Lambda extensions of rewrite orderings

Jean-Pierre Jouannaud
Albert Rubio

Report LSI–95–50–R

Lambda Extensions of Rewrite Orderings*

Jean-Pierre Jouannaud†** and **Albert Rubio**‡

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

Abstract. In this work we provide a new proof of the result by Gallier and Tannen that the combination of an arbitrary terminating first-order rewrite system with the simply typed lambda calculus is strongly normalizing. This proof proceeds via the explicit lifting of a rewrite ordering on first-order terms to a rewrite ordering on (first-order) algebraic λ-terms. For the definition of the ordering, we have used a technique developed by Kfoury and Wells, in order to delay the erasing β-reductions (also called K-redutions) which may destroy the monotonicity property. This technique is extended to be able to handle the extensionality rule.
As a particular case, the above construction yields an extension of the recursive path ordering of Dershowitz to a rewrite-ordering on (first-order) algebraic λ-terms which contains simply typed λ-derivations.

1 Introduction

The understanding of the functional paradigm has made a key step with the discovery of the Curry-Howard isomorphism. Functional programing languages, however, are more complex than purely functional definitions based on the typed lambda calculus. Definitions by pattern matching are now quite common, and their understanding can be based on an extension of the typed lambda calculus by algebraic operators, in which the reductions of the lambda calculus are combined with the algebraic ones. Of course, the use of such a model of computations requires that the key properties of the typed lambda calculus, namely, subject reduction, strong normalization, and confluence remain valid in the combination. Previous work in this area tries to identify under which syntactical conditions on the rewrite rules this is the case [3, 4, 9, 13, 10]. In these approaches, the two components of the language are usually separated, in the sense that the algebraic rules do not involve the lambda calculus operators. This is not completely true actually of [13, 10], but remains marginal in both works.

* This work was partly supported by the ESPRIT Basic Research Action CCL.
** This work was done while the author was in sabbatical, invited at the Technical University of Catalonia

Our goal in this paper is to elaborate tools for proving strong normalization of complex combinations in which the algebraic rules may use the lambda calculus operators in their left as well as in their right hand sides. For this, we will follow the traditionnal term rewriting approach, by comparing left and right-hand sides of rules in some *rewrite ordering*, that is a monotonic well-founded ordering on terms. These orderings will of course need to orient lambda terms as a subcase, and this will be done by the reduction relation on typed lambda terms. Our strategy is therefore to extend this ordering relation to combined terms by following an idea developped in [14] for the case of orderings on first order terms. In a first step, terms are normalized with respect to the lambda calculus reduction rules; in a second step different normal forms are compared in some ordering on algebraic terms, while equal ones are simply compared in the lambda calculus derivation relation. The main question is therefore to prove that it yields a rewrite ordering. This is unfortunately not true, because the first normalization step may erase subterms of the original term, a well known problematic property of so-called K-redexes in the lambda calculus which destroys the monotonicity property of the ordering. Our solution will be to develop a non-erasing version of the lambda calculus (usually called the λ-I-calculus) for computing normal forms, and relate them with the usual ones. A similar technique was used by Kfoury and Wells for their method of proving termination of β-reductions of various typed lambda calculi [12]. Based on these non-erasing calculi, our contributions are the following:

An extension of the technique by Kfoury and Wells (investigated in section 3), which allows to handle the extensionality rule 4.

A new proof of the result by Gallier and Tannen that the combination of an arbitrary terminating first-order rewrite system with the simply typed lambda calculus is strongly normalizing 5. This proof proceeds via the explicit lifting of a rewrite ordering on first-order terms to a rewrite ordering on (first-order) algebraic λ-terms.

As a particular case, the above construction yields an extension of the recursive path ordering of Dershowitz [5] to a rewrite-ordering on (first-order) algebraic λ-terms which contains simply typed λ-derivations 6.

2 Preliminaries

We expect the reader familiar with the basic concepts and notations of term rewriting systems and typed lambda calculi. We refer to [6] for definitions and notations of term rewriting, and to [1, 2] for the notations of lambda calculi. When notations differ, we will in general favour [6].

A *signature* $\mathcal{F}$ is a finite set of *function symbols* together with their (fixed) arity. $\mathcal{X}$ denotes a denumerable set of *variables*, $T(\mathcal{F})$ denotes the set of *ground terms* over $\mathcal{F}$ and $T(\mathcal{F}, \mathcal{X})$ denotes the set of *terms* built up from $\mathcal{F}$ and $\mathcal{X}$. Terms are identified with finite labelled trees as usual. *Positions* are strings of positive integers. Λ denotes the empty string (root position) and "." denotes string concatenation. We use $\mathcal{P}os(t)$ for the set of positions in t, and $\mathcal{F}\mathcal{P}os(t)$

for its set of non-variable positions. The prefix ordering (resp. lexicographic ordering) on positions is denoted by $>$ (resp. $>_{lex}^{N}$) and the strict subterm relationship by $\lhd$. The encompassment ordering, denoted by $\lhd$, is the strict part of the quasi-ordering: $u \trianglelefteq v$ if $v|_p = u\sigma$ for some position p and substitution σ and its equivalence corresponds to variable renaming. Subterm is a special case of encompassment, as well as subsumption for which u is an instance of v. The *subterm* of t at position p is denoted by $t|_p$ and the result of replacing $t|_p$ with u at position p in t is denoted by $t[u]_p$. This notation is also used to indicate that u is a subterm of t. $\mathcal{V}ar(t)$ denotes the set of variables appearing in t. A term is linear if variables in $\mathcal{V}ar(t)$ occur at most once in t, and ground if $\mathcal{V}ar(t) = \emptyset$.

λ-terms will be considered as particular terms, therefore allowing us to reuse the same notations: for each variable $x \in \mathcal{X}$, $\lambda x.$ will be seen as a unary (prefix) function symbol, while the hidden application operator (also denoted by @ when necessary) will be seen as a binary infix symbol. We use $\mathcal{V}ar(t)$ and $\mathcal{BV}ar(t)$ for respectively, the set of free variables and the set of bound variables of t. We will assume for convenience (thanks to α-conversion) that bound variables are all different, and are different from the free ones. Terms over the infinite signature $\mathcal{F} \cup \{\lambda x., @\}$ are called algebraic λ-terms, of which usual terms as well as λ-terms are particular cases.

Typing judgemments will be written as $\Gamma \vdash t : \tau$, meaning that the term t has type τ in the environment Γ. We will use Church simple types most of the time, although richer type disciplines will be considered at the end of the paper. We will sometimes use $\tau(t)$ for the type of t, the environment Γ being assumed.

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

A (possibly higher-order) *term rewriting system* is a set of rewrite rules $R = \{l_i \to r_i\}_i$, where l_i and r_i are algebraic λ-terms such that $l_i \notin \mathcal{X}$ and $\mathcal{V}ar(r_i) \subseteq \mathcal{V}ar(l_i)$. Given a term rewriting system R, a term t rewrites to a term u at position p with the rule $l \to r$ and the substitution σ, written $t \xrightarrow[l \to r]{p} u$, or simply $t \to_R u$, if $t|_p = l\sigma$ and $u = t[r\sigma]_p$. Such a term t is called *reducible*. Irreducible terms are said to be in *normal form*. A term t is *strongly normalizable* if every reduction sequence out of t is finite, hence ends in a normal form of t, denoted by $t{\downarrow_R}$. A substitution γ is strongly normalizable if $x\gamma$ is strongly normalizable for all x. We denote by $\to_R^+$ (resp. $\to_R^=, \to_R^*$) the transitive (resp. reflexive, transitive and reflexive) closure of the rewrite relation $\to_R$. The subindex R will be omitted when clear from the context. Reductions with R will be called *algebraic*.

We will distinguish two particular higher-order rules originating from the λ-calculus, called lambda reductions:

$$(\lambda x.u)v \xrightarrow[\beta]{} u\{x \mapsto v\}$$

$$\lambda x.(ux) \xrightarrow[\eta]{} u \quad \texttt{if } x \notin \mathcal{V}ar(u)$$

Putting together algebraic and lambda reductions yields *combined reductions*:

$$\underset{mix}{\longrightarrow} = \underset{R}{\longrightarrow} \cup \underset{\beta}{\longrightarrow} \cup \underset{\eta}{\longrightarrow}$$

A rewrite relation $\rightarrow$ is

- *confluent* if $t \rightarrow^* u$ and $t \rightarrow^* v$ implies $u \rightarrow^* s$ and $v \rightarrow^* s$ for some s,
- *terminating* (or *strongly normalizing*) if all reduction sequences are finite, in which case it is called a *reduction*,
- *convergent* if it is confluent and terminating.

We sometimes speak of a strongly-normalizing, or confluent, or convergent relation on a subset of the whole set of terms. This assumes of course that this subset is closed under rewriting.

We will make intensive use of well-founded orderings for proving strong normalization properties. As usual, we will use the vocabulary of rewrite systems for orderings: *rewrite orderings* are orderings generated by (possibly infinitely many) rewrite rules, and *reduction orderings* are in addition well-founded. The following results will play a key role, see [6]:

Assume $\rightarrow_1$ and $\rightarrow_2$ are well-founded orderings on sets S_1, S_2. Then $(\rightarrow_1, \rightarrow_2)_{lex}$ is a well-founded ordering on $S_1 \times S_2$.

Assume $\rightarrow_1$ and $\rightarrow_2$ are quasi orderings on sets S_1, S_2 whose strict part is well-founded. Then $(\rightarrow_1, \rightarrow_2)_{comp}$ defined as $(s,t)(\rightarrow_1, \rightarrow_2)_{comp}(u,v)$ iff $s \rightarrow_1 u$ and $t \rightarrow_2 v$ is a quasi ordering on $S_1 \times S_2$ whose strict part is well-founded

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

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

3 Garbage Collecting Lambda Calculus

Reducing a λ-term t requires reducing the existing redexes in t. These redexes may evolve along the derivation, becoming the so-called *residuals*. In particular, they may simply disappear in case a subterm is erased by a rewrite of the form

$$s[(\lambda x.u)v] \underset{\beta}{\longrightarrow} s[u] \;\; \texttt{if} \;\; x \notin \mathcal{V}ar(u).$$

New redexes may also be created along reductions. It is well known that there are three kinds of created β-redexes (which we subdivide into four categories):

1. *hidden redex:* $((\lambda x.\lambda y.u)w)v \underset{\beta}{\longrightarrow} (\lambda y.u\{x \mapsto w\})v \;\; \texttt{if} \;\; x \in \mathcal{V}ar(u)$
2. *projection redex:* $((\lambda x.\lambda y.u)w)v \underset{\beta}{\longrightarrow} (\lambda y.u)v \;\; \texttt{if} \;\; x \notin \mathcal{V}ar(u)$
3. *identity redex:* $((\lambda x.x)(\lambda y.u))v \underset{\beta}{\longrightarrow} (\lambda y.u)v$
4. *application redex:* $(\lambda x.w[xv])(\lambda y.u) \underset{\beta}{\longrightarrow} w\{x \mapsto \lambda y.u\}[(\lambda y.u)(v\{x \mapsto \lambda y.u\})]$

We of course make the implicit assumption that bound variables are renamed appropriately to avoid captures, by considering equivalence classes of terms modulo α-conversion.

We observe that *erasing* steps (associated with K-redexes in Barendregt's terminology) can create redexes of the hidden kind only, as in $((\lambda x.(\lambda y.y))z)u \underset{\beta}{\longrightarrow} (\lambda y.y)u$. They will play a particular role here, because erasing reductions complicate the construction of monotonic orderings. As we can see with the following example, orderings may not be compatible with β-reductions: assuming $v > w$, then $(\lambda x.u)v > (\lambda x.u)w$ by monotonicity. However, both terms $(\lambda x.u)v$ and $(\lambda x.u)w$ reduce to u if $x \notin \mathcal{V}ar(u)$.

In order to tackle this problem, we will split the relation $\longrightarrow_\beta$ into a *erasing* and a *non-erasing* rewrite relation as well as anticipate the creation of hidden redexes by erasing reductions. To this end, we will use a technique originating from [12], in which terms are restructured by distributing applications over the first of two successive abstractions. By pushing down existing redexes, this rule allows previously hidden redexes to show up. As a consequence, it now becomes possible to compute β-normal forms by postponing erasing β-reductions until non-erasing β-reductions are exhausted. In some sense, this technique implements a sort of garbage collector, hence the title of this section.

Definition 1. *Erasing* β-reductions, non-erasing β-reductions and *restructuring* γ-reductions are relations on well-typed terms defined as follows:

$$\begin{aligned} s[(\lambda x.u)v] &\underset{K}{\longrightarrow} s[u] && \texttt{if } x \notin \mathcal{V}ar(u) \\ s[(\lambda x.u)v] &\underset{I}{\longrightarrow} s[u\{x \mapsto v\}] && \texttt{if } x \in \mathcal{V}ar(u) \\ (\lambda x.(\lambda y.u))\ v &\underset{\gamma}{\longrightarrow} \lambda y.((\lambda x.u)\ v) && \texttt{if } x \neq y \end{aligned}$$

We will use $\underset{\beta}{\longrightarrow}$, as expected, for $\underset{K}{\longrightarrow} \cup \underset{I}{\longrightarrow}$, and $\longrightarrow$ for $\underset{I}{\longrightarrow} \cup \underset{\gamma}{\longrightarrow}$.

Note that we could get rid of the condition in the γ-rule by explicitely renaming the variable y into a new variable z, and substituting y by z in u in order to prevent captures. As already said, we prefer to leave these renamings implicit. Note also that we could restrict the application of the γ-rule to the necessary cases, that is when $x \notin \mathcal{V}ar(u)$. We will see later that this restriction does not simplify the framework.

Lemma 2. *(Subject Reduction [12]) Assume* $s \longrightarrow t$, *and* $\Gamma \vdash s : \tau$. *Then* $t : \tau$.

The following straightforward lemma relates γ-reductions with β-reductions:

Lemma 3. *(Correctness)*
Assume that s *and* t *are well-typed terms such that* $s \overset{*}{\longrightarrow} t$. *Then,* $s{\downarrow_\beta} = t{\downarrow_\beta}$.

Proof. Easy induction on the length of the derivation from s to t. For the base case, the result is trivial if $s \underset{I}{\longrightarrow} t$. If $s \underset{\gamma}{\longrightarrow} t$, we simply observe that $s \underset{\beta}{\longrightarrow} \underset{\beta}{\longleftarrow} t$, and then rely on the confluence property of β-reductions.

Since we cannot use $\underset{\beta}{\longrightarrow}$ to compute normal forms without erasing subterms, we will use $\longrightarrow$ instead. To show that $\longrightarrow$ can be used for that purpose, we show that it is terminating, confluent, and approximates β-reductions.

Lemma 4. *[12]* $\underset{\gamma\cup\beta}{\longrightarrow} = \longrightarrow \cup \underset{K}{\longrightarrow}$ *is terminating.*

Corollary 5. *(Strong Normalization)* $\longrightarrow$ *is terminating.*

Note that termination is not straightforward, since I-reductions may create new γ-redexes. In particular, rewriting with $\underset{I}{\longrightarrow}$ a term in γ-normal form may not result in a term in γ-normal form. This happens only when the argument of the abstraction is itself an abstraction which replaces a variable located below an abstraction, as in

$$(\lambda x.((\lambda z.x)\ v))(\lambda y.u) \underset{I}{\longrightarrow} (\lambda z.(\lambda y.u))\ (v\{x \mapsto \lambda y.u)\}$$

Since there must be a replacement for this to happen, this cannot arise with erasing β-reductions. This remark will be crucial in the sequel. On the other hand, this would still happen with the restriction of the γ-rule that we mentionned earlier, since z is not a free variable of u. This justifies the use of the full version.

Lemma 6. *(Local Commutation)*
Assume $s \underset{\gamma}{\longrightarrow} u$ *and* $s \underset{I}{\longrightarrow} t$. *Then* $t \overset{+}{\underset{\gamma}{\longrightarrow}} v$ *and* $u \underset{I}{\longrightarrow} v$ *for some* v.

Proof. By standard case analysis on rewrite positions. Note that there is one critical pair originating from the left hand side of the γ-rule.

Lemma 7. *(Confluence)* $\longrightarrow$ *is confluent.*

Proof. The only critical pair is confluent, by lemma 6. We can therefore conclude by Newmann's lemma, since $\longrightarrow$ is terminating by lemma 5.

Let $s{\downarrow}$ be the normal form of s for $\longrightarrow$. This normal form exists as a consequence of lemma 5 and is unique by lemma 7.

Lemma 8. *(Normal Form Preservation by K)*
Let $s = s{\downarrow}$ *and* $s \underset{K}{\longrightarrow} t$. *Then* $t = t{\downarrow}$.

Proof. K-reductions may create redexes of the projection kind only. But this is of course not posible if the starting term is in γ-normal form.

This lemma suggests a way to compute the β-normal form of a term s, by first computing its γ-normal form t, then its I-normal form u, and finally its K-normal form v. To show this claim, note that $u = s{\downarrow}$ by lemma 7, and v is in I-normal form by lemma 8, hence it is in β-normal form.

This calculus has a other important properties that will turn out to be most useful for the design of orderings. This is the case of the preservation of $\longrightarrow$-normal forms by first-order reductions.

Lemma 9. *(Normal Form Preservation by R)*
Let R be an arbitrary set of first-order rules, $s = s{\downarrow}$ and $s \longrightarrow_R t$. Then $t = t{\downarrow}$.

Proof. Since $\xrightarrow[\beta]{}$ as well as $\xrightarrow[\gamma]{}$ rules are left-linear, the only possibility for creating an $\longrightarrow$-redex by a algebraic reduction is to pop up an abstraction either on the left branch of an application (possibly creating a I-redex), or below another abstraction (hence creating a γ-redex). This requires the algebraic rule to be collapsing, of the form $l \rightarrow x$ with $x \in \mathcal{V}ar(l)$. But then, x must be of base type, hence cannot be instantiated by an abstraction.

We can see here how important the first-order assumption is. An extension of these techniques to higher-order rules seems therefore problematic. We are now ready for generalizing these lemmas to the case of terms which are not yet in normal form:

Lemma 10. *(Normal Form Commutation with K)*
Let $s \xrightarrow[K]{} t$. Then $s{\downarrow} \xrightarrow[K]{+} t{\downarrow}$.

Proof. The proof is by induction on the number of steps from s to $s{\downarrow}$. If $s = s{\downarrow}$, the result follows from lemma 8. Otherwise, let $s \longrightarrow u \xrightarrow{*} s{\downarrow}$. Assuming the existence of a term v such that $t \xrightarrow{*} v$ and $u \xrightarrow[K]{+} v$, we can easily conclude by repeated applications of the induction hypothesis. We are therefore left to prove our assumption. The discussion is similar to the one in the proof of the critical pair lemma [8].

- (Disjoint case). If $s \xrightarrow[K]{} t$ and $s \longrightarrow u$ apply at disjoint positions of s, then they commute, and we are done.
- (Critical pair case). There are no critical pairs.
- (First ancestor case). The $\longrightarrow$-step is above. It applies again to t, yielding v, since the $(\gamma \cup I)$-rules are left-linear. Since they are non-erasing, but may be duplicating, we can also apply the K-rule to u, hence $u \xrightarrow[K]{+} v$.
- (Second ancestor case). The K-step is above. Then, it still applies to u, yielding v, since K-rewrites are left-linear too. On the other hand, the $\longrightarrow$-redex is still in t if it was in the left branch of the application, or has disappeared in case it was in the right branch. Hence $t \xrightarrow{=} v$.

Lemma 11. *(Normal Form Commutation with R)*
Let R be an arbitrary set of first-order rules, and $s \xrightarrow[R]{} t$. Then $s{\downarrow} \xrightarrow[R]{+} t{\downarrow}$.

Proof. The proof is slightly more difficult, since the rules in R may not be left-linear. As a consequence, a simple induction on the length of the derivation from s to $s{\downarrow}$ does not work. We use instead a neotherian induction on $\xrightarrow{+}$ (although an induction on the longest derivation issued from s would do as well).

If s is in $\longrightarrow$-normal form, we are done by lemma 9. Otherwise, $s \longrightarrow u$ for some u. Assuming now the existence of two terms v, w such that $s \xrightarrow{+} v \underset{R}{\longrightarrow} w \xleftarrow{*} t$, we can conclude by noetherian induction that $v\downarrow \xrightarrow[R]{+} w\downarrow$. But the confluence lemma ensures now that $v\downarrow = s\downarrow$, and $w\downarrow = t\downarrow$.

We are left with proving our assumption, and we again discuss by cases as previously. The first three cases are the same, we discuss the last one only: the R-redex, say $l\sigma$, is above the $\longrightarrow$-redex. The problem here is that u may not contain an instance of l anymore in case l is not linear. We must therefore apply further $\longrightarrow$-reductions before to find a term v which contains an instance of l, say $l\sigma'$. v now rewrites to w, replacing $l\sigma'$ by $r\sigma'$. And of course, $t \xrightarrow[R]{*} w$ by rewriting the subterm $r\sigma$ of t into $r\sigma'$.

4 Garbage Collecting Extensional Lambda Calculus

Postponing η-reductions as well as K-reductions is of course possible, since $\beta\eta$-normal forms can be obtaining by postponing the η-rule. But the η-rule is a kind of non-erasing reduction, hence we should apply the η-steps before the erasing β-steps. There is another reason for not postponing the η-rule: this destroys monotonicity of the resulting ordering. This section will therefore follow the same pattern as section 3, starting with an analysis of the rules needed for restructuring terms.

First, we need as previously to make sure that η-redexes cannot be created along K-reductions. Again, there are four kinds of η-redexes created along β-reductions:

1. *hidden η-redex:* $\lambda y.(\lambda x.uy)v \underset{\beta}{\longrightarrow} \lambda y.(u\{x \mapsto v\})y$ if $y \notin \mathcal{V}ar(u, v)$ and $x \in \mathcal{V}ar(u)$.
2. *projection η-redex:* $\lambda x.u((\lambda y.x)v) \underset{\beta}{\longrightarrow} \lambda x.ux$ if $x \neq y$ and $x \notin \mathcal{V}ar(u)$.
3. *identity η-redex:* $\lambda x.u((\lambda y.y)x) \underset{\beta}{\longrightarrow} \lambda x.ux$ if $x \notin \mathcal{V}ar(u)$.
4. *η-redex:* $\lambda x.u((\lambda y.x)v) \underset{\beta}{\longrightarrow} \lambda x.ux$ if $x \neq y$ and $x \notin \mathcal{V}ar(u)$.

Again, *erasing* β-steps can create redexes of the projection kind only, as in $\lambda y.((\lambda x.u\ y)\ v) \underset{\beta}{\longrightarrow} \lambda y.uy \longrightarrow_\eta u$ if $x, y \notin \mathcal{V}ar(u)$ and $x \neq y$. Anticipating the obtained η-reduction will require to restructure the starting term as $(\lambda x.\lambda y.z\ y)\ v$. This is exactly a right to left use of the γ-rule, which must therefore be abandonned.

The rules below are rule schemas rather than rules. But one can convince oneself that they are easily implementable and yield a linear-time algorithm for computing the restructured version of any term. They will allow us to postpone erasing β-reductions by poping up K-redexes (instead of pushing them down as implicitely done by the γ-reductions of section 3).

Definition 12. *Distributive* reductions are relations on well-typed terms defined as follows:

$$\lambda y.s\{z \mapsto u\} \underset{D_\lambda}{\longrightarrow} s\{z \mapsto \lambda y.u\} \qquad \texttt{if } s \xrightarrow[K \cup R]{+} z$$

$$(s\{z \mapsto \lambda y.u\})v \underset{D_@}{\longrightarrow} s\{z \mapsto (\lambda y.u)\ v\} \texttt{ if } s \xrightarrow[K \cup R]{+} z$$

We will use $\underset{D}{\longrightarrow}$ for $\underset{D_@}{\longrightarrow} \cup \underset{D_\lambda}{\longrightarrow}$, and $\longrightarrow$ for $\underset{I}{\longrightarrow} \cup \underset{\eta}{\longrightarrow} \cup \underset{D}{\longrightarrow}$.

To avoid captures, it is assumed in the above rule that the variable y does not appear either free or bound in the context of the redex. Note that the distributivity rule for abstractions may create free variables. Indeed, if $y \in \mathcal{V}ar(s[\,])$, y is bound on the left hand side, but free on the right hand one. That a free variable is created is not a problem, however, since the term $s[\,]$ will eventually disappear along the erasing β-reductions $s[\lambda y.u] \xrightarrow[K]{*} \lambda y.u$.

One may think that the simplest possible distributivity rules are enough to cover all cases, but this is not the case. Consider as an example the rule for abstractions:

$$\lambda y.((\lambda x.u)v) \underset{simple\ D_\lambda}{\longrightarrow} (\lambda x.(\lambda y.u))v \texttt{ if } y \in \mathcal{V}ar(u) \texttt{ and } x \notin \mathcal{V}ar(u)$$

Taking now a more general pattern of the distributivity rule for abstractions, say $\lambda y.(\lambda x_1.((\lambda x_2.u)\ v_2))v_1$ with $x_2 \notin \mathcal{V}ar(u), x_1 \notin \mathcal{V}ar(u), x_1 \in \mathcal{V}ar(v_2)$ and $y \in \mathcal{V}ar(u)$, we get:

$$\lambda y.(\lambda x_1.((\lambda x_2.u)\ v_2))\ v_1 \underset{D_\lambda}{\longrightarrow} (\lambda x_1.(\lambda x_2.(\lambda y.u))\ v_2)\ v_1$$

since $(\lambda x_1.((\lambda x_2.z)\ v_2))\ v_1 \longrightarrow_+^2 z$, but $\lambda y.(\lambda x_1.((\lambda x_2.u)\ v_2))\ v_1$ is in normal-form for $\underset{simpleD_\lambda}{\longrightarrow}$. The $simpleD_\lambda$-rule will be enabled, of course, after rewriting the K-redex $(\lambda x_2.z)\ v_2$, but we will see that we actually need the property that K-rewrites do not create new D-redexes. We could of course think of adopting a more liberal condition for the $simpleD_\lambda$ rule, namely

$$\lambda y.((\lambda x.u)v) \underset{simple\ D_\lambda}{\longrightarrow} (\lambda x.(\lambda y.u))v \texttt{ if } y \in \mathcal{V}ar((\lambda x.u)\ v) \texttt{ and } x \notin \mathcal{V}ar(u)$$

This variation together with the I-rule yields non-confluent derivations:

$$(\lambda x.u)\ v\{y \mapsto w\} \underset{I}{\longleftarrow} (\lambda y.((\lambda x.u)v))\ w \underset{simpleD_\lambda}{\longrightarrow} (\lambda y.(\lambda x.u)\ v)\ w$$

and we will again see that confluence is needed.

In section 3, algebraic rewrite rules could not create λ-redexes nor γ-redexes 9, hence the γ-rule could be studied inside the λ-calculus alone. It is no more the case with the above restructuring rules which refer to the algebraic rewrite system. We could avoid it, to the price of giving another restructuring rule for distributing algebraic symbols over K-redexes:

$$f(\ldots,(\lambda x.u)\ v,\ldots) \underset{D_\mathcal{F}}{\longrightarrow} (\lambda x.f(\ldots,u,\ldots))\ v \texttt{ if } x \notin \mathcal{V}ar(f(\ldots,u,\ldots))$$

Besides, confluence would require to forbid the application of this rule if there is a possibility later in the computation that a γ-redex surfaces at the left of the current γ-redex $(\lambda x.u)\ v$. The present solution gives less restructuring and is easier to work out technically, which explains our choice.

Lemma 13. *(Subject Reduction) Assume* $s \longrightarrow t$, *and* $\Gamma \vdash s : \tau$. *Then* $t : \tau$.

Proof. Since the first-order rules preserve types by definition, and $\beta\eta$-reductions as well, then s and z must have the same type if $s \underset{K \cup R}{\longrightarrow} z$. Hence, the distributivity rules preserve types as well.

The following straightforward lemma relates distributive reductions with β-reductions:

Lemma 14. *(Correctness)*
Assume that s *and* t *are well-typed* λ*-terms such that* $s \xrightarrow[D]{*} t$. *Then* $s{\downarrow_\beta} = t{\downarrow_\beta}$.

Since we cannot use $\underset{\beta}{\longrightarrow}$ to compute normal form without erasing subterms, we will use $\longrightarrow$ instead. To show that $\longrightarrow$ can be used for that purpose, we show that it is terminating, confluent, and that it approximates $\beta\eta$-reductions. Some elaboration will be needed.

Lemma 15. $\underset{D}{\longrightarrow}$ *terminates.*

Proof. We interpret a term t by the multiset of lengths of redex positions in t. A simple analysis shows that this multiset decreases along distributive reductions (and shows as well that the condition $y \in \mathcal{V}ar(u)$ is crucial for both rules).

Lemma 16. *Let* $s \underset{I \cup \eta}{\longrightarrow} t$. *Then* $s{\downarrow_D} \xrightarrow[I \cup \eta]{+} t{\downarrow_D}$.

Proof. Routine check. Note that $\lambda y.(s'[u]@y)$ cannot be at the same time an η-redex (requiring $y \notin \mathcal{V}ar(u)$ and a D_λ-redex (requiring $y \in \mathcal{V}ar(u)$). The only critical pair case is therefore obtained by rewriting the term $s[\lambda y.(u\ y)]v \underset{D_@}{\longrightarrow} s[(\lambda y.(u\ y))\ v]$, but the η-redex is still in the result.

Corollary 17. $\underset{D \cup I \cup \eta}{\longrightarrow}$ *is terminating.*

Proof. We interpret a term t by the number of I-redexes in $t{\downarrow_D}$. Lemma 16 shows that this number decreases along I-derivations.

Lemma 18. $\longrightarrow$ *is confluent.*

Proof. Since $\longrightarrow$ is terminating, by Newmann's lemma, we need to prove local confluence only. As usual, it is enough to consider the critical pairs, actually those critical pairs involving one of the distributivity rules, since the others are already confluent in the simply typed λ-calculus (noticing that erasing β-reductions are not used to close diagrams of non-erasing β-reductions or η-reductions).

- Distributivity rules with themselves: let their respective left-hand sides be $\lambda y.s[u]$ and $s'[\lambda y'.u']@v$.
  - Case 1: D_λ inside $D_@$ at the position of $\lambda y'.u'$. Since the subterm $\lambda y'.u'$ is just copied by the rule $D_@$, the corresponding critical pair commutes trivially.
  - Case 2 : D_λ inside $D_@$ at a position in s'. This cannot be an ancestor position of the position of $\lambda y'.u'$, because the condition $y \in \mathcal{V}ar(u)$ wouyld then prevent $s'[z]$ to reduce to z via K-reductions. Hence $\lambda y'.u'$ and $\lambda y.s[u]$ are at disjoint positions, and then the critical pair can be made confluent by means of K-reductions.
  - Case 3 : $D_@$ inside D_λ at a position in s. For the same reason as above, the two subterms u and $s'[\lambda y'.u']@v$ must be at disjoint positions, and the critical pair can be made confluent by means of K-reductions.
- Distributivity rule with the non-erasing β-rule or η-rule:
  - Case 1: η inside $D_@$ at the position of $\lambda y'.u'$. This case commutes as previously.
  - Case 2: the non-erasing β-rule (or the η-rule) must apply at a position in s (resp. s'), which must be disjoint from the position of u (resp. $\lambda y'.u'$) for the same reason as above, and the same technique applies.

Let $s{\downarrow}$ be the normal form of s for $\longrightarrow$. This normal form exists as a consequence of lemma 17 and is unique by lemma 18.

Lemma 19. *(Normal Form Preservation by K)*
Let $s = s{\downarrow}$ and $s \underset{K}{\longrightarrow} t$. Then $t = t{\downarrow}$.

Proof. By assumption, $s = s[(\lambda x.u)v]_p$ with $x \notin \mathcal{V}ar(u)$ and $t = s[u]_p$. We show by contradiction that t must itself be in normal form for $\longrightarrow$.

Assume $t \underset{\eta}{\longrightarrow} t'$. Since s is in normal form, necessarily $p = q.2$, $t = s[\lambda y.u]_q$, and $u = (u'y)$. Hence $s = s[\lambda y.(\lambda x.u)v]_q$. Since $(\lambda x.z)v \underset{K}{\longrightarrow} z$, the distributivity rule for abstractions applies to s which contradicts the assumption.

Assume $t \underset{I}{\longrightarrow} t'$. Since s is in normal form, necessarily $p = q.1$, $t = s[(uw)]_q$ and u is an abstraction. Hence $s = s[((\lambda x.u)v)w]_q$. Since $(\lambda x.z)v \underset{K}{\longrightarrow} z$ the distributivity rule for applications applies to s which contradicts the assumption.

Assume $t \underset{D@}{\longrightarrow} t'$. Then $p = q.1.r$, $t = s[(w[u']_{r.r'})v']_q$ $(u' = u|_{r'})$, and $s = s[(w[(\lambda x.w|_r)v]_r)v']_q$. Since $w[z]_{r.r'} \underset{K}{\longrightarrow} z$, $w[(\lambda x.w|_r[z]_{r'})v]_r \underset{K}{\longrightarrow} z$ and the distributivity rule for applications applies, contradicting our assumption.

Assume $t \underset{D\lambda}{\longrightarrow} t'$. The proof is similar as above.

A consequence of the above lemma is that we can compute the $\beta\eta$-normal form of t by computing first its $DI\eta$-normal form, then its K-normal form. We could actualy again decompose the first step by postponing η-steps.

Lemma 20. *(Normal Form Preservation by R)*
Let $s = s{\downarrow}$ and $s \underset{R}{\longrightarrow} t$. Then $t = t{\downarrow}$.

Proof. By assumption, $s = s[l\sigma]_p$ and $t = s[r\sigma]_p$. We show that t must itself be in normal form for $\longrightarrow$.

The result is straightforward if $r \notin X$. Otherwise, the reasonning is similar to the previous proof.

Lemma 21. *(Normal Form Commutation with K)*
Let $s \underset{K}{\longrightarrow} t$. *Then* $s{\downarrow} \underset{K}{\overset{+}{\longrightarrow}} t{\downarrow}$.

Proof. By induction on the number m of steps from s to $s{\downarrow}$, as in lemma 10.

Lemma 22. *(Normal Form Commutation with R)*
Let $s \underset{R}{\longrightarrow} t$. *Then* $s{\downarrow} \underset{R}{\overset{+}{\longrightarrow}} t{\downarrow}$.

Proof. By noetherian induction on $\longrightarrow$, as in lemma 11. The proof is basically the same, since all rules in $\longrightarrow$ are non-erasing and left linear.

These results have been proved for plain rewriting. They generalize without difficulty to the case of rewriting modulo an equational theory whose equivalence classes are finite, to the case of reductive conditional rewrite rules [11], as well as to their combination. Of course, we will be primarily interested in the associative-commutative case.

5 λ-Extension of a Rewrite Ordering

We are now ready to introduce our ordering on mixed terms and prove that it is indeed a rewrite ordering.

Definition 23. Assume $\underset{R}{\overset{+}{\longrightarrow}}$ is a reduction relation (generated by a set R of typed rewrite rules) on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ naturally extended to $\mathcal{T}(\mathcal{F} \cup \{\lambda, @\}, \mathcal{X})$.

We define $s > t$ iff
(i) $s{\downarrow} \underset{R \cup K}{\overset{+}{\longrightarrow}} t{\downarrow}$ or
(ii) $s{\downarrow} = t{\downarrow}$ and $s \ (\longrightarrow \cup \underset{K}{\longrightarrow})^+ \ t$.

Note that the ordering $>$ is well-defined, since $\longrightarrow$ is terminating and confluent. Note also that this definition applies to the typed lambda calculus with or without extensionality, just by appropriately instanciating the relation $\longrightarrow$. We now elaborate the proof of our main theorem that $>$ is indeed a reduction ordering.

Lemma 24. *Let g be a function symbol not occurring in a TRS R. Then* $R \cup \{g(\ldots, x_i, \ldots) \rightarrow x_i\}$ *is terminating if and only if R is.*

Let G be $\{g(\ldots, x_i, \ldots) \rightarrow x_i\}$ and let R_g be $R \cup G$. To prove this lemma we will define a well-founded extension ordering $\succ_g$ including $\underset{R_g}{\overset{+}{\longrightarrow}}$.

Let us first define two mappings, one from terms to non-negative integers and another one from terms to multisets of terms:

$$|f(t_1,\ldots,t_m)|_g = \begin{cases} max(\{|t_1|_g,\ldots,|t_m|_g\}) & \text{if } f \neq g \\ 1 + max(\{|t_1|_g,\ldots,|t_m|_g\}) & \text{otherwise} \end{cases}$$

$$A(f(t_1,\ldots,t_m)) = \begin{cases} A(t_1) \cup \ldots \cup A(t_m) & \text{if } f \neq g \\ \{t_1,\ldots,t_m\} & \text{otherwise} \end{cases}$$

We have $s \succ_g t$ iff

- $|s|_g > |t|_g$,
- $|s|_g = |t|_g$ and $s{\downarrow_G} \xrightarrow[R]{+} t{\downarrow_G}$,
- $|s|_g = |t|_g$ and $s{\downarrow_G} = t{\downarrow_G}$ and $A(s)(\succ_g)_{mul}A(t)$

Lemma 25. *For all term t and for all $t' \in A(t)$, we have $|t|_g > |t'|_g$.*

Proof. Let t be $f(t_1,\ldots,t_n)$. We proceed by induction on the size of t.

1. $f \neq g$. By definition, $A(t) = A(t_1) \cup \ldots \cup A(t_n)$. By induction hypothesis, for all $t' \in A(t_i)$, we have $|t_i|_g > |t'|_g$. Then, since $|t|_g \geq |t_i|_g$ and for all $t' \in A(t)$ there is some t_i s.t. $t' \in A(t_i)$, we have $|t|_g > |t'|_g$ for all $t' \in A(t)$.
2. $f = g$. Then $A(t) = \{t_1,\ldots,t_n\}$ and $|t|_g = 1 + max\{|t_1|_g,\ldots,|t_m|_g\} > |t_i|_g$.

Lemma 26. *If R is terminating then $\succ_g$ is a well-founded ordering.*

Proof. Transitivity and irreflexivity are proved by a simple case analysis on the definition. For well-foundedness we will obtain a contradiction from the existance of some term s_1, minimal wrt. $|s_1|_g$, starting an infinite decreasing sequence $s_1 \succ_g s_2 \succ_g \ldots$

Since $s \succ_g t$ implies $|s|_g \geq |t|_g$ for all s and t, by minimality of s_1 we have $|s_1|_g = |s_i|_g$ for all i. Then by well-foundedness of $\xrightarrow[R]{+}$, there must exist some s_k, s.t. for all $j > k$ we have $s_k{\downarrow_G} = s_j{\downarrow_G}$. Let t_j be s_{k+j-1} (remind that $|s_1|_g = |t_1|_g$ and $|t_1|_g = |t_i|_g$ for all i). Then we have $A(t_1)(\succ_g)_{mul}A(t_2)(\succ_g)_{mul}\ldots$. Now by (the proof of the double implication between the well-foundedness of an ordering and its multiset extension in) [7], we can extract an infinite sequence $t'_1 > t'_2 > \ldots$ starting from an element in $A(t_1)$. Now by lemma 25, we have $|s_1|_g = |t_1|_g > |t'_1|_g$, which contradicts the minimality of s_1.

Lemma 27. *If $s \xrightarrow[R]{} t$ then $|s|_g \geq |t|_g$.*

Proof. We first prove that for all terms l built over the signature of R (i.e. not containing any g) we have $|l\sigma|_g = max\{|x\sigma|_g \mid x \in \mathcal{V}ar(l)\}$. We proceed by induction on the size of l.

If l is a constant or variable the result trivially holds. If $l = f(l_1,\ldots,l_n)$ (with $f \neq g$) then $|l\sigma|_g = max\{|l_1\sigma|_g,\ldots,|l_n\sigma|_g\}$, and by induction hypothesis, we have $|l_i\sigma|_g = max\{|x\sigma|_g \mid x \in \mathcal{V}ar(l_i)\}$ for all i, which implies the result.

Now we will prove $s \xrightarrow[R]{} t$ implies $|s|_g \geq |t|_g$. Assume $s = C[l\sigma]$ and $t = C[r\sigma]$ for some rule $l \to r$ in R and context C. By the previous result, since $\mathcal{V}ar(l) \supseteq \mathcal{V}ar(r)$, we have $|l\sigma|_g \geq |r\sigma|_g$. Then it follows that $|C[l\sigma]|_g \geq |C[r\sigma]|_g$.

Lemma 28. *If $|s|_g \geq |t|_g$ then there exists an $s' \in A(s)$ s.t. for all $t' \in A(t)$ we have $|s'|_g > |t'|_g$, and hence $A(s)(\succ_g)_{mul}A(t)$.*

Proof. Assume $s = f(s_1, \ldots, s_m)$ and $t = h(t_1, \ldots, t_n)$. We proceed by induction on the (sum of the) sizes of s and t.

1. $f \neq g$. By definition, $|s|_g = max\{|s_1|_g, \ldots, |s_m|_g\} > |t|_g$. Then there is some s_i s.t. $|s_i|_g > |t|_g$, and by induction hypothesis we have that there exists some $s' \in A(s_i)$ s.t. for all $t' \in A(t)$ we have $|s'|_g > |t'|_g$. Since $A(s) = \ldots \cup A(s_i) \cup \ldots$, the result holds.
2. $h \neq g$. By definition, $|s|_g > max\{|t_1|_g, \ldots, |t_n|_g\} = |t|_g$. Hence $|s|_g > |t_i|_g$ for all i.Then, by induction hypothesis, we have that for all i there exists some $s' \in A(s)$ s.t. for all $t' \in A(t_i)$ we have $|s'|_g > |t'|_g$. Then taking the maximal such s', since $A(t) = A(t_1) \cup \ldots \cup A(t_n)$, the result follows.
3. $f = h = g$. By definition, $|s|_g = 1 + max\{|s_1|_g, \ldots, |s_m|_g\} > 1 + max\{|t_1|_g, \ldots, |t_m|_g\} = |t|_g$, which implies $max\{|s_1|_g, \ldots, |s_m|_g\} > max\{|t_1|_g, \ldots, |t_m|_g\}$. Therefore there is some $s' \in A(s) = \{s_1, \ldots, s_m\}$ s.t. for all $t' \in A(t) = \{t_1, \ldots, t_m\}$, we have $|s'|_g > |t'|_g$.

Lemma 29. *If $s \xrightarrow[R_g]{} t$ then $s \succ_g t$.*

Proof. We proceed by induction on the (sum of the) sizes of s and t. Assume $s = C[l\sigma]$ and $t = C[r\sigma]$ for some rule $l \to r$ in R_g, substitution σ and context C.

If C is the empty context then there are two cases:

1. if $l \to r$ in G then we have $|l\sigma|_g = 1 + max\{\ldots, |x_i\sigma|_g, \ldots\} > |x_i\sigma|_g = |r\sigma|_g$.
2. if $l \to r$ in R then, by lemma 27, we have $|l\sigma|_g \geq |r\sigma|_g$, and $l\sigma{\downarrow_G} = l\sigma'$ and $r\sigma{\downarrow_G} = r\sigma'$ for some σ' (with $x\sigma' = x\sigma{\downarrow_G}$ for all x in $\mathcal{D}om(\sigma)$). Then, since $l\sigma' \xrightarrow[R]{+} r\sigma'$ the result holds.

If C is not empty, assume $s = f(\ldots, s', \ldots)$ and $t = f(\ldots, t', \ldots)$ with $s' = C'[l\sigma]$ and $t' = C'[r\sigma]$. By induction hypothesis, we have $s' \succ_g t'$. By lemma 27 we have $|s|_g \geq |t|_g$. Suppose $|s|_g = |t|_g$ (otherwise it is already finished). On the other hand, either $s{\downarrow_G} = t{\downarrow_G}$ or $s{\downarrow_G} \xrightarrow[R]{+} t{\downarrow_G}$. Suppose $s{\downarrow_G} = t{\downarrow_G}$ (otherwise it is again finished). Then we distinguish the following two cases:

1. $f \neq g$. We have $s'{\downarrow_G} = t'{\downarrow_G}$ (otherwise $s{\downarrow_G} \neq t{\downarrow_G}$, contradicting the assumption) and $A(s) = S \cup A(s')$ and $A(t) = S \cup A(t')$ for some S. If $|s'|_g > |t'|_g$ then by lemma 28 we have $A(s')(\succ_g)_{mul}A(t')$, implying $A(s)(\succ_g)_{mul}A(t)$. Otherwise we have $|s'|_g = |t'|_g$ and $s'{\downarrow_G} = t'{\downarrow_G}$, and therefore $A(s')(\succ_g)_{mul}A(t')$, which implies again $A(s)(\succ_g)_{mul}A(t)$.
2. $f = g$. We have $A(s) = S \cup s'$ and $A(t) = S \cup t'$ for some S, and hence $A(s)(\succ_g)_{mul}A(t)$.

Lemma 30. *Assume that R is a reduction relation. Then $\xrightarrow[R \cup K]{+}$ is well-founded.*

Proof. Consider $R' = R \cup \{@(x,y) \to x\} \cup \bigcup_{x \in \mathcal{X}} \{\lambda x.y \to y\}$. Since R is terminating, applying lemma 24 for the first projection rule, then for the projection rules for the λ-operators proves that R' is terminating.

Then, as $\xrightarrow[R \cup K]{} \subseteq \xrightarrow[R']{}$ (note that the K-steps can be encoded as two steps using the projection rules added to R), we conclude that $\xrightarrow[R \cup K]{}$ is terminating.

Lemma 31. *$>$ is an ordering on mixed terms.*

Proof. Irreflexivity. By uniqueness of normal forms w.r.t. $\longrightarrow$, we have $s{\downarrow} = s{\downarrow}$. Now, $\xrightarrow[R \cup K]{+}$ and $\xrightarrow[\gamma \cup \beta]{+}$ being well-founded by lemmas 30 and 4, they are irreflexive. Hence $s \not> s$.
Transitivity. It follows from the transitivity of the rewrite relations.

Lemma 32. *(Well-foundedness) $>$ is well-founded.*

Proof. Suppose there exists an infinite sequence $s_1 > s_2 > \ldots$ Then since $\xrightarrow[R \cup K]{+}$ is well-founded, there must be some s_j s.t. $s_j{\downarrow} = s_k{\downarrow}$ for all $k \geq j$. But then we have $s_j \xrightarrow[\gamma \cup \beta]{+} s_{j+1} \xrightarrow[\gamma \cup \beta]{+} \ldots$ contradicting the well-foundedness of $\xrightarrow[\gamma \cup \beta]{+}$.

Lemma 33. *(Monotonicity) $>$ is monotonic.*

Proof. If $s > t$ then $u[s] > u[t]$ for all terms s and t and context u. Remark that $u[s]{\downarrow} = u[s{\downarrow}]{\downarrow}$ and $u[t]{\downarrow} = u[t{\downarrow}]{\downarrow}$ by confluence property of $\longrightarrow$.

1. if $s{\downarrow} \xrightarrow[R \cup K]{+} t{\downarrow}$ then $u[s{\downarrow}] \xrightarrow[R \cup K]{+} u[t{\downarrow}]$, hence $u[s]{\downarrow} \xrightarrow[R \cup K]{+} u[t]{\downarrow}$ by lemmas 10 and 11 and the previous remark.
2. if $s{\downarrow} = t{\downarrow}$ then $u[s]{\downarrow} = u[s{\downarrow}]{\downarrow} = u[t{\downarrow}]{\downarrow} = u[t]{\downarrow}$ and $s \xrightarrow[\gamma \cup \beta]{+} t$ implies $u[s] \xrightarrow[\gamma \cup \beta]{+} u[t]$.

Lemma 34. *$>$ is stable under substitutions.*

Proof. If $s > t$ then $s\sigma > t\sigma$ for all terms s and t and substitution σ. By confluence $(s\sigma){\downarrow} = (s{\downarrow}\,\sigma){\downarrow}$ and $(t\sigma){\downarrow} = (t{\downarrow}\,\sigma){\downarrow}$.

1. if $s{\downarrow} \xrightarrow[R \cup K]{+} t{\downarrow}$ then $s{\downarrow}\,\sigma \xrightarrow[R \cup K]{+} t{\downarrow}\,\sigma$, and by lemmas 10 and 11 and the previous remark $(s\sigma){\downarrow} \xrightarrow[R \cup K]{+} (t\sigma){\downarrow}$.
2. if $s{\downarrow} = t{\downarrow}$ then $(s\sigma){\downarrow} = (t\sigma){\downarrow}$ by the previous remark, and $s \xrightarrow[\gamma \cup \beta]{+} t$ implies $s\sigma \xrightarrow[\gamma \cup \beta]{+} t\sigma$.

Lemma 35. *Assume $\xrightarrow[R]{+}$ is given by a set R of (typed) rewrite rules. Then, $\xrightarrow[R \cup \beta]{} \subseteq >$.*

Proof. Since $>$ is a reduction ordering, we simply need to prove the inclusion of $\underset{\beta}{\longrightarrow}$ and $\underset{R}{\longrightarrow}$.

Let $s \underset{I}{\longrightarrow} t$. By lemma 7, $s\downarrow = t\downarrow$. Hence $s > t$ by case (ii).

Let $s \longrightarrow_K t$. Then, by lemma 10, $s\downarrow \longrightarrow_K{}^+ t\downarrow$ and hence $s\downarrow \overset{+}{\underset{R\cup K}{\longrightarrow}} t\downarrow$, implying $s > t$ by case (i).

Let $s \underset{R}{\longrightarrow} t$. By lemma 11, $s\downarrow \underset{R}{\longrightarrow} t\downarrow$. Hence $s > t$ by case (i).

As a consequence,

Theorem 36. *$>$ is a reduction ordering on mixed terms which restricts to $\overset{+}{\underset{R}{\longrightarrow}}$ on algebraic terms and to $(\longrightarrow \cup \underset{K}{\longrightarrow})^+$ on typed lambda terms.*

Note that $\longrightarrow \cup \longrightarrow_K$ is the union of $\longrightarrow_\beta$ and the restructuring rules.

6 λ-Extension of the Recursive Path Ordering

In this section, we consider the lambda extension to mixed terms of the recursive path ordering, which can be seen as a particular reduction relation. Our function symbols will be split into three sets: the set Lex of lexicographic symbols, the set Mul of multiset symbols, and the set of monotonic symbols, that is application and λ-abstractions. The ordering will be generated by a well-founded ordering $>_{\mathcal{F}}$ on $\mathcal{F}$, considered as a well-founded ordering $>_{\mathcal{F}\cup\{@,\lambda\}}$ on the whole (infinite) signature.

The definition is an adaptation of the definition of the previous section, and of the definition of the lexicographic path ordering:

Definition 37. We define $s \succ_{lpo}^{\beta} t$ iff
(i) $[\![s\downarrow]\!] \succ_{lpo}^{e\beta} [\![t\downarrow]\!]$ or
(ii) $s\downarrow = t\downarrow$ and $s\ (\longrightarrow \cup \underset{K}{\longrightarrow})^+\ t$.
where
$s = f(\vec{s}) \succeq_{lpo}^{e\beta} g(\vec{t})$ iff

1. $f \in \mathcal{F}$ and $s_i \succeq_{lpo}^{e\beta} t$ for some $s_i \in \vec{s}$
2. $s = \lambda x(u)@v$, $x \notin Var(u)$ and $u \succeq_{lpo}^{e\beta} t$
3. $f >_{\mathcal{F}\cup\{@,\lambda\}} g$, and $f(\vec{s}) \succ_{lpo}^{e\beta} t_i$ for all $t_i \in \vec{t}$
4. $f = g \in Lex$, $f(\vec{s}) \succ_{lpo}^{e\beta} t_i$ for all $t_i \in \vec{t}$, and $\vec{s}\ (\succeq_{lpo}^{e\beta})_{lex}\ \vec{t}$
5. $f = g \in Mul$, and $\vec{s}\ (\succeq_{lpo}^{e\beta})_{mul}\ \vec{t}$
6. $f = g = @$, and $\vec{s}\ (\succeq_{lpo}^{e\beta})_{mon}\ \vec{t}$
7. $f = \lambda x, g = \lambda y$, and $s_1 \succeq_{lpo}^{e\beta} t_1\{y \mapsto x\}$

and
$[\![\lambda x.u]\!] = \lambda x([\![u]\!])$
$[\![x]\!] = x$
$[\![(uv)]\!] = [\![u]\!]@[\![v]\!]$
$[\![f(\vec{t})]\!] = f([\![\ \vec{t}\ ]\!])$

Theorem 38. *$\succ^{\beta}_{lpo}$ is a reduction ordering on mixed terms which restricts to $\succ_{lpo}$ on algebraic terms and to $(\longrightarrow \cup \underset{K}{\longrightarrow})^+ \supset \longrightarrow^+_\beta$ on typed lambda terms.*

Proof. Note that on one hand, cases 1, 3 , 4 and 5 of the definition of $\succeq^{e\beta}_{lpo}$ define a recursive path ordering $\succeq_{lpo}$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$, and on the other hand, cases 2, 6 and 7 define $\xrightarrow[K]{+}$.

Let R_{lpo} be an infinite TRS containing all rules $s \to t$ iff $s \succ_{lpo} t$. Here, we assume that we have a single basic type. If we have several, we will consider the associated single sorted rewrite system obtained from R by equating all sorts. The new system has a richer rewrite relation on terms, hence its termination will do.

Then we trivially have that $\xrightarrow[R_{lpo}]{+}$ is equivalent to $\succ_{lpo}$ and $\underset{R_{lpo} \cup K}{\longrightarrow}$ is equivalent to $\succ^{e\beta}_{lpo}$. Now by theorem 36 the result holds.

7 Conclusion

Besides a new proof of strong normalization of the simply typed λ-calculus combined with an arbitrary first-order rewrite system, we have made a significant contribution with the design of an explicit (rpo-like) rewrite orderings for the terms in the combination. Although we have considered the case of simply typed lambda calculus only, we believe that our techniques extend to the polymorphic and intersection type disciplines. We do not have ideas yet about the dependent type case for which we think that additional techniques are needed. Finally, we also like to mention the importance of lemma 24. It is well known that we cannot add two projection rules for a given new symbol to a terminating rewrite system without loosing termination [15]. We have shown here that we can add one, drawing a precise line between termination and non-termination for this sort of first-order combination.

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. Val Breazu-Tannen. Combining algebra and higher-order types. In *Proc. 3rd IEEE Symp. Logic in Computer Science, Edinburgh*, July 1988.

4. Val Breazu-Tannen and Jean Gallier. Polymorphic rewriting conserves algebraic strong normalization. *Theoretical Computer Science*, 1990. to appear.
5. Nachum Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17(3):279–301, March 1982.
6. Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–309. North-Holland, 1990.
7. Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, August 1979.
8. Gérard Huet and D. Oppen. Equations and rewrite rules: a survey. In R. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 349–405. Academic Press, 1980.
9. Jean-Pierre Jouannaud and Mitsuhiro Okada. Executable higher-order algebraic specification languages. In *Proc. 6th IEEE Symp. Logic in Computer Science, Amsterdam*, pages 350–361, 1991.
10. Jean-Pierre Jouannaud and Mitsuhiro Okada. Abstract data type systems. Research Report 975, Université de Paris Sud, June 1995.
11. Jean-Pierre Jouannaud and B. Waldmann. Reductive conditional term rewriting systems. In *Proc. Third IFIP Working Conference on Formal Description of Programming Concepts*, Ebberup, Denmark, 1986.
12. A.J. Kfoury and J. B. Wells. New notions of reduction and non-semantic proofs of strong normalization in typed λ-calculi. In *Proc. 10th IEEE Symp. Logic in Computer Science, San Diego*, 1995.
13. T. Nipkow. Higher order critical pairs. In *Proc. IEEE Symp. on Logic in Comp. Science*, Amsterdam, 1991.
14. Albert Rubio. Automated deduction with constrained clauses. PhD Thesis, Univ. de Catalunya, 1994.
15. Y. Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, April 1987.

This article was processed using the LaTeX macro package with LLNCS style