A Total AC-Compatible Reduction Ordering on Higher-Order Terms

Daria Walukiewicz*

Institute of Informatics, Warsaw University
Banacha 2, 02-097 Warsaw, Poland
email: daria@mimuw.edu.pl

Abstract. The higher-order rewriting in presence of associative and commutative (AC) symbols is considered. A higher-order reduction ordering which allows to state the termination of a given rewriting system is presented. This ordering is an AC-extension of λ-RPO and is defined on simply-typed λ-terms in β-normal η-long form. It is total on ground terms.

1 Introduction

Automated methods of proving termination of rewrite systems are used in many theorem provers and specification development systems. Most of these methods are based on finding a suitable reduction ordering, i.e., a well-founded partial order which is stable under context and substitution. The most popular reduction ordering in first-order rewriting is RPO – *recursive path ordering*. Its principle is to generate recursively an ordering on terms from a given ordering on function symbols called *precedence*. The recursive definition of RPO allows to apply and to implement it easily.

Higher-order rewrite rules are used in the programming languages, like ML, Elf, or theorem provers, like Isabelle. Unfortunately, there are only few methods for proving termination of higher-order systems. In the case of first-order pattern matching it is known that a higher-order rewrite system has termination property when its rules follow a generalised form of a primitive recursive schema of higher type (see [5, 4]). In the general case of higher-order pattern matching one can use λ-RPO proposed by Jouannaud and Rubio [6].

λ-RPO is a *higher-order reduction ordering*, i.e., a well-founded ordering stable under ground contexts and substitutions and moreover compatible with the underlying typed λ-calculus. Since we are interested in higher-order pattern matching, compatibility with typed λ-calculus means compatibility with $\beta\eta$-conversions. This is easily achieved by defining λ-RPO on canonical representatives – terms in β-normal η-long form. Compared to RPO, λ-RPO needs not only a precedence on function symbols but also a precedence on types. Terms are compared first by type, then by top symbol, finally the comparison proceeds

* Partly supported by Polish KBN Grant 8 T11C 034 10.

recursively to subterms. Because of its recursive definition, λ-RPO is especially well suited for automation.

Practical applications often introduce AC-symbols (like $+$) into the signature. While rewriting terms with such symbols one has to have in mind that they satisfy the axioms of associativity ($x + (y + z) = (x + y) + z$) and commutativity ($x + y = y + x$). These axioms can't be just oriented and added to the system, because any orientation of commutativity yields an infinite reduction. The solution is to consider a new type of rewriting – rewriting modulo AC, rewriting of AC-equivalence classes (see [7]). In this case the ordering used to prove termination should be AC-compatible. An ordering is *AC-compatible* if the result of comparing of two AC-equivalence classes doesn't depend on representatives, i.e., $s' =_{AC} s > t =_{AC} t'$ implies $s' > t'$. The most intuitive method of making an order AC-compatible is *flattening* of terms (removing nested AC-symbols) before comparing them. Unfortunately, because of flattening monotonicity is lost. For example consider RPO with $f > g$, $f \in \mathcal{F}_{AC}$. We have $f(a,a) >_{rpo} g(a)$, but after flattening the terms $f(a, f(a,a))$, $f(a, g(a))$ we obtain $f(a,a,a) \not>_{rpo} f(a, g(a))$.

Recently Rubio and Nieuwenhuis showed a new method of making RPO AC-compatible [10]. Their method consists of computing interpretations of terms before comparing them in RPO. An interpretation of the term is a set of normal forms in the system containing flattening and the rules $f(u_1, \ldots, g(v_1, \ldots, v_n) \ldots, u_m) \longrightarrow f(u_1, \ldots, v_i \ldots, u_m)$ for $i = 1 \ldots n$, f an AC-symbol and any g smaller than f in the precedence.

The aim of this paper is to bring together two types of rewriting described above and construct an ordering which can be used to prove termination of higher-order rewrite systems modulo AC. The main idea is to apply the method of Rubio and Nieuwenhuis to λ-RPO. It is worth pointing out that our construction does not impose any restriction not already present in the definition of λ-RPO or Rubio and Nieuwenhuis's method. The obtained ordering is defined in a simple way by means of rewrite rules, and can be easily implemented since its main component is λ-RPO. Moreover, if there are no AC-symbols in the signature, AC-λ-RPO is just λ-RPO.

The paper is structured in the following way. Section 2 presents some preliminaries and definitions. In Sect.3 the ordering AC-λ-RPO is defined. Section 4 is devoted to the study of the properties of AC-λ-RPO which allow us to prove the main result. We present examples in Sect.5. The full version of this paper can be found at `http:\\zls.mimuw.edu.pl\~daria\papers`.

2 Definitions

We expect the reader to be familiar with the basic concepts and notations of term rewriting systems [3] and typed lambda calculi [1], [2].

2.1 Types and Terms

Let $\mathcal{S}$ denote the set of basic types and let $\mathcal{T}_{\mathcal{S}}$ denote the set of all types obtained from $\mathcal{S}$ using the type constructor $\rightarrow$.

An expression of the form $\sigma_1 \times \ldots \times \sigma_n \to \sigma$ is called the *type declaration*. $\sigma_1, \ldots, \sigma_n$ are input types, σ is the *output type*. We assume that the output type is always a basic type. Note that type declarations are not types.

A *signature* $\mathcal{F}$ is a set of function symbols subject to some restrictions. First, we assume that every function symbol $f \in \mathcal{F}$ comes with its type declaration. Next, we assume that there are only finitely many symbols of each output type and that for every type σ there is a symbol $\perp_\sigma : \sigma$ in $\mathcal{F}$. Finally there is a subset $\mathcal{F}_{\mathcal{AC}} \subseteq \mathcal{F}$ of AC-symbols. Each $f \in \mathcal{F}_{\mathcal{AC}}$ must have type declaration of the form $\sigma \times \sigma \to \sigma$ and satisfy the axioms of commutativity and associativity:

$$\begin{aligned} f(a,b) =& f(b,a) \\ f(a,f(b,c)) =& f(f(a,b),c) \end{aligned}$$

In our reasonings we will often allow $f \in \mathcal{F}_{\mathcal{AC}}$ to take an arbitrary, but not smaller than two, number of arguments. For example if we have $+ : int \times int \to int$ then we can have a term $+(a,b,c)$ and intuitively it is equivalent (among others) to $+(a,+(b,c))$.

We will denote by $\lambda\mathcal{F}$ the set $\mathcal{F}$ extended with symbols $\lambda_{\sigma\to\tau} : \tau \to (\sigma \to \tau)$ for each $\sigma, \tau \in \mathcal{T}_\mathcal{S}$.

The set of *terms with types* $\mathcal{T}$ is generated from a denumerable set of variables $\mathcal{X} = \bigcup_{\sigma \in \mathcal{T}_\mathcal{S}} \mathcal{X}_\sigma$, according to the grammar:

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

This set is then restricted to *simply typed λ-terms* $\mathcal{T}(\mathcal{F},\mathcal{X})$ by the following typing rules:

$$\textbf{(var)}\frac{}{x^\sigma : \sigma} \qquad \textbf{(abs)}\frac{u : \tau}{\lambda x^\sigma.u : \sigma \to \tau} \qquad \textbf{(app)}\frac{u : \sigma \to \tau \qquad v : \sigma}{u(v) : \tau}$$

$$\textbf{(fun)}\frac{u_1 : \sigma_1, \ldots, u_n : \sigma_n}{f(u_1, \ldots, u_n) : \sigma} \quad \text{for } f : \sigma_1 \times \ldots \times \sigma_n \to \sigma$$

Terms are identified with finite labelled trees by considering $\lambda x.$ as an unary function symbol. *Positions* of a term are strings of positive integers. Λ denotes the empty string (root position), $\cdot$ denotes concatenation. The subterm of s at position p is denoted by $s|_p$, while $s[t]_p$ stands for the result of replacing $s|_p$ by t. The expressions $\langle s_1, \ldots s_n\rangle$ and $\boldsymbol{s}$ stand for ordered tuple of terms with and without specifying the elements of the tuple.

For convenience we assume that bound variables are all different, and are different from free ones.

Substitutions are written as in $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, where t_i is assumed to be different from x_i. The substitution is called ground if all t_i are ground.

From now on, we will work only with terms in η-long β-normal form. This form is computed using the rules:

$$(\lambda x.v)(u) \longrightarrow_\beta u\{x \mapsto v\}$$

$$C[u]_p \longrightarrow_\eta C[\lambda x.u(x)]_p \;\text{ if } \begin{cases} u : \sigma \to \tau \\ x \notin Var(u) \\ u \text{ is not an abstraction} \\ C[u]|_q \text{ is not an application for } p = q \cdot 1 \end{cases}$$

The simply typed λ-calculus is confluent and terminating with respect to β-reductions and η-expansions. We write $u{\downarrow}$ for the unique normal form of u with respect to these rules. If $t = t{\downarrow}$ then t is called *normalised* and it is of the form $\lambda x_1 \dots x_m.\, f(t_1, \dots, t_n)$, where $m \geq 0$, $f \in \mathcal{F} \cup \mathcal{X}$, terms $t_1, \dots t_n$ are normalised and $f(t_1, \dots, t_n)$ is of basic type. In the sequel, $\mathcal{T}(\mathcal{F}, \mathcal{X}){\downarrow}$ stands for the set of normalised simply typed λ-terms.

2.2 Higher Order Rewriting modulo AC

A *higher-order term rewriting system* is a set of rewrite rules $R = \{l_i \to r_i\}_{i \in I}$, where for every $i \in I$, l_i, r_i are normalised terms of the same type. Given a system R, a term s rewrites modulo AC to a term t at a position p, with a rule $l \to r$ and a substitution γ if there is a term u such that $s =_{AC} u$, $u|_p = l\gamma{\downarrow}$ and $u[r\gamma{\downarrow}]_p =_{AC} t$. We denote this fact by $s \xrightarrow[l \to r/AC]{p} t$ or just $s \longrightarrow_R t$. A term s is in R-normal form if it can't be rewritten by R. We denote by $\longrightarrow_R^*$ the reflexive, transitive closure of $\longrightarrow_R$.

A rewrite system is said to be *confluent* if for any three terms s, u, t, $u \longrightarrow_R^* s$ and $u \longrightarrow_R^* t$, imply $s \longrightarrow_R^* v$ and $t \longrightarrow_R^* v$ for some v. A definition of *local confluence* is obtained when we replace $u \longrightarrow_R^* s$ and $u \longrightarrow_R^* t$ by $u \longrightarrow_R s$ and $u \longrightarrow_R t$.

A rewrite system is *terminating* (or *strongly normalising*) if there is no infinite rewriting sequence $s_1 \longrightarrow_R s_2 \longrightarrow_R \dots \longrightarrow_R s_n \longrightarrow_R \dots$.

2.3 Orderings

Throughout the paper we will make intensive use of orderings on λ-terms in η-long β-normal form.

A quasi-ordering $\geq$ is said to be *well-founded* if there is no infinite sequence $s_1 > s_2 \dots > s_i > \dots$, where $>$ is the strict part of $\geq$.

A quasi-ordering is *stable under ground contexts* if for all $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X}){\downarrow}$, $u \in \mathcal{T}(\mathcal{F}){\downarrow}$ and positions p, $s > t$ implies $u[s]_p > u[t]_p$. A quasi-ordering is *stable under ground substitutions* if for all $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X}){\downarrow}$ and for all ground substitutions γ, $s > t$ implies $s\gamma{\downarrow} > t\gamma{\downarrow}$.

A *higher order reduction ordering (reduction ordering on higher-order terms)* is a well-founded ordering stable under ground contexts and substitutions. Such an ordering is of principal significance in proving termination of rewrite system.

We can consider lexicographic ($\langle s_1, \dots, s_n \rangle \geq^{lex} \langle t_1, \dots, t_n \rangle$ iff $s_1 \geq t_1, \dots s_{i-1} \geq t_{i-1}, s_i > t_i$) and multiset (transitive closure of $\{s_1, \dots, s_i, \dots, s_n\} \gg \{s_1, \dots, t_1, \dots, t_m, \dots s_n\}$ for $s > t_i$ for all $i \in [1 \dots m]$) extensions of a reduction ordering. These are reduction orderings.

A quasi-ordering is *total* if any two ground terms are comparable.

Dealing with rewriting modulo AC we have to assure AC-compatibility of orderings in use. This means that whenever $s' =_{AC} s > t =_{AC} t'$ then $s' > t'$.

Now we are going to define several useful notions.

Definition 1. *A* top symbol *of a term is defined by* $T(f(s_1, \dots, s_n)) = f$ *for* $f \in \mathcal{F} \cup \mathcal{X}$ *and* $T(\lambda x^\sigma.s) = \lambda_{\sigma\to\tau}$ *if* $\lambda x^\sigma.s : \sigma \to \tau$.

A term v *is an* immediate subterm *of a term* u *if* $v \in st_\lambda(u)$, *where*

$$st_\lambda(u) = \begin{cases} \langle u_1, \dots, u_n\rangle \text{ if } u = f(u_1, \dots, u_n),\ f \in \mathcal{F} \cup \mathcal{X} \\ v\{x \mapsto \bot_\sigma\} \ \text{ if } u = \lambda x^\sigma.v,\ u : \sigma \to \tau \end{cases}$$

The subterm relation, *denoted by* $\trianglerighteq$, *is the transitive closure of the immediate subterm relation. In order not to confuse these subterms with standard ones we will write* $s|_p^\lambda$ *for the subterm of* s *at the position* p *according to* $\trianglerighteq$.

We are now in a position to recall the definition of λ-RPO ordering from [6]. λ-RPO is based on:

- precedence on types : quasi-ordering $\geq_{\mathcal{T}_S}$ on types, well-founded and compatible with term structure (i.e., for all $s : \sigma, t : \tau \in \mathcal{T}(\mathcal{F}, \mathcal{X})\!\downarrow$, such that $s \trianglerighteq t$ we have $\sigma \geq_{\mathcal{T}_S} \tau$; for details and examples see [6]).
- function *status* associating symbol *Lex* (for lexicographic) or symbol Mul (for multiset) to every function symbol and such that $status(f) = Mul$ if and only if $f \in \mathcal{F}_{\mathcal{AC}}$ (the "only if" assumption is made for notational simplicity).
- precedence on function symbols : well-founded quasi-ordering $\geq_{\mathcal{F}}$ on function symbols from $\lambda\mathcal{F}$ of the same (modulo $=_{\mathcal{T}_S}$) output type. If $\alpha =_{\mathcal{T}_S} \beta$ we put $\bot_\alpha =_{\mathcal{F}} \bot_\beta$ and $\lambda_\alpha =_{\mathcal{F}} \lambda_\beta$. $\bot_\alpha$ is always minimal, position of λ_α is arbitrary. $\geq_{\mathcal{F}}$ may equalise the symbols of the same status and arity only. Variables are not comparable with any other symbol.

For $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})\!\downarrow$ we let $s : \sigma \geq_{\lambda rpo} t : \tau$ iff

1. $\sigma >_{\mathcal{T}_S} \tau$, or
2. $\sigma =_{\mathcal{T}_S} \tau$, and
   (a) $T(s) \notin \mathcal{X}$ and $s_i \geq_{\lambda rpo} t$ for some $s_i \in st_\lambda(s)$, or
   (b) $T(s) >_{\mathcal{F}} T(t)$ and $s >_{\lambda rpo} t_i$ for all $t_i \in st_\lambda(s)$, or
   (c) $T(s) = T(t) \in \mathcal{F}_{\mathcal{AC}}$ and $st_\lambda(s) \geq^{mul}_{\lambda rpo} st_\lambda(t)$, or
   (d) $T(s) = T(t) \notin \mathcal{F}_{\mathcal{AC}}$ and $st_\lambda(s) \geq^{lex}_{\lambda rpo} st_\lambda(t)$ and $s >_{\lambda rpo} t_i$ for all $t_i \in st_\lambda(s)$, or
   (e) $T(s) = T(t) \in \mathcal{X}$ and $st_\lambda(s) =^{lex}_{Mul} st_\lambda(t)$,

where $s >_{\lambda rpo} t$ iff $s \geq_{\lambda rpo} t$ and $t \not\geq_{\lambda rpo} s$; $\geq^{mul}_{\lambda rpo}$, $\geq^{lex}_{\lambda rpo}$ are multiset and lexicographic extensions of $\geq_{\lambda rpo}$; and $=_{Mul}$ means that both sides are equal modulo permutations of equivalent arguments below AC-symbol.

λ-RPO is a higher-order reduction ordering. It is total if $\geq_{\mathcal{F}}$, $\geq_{\mathcal{T}_S}$ are total. It will serve as a basis in our construction of AC-compatible reduction ordering on λ-terms.

Finally we define a notion of embedding and well-order on type.

Definition 2 (Embedding). *We write* $\langle s_1, \dots, s_n \rangle \hookrightarrow_e \langle t_1, \dots t_m \rangle$ *if there are indexes* $1 \leq j_1 < j_2 < \dots < j_m \leq n$ *such that* $s_{j_i} \,{}_e\!\!\gtrsim t_i$ *for* $i = 1 \dots m$. *A term* t *is* embedded *in* s *(*$s \,{}_e\!\!\gtrsim t$*) iff:*

1. $s_i \,{}_e\!\!\gtrsim t$, *for some* $s_i \in st_\lambda(s)$, *or*
2. $T(s) =_{\mathcal{F}} T(t)$ *and* $st_\lambda(s) \hookrightarrow_e st_\lambda(t)$.

A quasi-ordering $\geq$ is a *well-order on type* if for every infinite sequence $(t_i)_{i \in \mathbb{N}}$ of terms of the same type there are indexes $k > l$ such that $t_k \geq t_l$. We leave without proof that $\mathcal{T}(\mathcal{F})\!\downarrow^n$ and $\mathcal{T}(\mathcal{F})\!\downarrow^*$ (tuples and words over $\mathcal{T}(\mathcal{F})\!\downarrow$) are well-ordered on type by $\hookrightarrow_e$.

3 The ordering

As it was mentioned in the introduction, comparing terms in AC-λ-RPO consist of two phases. First we have to calculate the interpretations of the terms. These are sets of normal forms in the auxiliary system — system R. Then we compare the interpretations using $\geq_{\lambda rpo}$.

3.1 System R

Following the notation from [10] we distinguish some symbols from $\lambda\mathcal{F}$ and call them *interpreted*, $\mathcal{F}_{\mathcal{I}}$. We assume that $\mathcal{F}_{\mathcal{AC}} \subseteq \mathcal{F}_{\mathcal{I}}$ and that no λ_α belongs to it. The possibility of having in $\mathcal{F}_{\mathcal{I}}$ some symbols not from $\mathcal{F}_{\mathcal{AC}}$ enriches the obtained family of orderings.

System R comes in two parts: rules for flattening and rules for interpreted symbols. As the latter make the system R non confluent it is necessary to work with sets of normal forms. We will write $snf_R(u)$ for the set of normal forms of u with respect to R and $mnf(u)$ for its maximal, in λ-RPO, normal form, which is unique if $u \in \mathcal{T}(\mathcal{F})\!\downarrow$ and both precedences are total.

Let us introduce two useful notations. Suppose that $f, g \in \lambda\mathcal{F}$ have output types σ and τ respectively. We write $f >_{\mathcal{T}_\mathcal{S}, \mathcal{F}} g$ iff $\sigma >_{\mathcal{T}_\mathcal{S}} \tau$ or $\sigma =_{\mathcal{T}_\mathcal{S}} \tau$ and $f >_{\mathcal{F}} g$.

A set of *Subterms of the Same Type* is defined by:

$$SST(f(s_1, \dots s_n)) = \begin{cases} \{s_i \mid \alpha_i =_{\mathcal{T}_\mathcal{S}} \alpha\} & \text{if } f : \alpha_1 \times \dots \times \alpha_n \to \alpha \\ & \text{and there is an } \alpha_i =_{\mathcal{T}_\mathcal{S}} \alpha \\ \{\bot_\alpha\} & \text{if no such } \alpha_i \text{ exists} \end{cases}$$

$$SST(\lambda x^\sigma . s) \qquad = \begin{cases} \{s\{x \mapsto \bot_\sigma\}\} & \text{if } \tau =_{\mathcal{T}_\mathcal{S}} \sigma \to \tau \\ \{\bot_{\sigma \to \tau}\} & \text{otherwise} \end{cases}$$

System R:

(flat)

$$f(x_1, \dots x_m, g(y_1, \dots y_r), z_1, \dots z_n) \longrightarrow_R f(x_1, \dots x_m, y_1, \dots y_r, z_1, \dots z_n)$$

for $f \in \mathcal{F}_{\mathcal{AC}}$ and $g =_{\mathcal{F}} f$,

(inter)

- $f(x_1, \ldots, x_m, g(y_1, \ldots, y_r), z_1, \ldots, z_n) \longrightarrow_R f(x_1, \ldots, x_m, a, z_1, \ldots, z_n)$ for $f \in \mathcal{F}_\mathcal{I}$, $f >_{\mathcal{T}_\mathcal{S}, \mathcal{F}} g$ and $a \in SST(g(y_1, \ldots, y_r))$,
- $f(x_1, \ldots, x_m, \lambda y^\sigma.Yy, z_1, \ldots, z_n) \longrightarrow_R f(x_1, \ldots, x_m, a, z_1, \ldots, z_n)$ for $f \in \mathcal{F}_\mathcal{I}$, $f >_{\mathcal{T}_\mathcal{S}, \mathcal{F}} \lambda_{\sigma \to \tau}$ and $a \in SST(\lambda y^\sigma.Yy)$.

The fact that we use the same constant $\perp$ both, in the subterm definition (cf. Definition 1) and in the definition of the system R is not essential to our work. All we need in the definition of the subterm is to prevent a variable that was bound from being substituted. For this purpose we could use any other constant instead of $\perp$. The other solution would be to syntactically separate free and bound variables and introduce loose bound variables (as it is done in [9]). In the definition of system R we really need $\perp$ as it is the smallest (in $\geq_\mathcal{F}$) element of the given output type (see the precedence on function symbols in the definition of λ-RPO).

Lemma 1. *The relation $\longrightarrow_R$ is included in $\geq_{\lambda rpo}$, hence terminating.*

Even though $\longrightarrow_R$ is not confluent we still have confluence for computing snf_R.

Lemma 2 (Confluence for computing snf_R).

$$snf_R(t[l\gamma]) = snf_R(t[r_1\gamma]) \cup \ldots \cup snf_R(t[r_n\gamma])$$

where γ is a substitution and $\{r_1, \ldots, r_n\}$ is the set of right-hand sides of all rules in R with left-hand side equal to l.

Proof. We begin by proving strong normalisation and confluence of the following rewriting of sets:

$$S \cup \{t[l\gamma]\} \longrightarrow_{SR} S \cup \{t[r_1\gamma], \ldots, t[r_n\gamma]\}$$

where γ, $r_1, \ldots, r_n$ are as above.

Strong normalisation of $\longrightarrow_{SR}$ is a consequence of the strong normalisation of $\geq_{\lambda rpo}$. Since by Newmann's lemma local confluence for terminating relation implies confluence, it remains to analyse all possible overlappings of redexes.

Let S, T, U be sets of terms satisfying:

$$T \longleftarrow_{SR} S \longrightarrow_{SR} U$$

We will show that there is a set W such that:

$$T \longrightarrow^*_{SR} W \longleftarrow^*_{SR} U$$

There are two possibilities: either $S = S' \cup \{t[l\gamma]\} \cup \{t'[g\delta]\}$ where $t \neq t'$ and

$$\begin{aligned} T =& S' \cup \{t[r_1\gamma]\} \cup \ldots \cup \{t[r_n\gamma]\} \cup \{t'[g\delta]\} \\ U =& S' \cup \{t[l\gamma]\} \cup \{t'[d_1\delta]\} \cup \ldots \cup \{t'[g_m\delta\} \\ W =& S' \cup \{t[r_1\gamma]\} \cup \ldots \cup \{t[r_n\gamma]\} \cup \{t'[d_1\delta]\} \cup \ldots \cup \{t'[g_m\delta\} \end{aligned}$$

and local confluence is obvious, or $S = S' \cup \{t\}$, where $t = t[l\gamma]_p = t[g\delta]_q$. In the latter case we have the following possibilities:

- disjoint redexes (neither p is a prefix of q, nor q is a prefix of p). Hence $t = t[l\gamma]_p[g\delta]_q$ and we set:

  $$W = S' \cup \{t[r_i\gamma]_p[d_j\delta]_q \mid i = 1 \ldots n, j = 1 \ldots m\}$$

  It is easily seen that $T \longrightarrow^*_{SR} W \longleftarrow^*_{SR} U$.
- variable overlap. This means that $q = pq'q''$ and $l|_{q'} = x$ is a variable. Hence $t = t[l\gamma]_p$, where $x\gamma|_{q''} = g\delta$. Since all the rules in R are linear we have:

  $$\begin{aligned} T =& S' \cup \{t[r_1\gamma]_p, \ldots, t[r_n\gamma]_p\} \\ U =& S' \cup \{t[l\gamma_1]_p, \ldots, t[l\gamma_m]_p\} \end{aligned}$$

  where $z\gamma_i = z\gamma[d_i\delta]$ if $z = x$ and $z\gamma_i = z\gamma$ otherwise. We set:

  $$W = S' \cup \{t[r_i\gamma_j]p \mid i = 1 \ldots n, j = 1 \ldots m\}$$

  It is easy to check that T, U rewrite to W.
- critical overlap. This means that $q = pq'$ and $l|_{q'}$ is not a variable. In this case we have to consider all "critical pairs". There are five such pairs, but as all five can be handled in much the same way, we will analyse only one of them. Let t be of the form:

  $$t = t[f(\boldsymbol{x'}, g(\boldsymbol{y'}, \lambda z^\sigma.Zz, \boldsymbol{y''}), \boldsymbol{x''})\gamma]_p$$

  where $|q| = |p| + 1$, $f, g \in \mathcal{F}_\mathcal{I}$, $f >_{\mathcal{T}_\mathcal{S},\mathcal{F}} g >_{\mathcal{T}_\mathcal{S},\mathcal{F}} \lambda_{\sigma\to\tau}$, $f : \alpha_1 \times \ldots \times \beta \times \ldots \alpha_n \to \alpha$, $g : \beta_1 \times \ldots \times (\sigma \to \tau) \times \ldots \beta_m \to \beta$, $\lambda z^\sigma.Zz : \sigma \to \tau$. The set $S = S' \cup \{t\}$ rewrites either by the first rule of **(inter)** to the set:

  $$T = S' \cup \{t[f(\boldsymbol{x'}, a, \boldsymbol{x''})\gamma]_p \mid a \in SST(g(\boldsymbol{y'}, \lambda z^\sigma.Zz, \boldsymbol{y''}))\}$$

  or by the second rule of **(inter)** to the set:

  $$U = S' \cup \{t[f(\boldsymbol{x'}, g(\boldsymbol{y'}, b, \boldsymbol{y''}), \boldsymbol{x''})\gamma]_p \mid b \in SST(\lambda z^\sigma.Zz)\}$$

  But these two sets rewrite to:

  $$W = S' \cup \{t[f(\boldsymbol{x'}, c, \boldsymbol{x''})\gamma]_p\}, \text{ where}$$

  $c \in C = \{y_i \mid \beta_i =_{\mathcal{T}_\mathcal{S}} \beta\} \cup \{Z\perp_\sigma \mid \tau =_{\mathcal{T}_\mathcal{S}} \sigma \to \tau =_{\mathcal{T}_\mathcal{S}} \beta\}$ or $c = \perp_\sigma$ (if the set C is empty), if one apply the first rule of **(inter)** to U and the second rule of **(inter)** to T.

The proof is completed by showing that given term t, term s belongs to $snf_R(t)$ if and only if s belongs to SR-normal form of $\{t\}$. Both implications can be proved by a routine induction. $\square$

By the above we obtain methods to compute snf_R and hence mnf:

Lemma 3.

1. $mnf(f(t_1,\dots,t_n)) = mnf(f(mnf(t_1),\dots mnf(t_n)))$,
2. $mnf(f(\dots t \dots)) \in \{mnf(f(\dots a \dots))\}$ *where* $a \in SST(t)$, *if* $f \in \mathcal{F}_{\mathcal{I}}$ *and* $f >_{\mathcal{T}_S,\mathcal{F}} T(t)$
3. $mnf(\lambda x^\sigma.s) =_{\lambda rpo} \lambda x^\sigma.mnf(s\{x \mapsto \bot_\sigma\})$.

3.2 Definition of AC-λ-RPO

For a given term u let us apply the rules of **(flat)** at the root position as long as possible. Call the result v. Then $st_\lambda(v)$ is the *top-flattening* of u (denoted by $tf(u)$).

Suppose we are given a λ-RPO with total precedences on types and function symbols. We define our AC-λ-RPO ordering ($\succ$) on $\mathcal{T}(\mathcal{F},\mathcal{X})\!\downarrow$ by $s \succ t$ iff

1. $\forall t' \in snf_R(t)\ \exists s' \in snf_R(s)$, such that $s' >_{\lambda rpo} t'$, or
2. $\forall t' \in snf_R(t)\ \exists s' \in snf_R(s)$, such that $s' \geq_{\lambda rpo} t'$, and
   (a) $T(s) \in F_{AC}$ and $tf(s) \succ^{mul} tf(t)$, or
   (b) $T(s) \notin F_{AC}$ and $tf(s) \succ^{lex} tf(t)$,

where $\succ^{mul}$, $\succ^{lex}$ are lexicographic and multiset AC-extensions of $\succ$ (it is left to the reader to verify that the case 2(b) is correct, i.e. the numbers of arguments of s and t are equal if s and t do not satisfy the case 1 and $T(s) \notin \mathcal{F}_{AC}$).

Note that if $s,t \in \mathcal{T}(\mathcal{F})\!\downarrow$, we can simply replace the first case of the above definition by $mnf(s) >_{\lambda rpo} mnf(t)$ and the second case by $mnf(s) \geq_{\lambda rpo} mnf(t)$.

4 Results

In this Section we show that AC-λ-RPO is an AC-compatible higher-order reduction ordering. To this end we proceed with the study of its properties.

It is easy to see that $\succ$ is irreflexive and transitive. The proof of AC-compatibility is little bit more involved. It is based on the observation that AC-equal terms have identical snf_Rs and AC-equal tfs (up to the permutation of arguments below AC-symbol). The desired conclusion is established by induction on term's size.

4.1 Subterm and deletion properties

Although subterm and deletion properties are not explicitly needed for our main result, it turns out that they will be extremely useful in all consecutive subsections. We begin by recalling that we have slightly modified the definition of subterm (cf. Definition 1).

We say that a quasi-ordering $\geq$ has the *subterm property* if for every $s_i \in st_\lambda(s)$ we have $s > s_i$. An easy verification shows that λ-RPO has the subterm property on $\mathcal{T}(\mathcal{F})\downarrow$. A subterm property of $\succ$ is an easy consequence of the following lemma.

Lemma 4. *For any $s \in \mathcal{T}(\mathcal{F})\downarrow$ and $t \in st_\lambda(s)$ we have $mnf(s) >_{\lambda rpo} mnf(t)$.*

Proof. Let $s : \sigma$ and $t : \tau$. We begin by recalling that by compatibility with term structure $\sigma \geq_{\mathcal{T_S}} \tau$.

If $\sigma >_{\mathcal{T_S}} \tau$ then we have $mnf(s) >_{\lambda rpo} mnf(t)$, because computing of mnf doesn't change the type.

Otherwise $\sigma =_{\mathcal{T_S}} \tau$. We first consider $T(s) = \lambda$. Since $s = \lambda x^\sigma.w$, $t = w\{x \mapsto \bot_\sigma\}$ we see that $mnf(s) >_{\lambda rpo} mnf(t)$ holds if only $mnf(\lambda x^\sigma.w) >_{\lambda rpo} mnf(w\{x \mapsto \bot_\sigma\})$ does. But this follows from case 3 of Lemma 3 and case 2(a) of the definition of $\geq_{\lambda rpo}$.
In case of $T(s) = f \in \mathcal{F}$ let us first prove by induction on term's size that for all $u : \alpha$, $f(\ldots \bot \ldots) : \beta$ in R-normal form we have $mnf(f(\ldots u \ldots)) >_{\lambda rpo} u$.

1. $\beta \geq_{\mathcal{T_S}} \alpha$. Our claim obviously holds since mnf does not change the type.
2. $f \notin \mathcal{F_I}$ or $T(u) >_{\mathcal{T_S},\mathcal{F}} f$. Then $mnf(f(\ldots u \ldots)) = f(\ldots u \ldots)$ and we obtain the result by the subterm property of $\geq_{\lambda rpo}$.
3. $f \in \mathcal{F_I}$, $f =_{\mathcal{F}} T(u)$. If $f \notin \mathcal{F_{AC}}$ the above proof applies, otherwise $mnf(f(\ldots u \ldots)) = f(\ldots u_1, \ldots, u_k \ldots)$ where $u_i \in st_\lambda(u)$. We finish by case 2(c) of the definition of $\geq_{\lambda rpo}$.
4. $f \in \mathcal{F_I}$, $f >_{\mathcal{T_S},\mathcal{F}} T(u)$. Then $mnf(f(\ldots u \ldots)) \in \bigcup_{a \in A} mnf(f(\ldots a \ldots))$, where $a \in SST(u)$. Set $Max = max_{\lambda rpo}\{a |\ a \in SST(u)\}$. Clearly $Max \geq_{\lambda rpo} u_i$ for $u_i \in st_\lambda(u)$. By induction hypothesis we have $mnf(f(\ldots Max \ldots)) >_{\lambda rpo} Max$. We have also $mnf(f(\ldots Max \ldots)) >_{\lambda rpo} u$, because it is equivalent to $mnf(f(\ldots Max \ldots)) >_{\lambda rpo} u_i$ for all $u_i \in st_\lambda(u)$, which follow immediately from transitivity of $\geq_{\lambda rpo}$, $mnf(f(\ldots Max \ldots)) >_{\lambda rpo} Max$ and $Max \geq_{\lambda rpo} u_i$. Since $mnf(f(\ldots u \ldots)) \geq_{\lambda rpo} mnf(f(\ldots Max \ldots))$, we conclude that $mnf(f(\ldots u \ldots)) >_{\lambda rpo} u$.

For the general case, we have $mnf(f(\boldsymbol{u}, t, \boldsymbol{v})) = mnf(f(\overrightarrow{mnf(u)}, mnf(t), \overrightarrow{mnf(v)})$ by case 1 of Lemma 3 and $mnf(f(\overrightarrow{mnf(u)}, mnf(t), \overrightarrow{mnf(v)})) >_{\lambda rpo} mnf(t)$ from what has already been proved. Hence $mnf(f(\boldsymbol{u}, t, \boldsymbol{v})) >_{\lambda rpo} mnf(t)$ □

Lemma 5 (Deletion property). *For each symbol $f \in \mathcal{F_{AC}}$, $f(\boldsymbol{v}, t, \boldsymbol{u}) \succ f(\boldsymbol{v}, \boldsymbol{u})$.*

4.2 Monotonicity

Monotonicity is a basis of the stability under context and well-foundedness of AC-λ-RPO.

The unusual form of this property is due to the definition of subterm and the way λ-RPO treats an abstraction. Since when taking an immediate subterm of $\lambda x^\sigma.s$ we replace x by $\bot_\sigma$, we can reconstruct it while applying a context. The other reason for choosing such formulation is the simplicity of notation (see the definition of $c\langle - \rangle$ below).

Definition 3 (Closure by λ-context). *The* closure by λ-context *of the term s, denoted by $\lambda x^\sigma.\hat{s}$, is defined to be any term $\lambda x^\sigma.w$ such that $w\{x_\sigma \mapsto \bot_\sigma\} = s$. Note that all closures by λ-context of a given term are $=_{\lambda rpo}$ equal.*

Example 1. Set $s = f(\bot_\sigma, h(\bot_\sigma))$. Then $\lambda x^\sigma.f(\bot_\sigma, h(\bot_\sigma))$, $\lambda x^\sigma.f(x^\sigma, h(\bot_\sigma))$, $\lambda x^\sigma.f(\bot_\sigma, h(x^\sigma))$, $\lambda x^\sigma.f(x^\sigma, h(x^\sigma))$ are its closures by λ-context.

Definition 4 (Monotonicity). *Let $s, t \in \mathcal{T}(\mathcal{F})\!\downarrow$ be terms of the same type σ. We call an irreflexive and transitive relation $>$* monotone *if $s > t$ implies $f(\ldots s \ldots) > f(\ldots t \ldots)$ and $\lambda x^\sigma.\hat{s} > \lambda x^\sigma.\hat{t}$ for $f(\ldots s \ldots), f(\ldots t \ldots) \in \mathcal{T}(\mathcal{F})\!\downarrow$ and $\lambda x^\sigma.\hat{s}, \lambda x^\sigma.\hat{t}$ defined as in Definition 3.*

We write $c\langle u \rangle$ for the closure of u by either $\mathcal{F}$- or λ-context.

An easy computation shows that $>_{\lambda rpo}$ is monotone. To prove the monotonicity of $\succ$ we need to know when $>_{\lambda rpo}$ between *mnf*s is preserved under addition of a context.

Lemma 6. *Let $s, t \in \mathcal{T}(\mathcal{F})\!\downarrow$ be terms of the same type σ, and let $c\langle - \rangle$ be a ground $\mathcal{F}$- or λ-context. Then:*

$$mnf(s) \geq_{\lambda rpo} mnf(t) \Longrightarrow mnf(c\langle s \rangle) \geq_{\lambda rpo} mnf(c\langle t \rangle)$$
$$mnf(s) >_{\lambda rpo} mnf(t) \Longrightarrow mnf(c\langle s \rangle) >_{\lambda rpo} mnf(c\langle t \rangle) \quad \textit{if} \quad T(s) \geq_{\mathcal{T}_S, \mathcal{F}} T(c)$$

Proof. By induction on $|s| + |t|$. Without loss of generality we can assume that $c\langle - \rangle$ is already in R-normal form. Let us first prove the reduced form of the theorem, when s, t are R-normalised, i.e

$$s \geq_{\lambda rpo} t \Longrightarrow mnf(c\langle s \rangle) \geq_{\lambda rpo} mnf(c\langle t \rangle)$$
$$s >_{\lambda rpo} t \Longrightarrow mnf(c\langle s \rangle) >_{\lambda rpo} mnf(c\langle t \rangle) \;\text{ if }\; T(s) \geq_{\mathcal{T}_S, \mathcal{F}} T(c)$$

1. $T(c) \notin \mathcal{F}_\mathcal{I}$ or $T(s) >_{\mathcal{T}_S, \mathcal{F}} T(c)$.
   Note first that $mnf(c\langle s \rangle) = c\langle s \rangle$. By monotonicity of $>_{\lambda rpo}$ we have $c\langle s \rangle \geq_{\lambda rpo} c\langle t \rangle$ and since $\rightarrow_R \subseteq \geq_{\lambda rpo}$ we deduce $c\langle t \rangle \geq_{\lambda rpo} mnf(c\langle t \rangle)$. Hence $mnf(c\langle s \rangle)$ $\geq_{\lambda rpo} mnf(c\langle t \rangle)$.
   Similar arguments apply to the case where the initial inequality is strict ($s >_{\lambda rpo} t$). In the sequel $T(c) \neq \lambda$ (since $\lambda \notin \mathcal{F}_\mathcal{I}$). Set $T(c) = f$.

2. $f \in \mathcal{F}_{\mathcal{I}}$ and $f >_{\mathcal{T}_{\mathcal{S}},\mathcal{F}} T(t)$.
   By case 2 of Lemma 3 we have:

$$mnf(f(\dots t \dots)) \in \{mnf(f(\dots a \dots))\} \tag{1}$$

   where $a \in SST(t)$. Combining $s \geq_{\lambda rpo} t$ and $t >_{\lambda rpo} a$ for all $a \in SST(t)$ we get $s >_{\lambda rpo} a$ and by induction hypothesis $mnf(f(\dots s \dots)) \geq_{\lambda rpo} mnf(f(\dots a \dots))$. Since by 1 $mnf(f(\dots t \dots)) = mnf(f(\dots a \dots))$ for some a, the proof is complete.
   If the initial inequality were strict and $T(s) =_{\mathcal{F}} f$ then the same reasoning would yield $mnf(f(\dots s \dots)) >_{\lambda rpo} mnf(f(\dots t \dots))$.
3. $f \in \mathcal{F}_{\mathcal{I}}$ and $T(t) \geq_{\mathcal{T}_{\mathcal{S}},\mathcal{F}} f \geq_{\mathcal{T}_{\mathcal{S}},\mathcal{F}} T(s)$.
   (a) $T(t) \geq_{\mathcal{T}_{\mathcal{S}},\mathcal{F}} f >_{\mathcal{T}_{\mathcal{S}},\mathcal{F}} T(s)$. Then:

$$mnf(f(\dots s \dots)) \in \{mnf(f(\dots a \dots))\} \tag{2}$$

   where $a \in SST(s)$. Since $s \geq_{\lambda rpo} t$ and $T(t) >_{\mathcal{T}_{\mathcal{S}},\mathcal{F}} T(s)$ there must be an $s_j \in st_\lambda(s)$ such that $s_j \geq_{\lambda rpo} t$. Moreover, this s_j belongs to $SST(s)$ (as $s, t : \sigma$). By induction hypothesis we have $mnf(f(\dots s_j \dots)) \geq_{\lambda rpo} mnf(f(\dots t \dots))$ and by 2 we get $mnf(f(\dots s \dots)) \geq_{\lambda rpo} mnf(f(\dots t \dots))$. As $f \geq_{\mathcal{T}_{\mathcal{S}},\mathcal{F}} T(s)$ we need not consider the case of strict inequality.
   (b) $T(t) >_{\mathcal{T}_{\mathcal{S}},\mathcal{F}} f =_{\mathcal{F}} T(s)$. There are two possibilities; if $f \in \mathcal{F}_{\mathcal{AC}}$ then:

$$mnf(f(\dots s \dots)) = f(\dots s_1, \dots, s_n \dots)$$

   As above, $s \geq_{\lambda rpo} t$, $T(t) >_{\mathcal{T}_{\mathcal{S}},\mathcal{F}} T(s)$ and $s, t : \sigma$ imply the existence of an $s_j \in st_\lambda(s)$ such that $mnf(f(\dots s_j \dots)) \geq_{\lambda rpo} mnf(f(\dots t \dots))$. By the deletion property we have $f(\dots s_1, \dots, s_j, \dots, s_n \dots) >_{\lambda rpo} f(\dots s_j \dots)$, while by inclusion $\rightarrow_R \subseteq \geq_{\lambda rpo}$ we get $f(\dots s_j \dots) \geq_{\lambda rpo} mnf(f(\dots s_j \dots))$. Hence $mnf(f(\dots s \dots)) >_{\lambda rpo} mnf(f(\dots t \dots))$. This reasoning also works for the case of strict inequality ($s >_{\lambda rpo} t$).
   If $f \notin \mathcal{F}_{\mathcal{AC}}$, both claims can easily be proved by monotonicity of $>_{\lambda rpo}$.
   (c) $T(t) =_{\mathcal{F}} f =_{\mathcal{F}} T(s)$. Once again we have two possibilities; if $f \in \mathcal{F}_{\mathcal{AC}}$ then:

$$\begin{aligned} mnf(c\langle s \rangle) &= f(\dots s_1, \dots, s_n \dots) \\ mnf(c\langle t \rangle) &= f(\dots t_1, \dots, t_m \dots) \end{aligned}$$

   But $\{s_1 \dots s_n\} \geq^{mul}_{\lambda rpo} \{t_1 \dots t_m\}$ since $s \geq_{\lambda rpo} t$ and $T(s) =_{\mathcal{F}} T(t) \in Mul$. Thus, by case 2(c) of the definition of $\geq_{\lambda rpo}$, we get our claim. Similar arguments apply to the case $s >_{\lambda rpo} t$.
   The case of $f \notin \mathcal{F}_{\mathcal{AC}}$ runs as before.

We now return to the general case, when s, t are not necessarily R-normalised.

If $T(c) = \lambda$ we use the observation that $mnf(\lambda x^\sigma.\hat{u}) =_{\lambda rpo} \lambda x^\sigma.mnf(u)$ (case 3 of Lemma 3) to reduce our claim to $\lambda x^\sigma.mnf(s) \geq_{\lambda rpo} \lambda x^\sigma.mnf(t)$. But this is an easy consequence of the assumption that $mnf(s) \geq_{\lambda rpo} mnf(t)$. Similar arguments apply to the case $mnf(s) >_{\lambda rpo} mnf(t)$.

Otherwise $T(c) = f \in \mathcal{F}$. Since $mnf(s) \geq_{\lambda rpo} mnf(t)$ it follows that for every $t' \in snf_R(t)$ there is an $s' \in snf_R(s)$ such that $s' \geq_{\lambda rpo} t'$. These s', t' are in R-normal form, and so $mnf(c\langle s'\rangle) \geq_{\lambda rpo} mnf(c\langle t'\rangle)$. As $mnf(f(\dots t \dots)) \in \{mnf(f(\dots t' \dots)) \mid t' \in snf_R(t)\}$ we can find a t' which yields $mnf(c\langle s\rangle) \geq_{\lambda rpo} mnf(c\langle s'\rangle) \geq_{\lambda rpo} mnf(c\langle t'\rangle) =_{\lambda rpo} mnf(c\langle t\rangle)$. The same reasoning applies to the case of strict inequality ($mnf(s) >_{\lambda rpo} mnf(t)$ and $T(s) \geq_{\mathcal{T}_\mathcal{S},\mathcal{F}} T(c)$). □

Lemma 7. *$\succ$ is monotone.*

Proof. $s \succ t$ means that $mnf(s) \geq_{\lambda rpo} mnf(t)$. This implies, by the lemma above, that $mnf(c\langle s\rangle) \geq_{\lambda rpo} mnf(c\langle t\rangle)$. If this inequality is strict then $c\langle s\rangle \succ c\langle t\rangle$ obviously holds. So we are left with the case when $mnf(c\langle s\rangle) =_{\lambda rpo} mnf(c\langle t\rangle)$.

1. $T(c) \notin \mathcal{F}_{\mathcal{AC}}$. Since $tf(c\langle s\rangle) = \langle \boldsymbol{u}, s, \boldsymbol{v}\rangle \succ^{lex} \langle \boldsymbol{u}, t, \boldsymbol{v}\rangle = tf(c\langle t\rangle)$ we conclude that $c\langle s\rangle \succ c\langle t\rangle$ by case 2(b) of the definition of $\succ$,
2. $T(c) \in \mathcal{F}_{\mathcal{AC}}$ and $mnf(s) >_{\lambda rpo} mnf(t)$. If $T(s) \geq_{\mathcal{T}_\mathcal{S},\mathcal{F}} T(c)$ then by Lemma 6 we get $mnf(c\langle s\rangle) >_{\lambda rpo} mnf(c\langle t\rangle)$. Otherwise $tf(c\langle s\rangle) = \langle \boldsymbol{u}, s, \boldsymbol{v}\rangle$, while $tf(c\langle t\rangle)$ is equal to either $\langle \boldsymbol{u}, t, \boldsymbol{v}\rangle$ or $\langle \boldsymbol{u}, t_1 \dots t_k, \boldsymbol{v}\rangle$, where t_i are proper subterms of t. In both cases $tf(c\langle s\rangle) \succ^{mul} tf(c\langle t\rangle)$ since $s \succ t$, $t \succ t_i$, $s \succ t_i$. Thus, by the case 2(a) of the definition of $\succ$, we have $c\langle s\rangle \succ c\langle t\rangle$,
3. $T(c) \in \mathcal{F}_{\mathcal{AC}}$ and $mnf(s) =_{\lambda rpo} mnf(t)$. Note that this forces $T(s) =_\mathcal{F} T(t)$. If $T(s) \neq_\mathcal{F} T(c)$ we have $tf(c\langle s\rangle) = \langle \boldsymbol{u}, s, \boldsymbol{v}\rangle \succ^{mul} \langle \boldsymbol{u}, t, \boldsymbol{v}\rangle = tf(c\langle t\rangle)$. Otherwise $tf(c\langle s\rangle) = \langle \boldsymbol{u}, s_1, \dots s_n, \boldsymbol{v}\rangle$ and $tf(c\langle t\rangle) = \langle \boldsymbol{u}, t_1, \dots t_k, \boldsymbol{v}\rangle$. But $\{s_1 \dots s_n\} \succ^{mul} \{t_1 \dots t_k\}$ since $s \succ t$ and $mnf(s) =_{\lambda rpo} mnf(t)$. Hence $tf(c\langle s\rangle) \succ^{mul} tf(c\langle t\rangle)$ and consequently $c\langle s\rangle \succ c\langle t\rangle$. □

Using Lemma 7, by induction on the size of the context we get:

Corollary 8. *$\succ$ is stable under ground contexts.*

4.3 Well-foundedness

To show the well-foundedness of AC-λ-RPO we will formulate an analogue of Kruskal's theorem. This theorem states that the relation of embedding is a well-order on ground terms. In our case we have to replace well-order by well-order on type and ground terms by ground normalised terms, but the main idea remains the same.

Lemma 9. *Suppose that the precedences on types and function symbols are total and well-founded and that $\geq_{\mathcal{T}_\mathcal{S}}$ is compatible with term structure. Then the relation of embedding $\underset{e}{\gtrsim}$ is a well-order on type on $\mathcal{T}(\mathcal{F})\!\downarrow$.*

Proof. This lemma states that for every infinite sequences $(t_i)_i$ of ground normalised terms of the same type, there are indexes $k > l$ such that $t_k \underset{e}{\gtrsim} t_l$. The proof is by contradiction.

Let $(s_i)_i : \sigma$ be the minimal counterexample subject to the ordering $(type, size)_{lex}$. More precisely let $Counter_0$ be a set of counterexamples of minimal type. We will inductively construct $(s_i)_i$. Assume we have $Counter_k$.

- $|s_{k+1}| \leq |n_{k+1}|$ for all sequences $(n_i)_i \in Counter_k$
- $Counter_{k+1} = \{(n_i)_i \in Counter_k \mid n_{k+1} = s_{k+1}\}$

Consider E — a set of immediate subterms of the terms in $(s_i)_i$. By compatibility with term structure, the output type of any term in E is less than or equal to σ. Let us first prove that E is well-ordered on type by $\mathrel{_e\!\!\gtrsim}$.

$$\text{For any infinite sequence } (u_i)_i : \tau \text{ of terms from } E \text{ of the same type there are indexes } k > l \text{ such that } u_k \mathrel{_e\!\!\gtrsim} u_l. \tag{3}$$

If $\tau <_{\mathcal{T}_\mathcal{S}} \sigma$, the claim follows from the assumption that σ is the type of the minimal counterexample.

Otherwise, $\tau =_{\mathcal{T}_\mathcal{S}} \sigma$. Suppose, contrary to our claim, that there are no $k > l$ such that $u_k \mathrel{_e\!\!\gtrsim} u_l$. Let s_a be the term from which u_1 comes from. By rejecting from $(u_i)_i$ all terms that come from $s_1, \ldots, s_{a-1}$ we still obtain infinite counterexample $(u_{i_j})_j$ with u_{i_1} equal to u_1. Consider now the sequence (Δ_i):

$$s_1, \ldots, s_{a-1}, u_{j_1}, u_{j_2}, \ldots$$

It is smaller than $(s_i)_i$ (since s_a is replaced by its subterm) and it is a counterexample, because the supposition that there are $k > l$ such that $\Delta_k \mathrel{_e\!\!\gtrsim} \Delta_l$ leads to a contradiction with the fact that $(s_i)_i$ and $(u_{i_j})_j$ are counterexamples. This completes the proof of 3. From properties of $\mathrel{_e\!\!\gtrsim}$ mentioned in Subsection 2.3 it follows that E^n and E^* (tuples and words over E) are well-ordered on type by $\hookrightarrow_e$.

We can now return to the main proof. As $>_\mathcal{F}$ is well-founded there exist an infinite subsequence of $(s_i)_i$, denoted by $(s_{i_j})_j$, consisting of terms with $= \mathcal{F}$ top symbols. Let g be a representative of these.

If $g \notin \mathcal{F}_{AC}$, we use the fact that $\hookrightarrow_e$ well-orders on type E^n, to find in the sequence of tuples $(st_\lambda(s_{i_j}))_j$ two elements such that $st_\lambda(s_{i_k}) \hookrightarrow_e st_\lambda(s_{i_l})$ for $k > l$. Similar arguments apply to $g \in \mathcal{F}_{AC}$. By case 2 of the definition of embedding we have $s_{i_k} \mathrel{_e\!\!\gtrsim} s_{i_l}$ which contradicts the initial assumption about $(s_i)_i$. □

Lemma 10. *Let $s, t \in \mathcal{T}(\mathcal{F})\downarrow$. If $s \mathrel{_e\!\!\gtrsim} t$, then $s \succ t$ or $s =_{AC} t$.*

Proof. By induction on $|s| + |t|$ considering the way $s \mathrel{_e\!\!\gtrsim} t$ was shown: (cf. Definition 2 in Subsection 2.3)

- $s \mathrel{_e\!\!\gtrsim} t$ by the first part of the definition. Then there is an $s_i \in st_\lambda(s)$ such that $s_i \mathrel{_e\!\!\gtrsim} t$. By inductive hypothesis we have either $s_i \succ t$, or $s_i =_{AC} t$. The latter gives $s \succ t$ since $\succ$ is AC-compatible, the former does so by the subterm property ($s \succ s_i$) and transitivity of $\succ$.
- $s \mathrel{_e\!\!\gtrsim} t$ by the second part. Then for every $t_i \in st_\lambda(t)$ there is an $s_{j_i} \in st_\lambda(s)$ such that $s_{j_i} \mathrel{_e\!\!\gtrsim} t_i$ and $j_k < j_{k+1}$ for all k. Inductive hypothesis yields either $s_{j_i} \succ t_i$, or $s_{j_i} =_{AC} t_i$. Applying Lemma 7 (monotonicity) and Lemma 5 (deletion property — in case of different number of arguments) we obtain our claim. □

Corollary 11. *$\succ$ is well-founded on $\mathcal{T}(\mathcal{F})\!\downarrow$.*

Proof. Suppose that there is an infinite sequence $(u_i : \tau_i)_{i\in\mathbb{N}}$ decreasing in $\succ$. Since $s : \sigma \succ t : \tau$ implies $\sigma \geq_{\mathcal{T}_S} \tau$ the sequence $(\tau_i)_{i\in\mathbb{N}}$ is constant from a certain position, because $>_{\mathcal{T}_S}$ is well-founded. Considering $(u_i)_{i\in\mathbb{N}}$ from this position we can use Lemma 9 to obtain two indexes $k > l$ such that $u_k \,_e\!\gtrsim u_l$. Previous lemma now yields $u_k \succ u_l$ or $u_k =_{AC} u_l$. Hence a contradiction. □

4.4 Stability under ground substitutions

Throughout this Subsection we consider only ground and normalised substitutions. Before we begin note that if $s >_{\lambda rpo} t$ or $s \succ t$ then $T(s) \notin \mathcal{X}$.

Lemma 12. *Let s, t be terms from $\mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $s =_{AC} t$, and γ be a ground substitution. Then $s\gamma\!\downarrow =_{AC} t\gamma\!\downarrow$.*

Proof. The proof will be divided into two steps. Let us first show by induction on $|u_1| + |u_2| + |w_1| + |w_2|$ that

$$u_1 =_{AC} u_2 \;\wedge\; w_1 =_{AC} w_2 \Longrightarrow u_1\{x \mapsto w_1\} =_{AC} u_2\{x \mapsto w_2\} \tag{4}$$

which gives:

$$s =_{AC} t \Longrightarrow s\gamma =_{AC} t\gamma \tag{5}$$

To complete the proof it remains to show that:

$$s =_{AC} t \Longrightarrow s\!\downarrow =_{AC} t\!\downarrow \tag{6}$$

which can be done by induction on $\langle \rightarrow_{\beta\eta}, |s| + |t| \rangle$ using the observation 4. □

Definition 5 (real subterm). *Let $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})\!\downarrow$. We will call t a* real subterm *of s, denoted by $s \blacktriangleright t$, iff $s \trianglerighteq t$ and no symbol on the path from the root of s to that of t belongs to $Var(s)$.*
For ground terms $\trianglerighteq \equiv \blacktriangleright$.

It is easy to check that the following two lemmas hold.

Lemma 13. *Let $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})\!\downarrow$ be terms of the same type. Suppose that $s \geq_{\lambda rpo} t$ and $T(t) \in \mathcal{X}$. Then there is a w such that $s =_{Mul} w$ and $w \blacktriangleright t$.*

Lemma 14. *If $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})\!\downarrow$ are such that $s \blacktriangleright t$ and if γ is a ground substitution than $s\gamma\!\downarrow \trianglerighteq t\gamma\!\downarrow$.*

Lemma 15. *Given $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})\!\downarrow$ in R-normal form suppose that γ is a ground substitution. Then:*

$$s >_{\lambda rpo} t \Longrightarrow mnf(s\gamma\!\downarrow) >_{\lambda rpo} mnf(t\gamma\!\downarrow) \;\; .$$

Proof. By induction on $|s| + |t|$ considering the proof of $s >_{\lambda rpo} t$. Let $s{:}\sigma$, $t{:}\tau$.

1. $\sigma >_{\mathcal{T}_S} \tau$. Since neither $\downarrow$ nor R normalisation changes the type, it follows that $mnf(s\gamma\Downarrow) >_{\lambda rpo} mnf(t\gamma\Downarrow)$. From now on $\sigma =_{\mathcal{T}_S} \tau$ and $T(s) \notin \mathcal{X}$.
2. $T(t) \in \mathcal{X}$. By Lemma 13 we have $s =_{Mul} w \blacktriangleright t$, by Lemma 14 $w\gamma\Downarrow \trianglerighteq t\gamma\Downarrow$, and by Lemma 4 $mnf(w\gamma\Downarrow) >_{\lambda rpo} mnf(t\gamma\Downarrow)$. Obviously $mnf(s\gamma\Downarrow) =_{\lambda rpo} mnf(w\gamma\Downarrow)$. Hence $mnf(s\gamma\Downarrow) >_{\lambda rpo} mnf(t\gamma\Downarrow)$. From now on $T(s), T(t) \notin \mathcal{X}$.
3. $T(t) >_{\mathcal{F}} T(s)$. Then there is an $s_i \in st_\lambda(s)$ such that $s_i \geq_{\lambda rpo} t$. By induction hypothesis we have $mnf(s_i\gamma\Downarrow) \geq_{\lambda rpo} mnf(t\gamma\Downarrow)$, and by Lemma 4 $mnf(s\gamma\Downarrow) >_{\lambda rpo} mnf(s_i\gamma\Downarrow)$.
4. $T(s) >_{\mathcal{F}} T(t)$. Then for every $t_i \in st_\lambda(t)$, $s >_{\lambda rpo} t_i$. By induction hypothesis we have $mnf(s\gamma\Downarrow) >_{\lambda rpo} mnf(t_i\gamma\Downarrow)$.

   If $T(t) = \lambda$ we have also $mnf(s\gamma\Downarrow) >_{\lambda rpo} \lambda x^\sigma.mnf(t_i\gamma\Downarrow)$ what implies $mnf(s\gamma\Downarrow) >_{\lambda rpo} mnf(t\gamma\Downarrow)$ by case 3 of Lemma 3.

   Otherwise $T(t) \in \mathcal{F}$ and $mnf(s\gamma\Downarrow) >_{\lambda rpo} T(t)(t_1\gamma\Downarrow, \ldots, t_m\gamma\Downarrow)$. Since $mnf(t\gamma\Downarrow) = mnf(T(t)(t_1\gamma\Downarrow, \ldots, t_m\gamma\Downarrow))$ and $\rightarrow_R \subseteq \geq_{\lambda rpo}$ we conclude that $mnf(s\gamma\Downarrow) >_{\lambda rpo} T(t)(t_1\gamma\Downarrow, \ldots, t_m\gamma\Downarrow) \geq_{\lambda rpo} mnf(T(t)(t_1\gamma\Downarrow, \ldots, t_m\gamma\Downarrow)) = mnf(t\gamma\Downarrow)$.
5. $T(t) =_{\mathcal{F}} T(s)$.
   (a) $T(s) \notin \mathcal{F}_{\mathcal{AC}}$. Then $s >_{\lambda rpo} t_k$ for all k and there is an i such that $s_i >_{\lambda rpo} t_i$ and $s_j =_{\lambda rpo} t_j$ for $j = 0 \ldots (i-1)$. By induction hypothesis we have $mnf(s\gamma\Downarrow) >_{\lambda rpo} mnf(t_k\gamma\Downarrow)$, $mnf(s_i\gamma\Downarrow) >_{\lambda rpo} mnf(t_i\gamma\Downarrow)$, $mnf(s_j\gamma\Downarrow) =_{\lambda rpo} mnf(t_j\gamma\Downarrow)$. If $T(t) = \lambda$ the reasoning as above suffices. Otherwise:

   $$mnf(s\gamma\Downarrow) = T(s)(s'_1, \ldots s'_n) \text{ where } T(s)(\bot, mnf(s_k\gamma\Downarrow)) = T(s)(\bot, s'_k)$$
   $$mnf(t\gamma\Downarrow) = T(t)(t'_1, \ldots t'_n) \text{ where } T(t)(\bot, mnf(t_k\gamma\Downarrow)) = T(t)(\bot, t'_k)$$

   It is easily seen that for $j = 0 \ldots (i-1)$ $s'_j =_{\lambda rpo} t'_j$ and that for all k we have $mnf(s_k\gamma\Downarrow) \geq_{\lambda rpo} s'_k$, $mnf(t_k\gamma\Downarrow) \geq_{\lambda rpo} t'_k$. But $mnf(s_i\gamma\Downarrow) = s'_i$ since s_i is not a variable (as $s_i >_{\lambda rpo} t_i$). Hence $s'_i = mnf(s_i\gamma\Downarrow) >_{\lambda rpo} mnf(t_i\gamma\Downarrow) \geq_{\lambda rpo} t'_i$ and for all k, $mnf(s\gamma\Downarrow) >_{\lambda rpo} mnf(t_k\gamma\Downarrow) \geq_{\lambda rpo} t_k$ which gives $mnf(s\gamma\Downarrow) >_{\lambda rpo} mnf(t\gamma\Downarrow)$.

   (b) $T(s) \in \mathcal{F}_{\mathcal{AC}}$. Then $\{s_1 \ldots s_n\} \geq^{mul}_{\lambda rpo} \{t_1 \ldots t_m\}$ and after removing pairs of AC-equal terms, for every t_i there is an s_j such that $s_j >_{\lambda rpo} t_i$, $s_j \neq \bot$, $T(s_j) \notin \mathcal{X}$ and $T(s_j) >_{\mathcal{F}} T(s)$ (as s was originally R-normalised). There is no loss of generality in assuming that these s_j have indexes from 1 to J. Thus

   $$\begin{aligned} mnf(s\gamma\Downarrow) &= mnf(T(s)(mnf(s_1\gamma\Downarrow), \ldots, mnf(s_n\gamma\Downarrow)) \\ &= T(s)(mnf(s_1\gamma\Downarrow), \ldots, mnf(s_J\gamma\Downarrow), \boldsymbol{u_{(J+1)}}, \ldots \boldsymbol{u_n}) \\ mnf(t\gamma\Downarrow) &= T(t)(\boldsymbol{v_1}, \ldots, \boldsymbol{v_m}) \end{aligned}$$

   where $\boldsymbol{u_i}$ come from $mnf(s_i\gamma\Downarrow)$, $\boldsymbol{v_i}$ from $mnf(t_i\gamma\Downarrow)$. More precisely

   $$\begin{aligned} T(s)(\bot, mnf(s_k\gamma\Downarrow)) =& T(s)(\bot, \boldsymbol{u_k}) \\ T(t)(\bot, mnf(t_k\gamma\Downarrow)) =& T(t)(\bot, \boldsymbol{v_k}) \end{aligned}$$

and $\{mnf(t_k\gamma\downarrow)\} \geq^{mul}_{\lambda rpo} \{\boldsymbol{v_k}\}$. Induction hypothesis applied to $s_j >_{\lambda rpo} t_i$ yields $mnf(s_j\gamma\downarrow) >_{\lambda rpo} mnf(t_i\gamma\downarrow)$. So when we compare $\{mnf(s_1\gamma\downarrow), \ldots, mnf(s_J\gamma\downarrow), \boldsymbol{u_{(J+1)}}, \ldots \boldsymbol{u_n}\}$ with $\{\boldsymbol{v_1}, \ldots, \boldsymbol{v_m}\}$ terms which come from AC-equal terms remain AC-equal, while for the others we have $\{mnf(s_j\gamma\downarrow)\} >^{mul}_{\lambda rpo} \{mnf(t_i\gamma\downarrow)\} \geq^{mul}_{\lambda rpo} \{\boldsymbol{v_i}\}$.

□

Lemma 16. *Given $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})\downarrow$ and a ground substitution γ, suppose that $s \succ t$. Then $s\gamma\downarrow \succ t\gamma\downarrow$.*

Proof. By induction on $|s| + |t|$ considering the proof of $s \succ t$.

1. $\forall t' \in snf_R(t)\ \exists s' \in snf_R(s)$ such that $s' >_{\lambda rpo} t'$. By Lemma 15 we have $mnf(s'\gamma\downarrow) >_{\lambda rpo} mnf(t'\gamma\downarrow)$, hence $mnf(s\gamma\downarrow) >_{\lambda rpo} mnf(t\gamma\downarrow)$ and $s\gamma\downarrow \succ t\gamma\downarrow$.
2. $\forall t' \in snf_R(t)\ \exists s' \in snf_R(s)$ such that $s' \geq_{\lambda rpo} t'$. Therefore $T(s) =_{\mathcal{F}} T(t)$ and $T(s) \notin \mathcal{X}$. By Lemma 15 we have $mnf(s'\gamma\downarrow) \geq_{\lambda rpo} mnf(t'\gamma\downarrow)$. Let $tf(s) = \langle s_1, \ldots, s_n \rangle$, $tf(t) = \langle t_1, \ldots, t_m \rangle$.
   (a) $T(s) \notin \mathcal{F}_{AC}$. Then $n = m$ and there is a k such that $s_k \succ t_k$ and $s_j =_{AC} t_j$ for $j = 1 \ldots (k-1)$. By induction hypothesis and by Lemma 12 we have $s_k\gamma\downarrow \succ t_k\gamma\downarrow$ and $s_j\gamma\downarrow =_{AC} t_j\gamma\downarrow$ for $j = 1 \ldots (k-1)$. This implies $s\gamma\downarrow \succ t\gamma\downarrow$.
   (b) $T(s) \in \mathcal{F}_{AC}$. Then for every $t_i \in tf(t)$ there is an $s_j \in tf(s)$ such that either $s_j =_{AC} t_i$ or $s_j \succ t_i$. It remains to determine dependences between rests of $s_j\gamma\downarrow$ and $t_i\gamma\downarrow$ in $tf(s\gamma\downarrow), tf(t\gamma\downarrow)$. If $s_j =_{AC} t_i$ then these rests also are AC-equal. Otherwise $s_j \succ t_i$ and of course $T(s_j) \notin \mathcal{X}$. If also $T(t_i) \notin \mathcal{X}$ then in $tf(s\gamma\downarrow)$, $tf(t\gamma\downarrow)$ we have $s_j\gamma\downarrow$ and $t_i\gamma\downarrow$ which are obviously oriented by induction hypothesis. If $T(t_i) \in \mathcal{X}$ then $t_i\gamma\downarrow$ may give rise to many terms $\boldsymbol{u_i}$ in $tf(t\gamma\downarrow)$. But $\{s_j\gamma\downarrow\} \succ^{mul} \{\boldsymbol{u_i}\}$ since $s_j\gamma\downarrow \succ t_i\gamma\downarrow$. Therefore $tf(s\gamma\downarrow) \succ^{mul} tf(t\gamma\downarrow)$ and $s\gamma\downarrow \succ t\gamma\downarrow$. □

4.5 Main Result

Having proved all the properties that make AC-λ-RPO an AC-compatible reduction ordering on higher-order terms in β-normal η-long form we can now formulate our main result.

Theorem 1. *Let H be a higher-order rewriting system modulo AC on normalised terms. If for every rule $l \to r$ of H we have $l \succ r$, then H is terminating.*

Proof. Suppose, on the contrary, that we have an infinite sequence $(u_i)_{i \in \mathbb{N}}$ in $\longrightarrow_H$. Substituting $\perp_\sigma$ for every free variable x^σ we can make $(u_i)_{i \in \mathbb{N}}$ ground. We now proceed to show that $(u_i)_{i \in \mathbb{N}}$ is decreasing with respect to $\succ$.

A rewrite step $s \xrightarrow[l \to r]{p} t$ means that $s|_p =_{AC} l\rho\downarrow$, $t =_{AC} s[r\rho\downarrow]_p$, and consequently that $s|^\lambda_p = l\rho\delta\downarrow$ where δ substitutes $\perp_\sigma$ for every free variable x^σ in $s|_p$. Note that $\rho\delta$ is a ground normalised substitution. We have $l\rho\delta\downarrow \succ r\rho\delta\downarrow$ by stability under ground substitutions (Lemma 16) and $s\langle l\rho\delta\downarrow \rangle_p \succ s\langle r\rho\delta\downarrow \rangle_p$ by stability under ground contexts (Corollary 8). As $s =_{AC} s\langle l\rho\delta\downarrow \rangle_p$ and $t =_{AC} s\langle r\rho\delta\downarrow \rangle_p$

we get by AC-compatibility $s \succ t$. Thus every infinite sequence in $\longrightarrow_H$ yields an infinite sequence decreasing in $\succ$ contradicting the well-foundedness of $\succ$ (Corollary 11). $\square$

Finally let us note that AC-λ-RPO is a total ordering, that is for any $s, t \in \mathcal{T}(\mathcal{F})\downarrow$ one of $s =_{AC} t$, $s \succ t$, $t \succ s$ holds. Totality of AC-λ-RPO is a simple consequence of totality of λ-RPO which follows from the totality of precedences on types and function symbols.

5 Examples

5.1 Map on non empty sets

The following rewriting system implements the function *map* on non empty sets of integers; *map* takes a function from integers to integers and a set, and applies this function to every element of the set. There are two basic types: Int and Set and two constructors of sets: *union* and *sing* (singleton). Of course *union* is an AC-symbol. S is a variable of type Set, n of type Int and X of type $Int \rightarrow Int$.

$$\begin{aligned} map(\lambda x^{Int}.Xx, sing(n)) &\longrightarrow sing(Xn) \\ map(\lambda x^{Int}.Xx, union(sing(n), S)) &\longrightarrow union(sing(Xn), map(\lambda x^{Nat}.Xx, S) \end{aligned}$$

To show the termination of this system consider RPO with $Set >_{\mathcal{T}_\mathcal{S}} Int >_{\mathcal{T}_\mathcal{S}} \rightarrow$ as a precedence on types and a precedence on function symbols where $map >_{\mathcal{F}} union >_{\mathcal{F}} sing$.

Analyse only the second rule. The snf_Rs of its left- and right-hand side are equal to $\{map(\lambda x^{Int}.Xx, union(\bot_{Set}, S))\}$ and $\{union(\bot_{Set}, map(\lambda x^{Int}.Xx, S)\}$ respectively. Applying case 2(b) of the definition of λ-RPO it remains to show

$$\begin{aligned} &map(\lambda x^{Int}.Xx, union(\bot_{Set}, S)) >_{\lambda rpo} \bot_{Set} \\ &map(\lambda x^{Int}.Xx, union(\bot_{Set}, S)) >_{\lambda rpo} map(\lambda x^{Int}.Xx, S) \end{aligned}$$

Both inequalities hold, because $\bot_{Set}$ is a subterm of the right-hand side and the multiset $\{\lambda x^{Int}.Xx, union(\bot_{Set}, S)\}$ is strictly greater than $\{\lambda x^{Int}.Xx, S\}$.

5.2 Some Normalisation Procedures in First-Order Logic

We have two basic types: t (terms), f (formulae) and the constants: $\vee, \wedge, \neg, \exists, \forall$. Quantifiers take as an argument a function from terms to formulae and return a formula.

$$\mathcal{S} = \{t, f\} \qquad \begin{aligned} \mathcal{F}_{f \rightarrow f} &= \{\neg\} \\ \mathcal{F}_{f \times f \rightarrow f} &= \{\vee, \wedge\} \\ \mathcal{F}_{(t \rightarrow f) \rightarrow f} &= \{\forall, \exists\} \end{aligned} \qquad \begin{aligned} \mathcal{X}_f &= \{P, Q\} \\ \mathcal{X}_{t \rightarrow f} &= \{P', Q'\} \\ \mathcal{F}_{AC} &= \{\vee, \wedge\} \end{aligned}$$

We consider two rewriting systems from [8].

Negation Normal Form.

$$\begin{aligned}
\neg\neg P &\longrightarrow P \\
\neg(P \wedge Q) &\longrightarrow \neg P \vee \neg Q \\
\neg(P \vee Q) &\longrightarrow \neg P \wedge \neg Q \\
\neg(\forall(\lambda x^t.P'x)) &\longrightarrow \exists(\lambda x^t.\neg(P'x)) \\
\neg(\exists(\lambda x^t.P'x)) &\longrightarrow \forall(\lambda x^t.\neg(P'x))
\end{aligned}$$

We choose the following precedences: all types are equal in $\geq_{\mathcal{T}_\mathcal{S}}$ and $\{\vee,\wedge\} >_\mathcal{F} \{\neg\} >_\mathcal{F} \{\exists,\forall\} >_\mathcal{F} \{\lambda_{t\to f}\}$. We set $\mathcal{F}_\mathcal{I} = \mathcal{F}_{\mathcal{AC}}$.

Let us consider only the second rule. The snf_Rs of its left- and right-hand side are equal to $\neg(P \wedge Q)$ and $P \vee Q$ respectively and the inequality $\neg(P \wedge Q) >_{\lambda rpo} P \vee Q$ holds because $\vee =_\mathcal{F} \wedge$.

Mini-Scoping.

$$\begin{aligned}
\forall\lambda x^t.P &\longrightarrow P \\
\forall\lambda x^t.(P'x \wedge Q'x) &\longrightarrow (\forall\lambda x^t.P'x) \wedge (\forall\lambda x^t.Q'x) \\
\forall\lambda x^t.(P'x \vee Q) &\longrightarrow (\forall\lambda x^t.P'x) \vee Q
\end{aligned}$$

To show the termination of this system we take the same $\geq_{\mathcal{T}_\mathcal{S}}, \geq_\mathcal{F}$ as above. It is easy to verify that all left-hand sides of the rules are strictly greater than right-hand sides.

Acknowledgement

The paper was written during my DEA studies at the University of Paris-Sud. I would like to thank my supervisor, professor Jean-Pierre Jouannaud, for his helpful comments, critiques and suggestions.

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. 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.
4. Jean-Pierre Jouannaud and Mitsuhiro Okada. Inductive data type systems. Technical report, Université de Paris Sud, 1996. submitted.
5. Jean-Pierre Jouannaud and Mitsuhiro Okada. Abstract data type systems. *Theoretical Computer Science*, 173(2):349–391, February 1997.

6. Jean-Pierre Jouannaud and Albert Rubio. A recursive path ordering for higher-order terms in η-long β-normal form. In Ganzinger H., editor, *Proc. 7th Rewriting Techniques and Applications, New-Jersey, LNCS 1103*. Springer-Verlag, 1996.
7. Dallas S. Lankford and A. M. Ballantyne. Decision procedures for simple equational theories with permutative axioms: Complete sets of permutative reductions. Research Report Memo ATP-37, Department of Mathematics and Computer Science, University of Texas, Austin, Texas, USA, August 1977.
8. Tobias Nipkow. Higher order critical pairs. In *Proc. IEEE Symp. on Logic in Comp. Science*, Amsterdam, 1991.
9. Tobias Nipkow. Functional Unification of Higher-Order Patterns. In *Proc. 8th IEEE Symp. Logic in Computer Science*, pages 64-74, 1993.
10. Albert Rubio and Robert Nieuwenhuis. A total AC-compatible ordering based on RPO. *Theoretical Computer Science*, 142(2):209–227, May 15, 1995.