overline{\sigma})$ and $\tau = d(\overline{\tau})$ with $c >_\mathcal{S} d$, and A*
4. *$c = d \in Mul$ and $\overline{\sigma}(\geq_{\mathcal{T}_\mathcal{S}})_{mul}\overline{\tau}$*
5. *$c = d \in Lex$ and $\overline{\sigma}(\geq_{\mathcal{T}_\mathcal{S}})_{lex}\overline{\tau}$, and A*
6. *$\sigma = \alpha \to \beta$, $\tau = \alpha' \to \beta'$ and $\alpha =_{\mathcal{T}_\mathcal{S}} \alpha'$ and $\beta \geq_{\mathcal{T}_\mathcal{S}} \beta'$*

where $>_{\mathcal{T}_\mathcal{S}}$ and $=_{\mathcal{T}_\mathcal{S}}$ are respectively the strict ordering and the equivalence associated with $\geq_{\mathcal{T}_\mathcal{S}}$.

We will also write $\geq_{rpo}$ for the full recursive path ordering on types generated by the given precedence and statuses on $\mathcal{S}$.

Note that, because of the subtem property for sort symbols, $\forall v \in \overline{\tau}\ \sigma \succ_{horpo} v$ or $u \succeq_{horpo} v$ for some $u \in \overline{\sigma}$ and A are equivalent propositions. We will later use this remark.

Definition 3.18 *A function symbol $f : \sigma_1 \times \ldots \times \sigma_n \to \sigma \in \mathcal{F}$ is* compatible *if $\forall \xi \in \mathcal{V}ar(\sigma)$, $\exists \tau \in \{\sigma_1, \ldots, \sigma_n\}$ such that $\tau|_p = \xi$ for some $p \in \mathcal{D}om(\tau)$, and $\forall q < p$, $\tau(q) \in Lex$.*

Compatibility can of course always be ensured by giving the lexicographic status to all sort symbols.

Proposition 3.19 *$\geq_{\mathcal{T}_\mathcal{S}}$ is a type ordering satisfying containement (with respect to $\geq_{rpo}$), well-foundedness and arrow-subterm property of $\geq_{rpo}$, functional preservation, stability under type substitution, and signature compatibility iff all function symbols are compatible.*

Proof: The first three properties follow from the fact that $\geq_{\mathcal{T}_\mathcal{S}} \cup \rhd_{\rightarrow}$ is included in the recursive path ordering generated by the precedence on the type constructors. Functionnal preservation is easy to verify. For stability, the if part is easy. For the converse, it is enough to show a counterexample to signature compatibility when a function symbol is not compatible. This is left to the reader. □

All examples given so far can use the above type ordering, provided the appropriate precedence on type constructors is given by the user.

3.6 A uniform ordering on terms and their types

Assuming we chose the above ordering as our type ordering, we are now going to reformulate the entire ordering by taking advantage of the facts that types are indeed first-order terms, that the above type ordering is a restriction of Dershowitz's recursive path ordering, and that the higher-order recursive path ordering restricts to the recursive path ordering for the case of first-order terms. This will enable us to have a single definition for ordering terms and types.

For making uniformity possible, we add a new constant in our language, $*$, such that all types have themselves type $*$. We omit the straightforward type system for typing types, which only aims at verifying arities of sorts symbols. $*$ will therefore be the only non-typable term in the language.

3.6.1 Statuses and Precedence

We assume given
- a partition $Mul \uplus Lex$ of $\mathcal{F} \cup \mathcal{S} \cup \{@\}$ such that $@ \in Mul$ and all function symbols are compatible.
- a well-founded quasi-ordering $\geq_{\mathcal{FS}}$ on $\mathcal{F} \cup \mathcal{S}$, called the *precedence*, such that

1. if $f : \overline{\sigma} \rightarrow \sigma =_{\mathcal{FS}} g : \overline{\tau} \rightarrow \tau$, then $\sigma =_{\mathcal{T}_\mathcal{S}} \tau$ and $f \in Lex$ iff $g \in Lex$ and $\overline{\sigma}(=_{\mathcal{FS}})_{mon}\overline{\tau}$;
2. variables are considered as constants incomparable in $>_{\mathcal{FS}}$ among themselves and with other function symbols.

3.6.2 Definition of the ordering

To ease the reading (as well as the writing!), we omit environments in this new definition.

Definition 3.20

$$\textit{Given } s : \sigma \textit{ and } t : \tau, \quad s \underset{horpo}{\succ} t \textit{ iff } \sigma = \tau = * \textit{ or } \sigma \underset{horpo}{\succeq} \tau \textit{ and}$$

1. $s = f(\overline{s})$ *with* $f \in \mathcal{FS}$, *and* $u \underset{horpo}{\succeq} t$ *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* $\overline{s}(\underset{horpo}{\succ})_{mul}\overline{t}$
4. $s = f(\overline{s})$ *and* $t = g(\overline{t})$ *with* $f =_{\mathcal{FS}} g \in Lex$ *and* $\overline{s}(\underset{horpo}{\succ})_{lex}\overline{t}$, *and* A
5. $s = @(s_1, s_2)$ *and* $u \underset{horpo}{\succeq} t$ *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. $s = @(s_1, s_2)$, $@(\overline{t})$ *is an arbitrary left-flattening of* t *and* $\{s_1, s_2\}(\underset{horpo}{\succ})_{mul}\overline{t}$
10. $s = \lambda x : \alpha.u,\ t = \lambda x : \beta.v$, $\alpha{=}\beta$ *and* $u \underset{horpo}{\succ} v$
11. $s = @(\lambda x.u, v)$ *and* $u\{x \mapsto v\} \underset{horpo}{\succeq} t$
12. $s = \alpha \to \beta$, *and* $\beta \underset{horpo}{\succeq} t$
13. $s = \alpha \to \beta,\ t = \alpha' \to \beta'$, $\alpha \underset{horpo}{=} \alpha'$ *and* $\beta \underset{horpo}{\succ} \beta'$

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

This uniform formulation of the higher-order recursive path ordering opens the way to its generalization to dependent type calculi such as the calculus of constructions. Indeed, the formulation of the ordering for the calculus of constructions should by now be clear for the cognitive reader. Our conjecture is that such an ordering should be well-founded as well, possibly imposing some extra conditions on big variables as in [38].

In the rest of this paper, we will consider the previous, more general formulation of the higher-order recursive path ordering in which the ordering on types is defined separately from the ordering on terms.

4 Computational Closure

The ordering is quite sensitive to innocent variations of the language, like adding (higher-order) dummy arguments to righthand sides or η-expanding higher-order variables. We will solve these problems by improving our definition in the light of the strong normalization proof. In that proof, it was crucial to show the computability of the righthand side subterms from the assumed computability property of the lefthand side subterms. In our definition, we actually require that for each righthand side subterm v, there exists a lefthand side subterm u such that $u \succeq_{horpo} v$. Assuming u is computable, then v is computable by Property 3.14 (ii). But any computability preserving operation applied to the lefthand side subterms in order to construct a term of the appropriate type would do as well. For example, the higher-order variable X is computable if and only if $\lambda x.X(x)$ is computable. Therefore, both forms may coexist. This discussion is formalized in subsection 4.1 with the notion of a computational closure borrowed from [3], with a slightly enhanced formulation.

4.1 The Computational Closure

Definition 4.1 *Given a term* $t = f(\overline{t})$ *with* $f \in \mathcal{F}$, *we define its* computable closure $\mathcal{CC}(t)$ *as* $\mathcal{CC}(t, \emptyset)$, *where* $\mathcal{CC}(t, \mathcal{V})$, *with* $\mathcal{V} \cap \mathcal{V}ar(t) = \emptyset$, *is the smallest set of well-typed terms containing all variables in* $\mathcal{V}$, *all terms in* $\overline{t}$, *and closed under the following operations:*

1. *subterm of minimal type: let* $s \in \mathcal{CC}(t, \mathcal{V})$, *and* $u : \sigma$ *be a subterm of* s *such that* $\sigma \in \mathcal{T}_{\mathcal{S}}^{min}$ *and* $\mathcal{V}ar(u) \subseteq \mathcal{V}ar(t)$; *then* $u \in \mathcal{CC}(t, \mathcal{V})$;

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

3. *recursive call: let* $\overline{s}$ *be a sequence of terms in* $\mathcal{CC}(t, \mathcal{V})$ *such that the term* $f(\overline{s})$ *is well typed and* $\overline{t}(\succ_{horpo} \cup \rhd_{>_{\mathcal{TS}}})_{stat_f} \overline{s}$*; then* $g(\overline{s}) \in \mathcal{CC}(t, \mathcal{V})$ *for every* $g =_{\mathcal{F}} f$*;*

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

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

6. *reduction: let* $u \in \mathcal{CC}(t, \mathcal{V})$*, and* $u \succeq_{horpo} v$*; then* $v \in \mathcal{CC}(t, \mathcal{V})$*;*

7. *weakening: let* $x \notin \mathcal{V}ar(u, t) \cup \mathcal{V}$*. Then,* $u \in \mathcal{CC}(t, \mathcal{V} \cup \{x\})$ *iff* $u \in \mathcal{CC}(t, \mathcal{V})$*.*

As a simple illustration of the use of the definition, let us show how to derive the following extensionality property: assuming $x \notin \mathcal{V}ar(u) \cup \mathcal{V}$, then $\lambda x.@(u, x) \in \mathcal{CC}(t, \mathcal{V})$ iff $u \in \mathcal{CC}(t, \mathcal{V})$. Let us prove first the if direction. Assuming without loss of generality that $x \notin \mathcal{V}ar(t)$, by weakening, $u \in \mathcal{CC}(t, \mathcal{V} \cup \{x\})$. By basic case, $x \in \mathcal{CC}(t, \mathcal{V} \cup \{x\})$, hence, by application, $@(u, x) \in \mathcal{CC}(t, \mathcal{V} \cup \{x\})$, and by abstraction, we get $\lambda x.@(u, x) \in \mathcal{CC}(t, \mathcal{V})$. For the converse, starting from $\lambda x.@(u, x) \in \mathcal{CC}(t, \mathcal{V})$, we get $\lambda x.u \in \mathcal{CC}(t, \mathcal{V})$ by reduction because $@(u, x) \succeq_{horpo} u$ by Case 5 of definition 3.3 and hence $\lambda x.@(u, x) \succeq_{horpo} \lambda x.u$ by Case 11 of definition 3.3, and then $u \in \mathcal{CC}(t, \mathcal{V})$ by reduction again, because $\lambda x.u \succeq_{horpo} u$ by Case 6 of definition 3.3.

Similarly, we can easily show that the abstraction rule is indeed an equivalence, by using weakening, application and reduction.

Note that we use $\succ_{horpo}$ instead of β-reduction as in [22] in Case 6. The derivation of the extensionality rule shows the usefulness of rule 6. On the other hand, it makes the membership of a term to the computational closure undecidable. But it becomes decidable if its use is bounded. In pratice, we can of course restrict the use of $\succeq_{horpo}$ by allowing a single step only, which is enough for all examples to come (it is enough for the extensionality property as well, by using an alternative proof). A similar remark applies to the recursive call case.

The rules we have given are meant to be used in a goal-oriented way. Using them in a forward chaining way makes essential use of weakening, but weakening is needed for backward chaining also, as seen from our coming examples. Note that the definition of the computational closure is easily extendable when new closure properties are established, since it is defined by a set of Horn clauses.

An important remark is that we use the previously defined ordering $\succeq_{horpo}$ in Case 6 and the relation $\succ_{horpo} \cup \rhd_{>_{\mathcal{TS}}}$ in Case 3 of the closure definition instead of simply β-reductions and $\beta \cup \rhd$-reductions respectively as in [22]. And indeed, we will consequently use $\succ_{horpo}$ and $\succ_{horpo} \cup \rhd_{>_{\mathcal{TS}}}$ as induction arguments in our proofs.

The following property of the computable closure can easily be shown by induction on the definition:

Lemma 4.2 *Assume that* $u \in \mathcal{CC}(t)$*. Then,* $u\gamma \in \mathcal{CC}(t\gamma)$ *for every substitution* γ*.*

Proof: We prove that if $u \in \mathcal{CC}(t, \mathcal{V})$ with $\mathcal{V} \subseteq \mathcal{X} \setminus (\mathcal{V}ar(t) \cup \mathcal{V}ar(t\gamma) \cup \mathcal{D}om(\gamma))$, then $u\gamma \in \mathcal{CC}(t\gamma, \mathcal{V})$. We proceed by induction on the definition of $\mathcal{CC}(t, \mathcal{V})$. Note that the property on $\mathcal{V}$ depends on t and γ, but not on u. It will therefore be trivially satisfied in all cases but abstraction and weakening. And indeed, this is the only cases in the proof which is not routine, hence we do them in detail as well as Case 1 to show its simplicity.

Case 1: let $s = f(\overline{s}) : \sigma \in \mathcal{CC}(t, \mathcal{V})$ with $f \in \mathcal{F}$ and $u : \tau$ be a subterm of s with $\sigma \in \mathcal{T}_{\mathcal{S}}^{min}$ and $\mathcal{V}ar(u) \subseteq \mathcal{V}ar(t)$. By induction hypothesis, $s\gamma \in \mathcal{CC}(t\gamma, \mathcal{V})$. By assumption on u, $u\gamma : \tau$ is a subterm of $s\gamma : \sigma$ and $\mathcal{V}ar(u\gamma) \subseteq \mathcal{V}ar(t\gamma)$. Therefore, $u\gamma \in \mathcal{CC}(t\gamma, \mathcal{V})$ by Case 1.

Case 5: let $u = \lambda x.s$ with $x \in \mathcal{X} \setminus (\mathcal{V} \cup \mathcal{V}ar(t))$ and $s \in \mathcal{CC}(t, \mathcal{V} \cup \{x\})$. To the price of renaming the variable x in s if necessary, we can assume in addition that $x \notin \mathcal{V}ar(t\gamma) \cup \mathcal{D}om(\gamma)$, and therefore $\mathcal{V} \cup \{x\} \subseteq \mathcal{X} \setminus (\mathcal{V}ar(t) \cup \mathcal{V}ar(t\gamma) \cup \mathcal{D}om(\gamma))$. By induction hypothesis, $s\gamma \in \mathcal{CC}(t\gamma, \mathcal{V} \cup \{x\})$. Since $x \notin \mathcal{V}ar(t\gamma) \cup \mathcal{V}$, by Case 5 of the definition $\lambda x.s\gamma \in \mathcal{CC}(t\gamma, \mathcal{V})$ and, since $x \notin \mathcal{D}om(\gamma)$, we have $u\gamma = \lambda x.s\gamma$.

Case 7: let $\mathcal{V}' = \mathcal{V} \cup \{x\}$. By definition, $u \in \mathcal{CC}(t, \mathcal{V}')$, with $x \notin \mathcal{V}ar(u, t) \cup \mathcal{V}$. By induction hypothesis we have $u\gamma \in \mathcal{CC}(t\gamma, \mathcal{V}')$, and, since $x \notin \mathcal{V}ar(u\gamma, t\gamma) \cup \mathcal{V}$, by Case 7, we have $u\gamma \in \mathcal{CC}(t\gamma, \mathcal{V})$.

Case 6: let $u \in \mathcal{CC}(t, \mathcal{V})$, and $u \succeq_{horpo} v$; then $v \in \mathcal{CC}(t, \mathcal{V})$; □

Lemma 4.3 *Assume that $u : \sigma \in \mathcal{CC}(t : \tau)$. Then, $u\xi : \sigma\xi \in \mathcal{CC}(t\xi : \tau\xi)$ for every type substitution ξ.*

Proof: We prove that if $u : \sigma \in \mathcal{CC}(t : \tau, \mathcal{V})$ with $\mathcal{V} \subseteq \mathcal{X} \setminus \mathcal{V}ar(t)$, then $u\xi : \sigma\xi \in \mathcal{CC}(t : \tau\xi, \mathcal{V}\xi)$. By $\mathcal{V}\xi$, we mean that the types of the variables in $\mathcal{V}$ are instantiated by ξ. We proceed by induction on the definition of $\mathcal{CC}(t, \mathcal{V}xi)$.

If $u : \sigma$ is in $\mathcal{V}\xi$ or in $\overline{t}\xi$ it holds directly. Otherwise there are several cases according to the definition.

Case 1: let $s = f(\overline{s}) : \sigma' \in \mathcal{CC}(t, \mathcal{V})$ with $f \in \mathcal{F}$ and $u : \sigma$ be a subterm of s with $\sigma \in \mathcal{T}_{\mathcal{S}}^{min}$ (which implies that σ is ground) and $\mathcal{V}ar(u) \subseteq \mathcal{V}ar(t)$. By induction hypothesis, $s\xi : \sigma'\xi \in \mathcal{CC}(t : \tau\xi, \mathcal{V}\xi)$. Therefore, since $\sigma = \sigma\xi \in \mathcal{T}_{\mathcal{S}}^{min}$ and $u\xi : \sigma\xi$ is a subterm of $s : \sigma'\xi$, we have $u\xi : \sigma\xi \in \mathcal{CC}(t : \tau\xi, \mathcal{V}\xi)$ by Case 1.

Case 2: let $u = g(\overline{u}) : \sigma$ with g such that $f >_{\mathcal{F}} g$, and $\overline{u} : \overline{\sigma} \in \mathcal{CC}(t : \tau, \mathcal{V})$. Then, by induction hypothesis, we have $\overline{u}\xi : \overline{\sigma}\xi \in \mathcal{CC}(t : \tau\xi, \mathcal{V}\xi)$ and hence $u\xi = g(\overline{u}\xi) : \sigma\xi \in \mathcal{CC}(t\xi : \tau\xi, \mathcal{V}\xi)$ by Case 2.

Case 3: let $u = g(\overline{u}) : \sigma$ and $t = f(\overline{t}) : \tau$ with $f =_{\mathcal{F}} g$, $\overline{u} : \overline{\sigma}$ in $\mathcal{CC}(t : \tau, \mathcal{V})$ and $\overline{t} : \overline{\tau}(\succ_{horpo} \cup \rhd_{>_{\mathcal{T}_{\mathcal{S}}}})_{stat_f} \overline{u} : \overline{\sigma}$. Then, by polymorphism of $\succ_{horpo}$ and $\rhd_{>_{\mathcal{T}_{\mathcal{S}}}}$ (the latter due to the stability under type substutions of the type ordering), we have $\overline{t} : \overline{\tau}\xi(\succ_{horpo} \cup \rhd_{>_{\mathcal{T}_{\mathcal{S}}}})_{stat_f} \overline{u} : \overline{\sigma}\xi$. Since, by induction hypothesis, $\overline{u}\xi : \overline{\sigma}\xi$ in $\mathcal{CC}(t\xi : \tau\xi, \mathcal{V}\xi)$, we can conclude $u\xi = g(\overline{u}\xi) : \sigma\xi \in \mathcal{CC}(t\xi : \tau\xi, \mathcal{V}\xi)$ by Case 3.

Case 4: let $u = @(s, u_1, \ldots, u_n) : \sigma$ with $s : \sigma_1 \to \ldots \to \sigma_n \to \sigma \in \mathcal{CC}(t : \tau, \mathcal{V})$ and $u_i : \sigma_i \in \mathcal{CC}(t : \tau, \mathcal{V})$ for every $i \in [1..n]$. Then, by induction hypothesis, we have $s\xi : \sigma_1\xi \to \ldots \to \sigma_n\xi \to \sigma\xi \in \mathcal{CC}(t\xi : \tau\xi, \mathcal{V}\xi)$ and $u_i\xi : \sigma_i\xi \in \mathcal{CC}(t\xi : \tau\xi, \mathcal{V}\xi)$ for every $i \in [1..n]$. Therefore $u\xi = @(s\xi, u_1\xi, \ldots, u_n\xi) : \sigma\xi \in \mathcal{CC}(t\xi : \tau\xi, \mathcal{V}\xi)$ by Case 4.

Case 5: let $u = \lambda x.s : \sigma_1 \to \sigma_2$ with $x \in \mathcal{X} \setminus (\mathcal{V} \cup \mathcal{V}ar(t))$ and $s : \sigma_2 \in \mathcal{CC}(t : \tau, \mathcal{V} \cup \{x\})$. By induction hypothesis, $s\xi : \sigma_2\xi \in \mathcal{CC}(t\xi : \tau\xi, \mathcal{V}\xi \cup \{x : \sigma_1\xi\})$, and hence $\lambda x.s\xi : \sigma_1\xi \to \sigma_2\xi \in \mathcal{CC}(t\xi : \tau\xi, \mathcal{V}xi)$ by Case 5 of the definition.

Case 6: let $s : \rho \in \mathcal{CC}(t : \tau, \mathcal{V})$ with $s : \rho \succeq_{horpo} u : \sigma$. By polymorphism of $\succeq_{horpo}$, we have $s : \rho\xi \succeq_{horpo} u : \sigma\xi$ and, by induction hypothesis, we have $s\xi : \rho\xi \in \mathcal{CC}(t\xi : \tau\xi, \mathcal{V}\xi)$. Therefore, $u\xi : \sigma\xi \in \mathcal{CC}(t\xi : \tau\xi, \mathcal{V}\xi)$; by Case 6.

Case 7: let $\mathcal{V}' = \mathcal{V} \cup \{x\}$. By definition, $u : \sigma \in \mathcal{CC}(t, \mathcal{V}')$, with $x \notin \mathcal{V}ar(u, t) \cup \mathcal{V}'$. By induction hypothesis we have $u\xi : \sigma\xi \in \mathcal{CC}(t\xi : \tau\xi, \mathcal{V}'\xi)$, and hence, by Case 7, we have $u\xi : \sigma\xi \in \mathcal{CC}(t\xi : \tau\xi, \mathcal{V}\xi)$. □

4.2 The Higher-Order Recursive Path Ordering with Closure

From now on, both orderings $\succ_{horpo}$ and $\succ_{chorpo}$ will coexist. We first give a more general formulation of the proposition A:

$$A = \forall v \in \overline{t}\ s \underset{chorpo}{\succ} v \text{ or } u \underset{chorpo}{\succeq} v \text{ for some } u \in \mathcal{CC}(s)$$

Definition 4.4

$$s : \sigma \underset{chorpo}{\succ} t : \tau \textit{ iff } \sigma \geq_{\mathcal{T}_S} \tau \textit{ and}$$

1. $s = f(\overline{s})$ *with* $f \in \mathcal{F}$, *and (i)* $u \underset{chorpo}{\succeq} t$ *for some* $u \in \overline{s}$ *or (ii)* $t \in \mathcal{CC}(s)$
2. $s = f(\overline{s})$ *with* $f \in \mathcal{F}$ *and* $t = g(\overline{t})$ *with* $f >_{\mathcal{F}} g$, *and* A
3. $s = f(\overline{s})$ *and* $t = g(\overline{t})$ *with* $f, g \in Mul$, $f =_{\mathcal{F}} g$, *and* $\overline{s}\ (\underset{chorpo}{\succ})_{mul}\ \overline{t}$
4. $s = f(\overline{s})$ *and* $t = g(\overline{t})$ *with* $f, g \in Lex$, $f =_{\mathcal{F}} g$, *and* $\overline{s}\ (\underset{chorpo}{\succ})_{lex}\ \overline{t}$ *and* A
5. $s = @(s_1, s_2)$, *and* $s_1 \underset{chorpo}{\succeq} t$ *or* $s_2 \underset{chorpo}{\succeq} t$
6. $s = \lambda x : \alpha.u$ *with* $x \notin \mathcal{V}ar(t)$, *and* $u \underset{chorpo}{\succeq} t$
7. $s = f(\overline{s})$ *with* $f \in \mathcal{F}$, $t = @(\overline{t})$ *is a partial 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* $s \underset{chorpo}{\succ} v$
9. $s = @(s_1, s_2)$, $t = @(\overline{t})$ *is a partial left-flattening of* t *and* $\{s_1, s_2\}(\underset{chorpo}{\succ})_{mul}\{\overline{t}\}$
10. $s = \lambda x : \alpha.u,\ t = \lambda x : \beta.v,\ \alpha =_{\mathcal{T}_S} \beta$, *and* $u \underset{chorpo}{\succeq} v$
11. $s = @(\lambda x : \alpha.u, v)$ *and* $u\{x \mapsto v\} \underset{chorpo}{\succeq} t$

The definition is recursive, and we can show that it is well-founded by comparing pairs of terms in the well-founded ordering $(\succ_{horpo}, \rhd)_{right-to-left-lex}$.

As an easy consequence of Case 1, we obtain that $s = f(\overline{s}) : \sigma \succ_{chorpo} t : \tau$ with $f \in \mathcal{F}$ if $\sigma = \tau$ and $t \in \mathcal{CC}(s)$. This shows that the technique based on the general schema, essentially based on the closure mecanism introduced by Blanqui, Jouannaud and Okada [3], is a very particular case of the present method. This does not necessarily mean that the latter outperforms the former. It could be that the ordering mecanism is kind of redundant with the closure mecanism. We do not think so, however, since we were able to prove many examples that the general schema could not. Besides, this new method inherits all the advantages of the recursive path ordering based methods, in particular it is possible to combine it with interpretations based techniques [35].

The proofs of Lemmas 3.7, 3.8 and 3.10 are easy to adapt (using now Lemma 4.2 for the proof of the new version of Lemma 3.8 and Lemma 4.3 for Lemma 3.10) to the ordering with closure:

Lemma 4.5 $\succ_{chorpo}$ *is monotonic for terms of equivalent types, stable, and polymorphic.*

4.3 Strong Normalization

We first show that terms in the computable closure of a term are computable under the appropriate hypotheses for their use in a comparison.

First, the computability predicate is of course now defined with respect to $\succ_{chorpo}$. To show that the computability properties remain valid, we simply observe that there is no change whatsoever in the proofs, since the rules applying to application-headed terms did not change. For this, it is crucial to respect the given formulation of Case 5, avoiding the use of the closure as in Case 1. Actually, we could have defined a specific, weaker closure for terms headed by an application, to the price of doing again the two most complex proofs of the computability properties. We did not think it was worth the trouble.

Second, we are still using the previous ordering $\succ_{horpo}$ in the definition of the new ordering $\succ_{chorpo}$, via the closure definition in particular. We will therefore need to prove that $\succ_{horpo} \subseteq \succ_{chorpo}$ in order to apply the induction arguments based on the well-foundedness of $\succ_{chorpo}$ on some appropriate set of terms. The following lemma is important for the proof of Property 4.7:

Lemma 4.6 $\succ_{horpo} \subseteq \succ_{chorpo}$.

The importance of this lemma comes from the computability properties. By the above inclusion, any term smaller than a computable term in the ordering $\succ_{horpo}$ will therefore be computable by Property 3.14 (ii).

We come to the main property of terms in the computable closure, which justifies its name:

Property 4.7 *Assume $\overline{t} : \overline{\tau}$ is computable, as well as every term $g(\overline{s})$ with $\overline{s}$ computable and $g(\overline{s})$ smaller than $t = f(\overline{t})$ in the ordering $(>_{\mathcal{F}}, (\succ_{horpo} \cup \rhd_{>_{\mathcal{T}_{\mathcal{S}}}})_{stat_f})_{lex}$ operating on pairs $\langle f, \overline{t}\rangle$. Then every term in $\mathcal{CC}(t)$ is computable.*

The precise formulation of this statement arises from its forthcoming use inside the proof of Lemma 4.8.

Proof: We prove that $u\gamma : \sigma$ is computable for every computable substitution γ of domain $\mathcal{V}$ and every $u \in \mathcal{CC}(t, \mathcal{V})$ such that $\mathcal{V} \cap \mathcal{V}ar(t) = \emptyset$. We obtain the result by taking $\mathcal{V} = \emptyset$. We proceed by induction on the definition of $\mathcal{CC}(t, \mathcal{V})$.

For the basic case: if $u \in \mathcal{V}$, then $u\gamma$ is computable by the assumption on γ; if $u \in \overline{t}$, we conclude by the assumption that $\overline{t}$ is computable, since $u\gamma = u$ by the assumption that $\mathcal{V} \cap \mathcal{V}ar(t) = \emptyset$; and if u is a subterm of $f(\overline{t})$, we again remark that γ acts as the identity on such terms, and therefore, they are computable by assumption.

For the induction step, we discuss the successive operations to form the closure:

case 1: u is a subterm of type in $\mathcal{T}_{\mathcal{S}}^{min}$ of some $v \in \mathcal{CC}(t, \mathcal{V})$. By induction hypothesis, $v\gamma$ is computable, hence strongly normalizable by Property 3.14 (i). By monotonicity of $\succ_{chorpo}$ for terms of equivalent types, $u\gamma$ is also strongly normalizable, and hence computable by Property 3.14 (vi).

case 2: $u = g(\overline{u})$ where $\overline{u} \in \mathcal{CC}(t, \mathcal{V})$. By induction hypothesis, $\overline{u}\gamma$ is computable. Since $f >_{\mathcal{F}} g$, $u\gamma$ is computable by our assumption that terms smaller than $f(\overline{t})$ are computable.

case 3: $u = g(\overline{u})$ where $f =_{\mathcal{F}} g$, $\overline{u} \in \mathcal{CC}(t, \mathcal{V})$ and $\mathcal{V}ar(u) \subseteq \mathcal{V}ar(t)$. By induction hypothesis, $\overline{u}\gamma$ is computable. By assumption, and monotonicity of reductions, $\overline{t}\gamma(\succ_{horpo} \cup \rhd_{>_{\mathcal{T}_{\mathcal{S}}}})_{stat_f} \overline{u}\gamma$. Therefore $u\gamma = g(\overline{u}\gamma)$ is computable by our assumption that terms smaller than $f(\overline{t})$ are computable.

case 4: by induction and Property 3.14 (iv).

case 5: let $u = \lambda x.s$ with $x \notin \mathcal{V}$ and $s \in \mathcal{CC}(t, \mathcal{V} \cup \{x\})$. To the price of possibly renaming x, we can assume without loss of generality that $x \notin \mathcal{D}om(\gamma) \cup \mathcal{V}ar(t)$. As a first consequence, $\mathcal{V} \cup \{x\} \cap \mathcal{V}ar(t) = \emptyset$; as a second, given an arbitrary computable term w, $\gamma' = \gamma \cup \{x \mapsto w\}$ is a computable substitution of domain $\mathcal{V} \cup \{x\}$. By induction hypothesis, $s\gamma'$ is therefore computable, and by Property 3.14 (v), $(\lambda x.s)\gamma$ is computable.

case 6: by Property 3.14 (ii).

case 7: by induction hypothesis. □

We now restate Property 3.15 and show in one case how the proof makes use of Property 4.7.

Lemma 4.8 *Let $f \in \mathcal{F}$ and let $\overline{t}$ be a set of terms. If $\overline{t}$ is computable, then $f(\overline{t})$ is computable.*

Proof: We prove that $f(\overline{t})$ is computable if terms in $\overline{t}$ are computable by an outer induction on the pair $\langle f, \overline{t}\rangle$ ordered lexicographically by the ordering $(>_{\mathcal{F}}, (\succ_{chorpo} \cup \rhd_{>_{\mathcal{T}_{\mathcal{S}}}})_{stat_f})_{lex}$, and an inner induction on the size of the reducts of t. Since terms in $\overline{t}$ are computable by assumption, they are strongly normalizable by Property 3.15(ii), hence, by Lemma 2.27 the ordering $(>_{\mathcal{F}}, ((\succ_{chorpo} \cup \rhd_{>_{\mathcal{T}_{\mathcal{S}}}})_{stat})_{lex}$ is well founded on the pairs satisfying the assumptions.

By Lemma 4.6, the set of pairs smaller than the pair $\langle f, \overline{t}\rangle$ for the ordering $(>_{\mathcal{F}}, (\succ_{chorpo} \cup \rhd_{>_{\mathcal{T}_{\mathcal{S}}}})_{stat})_{lex}$ contains the set of pairs that are smaller than that pair for the ordering $(>_{\mathcal{F}}, (\succ_{horpo} \cup \rhd_{>_{\mathcal{T}_{\mathcal{S}}}})_{stat})_{lex}$. This key remark allows us to use Property 4.7.

The proof is actually similar to the proof of Property 3.15, except for case 1 and for the cases using property A. We therefore do Cases 1 and 2, the latter using property A.

case 1: let $f(\overline{t}) \succ_{chorpo} s$ by case 1, hence $t_i \succeq_{horpo} s$ for some $t_i \in \overline{t}$ or $s \in \mathcal{CC}(t)$. In the former case, since t_i is computable, s is computable by Property 3.14 (ii); in the latter case, all terms smaller than $f(\overline{t})$ with respect to $(>_{\mathcal{F}}, (\succeq_{horpo})_{stat})_{lex}$ are computable by induction hypothesis and the above key remark, hence s is computable by Property 4.7.

case 2: let $t = f(\overline{t}) \succ_{chorpo} s$ by case 2. Then $s = g(\overline{s})$, $f >_{\mathcal{F}} g$ and for every $s_i \in \overline{s}$ either $t \succ_{chorpo} s_i$, in which case s_i is computable by the inner induction hypothesis, or $v \succeq_{chorpo} s_i$ for some $v \in \mathcal{CC}(t)$, in which case v is computable by Property 4.7 and hence s_i is computable by Property 3.14 (ii). We then conclude that s is computable the by outer induction hypothesis since $f >_{\mathcal{F}} g$. □

Theorem 4.9 *$\succ_{chorpo}$ is included in a polymorphic higher-order reduction ordering.*

Proof: Thanks to Property 4.7, the strong normalization proof of this improved ordering is exactly the same as previously, using of course Lemma 4.8 instead of Lemma 3.15. Using now Lemma 4.5, we therefore conclude that the transitive closure of $\succ_{chorpo}$ is a higher-order polymorphic reduction ordering. □

4.4 Examples

This new definition is much stronger than the previous one. In addition to allowing us proving the strong normalization property of the remaining rules of the sorting example, and for the same reason, the following rule can be added to the other rules of Example 2:

Example 8 $\quad n * m \;\; \rightarrow \;\; rec(n, 0, \lambda z_1 z_2.m + z_2)$

This additional rule can be proved terminating with the precedence: $* >_{\mathcal{F}} \{rec, +, 0\}$, since $\lambda z_1 z_2.m + z_2 \in CS(n*m)$: by base case, m and z_2 belong to $\mathcal{CC}(n*m, \{z_1, z_2\})$, hence $m+z_2 \in \mathcal{CC}(n*m, \{z_1, z_2\})$ by case 2 of the definition of the computational closure. Applying case 5 twice yields then the result. □

The coming example is quite classical too.

Example 9 Let $\mathcal{S} = \{NList : *,\ List : * \to *\}$, $\mathcal{S}^\forall = \{\alpha\}$, $\mathcal{F} = \{0, 1 :\to \alpha; + : \alpha \times \alpha \to \alpha; nil :\to NList; cons : \alpha \times NList \to List(\alpha); cons : \alpha \times List(\alpha) \to List(\alpha); foldl : (\alpha \to \alpha \to \alpha) \times \alpha \times NList \to \alpha; foldl : (\alpha \to \alpha \to \alpha) \times \alpha \times List(\alpha) \to \alpha; sum : List(\alpha) \to \alpha; +^c : \alpha \to \alpha \to \alpha\}$.
The rules are the following:

$$\begin{array}{rcrcl}
\{x : \alpha,\ F : \alpha \to \alpha \to \alpha\} & \vdash & foldl(F, x, nil) & \to & x \\
\{x, y : \alpha,\ F : \alpha \to \alpha \to \alpha,\ l : NList\} & \vdash & foldl(F, x, cons(y, l)) & \to & foldl(F, (F\ x\ y), l) \\
\{x, y : \alpha,\ F : \alpha \to \alpha \to \alpha,\ l : List(\alpha)\} & \vdash & foldl(F, x, cons(y, l)) & \to & foldl(F, (F\ x\ y), l) \\
 & & +^c & \to & \lambda xy.x + y \\
\{l : List(\alpha)\} & \vdash & sum(l) & \to & foldl(+^c, 0, l)
\end{array}$$

The first rule is by subterm case. For the second, we set a right-to-left lexicographic status for $foldl$, and, applying rule 4, we recursively have to show that $cons(y, l) \succ_{chorpo} l$, which succeeds by subterm, and F on the righthand side being taken care of by F on the lefthand one, we are left to show that $foldl(F, x, cons(y, l)) \succ_{chorpo} @(F, x, y)$, which succeeds easily by rule 7.

For the third, we show that $\lambda xy.x + y$ is in the closure of $+^c$, provided $+^c >_\mathcal{F} +$.

For the last, we need the precedence $sum >_\mathcal{F} \{flodl, +^c, 0\}$ in order to show that the righthand side is in the closure of the lefthand one. □

The following example, a definition of formal derivation, illustrates best the power of the computational closure.

Example 10 Let $\mathcal{S} = \{form\}$, $\mathcal{F} = \{D : (form \to form) \to (form \to form);\ \forall, \exists : (form \to form) \to form;\ 0, 1 :\to form;\ -, sin, cos, ln : form \to form;\ +, \times, / : form \times form \to form\}$.
The rules are the following:

$$\begin{array}{rcrcl}
 & & D(\lambda x.y) & \to & \lambda x.0 \\
 & & (\lambda x.x) & \to & \lambda x.1 \\
\{F : form \to form\} & \vdash & D(\lambda x.sin(F\ x)) & \to & \lambda x.cos(F\ x) \times (D(F)\ x) \\
\{F : form \to form\} & \vdash & D(\lambda x.cos(F\ x)) & \to & \lambda x. - sin(F\ x) \times (D(F)\ x) \\
\{F, G : form \to form\} & \vdash & D(\lambda x.(F\ x) + (G\ x)) & \to & \lambda x.(D(F)\ x) + (D(G)\ x) \\
\{F, G : form \to form\} & \vdash & D(\lambda x.(F\ x) \times (G\ x)) & \to & \lambda x.(D(F)\ x) \times (G\ x) + (F\ x) \times (D(G)\ x) \\
\{F : form \to form\} & \vdash & D(\lambda x.ln(F\ x)) & \to & \lambda x.(D(F)\ x)/(F\ x)
\end{array}$$

We take $D >_\mathcal{F} \{0, 1, \times, -, +, /\}$ for precedence and assume that the function symbols are in Mul. We do the first rule and the one before last.

Since $D >_\mathcal{F} 0$, $D(\lambda x.y) \succ_{chorpo} 0$ by precedence case, and since $x \notin \mathcal{V}ar(0)$, $D(\lambda x.y) \succ_{chorpo} \lambda x.0$ by precedence case for abstractions.

An alternative proof uses the computational closure : since $D >_\mathcal{F} 0$, $0 \in \mathcal{CC}(D(\lambda x.y), \{x\})$ by precedence case, hence $\lambda x.0 \in \mathcal{CC}(D(\lambda x.y))$ by abstraction. Therefore, $D(\lambda x.y) \succ_{chorpo} \lambda x.0$ by subterm case.

For the rule before last, we show again that the right hand side is in the closure of the lefthand one and apply the subterm case. Since it is more complicated, we will take this opportunity to do a goal-directed proof, using a stack of subgoals:
Initial goal (for sake of clarity, we rename the bound variables and make applications explicit):
$\lambda y.@(D(F), y) \times @(G, y) + @(F, y) \times @(D(G), y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))))$

By abstraction, we obtain the new subgoal:
$@(D(F), y) \times @(G, y) + @(F, y) \times @(D(G), y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$
By precedence, we obtain now two subgoals:
$@(D(F), y) \times @(G, y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$;
$@(F, y) \times @(D(G), y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$
By precedence again, we obtain two new subgoals:
$@(D(F), y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$;
$@(G, y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$;
$@(F, y) \times @(D(G), y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$
By application:
$D(F) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$;
$y \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$;
$@(G, y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$;
$@(F, y) \times @(D(G), y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$
By recursive call (note that $\lambda x.(@(F, x) \times @(G, x)) \rhd_{>_{\mathcal{T}_S}} F$):
$F \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$;
$y \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$;
$@(G, y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$;
$@(F, y) \times @(D(G), y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$
By extensionality (which has been proved already):
$\lambda x.@(F, x) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$;
$y \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$;
$@(G, y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$;
$@(F, y) \times @(D(G), y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$
By reduction (since $\lambda x.(@(F, x) \times @(G, x)) \succ_{horpo} \lambda x.@(F, x)$ by Cases 1 and 11):
$\lambda x.(@(F, x) \times @(G, x)) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{x, y\})$;
$y \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$;
$@(G, y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$;
$@(F, y) \times @(D(G), y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$
By basic case, the first goal now disappers:
$y \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$;
$@(G, y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$;
$@(F, y) \times @(D(G), y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$
By basic case again, the first subgoal disappears:
$@(G, y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$;
$@(F, y) \times @(D(G), y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$
By abstraction:
$\lambda y.@(G, y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))))$;
$@(F, y) \times @(D(G), y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$
By reduction as before (renaming the bound variable y into x):
$\lambda x.(@(F, x) \times @(G, x)) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))))$;
$@(F, y) \times @(D(G), y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$
By basic case, the first subgoal disappears and we are left with a single subgoal:
$@(F, y) \times @(D(G), y) \in \mathcal{CC}(D(\lambda x.(@(F, x) \times @(G, x))), \{y\})$ which is now solved as previously. □

This example can therefore be entirely solved by the mechanism of the closure. This does not mean that it could be already solved with the same proof by the technique developped in [3], where the computational closure was first introduced without any other mecanism. The point is that the closure defined here is more powerful thanks to case 6 based on $\succ_{horpo}$ instead of simply using β-reductions as in [3].

Note that our proof is goal-directed, but not syntax directed. The use of reductions (actually expansions in this backward chaining use of the rules) needs some user interaction to provide with the appropriate term reducing to the goal. This is however, at least in this example, a rather simple task that could be performed with an appropriate tactic.

We are not completely satisfied with this computation, though, because the structural definition of the ordering has been completely lost here. We would prefer to improve the ordering so as to do some of the computation via the ordering and delegate the hard parts only to the more complex closure mecanism. For the time being, however, the closure mechanism is the only one which applies to rules whose right-hand side is an abstraction with the bound variable occurring in the body, assuming that the lefthand side of rule is not itself an abstraction.

The next example (uncurrying) shows again the need of more user-interaction via the use of a middle term. So, the fact that $\succ_{chorpo}$ is not transitive appears also as a weakness of the ordering.

Example 11 First, to a signature $\mathcal{F}$, we associate the signature

$$F^{curry} = \left\{ \begin{array}{l} f_i : \sigma_1 \times \ldots \times \sigma_i \to (\sigma_{i+1} \to \ldots \to \sigma_p \to \tau) \text{ for every } i \in [0..p] \mid \\ f : \sigma_1 \times \ldots \times \sigma_n \to (\sigma_{n+1} \to \ldots \to \sigma_p \to \tau) \in \mathcal{F} \text{ where } \tau \text{ is a data-type} \end{array} \right\}$$

and we introduce the following rewrite rules for currying/uncurrying:

$$\begin{array}{lrcl} \{t_1 : \sigma_1, \ldots, t_{j+1} : \sigma_{j+1}\} & \vdash & f_{j+1}(t_1, \ldots, t_{j+1}) & \to & @(f_j(t_1, \ldots, t_j), t_{j+1}) \\ \{t_1 : \sigma_1, \ldots, t_{i+1} : \sigma_{i+1}\} & \vdash & @(f_i(t_1, \ldots, t_i), t_{i+1}) & \to & f_{i+1}(t_1, \ldots, t_{i+1}) \end{array}$$

Starting with the first rule, we set $f_{j+1} >_{\mathcal{F}} f_j$. Since the use of case 7 fails for type reason when comparing $f_{j+1}(t_1, \ldots, t_{j+1})$ and $f_{j+1}(t_1, \ldots, t_j)$, we proceed to show that the righthand side is in the closure of the lefthand one. By application, we have to show that both subterms $f_j(t_1, \ldots, t_j)$ and t_{j+1} are in the closure. This is trivial for the second, and results from the precedence case for the first.

Trying now to prove that the lefthand side of the second rule is bigger than the righthand one, we set $f_i >_{\mathcal{F}} f_{i+1}$, and the only possible case is subterm. But the comparison $f_i(t_1, \ldots, t_i) \succ_{chorpo} f_{i+1}(t_1, \ldots, t_{i+1})$ cannot succeed since there is a new free variable (t_{i+1}) in the righthand side. The computational closure does not help either here, since it is generated from the set $\{f_i(t_1, \ldots, t_i), t_{i+1}\}$, and there is no way, in general, to access the terms $t_1, \ldots, t_i$ in order to build the righthand side by the precedence rule.

The trick is to invent a middle term, and show that the lefthand side is bigger than the righthand one in the *transitive closure* of the ordering. The convenient middle term here is $@(\lambda x.f_{i+1}(t_1, \ldots, t_i, x), t_{i+1})$, which reduces to the righthand side by the use of Case 11. We therefore simply need to show that the lefthand side is bigger than the middle term. Since both terms are headed by an abstraction, we simply need to prove that $f_i(t_1, \ldots, t_i) \succ_{chorpo} \lambda x.f_{i+1}(t_1, \ldots, t_i, x)$, which we prove now by showing that the righthand side term is in the closure of the lefthand side one. Indeed, now, $t_1, \ldots, t_i, x \in \mathcal{CC}(f_i(t_1, \ldots, t_i), \{x\})$, and by precedence, $f_{i+1}(t_1, \ldots, t_i, x) \in \mathcal{CC}(f_i(t_1, \ldots, t_i), \{x\})$, and we conclude by abstraction. We see here that the middle term allowed us to pop up the subterm $f_i(t_1, \ldots, t_i)$ of the lefthand side of rule, allowing to make the terms $t_1, \ldots, t_i$ available in the closure of $f_i(t_1, \ldots, t_i)$. Although this looks as a trick, the (type) conditions for applying this trick sucessfully can be easily characterized and hence implemented, making it transparent for the user. We will indeed use it again for solving another example later.

Remark that we can accomodate a mixture of currying and uncurrying by chosing an appropriate well-founded precedence for selecting the right number of arguments desired for a given function symbol. □

We end this section with two examples showing yet another (temporary) weakness of our ordering.

Example 12 (adapted from [26]) Let $\mathcal{S} = \{List\}$, $\mathcal{S}^{\forall} = \{\alpha\}$, $\mathcal{F} = \{nil : List, cons : \alpha \times List \to List, fcons : (\alpha \to \alpha) \to List, dapply : \alpha \times (\alpha \to \alpha) \times (\alpha \to \alpha) \to \alpha, lapply : \alpha \times List \to \alpha\}$. The rules are the following:

$$\begin{array}{rcrcl} \{F, G : \alpha \to \alpha,\ x : \alpha\} & \vdash & dapply(x, F, G) & \to & F(G(x)) \\ \{x : \alpha\} & \vdash & lapply(x, nil) & \to & x \\ \{F : \alpha \to \alpha,\ x : \alpha,\ l : List\} & \vdash & lapply(x, cons(F, L)) & \to & F(lapply(x, L)) \end{array}$$

The first two rules are straightfoward. For the third, we face the problem that F is not in the closure of the lefthand side, as a higher-order variable occuring inside a strict subterm of a lefthand side argument. Although arbitrary subterms may yield non-terminating sequences as shown in [19], where a non-terminating example is constructed by using such a deep variable, this is not the case here: $cons$ being the free list constructor, its subterms are indeed computable whenever the whole term is computable. This example can therefore be solved by adding this property to the closure, see Section 7 for details. □

A second, more serious weakness shows up with the following example about process algebra, in which a quantifier named Σ binds variables via the use of a functional argument, that is, an abstraction:

Example 13 (taken from [11]) Let $\mathcal{S} = \{proc, data\}$, $\mathcal{F} = \{+ : proc \times proc \to proc, \cdot : proc \times proc \to proc, \delta :\to proc, \Sigma : (data \to proc) \to proc)\}$. Here, $+$ stands for the choice operator, $\cdot$ for sequential composition, δ for deadlock, and Σ for the data dependent choice. The rules are the following:

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

The first eight rules can be easily oriented by using our ordering (without the closure mechanism) with the precedence $\{\cdot, \Sigma\} >_{\mathcal{F}} +$. For the last rule, we cannot use the precedence case of the ordering (assuming now $\Sigma >_{\mathcal{F}} +$), since we would end up with $\lambda d : data.(P(d) \cdot x)$ on the right, whose type is $data \to proc$, hence cannot be smaller than the whole lefthand side. But the closure does not work either, since our two starting terms are $\Sigma(\lambda d : data.P(d))$ and x, which cannot help for building the righthand side. This is quite surprising, since no application occurs in the example, apart from the terms like $@(P, d)$ applying a higher-order variable to a term. Having inductive types would help again, provided we use the recent ideas from [5] : by reaching under the constructors $\cdot$ and Σ, $\lambda d : data.(P(d)$, hence $P(d)$ would be in the closure. It is then easy to reconstruct the righthand side. We develop an alternative technique in section 5 that could also be used here. □

5 Normalized Rewriting

The purpose of this section is to give a β-stable subrelations $(\succ_{horpo})^{\eta}_{\beta}$ of $\succ_{horpo}$ and $(\succ_{chorpo})^{\eta}_{\beta}$ of $\succ_{chorpo}$, in order to prove the strong normalization property of normalized rewriting.

First, we remark that $\succ_{horpo}$ and $\succ_{chorpo}$ themselves (without neutralized positions) are not β-stable. The following example shows the kind of terms which cause a violation of β-stability by $\succ_{chorpo}$:

Example 14 $\@(X, f(a)) = X(f(a)) \succ_{horpo} X(a) = \@(X, a)$ and $X(a) \succ_{horpo} a$
while instantiating these comparisons with the substitution $\gamma = \{X \mapsto \lambda y.a\}$ yields:
$X(f(a))\gamma\downarrow \;=\; X(a)\gamma\downarrow \;=\; a$ which contradicts β-stability. □

Fortunatelly this is the only situation in which β-stability can be violated. We will therefore restrict the orderings $\succ_{horpo}$ and $\succ_{chorpo}$ by forbidding the kind of comparison suggested by example 14.

5.1 The Normalized Higher-Order Recursive Path Ordering

We sketch here the definition of $(\succ_{horpo})^{\eta}_{\beta}$ on (arbitrary) terms:

Definition 5.1 *The relation $(\succ_{horpo})^{\eta}_{\beta}$ is defined as $\succ_{horpo}$ but restricting cases 5 and 9 as follows :*

5. *$s = \@(s_1, s_2)$, s_1 is not an abstraction nor a variable and $u(\underset{horpo}{\succeq})^{\eta}_{\beta}\, t$ for some $u \in \{s_1, s_2\}$*

9. *$s = \@(s_1, s_2)$, $t = \@(\overline{t})$ is a partial left-flattening of t, s_1 is not an abstraction nor a variable and $\{s_1, s_2\}((\underset{horpo}{\succ})^{\eta}_{\beta})_{mul}\{\overline{t}\}$*

Lemma 5.2 *$(\succ_{horpo})^{\eta}_{\beta}$ is a β-stable and η-polymorphic subrelation of $\succ_{horpo}$.*

Proof: The fact that $(\succ_{horpo})^{\eta}_{\beta}$ is a subrelation of $\succ_{horpo}$ is an easy consequence of their definitions: both relations differ only in Cases 5 and 9, by forbidding variables and abstractions as first argument of the lefthand side application in both cases. The proofs of stability and polymorphism can then be obtained from the proofs of Lemmas 5.19 and 5.9 as a particular case. □

5.2 The Normalized Computational Closure

We can now define the adequate version of the computational closure. Notice that we do not assume terms to be normalized in this definition. The reason is that taking possibly non-normalized terms allows to have more normalized terms in the closure on the one hand, and eases the coming stability proof on the other hand. In practice, we only need to collect normalized terms from the closure, since the righthand sides of rules (as well as the lefthand ones indeed) are assumed to be normalized.

First we need to restrict the type decreasing subterm relation $\rhd_{>_{\mathcal{TS}}}$:

Definition 5.3 *Let $\geq_{\mathcal{TS}}$ be a type ordering, and let us write $s : \sigma \rhd^{x}_{>_{\mathcal{TS}}} t : \tau$ iff $t = s|_p$ is a subterm of s such that (i) no superterm $s|_{q<p}$ of t in s is of the form $\@(x, v)$ with x a variable, (ii) $\sigma \geq_{\mathcal{TS}} \tau$ and (iii) $\mathcal{V}ar(u) \subseteq \mathcal{V}ar(t)$.*

Definition 5.4 *Given a term $t = f(\overline{t})$ with $f \in \mathcal{F}$, we define its* normalized computable closure *$\mathcal{CC}^{\eta}_{\beta}(t)$ as the set of tail normal terms in $\mathcal{CC}^{\eta}_{\beta}(t, \emptyset)$, where $\mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$, with $\mathcal{V} \cap \mathcal{V}ar(t) = \emptyset$, is the smallest set of well-typed terms containing all variables in $\mathcal{V}$, all terms in $\overline{t}$ and $\overline{t}\uparrow^{\neq\Lambda}$, and closed under the following operations:*

1. *restricted subterm of minimal type: let* $s \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$*, and* $u : \sigma = s|_p$ *be a subterm of* s *such that (i) no superterm* $s|_{q<p}$ *of* u *in* s *is of the form* $@(w, v)$ *with* w *a variable or an abstraction, (ii)* $\sigma \in \mathcal{T}^{min}_{\mathcal{S}}$ *and (iii)* $\mathcal{V}ar(u) \subseteq \mathcal{V}ar(t)$*; then* $u \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$*;*
2. *precedence: let* g *such that* $f >_{\mathcal{F}} g$*, and* $\overline{s} \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$*; then* $g(\overline{s}) \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$*;*
3. *recursive call: let* $\overline{s}$ *be a sequence of terms in* $\mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$ *such that the term* $f(\overline{s})$ *is well typed and* $\overline{t}(\succ_{horpo} \cup \rhd_{>_{\mathcal{T}_{\mathcal{S}}}})_{stat_f} \overline{s}$*; then* $g(\overline{s}) \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$ *for every* $g =_{\mathcal{F}} f$*;*
4. *application: let* $s : \sigma_1 \to \ldots \to \sigma_n \to \sigma \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$*, and* $u_i : \sigma_i \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$ *for every* $i \in [1..n]$*; then* $@(s, u_1, \ldots, u_n) \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$*;*
5. *abstraction: let* $x \notin \mathcal{V}ar(t) \cup \mathcal{V}$ *and* $s \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V} \cup \{x\})$*; then* $\lambda x.s \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$*;*
6. *reduction: let* $u \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$*, and* $u \succeq_{horpo} v$*; then* $v \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$*;*
7. *weakening: let* $x \notin \mathcal{V}ar(u, t) \cup \mathcal{V}$*. Then,* $u \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V} \cup \{x\})$ *iff* $u \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$*.*

Lemma 5.5 *Let* $t = f(\overline{t})$*. Then* $\mathcal{CC}^{\eta}_{\beta}(t) \subseteq \mathcal{CC}(t)$*.*

Proof: The proof follows from an easy induction on the definition of $\mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$, using the fact that $(\succ_{horpo})^{\eta}_{\beta}$ is a subrelation of $\succ_{horpo}$. □

We now prove the stability properties of the closure.

Lemma 5.6 *Let* $u \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$ *for some* $\mathcal{V}$*. Then* $u\downarrow$*,* $u\uparrow$*,* $u\uparrow^{\neq\Lambda}$ *and* $u\updownarrow$ *belong to* $\mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$*.*

Proof: For $u\downarrow$, we simply use Case 6, and for $u\uparrow$ and $u\uparrow^{\neq\Lambda}$, we use successively Cases 7, 4 and 5.

Corollary 5.6.1 *Let* $t = f(\overline{t})$ *be a tail expanded term, and* $u \in \overline{t}\uparrow^{\neq\Lambda}$*. Then* $u \in \mathcal{CC}^{\eta}_{\beta}(t)$*.*

Lemma 5.7 *Let* $t = f(\overline{t})$ *be a term and* γ *be a substitution. If* $u \in \mathcal{CC}^{\eta}_{\beta}(t)$ *then* $u\gamma \in \mathcal{CC}^{\eta}_{\beta}(t\gamma)$*.*

Proof: The proof is the same as the proof of Lemma 4.2.

Lemma 5.8 *Let* $t = f(\overline{t})$ *be a term and* γ *be a substitution. If* $u \in \mathcal{CC}^{\eta}_{\beta}(t)$ *then* $u\gamma\downarrow \in \mathcal{CC}^{\eta}_{\beta}(t\gamma\downarrow)$*.*

Proof: Let $\mathcal{V}$ be a set of variables such that $\mathcal{V} \subseteq \mathcal{X} \setminus (\mathcal{V}ar(t) \cup \mathcal{V}ar(t\gamma) \cup \mathcal{D}om(\gamma))$.

We prove that $u \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$ implies $u\gamma\downarrow \in \mathcal{CC}^{\eta}_{\beta}(t\gamma\downarrow, \mathcal{V})$ by induction on the definition of $\mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$. We assume that $\mathcal{D}om(\gamma) \subseteq \mathcal{V}ar(t)$.

If $u \in \mathcal{V}$ then $u\gamma\downarrow = u \in \mathcal{V}$, and hence $u\gamma\downarrow \in \mathcal{CC}^{\eta}_{\beta}(t\gamma\downarrow, \mathcal{V})$. If $u \in \overline{t}$, then $u\gamma\downarrow \in \overline{t}\gamma\downarrow$, and therefore $u\gamma\downarrow \in \mathcal{CC}^{\eta}_{\beta}(t\gamma\downarrow, \mathcal{V})$. Otherwise, we discuss with respect to the cases of definition 5.4, of which Cases 1, 4 and 7 are the only not entirely straightforward ones.

1. If $u : \sigma \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$ by Case 1, then $u = s|_p$ for some $s \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$ with $p \in \mathcal{P}os(s)$, $\sigma \in \mathcal{T}^{min}_{\mathcal{S}}$ and $\mathcal{V}ar(u) \subseteq \mathcal{V}ar(t)$. By induction hypothesis, $s\gamma\downarrow \in \mathcal{CC}^{\eta}_{\beta}(t\gamma\downarrow, \mathcal{V})$. By assumption, no superterm of u in s is of the form $@(w, v)$ with w with w an abstraction or a variable, hence $u\gamma\downarrow = s\downarrow|_p$, with $p \in \mathcal{P}os(s\gamma\downarrow)$, $\mathcal{V}ar(u\gamma) \subseteq \mathcal{V}ar(t\gamma)$, and, by subject reduction property of $\longrightarrow_{\beta}$, $u\gamma : \sigma$. Therefore, $u\gamma\downarrow \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$ by Case 1.
2. If $u \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$ by case 4, then $u = @(s, u_1, \ldots, u_n) \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$, where $s, u_1, \ldots, u_n \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$. By induction hypothesis, $s\gamma\downarrow, u_1\gamma\downarrow, \ldots, u_n\gamma\downarrow \in \mathcal{CC}^{\eta}_{\beta}(t\gamma\downarrow, \mathcal{V})$, and therefore, $@(s\gamma\downarrow, u_1\gamma\downarrow, \ldots, u_n\gamma\downarrow) \in \mathcal{CC}^{\eta}_{\beta}(t\gamma\downarrow, \mathcal{V})$ by Case 4. Using Case 6 repeatedly, $u\gamma\downarrow \in \mathcal{CC}^{\eta}_{\beta}(t\gamma\downarrow, \mathcal{V})$.

3. Case 7 follows from the assumption on the variables in $\mathcal{V}$. □

Lemma 5.9 *Let $t = f(\overline{t})$ be a term and ξ be a type substitution. If $u \in \mathcal{CC}^{\eta}_{\beta}(t)$ then $u\xi \in \mathcal{CC}^{\eta}_{\beta}(t\xi)$.*

Proof: The proof is the same as the proof of Lemma 4.3.

Lemma 5.10 *Let $u \in \mathcal{CC}^{\eta}_{\beta}(f(\overline{t}), \mathcal{V})$. Then $u\uparrow^{\neq\Lambda}, u\uparrow \in \mathcal{CC}^{\eta}_{\beta}(f(\overline{t}\uparrow), \mathcal{V})$.*

Proof: By Lemma 5.6, it suffices to prove it for anyone of $u\uparrow^{\neq\Lambda}, u\uparrow$. We do it by induction on the definition of $\mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$, and proceed with the interesting cases:

1. Base case. If $u : \sigma \in \mathcal{V}$, then, by definition of the closure, $u\uparrow^{\neq\Lambda} = u \in \mathcal{CC}^{\eta}_{\beta}(f(\overline{t}\uparrow), \mathcal{V})$; If $u \in \overline{t}$, then $u\uparrow \in \overline{t}\uparrow$, hence $u\uparrow \in \mathcal{CC}^{\eta}_{\beta}(f(\overline{t}\uparrow), \mathcal{V})$.
2. Case 1. Assume that $s \in \mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$, and let $u : \sigma = s|_p$. By induction hypothesis, $s\uparrow^{\neq\Lambda} \in \mathcal{CC}^{\eta}_{\beta}(f(\overline{t}\uparrow), \mathcal{V})$. Now, $u\uparrow^{\neq\Lambda} = s\uparrow|_q$ for some q which satisfies the same properties as p, and we are done.
3. Case 4. Assume $u, s_1, \ldots, s_n$ are in $\mathcal{CC}^{\eta}_{\beta}(t, \mathcal{V})$. By induction hypothesis, $u\uparrow^{\neq\Lambda}, s_1\uparrow, \ldots, s_n\uparrow$ are in $\mathcal{CC}^{\eta}_{\beta}(f(\overline{t}\uparrow), \mathcal{V})$. By Case 4, $@(u\uparrow^{\neq\Lambda}, s_1\uparrow, \ldots, s_n\uparrow) = @(u, s_1, \ldots, s_n)\uparrow^{\neq\Lambda}$ is in $\mathcal{CC}^{\eta}_{\beta}(f(\overline{t}\uparrow), \mathcal{V})$. □

Since the closure contains only tail normal terms, we get:

Corollary 5.10.1 *Let $u \in \mathcal{CC}^{\eta}_{\beta}(f(\overline{t}))$. Then $u \in \mathcal{CC}^{\eta}_{\beta}(f(\overline{t}\uparrow))$.*

5.3 The Normalized Higher-Order Recursive Path Ordering

Definition 5.11 *Let the proposition A be:*

$$\forall v \in \overline{t}\ s(\underset{chorpo}{\succ})^{\eta}_{\beta}\ v \text{ or } u(\underset{chorpo}{\succeq})^{\eta}_{\beta}\ v \text{ for some } u \in \mathcal{CC}^{\eta}_{\beta}(s)$$

$$s : \sigma(\underset{chorpo}{\succ})^{\eta}_{\beta}\ t : \tau \text{ iff } \sigma \geq_{\mathcal{TS}} \tau \text{ and}$$

1. $s = f(\overline{s})$ with $f \in \mathcal{F}$, and (i) $u(\underset{chorpo}{\succeq})^{\eta}_{\beta}\ t$ for some $u \in \overline{s}$ or (ii) $t \in \mathcal{CC}^{\eta}_{\beta}(s)$
2. $s = f(\overline{s})$ with $f \in \mathcal{F}$ and $t = g(\overline{t})$ with $f >_{\mathcal{F}} g$, and A
3. $s = f(\overline{s})$, $t = g(\overline{t})$ with $f =_{\mathcal{TS}} g \in Mul$, and $\overline{s}((\underset{chorpo}{\succ})^{\eta}_{\beta})_{mul}\overline{t}$
4. $s = f(\overline{s})$, $t = g(\overline{t})$ with $f =_{\mathcal{TS}} g \in Lex$ and $\overline{s}((\underset{chorpo}{\succ})^{\eta}_{\beta})_{lex}\overline{t}$, and A
5. $s = @(s_1, s_2)$, $s_1 \notin \mathcal{X}$ and $s_1(\underset{chorpo}{\succeq})^{\eta}_{\beta}\ t$ or $s_2(\underset{chorpo}{\succeq})^{\eta}_{\beta}\ t$
6. $s = \lambda x : \alpha.u$ with $x \notin \mathcal{V}ar(t)$, and $u(\underset{chorpo}{\succeq})^{\eta}_{\beta}\ t$
7. $s = f(\overline{s})$ with $f \in \mathcal{F}$, $t = @(\overline{t})$ is a partial 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 $s(\underset{chorpo}{\succeq})^{\eta}_{\beta}\ v$

9. $s = @(s_1, s_2)$, $t = @(\overline{t})$ *is a partial left-flattening of* t, $s_1 \notin \mathcal{X}$ *and* $\{s_1, s_2\}((\underset{chorpo}{\succ})^{\eta}_{\beta})_{mul}\{\overline{t}\}$

10. $s = \lambda x : \alpha.u,\ t = \lambda x : \beta.v,\ \alpha =_{\mathcal{TS}} \beta$, *and* $u(\underset{chorpo}{\succeq})^{\eta}_{\beta}\ v$

11. $s = @(\lambda x : \alpha.u, v)$ *and* $u\{x \mapsto v\}(\underset{chorpo}{\succeq})^{\eta}_{\beta}\ t$

12. $t = \lambda x.@(v, x)$ *with* $x \notin \mathcal{V}ar(v)$ *and* $s(\underset{chorpo}{\succ})^{\eta}_{\beta}\ v$

We start now proving its closure properties with respect to term and type instantiation. We start with polymorphism, whose proof is quite different from the proof of polymorphism for plain higher-order rewriting, and much more complicated. As a preparation, we study the behaviour of the ordering with respect to η-expansions.

Lemma 5.12 *Assume that* $s(\succeq_{chorpo})^{\eta}_{\beta}\ t$*, where* s *is tail* η*-expanded. Then* t *is tail* η*-expanded.*

Proof: By inspecting all cases in the definition. □

Lemma 5.13 *Assume* $s(\succ_{chorpo})^{\eta}_{\beta}\ t : \sigma$*, where* $\sigma = \overline{\sigma} \to \tau$ *is canonical. Then* $s(\succ_{chorpo})^{\eta}_{\beta}\ \lambda\overline{x} : \overline{\sigma}.@(t, \overline{x})$.

Note that the form of the type $\overline{\sigma} \to \tau$ is the most general possible.
Proof: By induction on the size of $\overline{\sigma}$ and applications of Case 12 of the definition. □

Corollary 5.13.1 *Assume* $s(\succ_{chorpo})^{\eta}_{\beta}\ t$ *for some tail expanded terms* s, t*. Then* $s(\succ_{chorpo})^{\eta}_{\beta}\ t\uparrow$.

Lemma 5.14 *Assume that* $s : \sigma(\succeq_{chorpo})^{\eta}_{\beta}\ t : \sigma'$*, and* σ *is the canonical type* $\sigma_1 \to \ldots \sigma_n \to \tau$*. Then* $\lambda\overline{x} : \overline{\sigma}.@(s, \overline{x})(\succeq_{chorpo})^{\eta}_{\beta}\ t$.

Proof: By induction on n. If $n = 0$, we are done. Otherwise, by Case 5, $@(s, x_n)(\succeq_{chorpo})^{\eta}_{\beta}\ t$ for some $x_n : \sigma_n \notin \mathcal{V}ar(t)$, and therefore, by Case 6, we get $\lambda x_n : \sigma_n.@(s, x_n)(\succeq_{chorpo})^{\eta}_{\beta}\ t$, and we can now conclude by induction hypothesis. □

Corollary 5.14.1 *Assume* $s(\succeq_{chorpo})^{\eta}_{\beta}\ t$ *for some tail expanded terms* s, t*. Then* $s \uparrow (\succeq_{chorpo})^{\eta}_{\beta}\ t$.

Lemma 5.15 *Assume that* $s : \sigma(\succeq_{chorpo})^{\eta}_{\beta}\ t : \sigma'$*,* σ *is the canonical type* $\sigma_1 \to \ldots \sigma_n \to \tau$ *and* σ' *is the canonical type* $\sigma'_{i_1} \to \ldots \sigma_{i_p} \to \tau_1 \to \ldots \to \tau_q \to \tau'$ *with*
(i) $1 \leq i_1 < \ldots < i_p \leq n$,
(ii) $\sigma'_{i_k} =_{\mathcal{TS}} \sigma_{i_k}$ *for every* $k \in [1..p]$,
(iii) $\tau \geq_{\mathcal{TS}} \tau_1 \to \ldots \to \tau_q \to \tau'$.
Then $\lambda\overline{x} : \overline{\sigma}.@(s, \overline{x})(\succeq_{chorpo})^{\eta}_{\beta}\ \lambda\overline{y} : \overline{\sigma'}\lambda\overline{z} : \overline{\tau}.@(t, \overline{y}, \overline{z})$.

Note that the form of the type σ' is the most general possible, since it follows from the fact that $\sigma \geq_{\mathcal{TS}} \sigma'$ which itself follows from the assumption that $s : \sigma(\succeq_{chorpo})^{\eta}_{\beta}\ t : \sigma'$. Besides, in case s and t are tail expanded (resp. tail normal), then $\lambda\overline{x} : \overline{\sigma}.@(s, \overline{x})$ and $\lambda\overline{y} : \overline{\sigma'}\lambda\overline{z} : \overline{\tau}.@(t, \overline{y}, \overline{z})$ are η-expended (resp. normalized).

Proof: First of all, in case s is a variable X, then $t = X$ and the result is clear. Otherwise, we discuss by induction on $n + p + q$ and by cases on their respective values.

1. Assume that $q \neq 0$. Then, by Case 12 applied to the assumption, we get $s : \sigma(\succeq_{chorpo})^{\eta}_{\beta}\ \lambda z_q.@(t, z_q)$, and we can conclude by the induction hypothesis.

2. Assume that $q = 0$ and $p = 0$. Then $q = 0$ and we are done.

3. Assume that $q = 0$, $p \neq 0$ and $i_p = n$. Then, by Case 10 applied to the assumption (and assuming without loss of generality that $x_n = y_{i_p}$), we get $\lambda x_n.@(s, x_n)(\succeq_{chorpo})^{\eta}_{\beta} \lambda y_{i_p}.@(t, y_{i_p})$, and we can conclude again by the induction hypothesis.

4. Assume that $q = 0$, $p \neq 0$ and $i_p \neq n$. Since s is not a variable, $@(s, x_n)(\succeq_{chorpo})^{\eta}_{\beta} t$ for some $x_n : \sigma_n \notin \mathcal{V}ar(t)$ by Case 5, and therefore, by Case 6, $\lambda x_n.@(s, x_n)(\succeq_{chorpo})^{\eta}_{\beta} t$, and we can conclude again by the induction hypothesis. Note that the typing conditions in order to apply the induction hypothesis are all fullfilled. □

Corollary 5.15.1 *Assume $s(\succeq_{chorpo})^{\eta}_{\beta} t$ for some tail expanded terms s, t. Then $s \uparrow (\succeq_{chorpo})^{\eta}_{\beta} t \uparrow$.*

Lemma 5.16 *Assume that $s(\succ_{chorpo})^{\eta}_{\beta} t$. Then $s\uparrow^{\neq\Lambda} (\succ_{chorpo})^{\eta}_{\beta} t\uparrow^{\neq\Lambda}$.*

Proof: The proof that $s\uparrow^{\neq\Lambda} (\succeq_{chorpo})^{\eta}_{\beta} t\uparrow^{\neq\Lambda}$ is by induction on the definition of the ordering.

First, we remark that property $A : \forall v \in \overline{t}$ either (i) $f(\overline{s})(\succeq_{chorpo})^{\eta}_{\beta} v$ or (ii) $u(\succeq_{chorpo})^{\eta}_{\beta} v$ for some (tail normal) $u \in \mathcal{CC}^{\eta}_{\beta}(f(\overline{s}))$ implies property $A \uparrow$: $\forall v \in \overline{t} \uparrow$ either (i) $f(\overline{s} \uparrow)(\succeq_{chorpo})^{\eta}_{\beta} v$ or (ii) $u(\succeq_{chorpo})^{\eta}_{\beta} v$ for some (tail normal) $u \in \mathcal{CC}^{\eta}_{\beta}(f(\overline{s} \uparrow))$.

For (i), we get $f(\overline{s} \uparrow)(\succeq_{chorpo})^{\eta}_{\beta} v\uparrow^{\neq\Lambda}$ by induction hypothesis, and therefore $f(\overline{s} \uparrow)(\succeq_{chorpo})^{\eta}_{\beta} v \uparrow$ by Corollary 5.13.1.

For (ii), we get $u\uparrow^{\neq\Lambda} = u(\succeq_{chorpo})^{\eta}_{\beta} v\uparrow^{\neq\Lambda}$ by induction hypothesis, and therefore $u(\succeq_{chorpo})^{\eta}_{\beta} v \uparrow$ by Corollary 5.13.1. We conclude by Corollary 5.10.1.

Property $A \uparrow$ assumes that v is always fully expanded. This is actually not the case when it is used in Case 7, because the first argument of a fully expanded application is tail expanded. There is no difficulty in proving this slightly modified form of $A \uparrow$, and we will use it without mentioning further this subtlety.

1. Case 1 (i). By induction hypothesis, $u\uparrow^{\neq\Lambda} (\succeq_{horpo})^{\eta}_{\beta} t\uparrow^{\neq\Lambda}$. By Corollary 5.14.1, $u \uparrow (\succeq_{horpo})^{\eta}_{\beta} t\uparrow^{\neq\Lambda}$, and therefore, $f(\overline{s} \uparrow)(\succeq_{chorpo})^{\eta}_{\beta} t\uparrow^{\neq\Lambda}$ by Case 1 (i).

   Case 1 (ii). By Lemma 5.10, $t\uparrow^{\neq\Lambda} = t \in \mathcal{CC}^{\eta}_{\beta}(f(\overline{s} \uparrow))$, hence $f(\overline{s} \uparrow)(\succeq_{chorpo})^{\eta}_{\beta} t\uparrow^{\neq\Lambda}$ by Case 1 (ii).

2. Case 2. By the above property $A \uparrow$, we get $f(\overline{s} \uparrow)(\succ_{chorpo})^{\eta}_{\beta} g(\overline{t} \uparrow)$, which is our goal.

3. Case 3. By induction hypothesis, Corollary 5.15.1 and Case 3.

4. Case 4. By induction hypothesis, property $A \uparrow$, Corollary 5.15.1 and Case 4.

5. Case 5. By applying successively the induction hypothesis to $s_1(\succeq_{chorpo})^{\eta}_{\beta} t$, or the induction hypothesis to $s_2(\succeq_{chorpo})^{\eta}_{\beta} t$ followed by Corollary 5.14.1, and then Case 5.

6. Case 6. By induction hypothesis and Case 6.

7. Case 7. By property $A \uparrow$ and Case 7.

8. Case 8. By induction hypothesis and Case 8.

9. Case 9. By induction hypothesis, Corollarys 5.14.1, 5.13.1 and 5.15.1 when needed, and Case 9.

10. Case 10. By induction hypothesis and Case 10.

11. Case 11. By induction hypothesis $u\{x \mapsto v\}\uparrow^{\neq\Lambda} (\succeq_{chorpo})^{\eta}_{\beta} t\uparrow^{\neq\Lambda}$. Since $u\{x \mapsto v\}\uparrow^{\neq\Lambda} = u\uparrow^{\neq\Lambda} \{x \mapsto v\uparrow^{\neq\Lambda}\}$, we get $@(\lambda x : \sigma.u\uparrow^{\neq\Lambda}, v\uparrow^{\neq\Lambda}) = s\uparrow^{\neq\Lambda} (\succeq_{horpo})^{\eta}_{\beta} t\uparrow^{\neq\Lambda}$ by Case 11.

12. Case 12. By induction hypothesis, $s\!\uparrow^{\neq\Lambda} (\succeq_{chorpo})^{\eta}_{\beta} v\!\uparrow^{\neq\Lambda} = t\!\uparrow^{\neq\Lambda}$ and we are done. □

Lemma 5.17 *Let $s(\succeq_{horpo})^{\eta}_{\beta} t$ and ξ be a type substitution. Then $s\xi \succeq_{chorpo} t\xi$.*

Proof: The proof is the same as the proof of Lemma 3.10, using now Lemma 5.9. □

Theorem 5.18 *The ordering $(\succeq_{chorpo})^{\eta}_{\beta}$ is η-polymorphic.*

Proof: By Lemmas 5.17 and 5.16.
We are left with β-stability. In order to prove it, we show a technical property characterizing the exact behaviour of the ordering with respect to instantiations:

Lemma 5.19 *Let s and t be tail normal candidate terms. If $s\ (\succ_{chorpo})^{\eta}_{\beta}\ t$ then for all tail normal substitutions γ there is a candidate term w such that $s\gamma\downarrow\ (\succ_{chorpo})^{\eta}_{\beta} w \longrightarrow^{*}_{\beta} t\gamma\downarrow$.*

Proof: By induction on the definition of $(\succ_{chorpo})^{\eta}_{\beta}$, and by cases according to the definition.
First, Proposition $A:\ \forall v \in \overline{t}\ s(\underset{chorpo}{\succ})^{\eta}_{\beta} v$ or $u(\underset{chorpo}{\succeq})^{\eta}_{\beta} v$ for some $u \in \mathcal{CC}^{\eta}_{\beta}(s)$ implies Proposition

$A\gamma\downarrow$: $\forall v_i = t_i\gamma\downarrow\ \exists w_i$ such that $w_i \longrightarrow^{*}_{\beta} t_i\gamma\downarrow$ and $\begin{cases} \text{(i)} & s\gamma\downarrow\ (\underset{chorpo}{\succ})^{\eta}_{\beta} w_i, \text{ or} \\ \text{(ii)} & u(\underset{chorpo}{\succeq})^{\eta}_{\beta} w_i \text{ for some } u \in \mathcal{CC}^{\eta}_{\beta}(s\gamma\downarrow) \end{cases}$

by an easy use of the induction hypothesis, noting that terms from the closure are tail normal.

1. Case 1. Let $s = f(\overline{s})(\succ_{chorpo})^{\eta}_{\beta} t$ with $f \in \mathcal{F}$, and
   - (i) $u(\succeq_{chorpo})^{\eta}_{\beta} t$ for some $u \in \overline{s}$. By induction hypothesis, there exists a term w such that $u\gamma\downarrow\ (\succeq_{chorpo})^{\eta}_{\beta} w \longrightarrow^{*}_{\beta} t\gamma\downarrow$. By Case 1 (i), we get $s\gamma\downarrow\ (\succ_{chorpo})^{\eta}_{\beta} w \longrightarrow^{*}_{\beta} t\gamma\downarrow$.
   - (ii) $t \in \mathcal{CC}^{\eta}_{\beta}(s)$. By lemma 5.8, $t\gamma\downarrow \in \mathcal{CC}^{\eta}_{\beta}(s\gamma\downarrow)$, hence $s\gamma\downarrow\ (\succ_{chorpo})^{\eta}_{\beta} t\gamma\downarrow$ by Case 1 (ii) and we can take $w = t\gamma\downarrow$.
2. Case 2. Let $s = f(\overline{s})(\succ_{chorpo})^{\eta}_{\beta} t = g(\overline{t})$ with $f, g \in \mathcal{F}$, $f >_{\mathcal{F}} g$, and A. By Proposition $A\gamma\downarrow$, taking $w = g(\overline{w})$ yields $g(\overline{w}) \longrightarrow^{*}_{\beta} t\gamma\downarrow$. Now, $s\gamma\downarrow = f(\overline{s\gamma\downarrow})(\succeq_{chorpo})^{\eta}_{\beta} g(\overline{w})$ by Case 2, because $f >_{\mathcal{F}} g$ and for every $w \in \overline{w}$, either (i) $s\gamma\downarrow\ (\succ_{chorpo})^{\eta}_{\beta} w$ or (ii) $u(\succeq_{chorpo})^{\eta}_{\beta} w$ for some $u \in \mathcal{CC}^{\eta}_{\beta}(s\gamma\downarrow)$.
3. Case 3. If $s = f(\overline{s})\ (\succ_{chorpo})^{\eta}_{\beta}\ t = g(\overline{t})$ with $f =_{\lambda\mathcal{F}} g \in Mul$ and $\overline{s}((\succeq_{chorpo})^{\eta}_{\beta})_{mul}\overline{t}$. By induction hypothesis, there is some multiset $\overline{w}$ such that $\overline{s}\gamma\downarrow\ (\succeq_{chorpo})_{mul}\overline{w}(\longrightarrow^{*}_{\beta})_{mon}\overline{t}\gamma\downarrow$. Taking $w = g(\overline{w})$, we get $s\gamma\downarrow\ (\succ_{chorpo})^{\eta}_{\beta} w \longrightarrow^{*}_{\beta} t$ by Case 3.
4. Case 4. Let $s = f(\overline{s})(\succ_{chorpo})^{\eta}_{\beta}\ t = g(\overline{t})$ with $f =_{\lambda\mathcal{F}} g \in Lex$, $\overline{(}s)((\succeq_{chorpo})^{\eta}_{\beta})_{lex}\overline{(}t)$ and A. By definition of the lexicographic extension, there exists some index k such that $s_i = t_i$ for $i \in [1..k-1]$ and $s_k(\succ_{chorpo})^{\eta}_{\beta} t_k$. We therefore set $w_i = s_i\gamma\downarrow$ for $i \in [1..k-1]$, w_k is defined by the induction hypothesis, and the rest of them by Property $A\gamma\downarrow$. As a consequence, we get $\overline{s}\gamma\downarrow\ ((\succeq_{chorpo})^{\eta}_{\beta})_{lex}\overline{w}(\longrightarrow^{*}_{\beta})_{mon}\overline{t}\gamma\downarrow$, and property A is true of $\overline{w}$. Taking $w = g(\overline{w})$, we get $s\gamma\downarrow\ (\succ_{chorpo})^{\eta}_{\beta} w \longrightarrow^{*}_{\beta} t$ by Case 4.
5. Case 5. Let $s = @(s_1, s_2)(\succ_{chorpo})^{\eta}_{\beta} t$, $s_1 \notin \mathcal{X}$ and $s_1(\succeq_{chorpo})^{\eta}_{\beta} t$. By assumption on s_1, $s_1\gamma\downarrow$ cannot be a variable. By assumption on s, it cannot be an abstraction either. Therefore, $s\gamma\downarrow = @(s_1\gamma\downarrow, s_2\gamma\downarrow)$.

   By induction hypothesis, there is some w such that $s_1\gamma\downarrow\ (\succeq_{chorpo})^{\eta}_{\beta} w \longrightarrow^{*}_{\beta} t\gamma\downarrow$, or $s_1\gamma\downarrow\ (\succeq_{chorpo})^{\eta}_{\beta} w \longrightarrow^{*}_{\beta} t\gamma\downarrow$ Since $s_1\gamma\downarrow$ is not a variable, we get $s\gamma\downarrow = @(s_1\gamma\downarrow, s_2\gamma\downarrow)(\succ_{chorpo})^{\eta}_{\beta} w \longrightarrow^{*}_{\beta} t\gamma\downarrow$ by Case 5 and we are done.

6. Case 6. Let $s = \lambda x : \alpha.u(\succ_{chorpo})^{\eta}_{\beta} t$ with $x \notin \mathcal{V}ar(t)$, and $u(\succeq_{chorpo})^{\eta}_{\beta} t$. Since $s\gamma\downarrow = \lambda x : \alpha.u\gamma\downarrow$ with $x \notin \mathcal{V}ar(t\gamma\downarrow)$, and, by induction, there is some w s.t. $u\gamma\downarrow\ (\succ_{chorpo})^{\eta}_{\beta} w \longrightarrow^*_{\beta} t\gamma\downarrow$, we can conclude by case 6 that $\lambda x : \alpha.u\gamma\downarrow\ (\succ_{chorpo})^{\eta}_{\beta} w \longrightarrow^*_{\beta} t\gamma\downarrow$.

7. Case 7. Let $s = f(\overline{s})(\succ_{chorpo})^{\eta}_{\beta} t = @(\overline{t})$ with $f \in \mathcal{F}$, $@(\overline{t})$ is a partial left-flattening, and A. Mimicking the proof of case 2, we get $s\gamma\downarrow\ (\succ_{chorpo})^{\eta}_{\beta} w \longrightarrow^*_{\beta} @((\overline{t})\gamma\downarrow)$ for some w. Since $t\gamma\downarrow = @((\overline{t})\gamma\downarrow)\downarrow$, we get $w \longrightarrow^*_{\beta} t\gamma\downarrow$, and we are done.

8. Case 8. Let $s = f(\overline{s})(\succ_{chorpo})^{\eta}_{\beta} t = \lambda x : \alpha.v$ with $f \in \mathcal{F}$, $x \notin \mathcal{V}ar(v)$ and $s(\succeq_{chorpo})^{\eta}_{\beta} v$. Since $t\gamma\downarrow = \lambda x : \alpha.v\gamma\downarrow$ with $x \notin \mathcal{V}ar(v\gamma\downarrow)$, by induction hypothesis, there is some w such that $s\gamma\downarrow\ (\succ_{chorpo})^{\eta}_{\beta} w \longrightarrow^*_{\beta} v\gamma\downarrow$. We can conclude by Case 8 that $\lambda x : \alpha.u\gamma\downarrow\ (\succ_{chorpo})^{\eta}_{\beta} w \longrightarrow^*_{\beta} t\gamma\downarrow$.

9. Case 9. Let $s = @(s_1, s_2)(\succ_{chorpo})^{\eta}_{\beta} t = @(\overline{t})$ with $s_1 \notin \mathcal{X}$, $\overline{t}$ is a partial left-flattening and $\{s_1, s_2\}((\succeq_{chorpo})^{\eta}_{\beta})_{mul}\{\overline{t}\}$. Since $s_1 \notin \mathcal{X}$, $s_1\gamma$ is not a variable either, nor an abstraction by assumption on s. Therefore, $s\gamma\downarrow = @(s_1\gamma\downarrow, s_2\gamma\downarrow)$. By induction hypothesis there is a term $w = @(\overline{w})$ such that $\{s_1\gamma\downarrow, s_2\gamma\downarrow\}((\succeq_{chorpo})^{\eta}_{\beta})_{mul}\overline{w}$ and $@(\overline{w}) \longrightarrow^*_{\beta} @(\overline{t\gamma}\downarrow)$. Therefore, by case 9, $s\gamma\downarrow\ (\succ_{chorpo})^{\eta}_{\beta} w$. Since $w \longrightarrow^*_{\beta} @(\overline{t\gamma}\downarrow) \longrightarrow^*_{\beta} t\gamma\downarrow$, the result holds.

10. Case 10. Let $s = \lambda x : \alpha.u(\succ_{chorpo})^{\eta}_{\beta} t = \lambda x : \beta.v$ with $\alpha =_{\mathcal{TS}} \beta$, and $u(\succeq_{chorpo})^{\eta}_{\beta} v$. Since $s\gamma\downarrow = \lambda x : \alpha.u\gamma\downarrow$ and $t\gamma\downarrow = \lambda x : \beta.v\gamma\downarrow$, by induction hypothesis, there is some w such that $u\gamma\downarrow\ (\succ_{chorpo})^{\eta}_{\beta} w \longrightarrow^*_{\beta} v\gamma\downarrow$. Therefore, by case 10, $s\gamma\downarrow\ (\succ_{chorpo})^{\eta}_{\beta} \lambda x : \beta.w \longrightarrow^*_{\beta} t\gamma\downarrow$.

11. Case 11 does not apply since s is tail normal by assumption.

12. Case 12. Let $t = \lambda x : \sigma.@(v, x)$ with $x \notin \mathcal{V}ar(v)$ and $s(\succ_{chorpo})^{\eta}_{\beta} v$. By induction hypothesis, there exists some term w such that $s\gamma\downarrow\ (\succ_{chorpo})^{\eta}_{\beta} w \longrightarrow^*_{\beta} v\gamma\downarrow$. Without loss of generality, we can assume that $x \notin \mathcal{V}ar(v\gamma)$, and therefore, by Case 12, $s\gamma\downarrow\ (\succ_{chorpo})^{\eta}_{\beta} \lambda x : \sigma.@(w, x) \longrightarrow^*_{\beta} \lambda x : \sigma.@(v\gamma\downarrow, x)$. Additional β-rewrite steps if necessary yield the result. □

Lemma 5.20 *$((\succ_{chorpo})^{\eta}_{\beta})^+$ is a β-stable subrelation of $(\succ_{chorpo})^+$.*

Proof: By induction on the length of the derivation. Assume that $s(\succ_{chorpo})^{\eta}_{\beta} u((\succ_{chorpo})^{\eta}_{\beta})^* t$. By lemma 5.19, $s\gamma\downarrow\ (\succ_{chorpo})^+ u\gamma\downarrow$, since β-reduction is included in $\succ_{chorpo}$. If $u = t$, we are done. Otherwise, by induction hypothesis, $u\gamma\downarrow\ (\succ_{chorpo})^+ t\gamma\downarrow$ and hence, $s\gamma\downarrow\ (\succ_{chorpo})^+ t\gamma\downarrow$. □

We can finally conclude:

Theorem 5.21 *$(\succ_{chorpo})^{\eta}_{\beta}$ is included in a polymorphic normalized higher-order reduction ordering.*

6 Neutralizing abstractions

In our ordering, all arrow type terms are treated as if they could become applied and serve in a β-reduction. This makes it difficult to prove the termination of rules whose righthand side has arrow type subterms which do not occur as lefthand side arguments. In such a case, the use of the computable closure may sometimes help, but Example 13 shows that this is not always the case. In this example, the righthand side arrow type subterm $\lambda y.P(y) \cdot x$ does not receive a special treatment, although it cannot serve creating a redex in a derivation. We will now improve our ordering by introducing a special treatment for these terms. The idea is to equip every function symbol with a set of neutralized positions, which can be seen as an additional status for that purpose.

Notation: In this section, we assume given a new constant $\bot_\sigma$ for every type σ. We will denote by $\mathcal{F}_{new}$ the augmented signature. Because $\bot_\sigma$ has type σ which may be polymorphic, there is no

garantee that $\mathcal{F}_{new}$ is regular. We already discussed this issue in Section 2, and noted that the unique typing property was preserved. Note also that for any type instantiation ξ and any environment Γ, $\Gamma \vdash_{\mathcal{F}_{new}} (\bot_\sigma)\xi : \tau$ iff $\Gamma \vdash_{\mathcal{F}_{new}} \bot_{\sigma\xi} : \tau$. In the sequel, we will actually identify $(\bot_\sigma)\xi$ with the term $\bot_{\sigma\xi}$, and assume that the precedence $\geq_{\mathcal{F}}$ on the extended signature has the following stability property: $f \geq_{\mathcal{F}} g$ implies $f\xi \geq_{\mathcal{F}} g\xi$ for every type instantiation ξ.

Of course, the higher-ordre rules we want to prove terminah ting are built from terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, not in $\mathcal{T}(\mathcal{F}_{new}, \mathcal{X})$.

6.1 Neutralization and Normalization

Definition 6.1 *The* i-neutralization, *or neutralization of level* i *of a term* $t \in \mathcal{T}(\mathcal{F}_{new}, \mathcal{X})$ *is the term* $\mathcal{N}_i(t)$ *defined as follows :*

1. *if* $i > 0$ *and* t *is of the form* $\lambda x : \sigma.u$ *then* $\mathcal{N}_i(t) = \mathcal{N}_{i-1}(u\{x \mapsto \bot_\sigma\})$*;*
2. *Otherwise* $\mathcal{N}_i(t) = t$*.*

Lemma 6.2 *Let* t *be an* η*-expanded term and* γ *be a substitution. Then* $\mathcal{N}_i(t\gamma) = \mathcal{N}_i(t)\gamma$ *for all* i*.*

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

1. Assume $i > 0$ and t is of the form $\lambda x : \sigma.u$. Without loss of generality, we can further assume that $x \notin \mathcal{D}om(\gamma)$. Then $\mathcal{N}_i(t\gamma) = \mathcal{N}_i((\lambda x : \sigma.u)\gamma) = \mathcal{N}_i(\lambda x : \sigma.u\gamma) = \mathcal{N}_{i-1}(u\gamma\{x \mapsto \bot_\sigma\}) = \mathcal{N}_{i-1}(u\gamma')$ where $\gamma' = \gamma \cup \{x \mapsto \bot_\sigma\}$. By induction hypothesis, $\mathcal{N}_{i-1}(u\gamma') = \mathcal{N}_{i-1}(u)\gamma' = \mathcal{N}_{i-1}(u)\{x \mapsto \bot_\sigma\}\gamma = \mathcal{N}_i(t)\gamma$.
2. Otherwise, $\mathcal{N}_i(t) = t$. Since t is η-expanded, it cannot be a variable of a functionnal type. Hence $t\gamma$ cannot be an abstraction unless $i = 0$, and therefore $\mathcal{N}_i(t\gamma) = t\gamma = \mathcal{N}_i(t)\gamma$ by definition. □

Lemma 6.3 *Let* t *be a normalized term and* γ *be a tail normal substitution. Then* $\mathcal{N}_i(t\gamma\downarrow) = \mathcal{N}_i(t)\gamma\downarrow$*.*

Proof: We proceed by induction on i.

1. If $i > 0$ and t is of the form $\lambda x : \sigma.u$ then $\mathcal{N}_i(t) = \mathcal{N}_{i-1}(u\{x \mapsto \bot_\sigma\})$; by induction hypothesis $\mathcal{N}_{i-1}(u\{x \mapsto \bot_\sigma\})\gamma\downarrow = \mathcal{N}_{i-1}(u\{x \mapsto \bot_\sigma\}\gamma\downarrow)$. Thus, we have $\mathcal{N}_i(t)\gamma\downarrow = \mathcal{N}_{i-1}(u\{x \mapsto \bot_\sigma\}\gamma\downarrow)$, and, since, we assume that x does not occur in γ, we have $\mathcal{N}_i(t)\gamma\downarrow = \mathcal{N}_{i-1}((u\gamma)\{x \mapsto \bot_\sigma\}\downarrow) = \mathcal{N}_i(t)\gamma\downarrow = \mathcal{N}_{i-1}((u\gamma\downarrow)\{x \mapsto \bot_\sigma\})$, and therefore $\mathcal{N}_i(t)\gamma\downarrow = \mathcal{N}_i(\lambda x : \sigma.u\gamma\downarrow) = \mathcal{N}_i((\lambda x : \sigma.u)\gamma\downarrow) = \mathcal{N}_i(t\gamma\downarrow)$.
2. Otherwise, t is not an abstraction unless $i = 0$, and hence $\mathcal{N}_i(t) = t$. Since t is normalized, it cannot be of a functionnal type, unless $i = 0$, and since t and $t\gamma$ have the same type in any environnement, $t\gamma\downarrow$ is not an abstraction either, unless $i = 0$. Therefore, $\mathcal{N}_i(t\gamma\downarrow) = t\gamma\downarrow = \mathcal{N}_i(t)\gamma\downarrow$. □

Since we want to neutralize only abstractions in terms, or terms that can become abstractions along reductions, we need to control neutralization via the function symbols heading these abstractions by neutralizing some of their arguments whose type is functional.

Definition 6.4 *To each symbol* $f : \sigma_1 \times \ldots \times \sigma_n \to \sigma \in \mathcal{F}$ *and each argument position* j *of* f*, we associate a natural number* $\mathcal{NL}_j^f \leq ar(\sigma_j)$*, called* neutralization level *of* f *at position* j*. We call* neutralized *those positions* j *with* $\mathcal{NL}_j^f > 0$*.*

We will now neutralize terms recursively. To this end, we need introducing new function symbols in the signature : for every algebraic symbol declaration $f : \sigma_1 \times \ldots \times \sigma_n \rightarrow \sigma$, we assume given a new symbol $f_{new} : \sigma'_1 \times \ldots \times \sigma'_n \rightarrow \sigma$ in the new neutralized signature. The type declaration of f_{new} symbol depends only upon the respective neutralization levels of the argument positions of f : if $\sigma_i = \tau_1 \rightarrow \ldots \rightarrow \tau_k \rightarrow \tau$ and $\mathcal{NL}_i^f = q \leq k$, then $\sigma'_i = \tau_{q+1} \rightarrow \ldots \rightarrow \tau_k \rightarrow \tau$. Besides, we assume that the extended signature is again regular, which is true when the neutralization level of a given argument position of the function symbol f is the same for all its type declarations.

Definition 6.5 *A term t is said to be* algebraically expanded *if it satisfies the following property : for all subterms of t of the form $f(u_1, \ldots, u_n)$, u_i is η-expanded for every i.*

The full neutralization *of an algebraically expanded term t is the term $\mathcal{FN}(t)$ defined as:*

1. *if $t \in \mathcal{X}$ or $t = \bot_\sigma$, then $\mathcal{FN}(t) = t$;*
2. *if $t = \lambda x.u$, then $\mathcal{FN}(t) = \lambda x.\mathcal{FN}(u)$;*
3. *if $t = @(t_1, \ldots, t_n)$, then $\mathcal{FN}(t) = @(\mathcal{FN}(t_1), \ldots, \mathcal{FN}(t_n))$;*
4. *if $t = f(t_1, \ldots, t_n)$ with $f \in \mathcal{F}$, then $\mathcal{FN}(t) = f_{new}(\mathcal{N}_{\mathcal{NL}_1^f}(\mathcal{FN}(t_1)), \ldots, \mathcal{N}_{\mathcal{NL}_n^f}(\mathcal{FN}(t_n)))$.*

Let us first remark that our definition makes sense, since :

- the term constructed in Case 4 is typable, thanks to the assumption that the arguments of an algebraic symbol are η-expanded, therefored ensuring that the type of f_{new} depends only upon the type of the arguments of f and its neutralization level, and not upon the form of the arguments themselves in case they have a functional type.

- the full normalization of an algebraically expanded term is itself algebraically expanded.

In the following, we are primarily interested in the full normalization of tail expanded (or tail normal) terms. Note first that a tail expanded term s is algebraically expanded, and that $\mathcal{FN}(s)$ is indeed itself tail expanded.

Lemma 6.6 *Let u be a tail expanded term and γ be a tail expanded substitution.*
Then $\mathcal{FN}(u\gamma) = \mathcal{FN}(u)\mathcal{FN}(\gamma)$.

Proof: We proceed by induction on $|u|$ and case analysis.

1. Assume that $\mathcal{D}om(\gamma) \cap \mathcal{V}ar(u) = \emptyset$. Then $u\gamma = u$ and since $\mathcal{D}om(\gamma) \cap \mathcal{V}ar(u) = \emptyset$, the result holds trivially.
2. Assume $u \in \mathcal{D}om(\gamma)$. Since $\mathcal{FN}(u) = u$, the result holds again.
3. Assume $u = \lambda y.w$. Then, $\mathcal{FN}((\lambda y.w)\gamma) = \mathcal{FN}(\lambda y.(w\gamma)) = \lambda y.\mathcal{FN}(w\gamma)$. By induction hypothesis, $\lambda y.\mathcal{FN}(w\gamma) = \lambda y.(\mathcal{FN}(w)\mathcal{FN}(\gamma)) = (\lambda y.\mathcal{FN}(w))\mathcal{FN}(\gamma) = \mathcal{FN}(u)\mathcal{FN}(\gamma)$.
4. Assume $u = f(u_1, \ldots, u_n)$. Then, $\mathcal{FN}(u\gamma) = \mathcal{FN}(f(u_1\gamma, \ldots, u_n\gamma)) = f_{new}(u'_1, \ldots, u'_n)$, where $u'_j = \mathcal{N}_{\mathcal{NL}_j^f}(\mathcal{FN}(u_j\gamma))$. By induction hypothesis $\mathcal{FN}(u_j\gamma) = \mathcal{FN}(u_j)\mathcal{FN}(\gamma)$. Since u is tail expanded, all u_j are η-expanded, and therefore $\mathcal{FN}(u_j)$ is η-expanded as well. By lemma 6.2, we have $\mathcal{N}_{\mathcal{NL}_j^f}(\mathcal{FN}(u_j)\gamma) = \mathcal{N}_{\mathcal{NL}_j^f}(\mathcal{FN}(u_j))\mathcal{FN}(\gamma)$ and the result holds.
5. Assume $u = @(w_1, w_2)$. Then, $\mathcal{FN}(@(w_1, w_2)\gamma) = \mathcal{FN}(@(w_1\gamma, w_2\gamma)) = @(\mathcal{FN}(w_1\gamma), \mathcal{FN}(w_2\gamma))$. By induction hypothesis, we get $\mathcal{FN}(@(w_1, w_2)\gamma) = @(\mathcal{FN}(w_1)\mathcal{FN}(\gamma), \mathcal{FN}(w_2)\mathcal{FN}(\gamma)) = @(\mathcal{FN}(w_1), \mathcal{FN}(w_2))\mathcal{FN}(\gamma) = \mathcal{FN}(u)\mathcal{FN}(\gamma)$ and we are done. □

We now show that full normalization preserves all kinds of normal forms, as can be expected from a transformation that drops abstractions and replaces their bound variables by bottom constants in the bodies. The first two lemmas are easy.

Lemma 6.7 *Let t be a normalized term. Then $\mathcal{FN}(t)\downarrow = \mathcal{FN}(t)$.*

Lemma 6.8 *Let t be a tail normal term. Then $\mathcal{FN}(t)\downarrow = \mathcal{FN}(t)$.*

Lemma 6.9 *Let t be a tail normal term and γ be a tail normal substitution. Then $\mathcal{FN}((t\gamma)\downarrow) = (\mathcal{FN}(t)\mathcal{FN}(\gamma))\downarrow$.*

Proof: By Lemma 6.6, it suffices to show that $\mathcal{FN}((t\gamma)\downarrow) = (\mathcal{FN}(t\gamma))\downarrow$, which we prove by induction on $\longrightarrow_\beta$. By Lemma 6.7, the result holds when $(t\gamma)\downarrow = t\gamma$. Otherwise, $t = @(s, t_1, \ldots, t_n)$, where $s\gamma = \lambda y.u$. Assuming without loss of generality that y does not occur in $t_2, \ldots, t_n$,
$\mathcal{FN}((t\gamma)\downarrow) = \mathcal{FN}((@(u\{y \mapsto t_1\}, t_2, \ldots, t_n)\gamma)\downarrow) = \mathcal{FN}((@(u, t_2, \ldots, t_n)\gamma')\downarrow)$
with $\gamma' = \gamma \cup \{y \mapsto (t_1\gamma)\downarrow\}$. Since $t\gamma \longrightarrow_\beta @(u, t_2, \ldots, t_n)\gamma'$, applying the induction hypothesis yields:
$\mathcal{FN}((@(u, t_2, \ldots, t_n)\gamma')\downarrow) = (\mathcal{FN}(@(u, t_2, \ldots, t_n))\mathcal{FN}(\gamma'))\downarrow =$
$(\mathcal{FN}(@(u, t_2, \ldots, t_n))(\{y \mapsto \mathcal{FN}(t_1\gamma\downarrow)\} \cup \mathcal{FN}(\gamma)))\downarrow$
and, by induction hypothesis again, the latter expression is equal to
$(\mathcal{FN}(@(u, t_2, \ldots, t_n))(\{y \mapsto (\mathcal{FN}(t_1)\mathcal{FN}(\gamma))\downarrow\} \cup \mathcal{FN}(\gamma)))\downarrow =$
$(\mathcal{FN}(@(u, t_2, \ldots, t_n))\{y \mapsto \mathcal{FN}(t_1)\downarrow\})\mathcal{FN}(\gamma))\downarrow =$
$((@(\mathcal{FN}(u), \mathcal{FN}(t_2), \ldots, \mathcal{FN}(t_n))\{y \mapsto \mathcal{FN}(t_1)\downarrow\})\mathcal{FN}(\gamma))\downarrow =$
$(@(\mathcal{FN}(u)\{y \mapsto \mathcal{FN}(t_1)\downarrow\}, \mathcal{FN}(t_2), \ldots, \mathcal{FN}(t_n))\mathcal{FN}(\gamma))\downarrow =$
$(@(\lambda y.\mathcal{FN}(u), \mathcal{FN}(t_1)\downarrow\}, \mathcal{FN}(t_2), \ldots, \mathcal{FN}(t_n))\mathcal{FN}(\gamma))\downarrow =$
$(@(\mathcal{FN}(\lambda y.u), \mathcal{FN}(t_1), \mathcal{FN}(t_2), \ldots, \mathcal{FN}(t_n))\mathcal{FN}(\gamma))\downarrow =$
$(@(s, \mathcal{FN}(t_1), \mathcal{FN}(t_2), \ldots, \mathcal{FN}(t_n))\mathcal{FN}(\gamma))\downarrow =$
$(\mathcal{FN}(@(s, t_1, t_2, \ldots, t_n))\mathcal{FN}(\gamma))\downarrow = (\mathcal{FN}(t)\mathcal{FN}(\gamma))\downarrow$. □

The following property is straightforward :

Lemma 6.10 *Let t be a term and x a variable of type σ. Then $t\{x \mapsto \bot_\sigma\}\uparrow^\eta = t\uparrow^\eta \{x \mapsto \bot_\sigma\}$.*

Since the level of neutralization $\mathcal{NL}_j^f$ of any argument is smaller than or equal to the arity of its type, we have the following properties.

Lemma 6.11 *Let $s : \sigma$ be an η-expanded term. Then $\mathcal{N}_i(s\xi\uparrow^\eta) = \mathcal{N}_i(s)\xi\uparrow^\eta$ for all $i \leq ar(\sigma)$.*

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

1. if $i = 0$ then $\mathcal{N}_i(s\xi\uparrow^\eta) = s\xi\uparrow^\eta$ and $\mathcal{N}_i(s) = s$. Hence, the result holds.
2. if $i > 0$ then, by assumption on σ, $\sigma = \rho \to \tau$ and $s = \lambda x : \rho.u$ by assumption on s. Therefore
   $\mathcal{N}_i(s\xi\uparrow^\eta) = \mathcal{N}_i((\lambda x : \rho\xi.u\xi)\uparrow^\eta) = \mathcal{N}_i(\lambda x : \rho\xi.(u\xi)\uparrow^\eta) = \mathcal{N}_{i-1}((u\xi)\uparrow^\eta \{x \mapsto \bot_{\rho\xi}\})$.
   By induction hypothesis, $\mathcal{N}_{i-1}((u\xi)\uparrow^\eta = \mathcal{N}_{i-1}(u)\xi\uparrow^\eta$, hence
   $\mathcal{N}_{i-1}((u\xi)\uparrow^\eta \{x \mapsto \bot_{\rho\xi}\}) = (\mathcal{N}_{i-1}(u)\xi\uparrow^\eta)\{x \mapsto \bot_\rho\xi\}$. By Lemma 6.10
   $(\mathcal{N}_{i-1}(u)\xi\uparrow^\eta)\{x \mapsto \bot_\rho\xi\} = (\mathcal{N}_{i-1}(u)\xi\{x \mapsto \bot_\rho\xi\})\uparrow^\eta = (\mathcal{N}_{i-1}(u)\{x \mapsto \bot_\rho\})\xi\uparrow^\eta$,
   and therefore $(\mathcal{N}_{i-1}(u)\{x \mapsto \bot_\rho\})\xi\uparrow^\eta = \mathcal{N}_i(\lambda x : \rho.u)\xi\uparrow^\eta = \mathcal{N}_i(s)\xi\uparrow^\eta$. □

Lemma 6.12 *Let $t : \sigma$ be a tail expanded term. Then $\mathcal{FN}(t\xi\uparrow^{\neq\Lambda}) = \mathcal{FN}(t)\xi\uparrow^{\neq\Lambda}$.*
Le $t : \sigma$ be an η-expanded term. Then $\mathcal{FN}(t\xi\uparrow^\eta) = \mathcal{FN}(t)\xi\uparrow^\eta$.

Proof: We first show that the second property follows from the first. If $\sigma\xi$ is not an arrow type, then both properties coincide. Otherwise, assume that $\sigma\xi = \tau_1 \to \tau_n \to \tau$. Then, $t\xi{\uparrow^\eta} = \lambda x_1 \ldots x_n.@(t\xi{\uparrow^{\neq\Lambda}}, x_1, \ldots, x_n)$. By definition $\mathcal{FN}(t\xi{\uparrow^\eta}) = \mathcal{FN}(\lambda x_1 \ldots x_n.@(t\xi{\uparrow^{\neq\Lambda}}, x_1, \ldots, x_n)) = \lambda x_1 \ldots x_n.@(\mathcal{FN}(t\xi{\uparrow^{\neq\Lambda}}), x_1, \ldots, x_n)$, which by the first property is $\lambda x_1 \ldots x_n.@(\mathcal{FN}(t)\xi{\uparrow^{\neq\Lambda}}, x_1, \ldots, x_n) = \mathcal{FN}(t)\xi{\uparrow^\eta}$.

We now prove the first property by induction on $|t|$.

1. Assume that $t \in \mathcal{X}$. Since $t\xi{\uparrow^{\neq\Lambda}} = t\xi$, $\mathcal{FN}(t) = t$ and $\mathcal{FN}(t\xi) = t\xi$, it follows that $\mathcal{FN}(t\xi{\uparrow^{\neq\Lambda}}) = t\xi = t\xi{\uparrow^{\neq\Lambda}} = \mathcal{FN}(t)\xi{\uparrow^{\neq\Lambda}}$.

2. Assume that $t = \bot_\sigma$. Then $t\xi{\uparrow^{\neq\Lambda}} = \bot_{\sigma\xi}$. Hence, $\mathcal{FN}(t\xi{\uparrow^{\neq\Lambda}}) = \bot_{\sigma\xi} = \bot_\sigma\xi = \mathcal{FN}(t)\xi{\uparrow^{\neq\Lambda}}$.

3. Assume that $t = \lambda x : \tau.u$. Then $\mathcal{FN}(t\xi{\uparrow^{\neq\Lambda}}) = \mathcal{FN}((\lambda x : \tau\xi.u\xi){\uparrow^{\neq\Lambda}}) = \mathcal{FN}(\lambda x : \tau\xi.(u\xi){\uparrow^{\neq\Lambda}}) = \lambda x : \tau\xi.\mathcal{FN}(u\xi{\uparrow^{\neq\Lambda}})$. By induction hypothesis, $\mathcal{FN}(u\xi{\uparrow^{\neq\Lambda}}) = \mathcal{FN}(u){\uparrow^{\neq\Lambda}}\ \xi$, which implies that $\lambda x : \tau\xi.\mathcal{FN}(u\xi{\uparrow^{\neq\Lambda}}) = \lambda x : \tau\xi.(\mathcal{FN}(u){\uparrow^{\neq\Lambda}})\xi = (\lambda x.\mathcal{FN}(u){\uparrow^{\neq\Lambda}})\xi = (\lambda x.\mathcal{FN}(u)){\uparrow^{\neq\Lambda}}\ \xi = \mathcal{FN}(t){\uparrow^{\neq\Lambda}}\ \xi$.

4. Assume that $t = @(t_1, \ldots, t_n)$. Then $\mathcal{FN}(t\xi{\uparrow^{\neq\Lambda}}) = \mathcal{FN}(@(t_1\xi, \ldots, t_n\xi){\uparrow^{\neq\Lambda}} = \mathcal{FN}(@(t_1\xi{\uparrow^{\neq\Lambda}}, t_2\xi{\uparrow^\eta}, \ldots, t_n\xi{\uparrow^\eta})) = @(\mathcal{FN}(t_1\xi{\uparrow^{\neq\Lambda}}), \mathcal{FN}(t_2\xi{\uparrow^\eta}), \ldots, \mathcal{FN}(t_n\xi{\uparrow^\eta}))$. By using the induction hypothesis for $\mathcal{FN}(t_1)\xi{\uparrow^{\neq\Lambda}}$, and the induction hypothesis together with the fact that the first property implies the second for $t_2\xi{\uparrow^\eta}, \ldots, t_2\xi{\uparrow^\eta}$, we get
$@(\mathcal{FN}(t_1\xi{\uparrow^{\neq\Lambda}}), \mathcal{FN}(t_2\xi{\uparrow^\eta}), \ldots, \mathcal{FN}(t_n\xi{\uparrow^\eta})) =$
$@(\mathcal{FN}(t_1)\xi{\uparrow^{\neq\Lambda}}, \mathcal{FN}(t_2)\xi{\uparrow^\eta}, \ldots, \mathcal{FN}(t_n)\xi{\uparrow^\eta}) =$
$@(\mathcal{FN}(t_1)\xi, \mathcal{FN}(t_2)\xi, \ldots, \mathcal{FN}(t_n)\xi){\uparrow^{\neq\Lambda}} =$
$@(\mathcal{FN}(t_1), \ldots, \mathcal{FN}(t_n))\xi{\uparrow^{\neq\Lambda}} =$
$\mathcal{FN}(@(t_1, \ldots, t_n))\xi{\uparrow^{\neq\Lambda}} = \mathcal{FN}(t)\xi{\uparrow^{\neq\Lambda}}$.

5. Assume finally that $t = f(t_1, \ldots, t_n)$ with $f \in \mathcal{F}$. Then $\mathcal{FN}(t\xi{\uparrow^{\neq\Lambda}}) = \mathcal{FN}(f(t_1, \ldots, t_n)\xi{\uparrow^{\neq\Lambda}}) = \mathcal{FN}(f(t_1\xi{\uparrow^\eta}, \ldots, t_n\xi{\uparrow^\eta})) = f_{new}(t'_1, \ldots, t'_n)$, where $t'_j = \mathcal{N}_{\mathcal{NL}^f_j}(\mathcal{FN}(t_j\xi{\uparrow^\eta}))$.

   By using the induction hypothesis together with the fact that the first property implies the second for $t_j\xi{\uparrow^\eta}$, $\mathcal{N}_{\mathcal{NL}^f_j}(\mathcal{FN}(t_j\xi{\uparrow^\eta})) = \mathcal{N}_{\mathcal{NL}^f_j}(\mathcal{FN}(t_j)\xi{\uparrow^\eta})$ and, by Lemma 6.11, $\mathcal{N}_{\mathcal{NL}^f_j}(\mathcal{FN}(t_j)\xi{\uparrow^\eta}) = \mathcal{N}_{\mathcal{NL}^f_j}(\mathcal{FN}(t_j))\xi{\uparrow^\eta}$. Let $t''_j = \mathcal{N}_{\mathcal{NL}^f_j}(\mathcal{FN}(t_j))$. Then $f_{new}(t'_1, \ldots, t'_n) = f_{new}(t''_1\xi{\uparrow^\eta}, \ldots, t''_n\xi{\uparrow^\eta}) = f_{new}(t''_1\xi, \ldots, t''_n\xi){\uparrow^{\neq\Lambda}} = f_{new}(t''_1, \ldots, t''_n)\xi{\uparrow^{\neq\Lambda}}$ with $t''_j = \mathcal{N}_{\mathcal{NL}^f_j}(\mathcal{FN}(t_j))$. Hence
   $f_{new}(t''_1, \ldots, t''_n)\xi{\uparrow^{\neq\Lambda}} = \mathcal{FN}(f(t_1, \ldots, t_n))\xi{\uparrow^{\neq\Lambda}} = \mathcal{FN}(t)\xi{\uparrow^{\neq\Lambda}}$. □

6.2 The Neutralized Ordering

We are now ready to define and study two modified orderings, one for plain higher-order terms and one for tail normal higher-order terms, by using the corresponding orderings on neutralized terms defined in Section 5.

Definition 6.13 *Given two tail expanded terms s and t, we define the neutralized orderings as follows:*

$$s \underset{nhorpo}{\succ} t \text{ if and only if } \mathcal{FN}(s) \underset{horpo}{\succ} \mathcal{FN}(t)$$

$$s \ (\underset{nhorpo}{\succ})^\eta_\beta \ t \text{ if and only if } \mathcal{FN}(s) (\underset{chorpo}{\succeq})^\eta_\beta \mathcal{FN}(t)$$

In order to make the proofs simnpler, we use the version of our ordering on normalized terms without the computable closure. Using the version with the computable closure would of course make the ordering stronger, but the proofs much more involved.

We now need to show that $\succ_{nhorpo}$ is well-founded and monotonic, compatible and functional on tail extanded terms of the same type, and that $(\succ_{nhorpo})^{\eta}_{\beta}$ is included in $\succ_{nhorpo}$, β-stable and η-polymorphic.

Lemma 6.14 *$\succ_{nhorpo}$ is well-founded and compatible.*

Proof: Compatibility trivially holds as for $\succ_{horpo}$ and well-foudedness follows directly from well-foundedness of $\succ_{horpo}$. □

Monotonicity needs some preparation:

Lemma 6.15 *Let s and t be two tail expanded terms (in the neutralized signature) of the same type and $u[s]$ be the η-expanded form of s. Then $s \succ_{horpo} t$ implies $\mathcal{N}_i(u[s]) \succ_{horpo} \mathcal{N}_i(u[t])$ for any i.*

Proof: We proceed by induction on i.

1. For the base case $i = 0$, the property holds by monotonicity of $\succ_{horpo}$.
2. Assume that $i > 0$ and u is not empty. Then $u[s] = \lambda x : \sigma.u'[@(s, x)]$ for some u' such that $x \notin \mathcal{V}ar(s)$ and $x \notin \mathcal{V}ar(u')$. Then $\mathcal{N}_i(u[s]) = \mathcal{N}_{i-1}(u'[@(s, \bot_\sigma)])$ and $\mathcal{N}_i(u[t]) = \mathcal{N}_{i-1}(u'[@(t, \bot_\sigma)])$. By monotonicity of $\succ_{horpo}$, $@(s, \bot_\sigma) \succ_{horpo} @(t, \bot_\sigma)$. Note that $@(s, \bot_\sigma)$ and $@(t, \bot_\sigma)$ are tail expanded terms and $u'[@(s, \bot_\sigma)]$ is the η-expansion of $@(s, \bot_\sigma)$. Therefore, by induction hypothesis, $\mathcal{N}_{i-1}(u'[@(s, \bot_\sigma)]) \succ_{horpo} \mathcal{N}_{i-1}(u'[@(t, \bot_\sigma)])$.
3. Assume now that $i > 0$ and u is empty. We prove that $\mathcal{N}_i(s) \succ_{horpo} \mathcal{N}_i(t)$ assuming that s is both η-expanded and tail expanded and t is tail expanded and have the same type.

   If s, and therefore t, has a non-functional type, then $\mathcal{N}_i(s) = s \succ_{horpo} t = \mathcal{N}_i(t)$.

   Otherwise, s and t have a type $\sigma \to \tau$. Since u is empty, $s = \lambda x.s'$ for some s' such that $x \in \mathcal{V}ar(s')$. The comparison $\lambda x.s' \succ_{horpo} t$ can only be by cases 6, 10 or 12. But, since t is tail expanded case 12 cannot apply and, since, by subterm property of the type ordering $\tau \not\geq_{\mathcal{T}_S} \sigma \to \tau$, case 6 cannot apply either. Therefore, $s \succ_{horpo} t$ holds by case 11, which implies that $t = \lambda x : \sigma.t'$ and $s' : \tau \succ_{horpo} t' : \tau$.

   By stability of $\succ_{horpo}$, we get $s'\{x \mapsto \bot_\sigma\} : \tau \succ_{horpo} t'\{x \mapsto \bot_\sigma\} : \tau$. Since s is both η-expanded and tail expanded, so is $s'\{x \mapsto \bot_\sigma\} : \tau$. And since t is tail expanded, so is $t'\{x \mapsto \bot_\sigma\} : \tau$. By induction hypothesis, $\mathcal{N}_{i-1}(s'\{x \mapsto \bot_\sigma\}) \succ_{horpo} \mathcal{N}_{i-1}(t'\{x \mapsto \bot_\sigma\})$. Therefore
   $\mathcal{N}_i(s) = \mathcal{N}_i(\lambda x.s') = \mathcal{N}_{i-1}(s'\{x \mapsto \bot_\sigma\}) \succ_{horpo} \mathcal{N}_{i-1}(t'\{x \mapsto \bot_\sigma\}) = \mathcal{N}_i(\lambda x.t') = \mathcal{N}_i(t)$. □

Lemma 6.16 *$\succ_{nhorpo}$ is monotonic on tail expanded terms.*

In the following, we will use without notice the property that $u[t]$ is tail-expanded whenever so are $s, u[s]$ and t, and s and t have the same type.

Proof: Assuming that s and t are arbitrary tail expanded terms of the same type as well as $u[s]$ and $u[t]$, we prove that $\mathcal{FN}(s) \succ_{horpo} \mathcal{FN}(t)$ implies $\mathcal{FN}(u[s]) \succ_{horpo} \mathcal{FN}(u[t])$. We proceed by induction on the size of u and case analysis. If u is empty, the result holds trivially. Otherwise, there are four cases:

1. Assume that $u[s] = f(w_1, \ldots, u'[s], \ldots, w_n)$, where $u'[s]$ must be of the form $v[w[s]]$ for some tail expanded term $w[s]$ such that $v[w[s]]$ is its η-expansion. By induction hypothesis $\mathcal{FN}(w[s]) \succ_{horpo} \mathcal{FN}(w[t])$, and, by Lemma 6.15, $\mathcal{N}_i(v[\mathcal{FN}(w[s])]) \succ_{horpo} \mathcal{N}_i(v[\mathcal{FN}(w[t])])$ for any i. By definition of v, $\mathcal{FN}(v[w[s]]) = v[\mathcal{FN}(w[s])]$ and $\mathcal{FN}(v[w[t]]) = v[\mathcal{FN}(w[t])]$, hence $\mathcal{N}_i(\mathcal{FN}(u'[s])) \succ_{horpo} \mathcal{N}_i(\mathcal{FN}(u'[t]))$, and by monotonicity of $\succ_{horpo}$, $\mathcal{FN}(f(w_1, \ldots, u'[s], \ldots, w_n)) = f_{new}(w'_1, \ldots, N_i(\mathcal{FN}(u'[s])), \ldots, w'_n) \succ_{horpo}$ $f_{new}(w'_1, \ldots, N_i(\mathcal{FN}(u'[t])), \ldots, w'_n) = \mathcal{FN}(f(w_1, \ldots, u'[s], \ldots, w_n))$.
2. Assume that $u[s] = @(u'[s], w_1, \ldots, w_n)$ where $u'[s]$ and $u'[t]$ are tail expanded terms. By induction hypothesis $\mathcal{FN}(u'[s]) \succ_{horpo} \mathcal{FN}(u'[t])$. Now, by monotonicity of $\succ_{horpo}$, we get $\mathcal{FN}(@(u'[s], w_1, \ldots, w_n)) = @(\mathcal{FN}(u'[s]), \mathcal{FN}(w_1), \ldots, \mathcal{FN}(w_n)) \succ_{horpo}$ $@(\mathcal{FN}(u'[t]), \mathcal{FN}(w_1), \ldots, \mathcal{FN}(w_n)) = \mathcal{FN}(@(u'[t], w_1, \ldots, w_n))$.
3. Assume that $u[s] = @(w_1, \ldots, u'[s], \ldots, w_n)$ where $u'[s]$ must be of the form $v[w[s]]$ for some tail expanded term $w[s]$ such that $v[w[s]]$ is its η-expansion. We proceed as in Case 1.
4. Assume that $u[s] = \lambda x.u'[s]$ where $u'[s]$ and u'[t] are tail expanded. We proceed again as in Case 1. □

Lemma 6.17 *$\succ_{nhorpo}$ is functional on tail expanded terms.*

Proof: Since $\mathcal{FN}(@(\lambda x.u, v)) = @(\lambda x.\mathcal{FN}(u), \mathcal{FN}(v)), \mathcal{FN}(@(\lambda x.u, v)) \longrightarrow_\beta \mathcal{FN}(u)\{x \mapsto \mathcal{FN}(v)\}$. By functionality of $\succ_{horpo}$, it follows that $\mathcal{FN}(@(\lambda x.u, v)) \succ_{horpo} \mathcal{FN}(u)\{x \mapsto \mathcal{FN}(v)\} = \mathcal{FN}(u\{x \mapsto v\}$ by Lemma 6.6. □

Lemma 6.18 *$(\succ_{nhorpo})^\eta_\beta$ is β-stable.*

Proof: Let s and t be tail normal terms, and γ be a tail normal substitution. We need to show that $s\ (\succ_{nhorpo})^\eta_\beta\ t$ implies $(s\gamma)\downarrow\ (\succ_{nhorpo})^\eta_\beta\ (t\gamma)\downarrow$.

Since $s\ (\succ_{nhorpo})^\eta_\beta\ t$, $\mathcal{FN}(s)\ (\succ_{nhorpo})^\eta_\beta\ \mathcal{FN}(t)$. By β-stability of $(\succ_{horpo})^\eta_\beta$, $\mathcal{FN}(s)\mathcal{FN}(\gamma)\downarrow\ (\succ_{horpo})^\eta_\beta\ \mathcal{FN}(t)\mathcal{FN}(\gamma)\downarrow$. By Lemma 6.9, $(\mathcal{FN}(s)\mathcal{FN}(\gamma))\downarrow = \mathcal{FN}((s\gamma)\downarrow)$ and $(\mathcal{FN}(t)\mathcal{FN}(\gamma))\downarrow = \mathcal{FN}((t\gamma)\downarrow)$, which implies that $\mathcal{FN}((s\gamma)\downarrow)(\succ_{horpo})^\eta_\beta\ \mathcal{FN}((t\gamma)\downarrow)$. □

Lemma 6.19 *$(\succ_{nhorpo})^\eta_\beta$ is polymorphic.*

Proof: Assume that $s\ (\succ_{nhorpo})^\eta_\beta\ t$. By definition, $s\ (\succ_{nhorpo})^\eta_\beta\ t$ implies $\mathcal{FN}(s)(\succ_{horpo})^\eta_\beta\ \mathcal{FN}(t)$. By polymorphism of $(\succ_{horpo})^\eta_\beta$, $\mathcal{FN}(s)\xi\ \uparrow^{\neq\Lambda}\ (\succ_{horpo})^\eta_\beta\ \mathcal{FN}(t)\xi\ \uparrow^{\neq\Lambda}$ and hence, by Lemma 6.12, $\mathcal{FN}(s\xi\uparrow^{\neq\Lambda}) = \mathcal{FN}(s)\xi\uparrow^{\neq\Lambda}\ (\succ_{horpo})^\eta_\beta\ \mathcal{FN}(t)\xi\uparrow^{\neq\Lambda} = \mathcal{FN}(t\xi\uparrow^{\neq\Lambda})$, which implies $s\xi\uparrow^{\neq\Lambda}\ (\succ_{nhorpo})^\eta_\beta\ t\xi\uparrow^{\neq\Lambda}$ and we are done. □

We conclude with our last unsuccessful example, showing that neutralization allows to orient the rule that could not be oriented before:

7 Further improvements and alternatives

We first consider the framework for normalized higher-order rewriting and orderings, before to move to the higher-order recursive path ordering itself, in which case the ideas presented in this section apply to all previous defined orderings, although they may be more useful for some versions than others. The second and third subsections present an improvement, in the sense that it gives some additional power to the orderings. The following three are alternatives, that allow defining variations of the previous orderings which can sometimes be useful for practical applications. The very last subsection describes a last example.

7.1 Higher-order rewriting

Our notion of higher-order rewriting is based on tail normal terms. One may wonder whether our work could have been carried out for the other natural generalization of Nipkow's rewriting, in which normalized terms are used instead, and the answer is positive. The only change to be made is to define η-polymorphism as $(\Gamma \vdash_{\mathcal{F}} s : \sigma) \succ (\Gamma \vdash_{\mathcal{F}} t : \sigma)$ implies $(\Gamma\xi \vdash_{\mathcal{F}} s\xi \uparrow^{\eta} : \sigma\xi) \succ (\Gamma\xi \vdash_{\mathcal{F}} t\xi \uparrow^{\eta} : \sigma\xi)$ for all type substitutions ξ.

One may also wonder whether any of these two is the good one, and also whether Nipkow's notion of rewriting (on terms of a data type) is the good one. We actually don't think so. The major role of η-expansions is to normalize the number of arguments of a higher-order term. Since functions symbols have an arity in our framework, it is not necessary to η-expand terms headed by a function symbol. Giving an arity to the variables, as done by Klop in his seminal work, would allow us getting rid completely of normal expansions. This new point of view is developped in a forthcoming paper.

7.2 Building in the permutative equality

So far, we have been very careful defining strict relations, in the sense that their transitive closure is irreflexive. We now move to non-strict ones, which will be the union of a strict part and an equality part.

For our purpose, an *equivalence* on higher-order terms is a reflexive, symmetric and transitive relation $=$ containing α-conversion, and a *compatible* relation $\succeq$ is the union of a *strict* part $\succ$ whose transitive closure $\succ^*$ is a strict ordering, and an equivalence $\simeq$ such that $\succ$ is compatible with $\simeq$, that is, $s' \simeq s \succ t \simeq t'$ implies $s' \succ t'$ (actually, $s' \succ^* t'$ would suffice) for all terms s, t, s', t'. A transitive compatible relations (the strict part of which being a strict ordering) is called a *quasi-ordering*, and is an *ordering* when the equivalence coincides with α-conversion.

The transitive closure $\succeq^*$ of the compatible relation $\succeq$ happens to be the quasi-ordering having $\succ^*$ as strict part and $=$ as equivalence. Indeed, apart from transitivity, compatible relations enjoy most properties of quasi-orderings which make them an appropriate tool for proving termination of rewrite systems. To this end, we say that a compatible relation $\succeq$ is well-founded when so is its strict part $\succ$. Note that $\succeq$ is a well-founded compatible relation iff $\succ$ is well-founded and compatible with $\simeq$.

We now define the multiset and lexicographic extensions of relations on a set S (resp. S_i) given as pairs $\simeq, \succ$ (resp. $\simeq_i, \succ_i$) such that $\simeq$ (resp. $\simeq_i$)is an equivalence relation, and $\succ$ (resp. $\succ_i$)is an arbitrary relation. By abuse of notation, we will allow us to write $\succeq$ (resp. $\succeq_i$) to denote such a pair. This notation makes of course sense when $\succeq$ (resp. $\succeq_i$) is known to be a compatible relation.

- $(s_1, \ldots, s_n)(\simeq_1, \ldots, \simeq_n)^n$ be the relation on $S_1 \times \ldots \times S_n$ defined as
  $(s_1, \ldots, s_n)(\simeq_1, \ldots, \simeq_n)^n(t_1, \ldots, t_n)$ iff $s_1 \simeq_1 t_1, \ldots, s_n \simeq_n t_n$;

  Then, $(\succeq_1, \ldots, \succeq_n)^n$ is an equivalence relation.

- $(\succeq_1, \ldots, \succeq_n)_{lex}$ be the relation on $S_1 \times \ldots \times S_n$ defined as the union of the relations $(\succeq_1, \ldots, \succeq_n)_{strict-lex}$ and $(\simeq_1, \ldots, \simeq_n)^n$, where
  $(s_1, \ldots, s_n)(\succeq_1, \ldots, \succeq_n)_{strict-lex}(t_1, \ldots, t_n)$ iff $s_1 \simeq_1 t_1, \ldots, s_{i-1} \simeq_{i-1} t_{i-1}$ and $s_i \succ_i t_i$ for some $i \in [1..n]$;

  Then $(\succeq_1, \ldots, \succeq_n)_{lex}$ is a well-founded compatible relation if so are $\succeq_1, \ldots, \succeq_n$.

- $\simeq_{mul}$ be the relation on the set of multisets of elements of S defined as
  $N \cup \{x\} \simeq_{mul} N' \cup \{y\}$ iff $N \simeq_{mul} N'$ and $x \simeq y$;

  Then, $\simeq_{mul}$ is an equivalence relation.

- $\succ_{mul}$ be the relation on the set of multisets of elements of S defined as $N \cup \{x\} \succ_{mul} N' \cup \{y_1, \ldots, y_n\}$ iff $N \succ_{mul} N'$ and $\forall i \in [1..n]\ x \succ y_i$;
- $\succeq_{mul}$ be the relation on the set of multisets of elements of S defined as the union of the relations $\succeq_{strict-mul}$ and $\simeq_{mul}$;

  Then, $\succeq_{mul}$ is a well-founded compatible relation if so is $\succeq$.

To show these results, it suffices to show that the above operations preserve compatible relations (this follows immediately from the compatibility property of the given relations), and then, by [13], the transitive closures of the resulting compatible relations are well-founded, and therefore, so are the compatible relations themselves. A direct proof of this second part would actually simplify a little bit the standard arguments.

We now define the relation $\succeq_{horpo} = \succ_{horpo} \cup =_{horpo}$, where

$$s \underset{horpo}{=} t \text{ iff } \sigma =_{\mathcal{T}_S} \tau \text{ and}$$

1. $s = f(\overline{s}),\ t = g(\overline{t}),\ f =_{\mathcal{F}} g \in Mul$ and $\overline{s}\ (\underset{horpo}{=})_{mul}\ \overline{t}$
2. $s = f(\overline{s}),\ t = g(\overline{t}),\ f =_{\mathcal{F}} g \in Lex$ and $\overline{s}(\underset{horpo}{=})_{mon}\overline{t}$
3. $s = @(s_1, s_2),\ t = @(t_1, t_2)$, and $\{s_1, s_2\}\ (\underset{horpo}{=})_{mul}\ \{t_1, t_2\}$
4. $s = \lambda x : \alpha.v,\ t = \lambda x : \beta.w,\ \alpha =_{\mathcal{T}_S} \beta$ and $v \underset{horpo}{=} w$.

$$s \underset{horpo}{\succ} t \text{ iff } \sigma \geq_{\mathcal{T}_S} \tau \text{ and}$$

1. $s = f(\overline{s})$ with $f \in \mathcal{F}$, and $u \underset{horpo}{\succeq} t$ for some $u \in \overline{s}$
2. $s = f(\overline{s})$ with $f \in \mathcal{F}$ and $t = g(\overline{t})$ with $f >_{\mathcal{F}} g$, and A
3. $s = f(\overline{s})$ and $t = g(\overline{t})$ with $f =_{\mathcal{F}} g \in Mul$ and $\overline{s}\ (\underset{horpo}{\succeq})_{mul}\ \overline{t}$
4. $s = f(\overline{s})$ and $t = g(\overline{t})$ with $f =_{\mathcal{F}} g \in Lex$ and $\overline{s}\ (\underset{horpo}{\succeq})_{lex}\ \overline{t}$, and A
5. $s = @(s_1, s_2)$, and $s_1 \underset{horpo}{\succeq} t$ or $s_2 \underset{horpo}{\succeq} t$
6. $s = \lambda x : \alpha.u$ with $x \notin \mathcal{V}ar(t)$, and $u \underset{horpo}{\succeq} t$
7. $s = f(\overline{s})$ with $f \in \mathcal{F}$, $t = @(\overline{t})$ is a partial 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 $s \underset{horpo}{\succ} v$
9. $s = @(s_1, s_2)$, $t = @(\overline{t})$ is a partial left-flattening of t and $\{s_1, s_2\}(\underset{horpo}{\succeq})_{mul}\overline{t}$
10. $s = \lambda x : \alpha.u,\ t = \lambda x : \beta.v$, $\alpha =_{\mathcal{T}_S} \beta$ and $u \underset{horpo}{\succ} v$

11. $s = @(\lambda x : \alpha.u, v)$ and $u\{x \mapsto v\} \underset{horpo}{\succeq} t$

where, as before

$$A = \forall v \in \overline{t}\ \ s \underset{horpo}{\succ} v \text{ or } u \underset{horpo}{\succeq} v \text{ for some } u \in \overline{s}$$

We can then mimic the study of the ordering done in Section 3, with some adaptations:

Firstly, we need to show (by an easy induction on $|s| + |u|$) that $s : \sigma =_{horpo} t : \tau$, $s' \succ_{horpo} s$ and $t =_{horpo} t'$ imply $s' \succ_{horpo} t'$. This will imply that $\succeq_{horpo}$ is a well-founded compatible relation for any set of terms for which $\succ_{horpo}$ is itself well-founded.

Secondly, the properties of the ordering such as monotonicity for terms of equivalent types, stability, polymorphism have now to be proved for both $=_{horpo}$ and $\succ_{horpo}$.

Thirdly, the computability property 3.14(ii) need to be reproved to take $=_{horpo}$ into account, which is routine. Everything else remains the same.

The changes are similar for the orderings defined in the other sections.

7.3 Building in inductive types

This makes sense when using the closure mecanism only. Inductive types are sorts equipped with a set of strictly positive typed free constructors, see [4] for precise definitions. For an example, the set of natural numbers in Peano notation is a strictly positive inductive types:

$$\texttt{Inductive type Nat} = \texttt{0} \oplus \texttt{S}(\texttt{Nat})$$

When the syntax allows to declare explicitely strictly positive inductive types, it is possible to define the computability predicates so as to ensure a new computability property for terms headed by a constructor symbol:

$$C(\overline{t}) \text{ is computable iff all terms in } \overline{t} \text{ are computable.}$$

This property can now be built in the definition of the closure, by adding the rule

8. arguments of constructor headed terms: let $C(\overline{s}) \in \mathcal{CC}(t, \mathcal{V})$ for some constructor symbol C; then $u \in \mathcal{CC}(t, \mathcal{V})\ \forall u \in \overline{s}$;

This rule allows to simplify the treatment of some examples presented in this paper.

7.4 Building in the type ordering

In all definitions of the ordering $\succ_{horpo}$ the type ordering has only be used to restrict the pair of terms that could be compared. However, the type ordering can also be used inside the ordering like the precedence. That is, we can add the following case to the definition

12. $s = f(\overline{s})$ with $f \in \mathcal{F}$ and $t = g(\overline{t})$ with $\sigma >_{\mathcal{TS}} \tau$, and A

Since we always have $\sigma \geq_{\mathcal{TS}} \tau$ there is no need to add a condition $\sigma =_{\mathcal{TS}} \tau$ in any other case.

The resulting ordering is more powerful than the ordering without the computable closure but incomparable with the ordering with the computable closure. The reason is that the inclusion of this new case requires to modify the induction argument of lemma 4.8 to include the type of the term as first component in the lexicographic comparison. Therefore, in the definition of the computable closure we can include a new case

10. type-ordering: let $g : \rho_1 \times \ldots \times \rho_n \to \tau$ such that $\sigma >_{\mathcal{T}_S} \tau$, and $\overline{s} : \overline{\rho} \in \mathcal{CC}(t : \sigma, \mathcal{V})$; then $g(\overline{s}) : \tau \in \mathcal{CC}(t : \sigma, \mathcal{V})$;

but we have to restrict the precedence case in the following way:

2. precedence: let $g : \rho_1 \times \ldots \times \rho_n \to \tau$ such that $\sigma =_{\mathcal{T}_S} \tau$ and $f >_{\mathcal{F}} g$, and $\overline{s} : \overline{\rho} \in \mathcal{CC}(t : \sigma, \mathcal{V})$; then $g(\overline{s}) : \tau \in \mathcal{CC}(t : \sigma, \mathcal{V})$;

It is easy to see that the new computable closure is incomparable with the previous one, which implies that the obtained ordering is incomparable with the previous one.

7.5 Building in the subterm type decreasing relation

Since the ordering $\succ_{horpo}$ defined in Section 3 is compatible with the ordering $\rhd_{>_{\mathcal{T}_S}}$, it is natural to try building the latter in the former. The proposal below is not an extension of $\succ_{horpo}$, however, since it may fail orienting some pairs oriented by $\succ_{horpo}$. But we think it is worthwhile a study that we have not done in enough detail so as to make a well-foundedness claim. We do it for the ordering of Section 3 first, before to comment on the impact on the closure definition.

Let $Subt_{>_{\mathcal{T}_S}}(t) = \{s \mid t \rhd_{>_{\mathcal{T}_S}} s\}$ if t is neutral, and $Subt_{>_{\mathcal{T}_S}}(t) = \emptyset$ otherwise. Let now

$$A = \forall v \in Subt_{>_{\mathcal{T}_S}}(t) \;\; s \underset{horpo}{\succ} v \text{ or } u \underset{horpo}{\succeq} v \text{ for some } u \in Subt_{>_{\mathcal{T}_S}}(s)$$

We define now

$$s \underset{horpo}{\succ} t \text{ iff } \sigma \geq_{\mathcal{T}_S} \tau \text{ and}$$

1. $s = f(\overline{s})$ with $f \in \mathcal{F}$, and $u \underset{horpo}{\succeq} t$ for some $u \in Subt_{>_{\mathcal{T}_S}}(s)$.
2. $s = f(\overline{s})$ with $f \in \mathcal{F}$ and $t = g(\overline{t})$ with $f >_{\mathcal{F}} g$, and A
3. $s = f(\overline{s})$ and $t = g(\overline{t})$ with $f =_{\mathcal{F}} g \in Mul$ and $\overline{s}$) $(\underset{horpo}{\succ})_{mul}\, \overline{t}$ and A
4. $s = f(\overline{s})$ and $t = g(\overline{t})$ with $f =_{\mathcal{F}} g \in Lex$ and $\overline{s}\; (\underset{horpo}{\succ})_{lex}\, \overline{t}$, and A
5. $s = @(s_1, s_2)$, and $s_1 \underset{horpo}{\succeq} t$ or $s_2 \underset{horpo}{\succeq} t$ and A
6. $s = \lambda x : \alpha.u$ with $x \notin \mathcal{V}ar(t)$, and $u \underset{horpo}{\succeq} t$
7. $s = f(\overline{s})$ with $f \in \mathcal{F}$, $t = @(\overline{t})$ is a partial 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 $s \underset{horpo}{\succ} v$
9. $s = @(s_1, s_2)$, $t = @(\overline{t})$ is a partial left-flattening of t, $\{s_1, s_2\}(\underset{horpo}{\succ})_{mul}\overline{t}$ and A
10. $s = \lambda x : \alpha.u,\ t = \lambda x : \beta.v$, $\alpha =_{\mathcal{T}_S} \beta$ and $u \underset{horpo}{\succ} v$
11. $s = @(\lambda x : \alpha.u, v)$, $u\{x \mapsto v\} \underset{horpo}{\succeq} t$ and A

The proof of this variant needs a bit more work. Indeed, the computability definition of the arrow type terms should now be changed to:

If $s :_C \sigma = \tau \to \rho$ then $s \in [\![\sigma]\!]$ if $@(s,t) \in [\![\rho]\!]$ for every t such that $v \in [\![\tau]\!]$ for every $v \in Subt_{>_{\mathcal{T}_\mathcal{S}}}(t)$.

For this variant, any term smaller than a computable term will be computable, and therefore the closure of a term will be closed by taking subterms of smaller type. This has a clear practical impact, since the ability of taking subterms, in particular for the starting term, is crucial for many proofs.

7.6 EXAMPLE

Section 5 relies on the property that terms in normal form should not need any η-expansions when instantiated by a type substitution. To ensure that property, we required that output types of both function symbols and variables be different from a type variable. An alternative allows to cope with type variables as output types, provided they are never instantiated by arrow types. This extension is straightfoward, but useful. It allows in particuler to deal with polymorphic recursors, as the one given in example 5.

On the other hand, there are situations where we need more flexibility. Here comes such an example.

Example 15 (Encoding Natural Deduction; taken from [9]) Let

$\mathcal{S} = \{o : *,\ c : * \times * \to *\}, \quad \mathcal{S}^\forall = \{\sigma, \tau, \rho\}.$

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

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

The rules are the following:

$$
\begin{array}{rcll}
app_{\sigma,\tau}(abs_{\sigma,\tau}(F), X) & \to & @(F, X) & (1)\\
\Pi^0_{\sigma,\tau}(\Pi_{\sigma,\tau}(X, Y)) & \to & X & (2)\\
\Pi^1_{\sigma,\tau}(\Pi_{\sigma,\tau}(X, Y)) & \to & Y & (3)\\
\exists^-_{\sigma,\tau}(\exists^+_\sigma(Z, X), G) & \to & @(G, Z, X) & (4)
\end{array}
$$

$$
\begin{array}{rcll}
app_{\sigma,\tau}(\exists^-_{\rho,\sigma\to\tau}(T, H), X) & \to & \exists^-_{\rho,\tau}(T, \lambda x : o\ y : \rho.app_{\sigma,\tau}(@(H, x, y), X)) & (5)\\
\Pi^0_{\sigma,\tau}(\exists^-_{\rho,c(\sigma,\tau)}(T, I)) & \to & \exists^-_{\rho,\tau}(T, \lambda x : o\ y : \rho.\Pi^0_{\sigma,\tau}(@(I, x, y))) & (6)\\
\Pi^1_{\sigma,\tau}(\exists^-_{\rho,c(\sigma,\tau)}(T, I)) & \to & \exists^-_{\rho,\tau}(T, \lambda x : o\ y : \rho.\Pi^1_{\sigma,\tau}(@(I, x, y))) & (7)\\
\exists^-_{\sigma,\tau}(\exists^-_{\rho,c(o,\sigma)}(T, J), G) & \to & \exists^-_{\rho,\tau}(T, \lambda x : o\ y : \rho.\exists^-_{\sigma,\tau}(@(J, x, y), G)) & (8)
\end{array}
$$

Rules 1, 2, 3 and 4 can be easily proved by applying the definition of the ordering. Rules 6 and 7, follow easily as well, using first case 1 (i) and considering $\exists_{\rho,c(\sigma,\tau)} >_\mathcal{F} \{\exists_{\rho,\tau}, \Pi^0_{\sigma,\tau}, \Pi^1_{\sigma,\tau}\}$ for the rest. Rule 5 is requires more work, as sketched below.

First, we introduce a family of ternary symbols $\exists_{\rho,\sigma\to\tau,\sigma} : c(o,\rho) \times o \to \rho \to (\sigma \to \tau) \times \sigma \to \tau$ parameterized by the types σ, τ, ρ, together with the precedences: $\exists_{\rho,\sigma\to\tau} >_\mathcal{F} \exists_{\rho,\sigma\to\tau,\sigma} >_\mathcal{F} \{\exists_{\rho,\tau}, app_{\sigma,\tau}\}$.

We then construct the "middle term" $@(\lambda y.\exists_{\rho,\sigma\to\tau,\sigma}(T, H, y), X)$ and show that

$$app_{\sigma,\tau}(\exists^-_{\rho,\sigma\to\tau}(T, H), X)(\underset{chorpo}{\succ})^\eta_\beta\ @(\lambda y.\exists_{\rho,\sigma\to\tau,\sigma}(T, H, y), X)$$

and

$$@(\lambda y.\exists_{\rho,\sigma\to\tau,\sigma}(T, H, y), X)(\underset{chorpo}{\succ})^\eta_\beta\ \exists^-_{\rho,\tau}(T, \lambda x : o\ y : \rho.app_{\sigma,\tau}(@(H, x, y), X))$$

therefore proving that the rule is in the transitive closure of $(\succ_{chorpo})^\eta_\beta$.

To show the first comparison we apply case 7 and then show that

$$\exists^{-}_{\rho,\sigma\to\tau}(T,H)(\underset{chorpo}{\succ})^{\eta}_{\beta}\ \lambda y.\exists_{\rho,\sigma\to\tau,\sigma}(T,H,y)$$

using case 1 (iii).

For the second comparison we apply case 11, and show that

$$\exists_{\rho,\sigma\to\tau,\sigma}(T,H,X)(\underset{chorpo}{\succ})^{\eta}_{\beta}\ \exists^{-}_{\rho,\tau}(T,\lambda x:o\ y:\rho.app_{\sigma,\tau}(@(H,x,y),X))$$

using case 2 and showing that $\lambda x : o\ y : \rho.app_{\sigma,\tau}(@(H,x,y),X)$ is in the computable closure of $\exists_{\rho,\sigma\to\tau,\sigma}(T,H,X)$.

Finally, let us mention that Rule 8 cannot be proved terminating by our ordering. □

8 Conclusion

We have defined a powerful mecanism for defining orderings operating on higher-order terms. Based on the notion of the computable closure of a term, we have succeeded defining a conservative extension of Dershowitz's recursive path ordering, which is indeed a polymorphic reduction ordering including β-reductions. To our knowledge, this is the first such ordering ever.

Another hidden achievement is a new, simple, easy to teach well-foundedness proof for Dershowitz's recursive path ordering. This proof can be easily extracted from the strong normalization proof given here: it actually reduces to a simplified version of the proofs of Lemma 3.15, spelled out in full detail in [7]. A similar proof also appears in [17]. Of course, this does not prove the stronger (but less usefull) property that the recursive path ordering is a well-order.

The idea of the computable closure originates from [3], where it was used to define a syntactic class of higher-order rewrite rules that are compatible with beta reductions and with recursors for arbitrary positive inductive types. The language there is indeed richer than the one considered here, since it is the calculus of inductive constructions generated by a monomorphic signature. The usefulness of the notion of closure in this different context shows the strength of the concept.

Our definition of the higher-order recursive path ordering can be developped now in several different directions, besides thoses mentionned in Section 7, that we discuss in turn.

First of all, we are not satisfied with our treatment of abstractions based on neutralization. This technique is a little bit complicated and technical, and compromises in some sense the elegance of the ordering and computational closure definitions. We would instead like to incorporate a better handling of abstractions directly in the recursive definition of the ordering.

The idea of the recursive path ordering is to prove the strong normalization property of the rewrite relation generated by a pair $l \to r$ by checking the rewrite relation generated by a set of pairs $l' \to r'$ obtained from $l \to r$ by taking various subterms of l (for l') and subterms of r for r'. The computation will of course fail if the new relation is not strongly normalizing. A possible patch is to improve the ordering by combining it with another ordering based, e.g. on interpretations. This has been done for the first-order case [23, 34], and for the higher-order case as well [35]. There is still room for improvement, though, since the interpretations used there are essentially first-order, and using interpretations "a la Van de Pol" would give a quite stronger ordering.

Assume that the two terms u and v are comparable, having for example the same type. Then, the two terms $\lambda x : \sigma.\lambda y : \tau.u$ and $\lambda y : \tau.\lambda x : \sigma.v$ are however incomparable for type reason if the types σ and τ are not equivalent in the type ordering. Even more, $\lambda x : \sigma.\lambda y : \tau.u$ and $\lambda y : \tau.\lambda x : \sigma.v$ are not equivalent

although they denote semantically identical functions. A solution to this problem would be to build Di Cosmo's type isomorphism in the type ordering. This does not seem so easy, however, since our current conditions on the type ordering would be violated.

Associative commutative operators are common in practice, and therefore, it is important to adapt our ordering to this case. This will be one of the directions we want to pursue.

Truly important is the development of an ordering able to compare terms of the calculus of constructions. Such an ordering can be obtained by replacing case 11 for abstractions by a clause ensuring monotonicity of dependent products. This is done in [38], by starting from the original version of the higher-order recursive path ordering introduced in [22]. Extending the uniform presentation of the ordering on terms and types will yield a much stronger ordering. First, this would improve the ordering on object-level terms of a dependent type, since such terms of different but convertible types such as $list(1+1)$ and $list(2)$ cannot be compared in [38]. Besides, this ordering should be able to compare arbitrary terms of the calculus, not only object-level terms, but type and kind level ones as well, hence generalizing what is called strong elimination in the calculus of inductive constructions. This is our conjecture sketched in Section 3.6 that such an ordering should be terminating. The fact that this conjecture generalizes strong elimination hints at its difficulty.

Using the computability technique instead of the Kruskal theorem is intriguing, and raises the question whether it is possible to exhibit a suitable extension of Kruskal's theorem that would allow proving that the higher-order recursive path ordering is a well-order of the set of higher-order terms ? Here, we must confess that we actually failed in our initial quest to find a suitable extension of Kruskal theorem on higher-order terms. In retrospect, the reason is that we were looking for too strong a statement. It may be that a version of Kruskal's theorem holds, and would imply our result, based on an adequate notion of subterm. If this is the case, this notion of subterm should somehow be related with the computability properties: not all subterms of a computable term are themselves computable. Related to this problem is the question of the ordinality of our ordering (or of a total extansion of it).

Acknowledgments: we are grateful to Femke Van Ramsdoonk for a couple of useful remarks.

References

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

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

[3] F. Blanqui, J.-P. Jouannaud, and M. Okada. The Calculus of Algebraic Constructions. In Narendran and Rusinowitch [29]. pp.

[4] F. Blanqui, J.-P. Jouannaud, and M. Okada. Inductive Data Types. *Theoretical Computer Science*, 277:. 2001.

[5] F. Blanqui. Inductive Types Revisited. submitted.

[6] Alain Colmerauer. Equations and Inequations on Finite and Infinite Trees. In Proc. Fifth Generation Computer Systems, Tokyo, Japan, pp 85–99. North Holland, 1984.

[7] Hubert Comon and Jean-Pierre Jouannaud. Les termes du premier ordre. Cours du DEA "Programmation". Available from the web at http://www.lix.polytechnique.fr/[̃]jouannaud/biblio.html.

[8] T. Coquand. Inductive definitions and type theory. Lecture notes. Int. summer school on automated deduction, Chambéry, July 1994.

[9] J. Van de Pol. *Termination of Higher-Order Rewrite Systems*. PhD thesis, Department of Philosophy–Utrecht University, 1996.

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

[11] J. Van de Pol. Termination proofs for higher-order rewrite systems. In *Higher-Order Algebra, Logic and Term Rewriting*, 1993. LNCS 816, pp 305–325, Springer-Verlag, 1994.

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

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

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

[15] Maribel Fernández and Jean-Pierre Jouannaud. Modular termination of term rewriting systems revisited. In Egidio Astesiano, Gianni Reggio, and Andrzej Tarlecki, editors, *Recent Trends in Data Type Specification*, LNCS 906:255–272. Springer-Verlag, 1995. Refereed selection of papers presented at ADT'94.

[16] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1989.

[17] Jean Goubault. Well founded recursive relations. *Computer Science Logic Workshop*, Paris, 2001. In LNCS 2142:484–497, Springer-Verlag, 2002.

[18] Gérard Huet, Gilles Kahn and Christine Paulin-Mohring. The Coq Proof Assistant. A Tutorial. Version 7.3. INRIA Rocquencourt and ENS Lyon.

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

[20] Jean-Pierre Jouannaud and Mitsuhiro Okada. Higher-Order Algebraic Specifications. In , editor, *Annual IEEE Symposium on Logic in Computer Science*, Amsterdam, The Netherlands, 1991. IEEE Comp. Soc. Press.

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

[22] 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.

[23] Sam Kamin and Jean-Jacques Levy. Two generalizations of the recursive path ordering. Unpublished notes, 1980.

[24] Jan Wilhelm Klop. Combinatory Reduction Relations. Mathematical Centre Tracts 127. Mathematisch Centrum, Amsterdam, 1980.

[25] 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.

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

[27] Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(1):3–29, February 1998.

[28] Dale Miller. Lambda PROLOG. environ 90.

[29] Paliath Narendran and Michael Rusinowitch, editors. *10th International Conference on Rewriting Techniques and Applications*, Trento, Italy, July 1999. LNCS 1631, Springer-Verlag.

[30] Tobias Nipkow. Higher-order critical pairs. In *6th IEEE Symp. on Logic in Computer Science*, pages 342–349. IEEE Computer Society Press, 1991.

[31] Mitsuhiro Okada and Gaisi Takeuti. On the theory of quasi ordinal diagrams. In S. G. Simpson, editor, *Logic and Combinatorics*. American Mathematical Society, 1986.

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

[33] Albert Rubio. A fully syntactic AC-RPO. In Narendran and Rusinowitch [29].

[34] Cristina Borralleras, Maria Ferreira and Albert Rubio. Complete monotonic semantic path orderings. In *Conference on automated deduction*, Pittsburg, 2000. LNAI 1831:346–364, Springer-Verlag.

[35] Cristina Borralleras and Albert Rubio. A monotonic, higher-order semantic path ordering. submitted, 2001.

[36] Hans Zantema. Termination. In *Term Rewriting Systems*, M. Bezem, J.W. Kop and R. de Vrijer eds, Cambridge Tracts in Theoretical Computer Science 55, Cambridge University Press, 2003.

[37] Femke Van Raamsdonk. Confluence and Normalization for Higher-Order Rewrite Systems. phd thesis, Vrije Universiteit, Amsterdam, The Netherland, 1996.

[38] 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.