Certified Higher-Order Recursive Path Ordering

Adam Koprowski

Eindhoven University of Technology
Department of Computer Science
P.O. Box 513, 5600 MB, Eindhoven, The Netherlands
`A.Koprowski@tue.nl`

Abstract. The paper reports on a formalization of a proof of well-foundedness of the higher-order recursive path ordering (HORPO) in the proof checker Coq. The development is axiom-free and fully constructive. Three substantive parts that could be used also in other developments are the formalizations of the simply-typed lambda calculus, of finite multisets and of the multiset ordering. The Coq code consists of more than 1000 lemmas and 300 definitions.

1 Introduction

A term rewriting system is terminating if all rewrite sequences are finite. Termination of first-order term rewriting, although in general undecidable, is considered to be an important problem in term rewriting. Several techniques have been developed for dealing with this problem and also a number of tools that attempt at proving termination automatically. One of the well-known techniques is the recursive path ordering (RPO) introduced by Dershowitz [6]. It is a well-founded reduction ordering and hence is suitable for proving termination.

In case of higher-order rewriting, a natural extension of first-order rewriting where bound variables may be present, significantly less results are available. Jouannaud and Rubio generalized RPO to higher-order case thus giving rise to higher-order recursive path ordering (HORPO) [8]. Using the notion of computability, introduced by Tait and Girard to prove termination of simply typed lambda calculus ($\lambda^{\rightarrow}$), they succeeded in proving well-foundedness of the union of HORPO and β-reduction of $\lambda^{\rightarrow}$. This is the essential part of the justification why HORPO can be used for proving termination of higher-order rewriting. A corollary of this result is termination of $\lambda^{\rightarrow}$ and well-foundedness of RPO which is embedded in HORPO. Later in [9] they extended and improved the ordering.

Based on those developments we made a formalization of HORPO in the theorem prover Coq, which is the subject of this paper. The formalization is complete (i.e., it does not contain any axioms) and fully constructive. The definition of HORPO is taken from [8] without extensions and improvements from [9], however parts of the theory presented in the latter was formalized as well. The formalization contains all the proofs required to justify the use of HORPO for proving termination of higher-order rewriting. More information along with Coq proof scripts can be found at:

`http://www.win.tue.nl/~akoprows/coq-horpo`.

To give the reader an impression of the size of this formalization we would like to mention that the Coq scripts consist of 29 files with $> 25,000$ lines of code and with $> 700,000$ total characters. They contain $> 1,100$ lemmas and > 300 definitions.

This formalization has become part of the CoLoR project[1]. CoLoR – Coq library on rewriting and termination – is an initiative to formalize the theory of term rewriting in Coq and ultimately to certify termination proof candidates produced by existing termination proving tools. This work can be seen as a contribution to the CoLoR project.

The structure of this paper is as follows. In Section 2 we shortly discuss motivation of this development, its short history and related work. Then in Section 3 we give a broad overview of the formalization. In Section 4 we introduce some preliminaries and we continue with the definition of higher-order terms and higher-order rewriting in Section 5. Section 6 is devoted to introduction of computability predicate proof method and contains proofs of all the computability properties required in Section 7 where we introduce HORPO ordering and prove some of its properties the main one being well-foundedness. We conclude in Section 8.

2 Motivation, History and Related Work

Formal theorem proving is rather time consuming and often requires enormous amount of work to be completed. Thus one may wonder what is the motivation behind this effort. We will mention three main motivational factors for this development.

- Verification of the proof. Sometimes the goal may be simply to verify the correctness of the proof. This is especially true for complicated proofs that are not very well known such as the work in [8].
  The results from [8, 9] are impressive and complicated and as such are inevitably subject to some small slips. This justifies the effort of verification of such results. Indeed in the course of formalization we were able to detect a small flaw, concerning the use of multiset extension of an arbitrary relation, that could be easily repaired (we will discuss it shortly in Sections 4 and 7.1). In general [8, 9] turned out to be a very favorable subject for formalization and the structure of the proofs could be followed to the letter in the formalization process.[2]
- Theorem proving is still a rather laborious task. But with constantly improving proof assistants this may not necessarily be so in the future and one

[1] `http://color.loria.fr`

[2] Obviously providing formal proofs requires to be more explicit and to include all the results that in normal presentation would be omitted as considered to be straightforward or irrelevant.

of the main stimulus for improvement and growth of theorem provers are big developments accomplished with their use.
- The most pragmatic motivation is the CoLoR initiative. As described in introduction, CoLoR is a project aiming at proving theoretical results from term rewriting in the theorem prover Coq. The ultimate goal is to (automatically) transform termination proof candidates produced by termination tools into formal Coq proofs certifying termination. This requires formalization of the term rewriting theory and this development has become part of the CoLoR library and can be seen as a contribution to this project.

This development started in January 2004 as the author's Master's Thesis [10] at the Free University Amsterdam, supervised by Femke van Raamsdonk. After half a year it was completed only with computability properties left as axioms. The eagerness of having axiom-free development resulted in another one and half year of work at the Eindhoven University of Technology, was finished in February 2006 and is the subject of this paper.

The ideas of formalizing $\lambda^{\rightarrow}$ and RPO are not new and there are some existing formalizations. Persson [14] in his PhD thesis presents a constructive proof of well-foundedness of a general form of recursive path relations. Leclerc [12] presents a formalization in Coq of well-foundedness of RPO with the multiset ordering. Murthy [13] formalizes a classical proof (due to Nash-Williams) of Higman's lemma in Nuprl 3. Berghofer [3] presents a constructive proof (due to Coquand and Fridlender) of Higman's lemma in Isabelle. Coupet-Grimal and Delobel [5] recently formalized RPO within the CoLoR project using parts of our development (for finite multisets and multiset ordering). Berger et al. [2] proved termination of $\lambda^{\rightarrow}$ in three different theorem provers including Coq. They also used the computability proof method but their work was completely independent of this formalization and focuses on extraction of normalization algorithm. However we would like to stress that we are not aware of any attempt at formalizing HORPO so, to the best of our knowledge, the main part of this work was never before a subject of formalization.

3 Overview of the Formalization

The main result of this formalization is well-foundedness of the union of HORPO and β-reduction relation of $\lambda^{\rightarrow}$. The formalization is complete and hence contains development of all the dependant results, most notably formalization of $\lambda^{\rightarrow}$ and of finite multisets and multiset extension of a relation. In this section we give overview of those results.

The development can be divided into 4 main parts. Their very brief description follows.

- **Auxiliary results**. A number of rather simple definitions and results that were not present in the Coq standard library.
- **Multisets**. Since HORPO uses multiset extension to compare arguments of functions some results about finite multisets and multiset ordering were required.

  - Finite multisets have been defined using abstract data type paradigm. That is a number of primitive operations for multisets has been specified along with their specifications.
  - To show that this axiomatic specification can be fulfilled an implementation of multisets using lists has been provided.
  - A number of abstract properties about multisets has been proven. Abstract in the sense that the proofs rely only on abstract specification of multisets. This ensures that given another (say more efficient) implementation of multisets all those results carry on automatically.
  - Multiset extension of arbitrary relation has been defined and it has been proven to preserve main properties of the relation, including well-foundedness.
- **Simply typed lambda calculus ($\lambda^{\rightarrow}$).** Higher-order rewriting uses some form of higher-order metalanguage (we will elaborate more on this subject in Section 5). In this development we decided to use $\lambda^{\rightarrow}$ in its pure form due to its universality and in the hope that this part of the development can be useful not only for the purpose of this project.
  - $\lambda^{\rightarrow}$ terms over arbitrary possibly many-sorted signature have been defined using de Bruin indices [4].
  - A number of properties concerning $\lambda^{\rightarrow}$ terms has been proven. This includes some results concerning typing, including a constructive proof of the decidability of typing and of the decidability of β-reduction from which a certified code for normalization could be extracted.
  - Typed substitution has been defined. Note that this is far from trivial as the substitution operates on typed terms so essentially the entities being substituted are typing judgements and one has to ensure that types and environments do not clash.
  - An equivalence relation on terms has been defined as an extension of α-convertibility to free variables. Convertibility of terms that are equal up to the names of free variables or irrelevant differences in environments and of lifted terms is captured by this relation.
  - Encoding of algebraic terms. As we will see in Section 5 the variant of higher-order rewriting that is of interest for us uses algebraic terms which have been encoded as $\lambda^{\rightarrow}$ terms.
  - A corollary of the main result of this development is the termination of $\lambda^{\rightarrow}$.
- **HORPO**. The definition of HORPO and proofs of its properties that constitute the main part of this paper.
  - Definition of HORPO as a slight variant of HORPO from [8].
  - Proofs of computability properties required for the main proof using computability predicate proof method due to Tait and Girard.
  - Main result: well-foundedness of the union of HORPO and β-reduction of $\lambda^{\rightarrow}$.
  - A consequence of this fact is that HORPO is a higher-order reduction ordering and hence is suitable for proving termination of higher-order term rewriting systems.

The part concerning HORPO will be treated in more details in the following sections. The reader interested in other parts of the development is encouraged to consult [11]. For introduction to $\lambda^{\rightarrow}$ we refer to [1].

4 Preliminaries

For a set A and a relation $>$ we will say that $a \in A$ is *accessible*, $a \in \mathcal{A}cc_{>}$ if a does not start any infinite reduction $a > a' > \ldots$. If the relation $>$ is clear from the context we will omit the subscript and write $a \in \mathcal{A}cc$. Note that the relation $>$ is well-founded if $\forall a \in A \;.\; a \in \mathcal{A}cc_{>}$.

Given non-empty set A we define finite multiset over A in the usual way. Now given an arbitrary relation $>$ on A we define its extension to the relation $>_{mul}$ on multisets over A as follows:

$$M >_{mul} N \iff \exists X, Y, Z \;.\; \begin{cases} Y \neq \emptyset \\ M = X \cup Y \\ N = X \cup Z \\ \forall z \in Z \;.\; \exists y \in Y \;.\; y > z \end{cases}$$

We will use following properties of this multiset ordering:

- $(\mathbf{M_1})$ If $M >_{mul} N$ then $\forall n \in N \;.\; \exists m \in M \;.\; m \geq n$.
- $(\mathbf{M_2})$ If $\forall m \in M \;.\; m \in \mathcal{A}cc_{>}$ then $M \in \mathcal{A}cc_{>_{mul}}$ (so $>_{mul}$ preserves well-foundedness).
- $(\mathbf{M_3})$ If for every $m \in M$, $n \in N$ problem whether $m > n$ is decidable then the problem whether $M >_{mul} N$ is decidable.

Note that in [8] alternative definition of multisets has been used. For orders those definitions are equivalent (which we verified in Coq) however for arbitrary relations with this alternative definition only a weaker variant of (M_2) holds where the conclusion is $m >^{*} n$. This led to some difficulties as remarked in Section 7.1. See [11] for details.

Given a set A and two relations $>_1$ and $>_2$ on A we define their lexicographic extension $(>_1, >_2)_{lex}$ as:

$$(m_1, m_2)(>_1, >_2)_{lex}(n_1, n_2) \iff m_1 >_1 n_1 \vee (m_1 = n_1 \wedge m_2 >_2 n_2)$$

We state the well-known property:

- $(\mathbf{L_1})$ If $>_1$ and $>_2$ are well-founded then so is $(>_1, >_2)_{lex}$.

5 Higher-Order Rewriting

There are several variants of higher-order rewriting. Here we use the algebraic-functional systems (AFSs) introduced by Jouannaud and Okada [7]. The main difference between AFSs and another popular format of higher-order rewriting systems (HRSs) is that in the second we work modulo beta-eta (using pure $\lambda^{\rightarrow}$

terms) whereas in AFSs we do not (and function symbols have fixed arity). As a consequence rewriting for AFSs is defined using plain pattern matching compared to rewriting modulo $\beta\eta$ of $\lambda^{\rightarrow}$ in HRSs framework. For a broader discussion on this subject we refer the reader to, for instance, [15].

Given a set of *sorts* $\mathcal{S}$ we inductively define a set of simple types $\mathcal{T}$ as:

- $\alpha \in \mathcal{T}$ if $\alpha \in \mathcal{S}$ (*base type*),
- $\alpha \rightarrow \beta \in \mathcal{T}$ if $\alpha, \beta \in \mathcal{T}$ (*arrow type*).

We define a *signature* Σ as a set of function symbols with a fixed arity. For declaration of f expecting n arguments of types $\alpha_1, \dots, \alpha_n$ and an output type β we will write $f : \alpha_1 \times \ldots \times \alpha_n \rightarrow \beta$. *Environment* is defined as a set of variable declarations, that is: $\Gamma = \{x_1 : \alpha_1, \dots, x_n : \alpha_n\}$ with $x_i \in \mathcal{V}$ and $\alpha_i \in \mathcal{T}$ for every i and with $x_i \neq x_j$ for $i \neq j$.

The set of preterms over given signature Σ and a set of variables $\mathcal{V}$ is generated according to the following grammar:

$$\mathcal{P}t := \mathcal{V} \mid @(\mathcal{P}t, \mathcal{P}t) \mid \lambda\mathcal{V}{:}\mathcal{T}.\mathcal{P}t \mid \Sigma(\mathcal{P}t, \dots, \mathcal{P}t)$$

denoting variable, application, abstraction and function application. As usually application is left-associative.

Typed terms are identified with *typing judgements* of the form $\Gamma \vdash t : \alpha$ stating that in the environment Γ preterm t has type α. They conform to the following type inference system:

$$\frac{x : \alpha \in \Gamma}{\Gamma \vdash x : \alpha} \qquad \frac{\begin{array}{c} f : \alpha_1 \times \ldots \times \alpha_n \rightarrow \beta \in \Sigma \\ \Gamma \vdash t_1 : \alpha_1, \dots, \Gamma \vdash t_n : \alpha_n \end{array}}{\Gamma \vdash f(t_1, \dots, t_n) : \beta}$$

$$\frac{\Gamma \vdash t : \alpha \rightarrow \beta \qquad \Gamma \vdash u : \alpha}{\Gamma \vdash @(t, u) : \beta} \qquad \frac{\Gamma \cup \{x : \alpha\} \vdash t : \beta}{\Gamma \vdash \lambda x{:}\alpha.t : \alpha \rightarrow \beta}$$

From here onwards we assume terms to be typed and often we will omit the environments writing $t : \alpha$ or even only t instead of $\Gamma \vdash t : \alpha$.

Free variables occurring in term t are denoted as $\mathsf{Vars}(t)$. We define the *replacement* of term u in term t at position p in the usual way and denote it by $t[u]_p$. We define the *strict subterm relation* in the standard way and denote it by $\sqsubset$. It is well-founded and will be used for the induction on the structure of terms. List of terms $@(t, u_1, \dots, u_i), u_{i+1}, \dots u_n$ is called *partial left-flattening* of term $@(t, u_1, \dots, u_n)$ for $1 \leq i \leq n$. We will say that a term t is *neutral* if it is not an abstraction. By $\sim$ we denote the equivalence relation on terms that, roughly speaking, extends α-convertibility to free variables. See [11] for details.

We will write substitution as $\gamma = [x_1/u_1, \dots, x_n/u_n]$ with its domain denoted as $Dom(\gamma) = \{x_1, \dots, x_n\}$ and the application of substitution γ to term t as $t\gamma$ resulting in a term in which all free occurrences of variables x_i are replaced with term u_i for $1 \leq i \leq n$. Note that substitution is defined on typing judgements.

The format of *higher-order term rewriting systems* and their *rewrite relation* are of no direct interest in this paper and will be omitted here – they are defined as in [9] (also see [11]).

Note that in the formalization as a metalanguage the pure $\lambda^{\rightarrow}$ terms were used. To avoid dealing with arities an assumption has been made that output types of functions are base types and hence a functional $f : \alpha_1 \times \ldots \times \alpha_n \rightarrow \beta$ is encoded using $\lambda^{\rightarrow}$ constant f of type $\alpha_1 \rightarrow \ldots \rightarrow \alpha_n \rightarrow \beta$ with its application $f(t_1, \ldots, t_n)$ encoded as $@(f, t_1, \ldots, t_n)$.

Again for more throughout introduction to notions presented in this section we refer the reader to [11].

6 Computability

In this section we present the computability predicate proof method due to Tait and Girard. In Section 7 we will use computability with respect to a particular relation (being union of HORPO and β-reduction) but we present computability for an arbitrary relation satisfying given properties.

We begin by defining computability in 6.1, in 6.2 we prove some computability properties and finally in 6.3 we make some remarks on the formalization of computability predicate in Coq.

6.1 Definition of Computability

Definition 1 (Computability). *A term $t : \delta$ is computable with respect to a relation on terms $\rhd$, denoted as $t \in \mathbb{C}_\delta$ (or simply $t \in \mathbb{C}$), if:*

- *δ is a base type and t is strongly normalizable ($t \in \mathcal{A}cc_{\rhd}$) or*
- *$\delta = \alpha \rightarrow \beta$ and $@(t, u) \in \mathbb{C}_\beta$ for all $u \in \mathbb{C}_\alpha$.*

Note that it is usual to assume that variables are computable. We do not do that, following the presentation in [8]. Computability of variables will follow from computability properties.

6.2 Computability Properties

In the formalization we made an attempt at proving computability in an abstract way, that is for an arbitrary relation $\rhd$. Below we present the list of required properties of $\rhd$ that we needed to complete all the computability proofs.

($\mathbf{P_1}$) Subject reduction: $t : \alpha \rhd u : \beta \implies \alpha = \beta$,
($\mathbf{P_2}$) Preservation of environments: $\Gamma_t \vdash t : \delta \rhd \Gamma_u \vdash u : \eta \implies \Gamma_t = \Gamma_u$.
($\mathbf{P_3}$) Free variables consistency: $t \rhd u \implies \mathsf{Vars}(u) \subseteq \mathsf{Vars}(t)$.
($\mathbf{P_4}$) Normal form of variables: $\neg(x \rhd u)$.
($\mathbf{P_5}$) Compatibility with $\sim$.[3]
($\mathbf{P_6}$) Stability under substitution: $t \rhd u \implies t\gamma \rhd u\gamma$.
($\mathbf{P_7}$) Monotonicity: $u \rhd u' \implies t[u]_p \rhd t[u']_p$.

[3] Precise formulation of this property requires more detailed introduction of $\sim$ relation; consult [11] for details.

($\mathbf{P_8}$) Reductions of abstraction: $\lambda x\!:\!\alpha.t_b \rhd u \implies \exists u_b \,.\, u = \lambda x\!:\!\alpha.u_b \wedge t_b \rhd u_b$.

($\mathbf{P_9}$) Reductions of application: $t = @(t_l, t_r) \rhd u$ implies
- $u = @(u_l, u_r)$, $t_l \unrhd u_l$ and $t_r \unrhd u_r$ and $t_l \rhd u_l \vee t_r \rhd u_r$ or
- $t \rhd u$ is a β-reduction step, so $t = @(\lambda x\!:\!\alpha.t_l, t_r) \rhd t_l[x/t_r] = u$.

All those properties but the last one are somehow standard. The last one demands reductions of application to operate argument-wise or be a β-reduction step. This property is specific for the $\rhd$ relation being union of HORPO and β-reduction relation as we will use it in Section 7.

Let us recall that we did not assume variables to be computable. Variables of a base type are computable due to the definition of computability and (P_4). Variables of a functional type are computable by property (C_3) presented in the following lemma which forbids us to prove the following computability properties (C_1), (C_2) and (C_3) separately.

Lemma 1. *For all terms $\Gamma \vdash t : \delta$, $\Delta \vdash u : \delta$ we prove that:*

($\mathbf{C_1}$) $t \in \mathbb{C}_\delta \implies t \in \mathcal{A}cc$

($\mathbf{C_2}$) $t \in \mathbb{C}_\delta \wedge t \rhd u \implies u \in \mathbb{C}_\delta$

($\mathbf{C_3}$) *if t-neutral then* $(\forall w : \delta \,.\, t \rhd w \implies w \in \mathbb{C}_\delta) \iff t \in \mathbb{C}_\delta$

Proof. Induction on type δ. Note that 'if' part of (C_3) is (C_2) so below we only prove the 'only if' part of this property.

- δ is a base type.
  - (C_1) $t \in \mathbb{C}_\delta$ and δ is a base type so $t \in \mathcal{A}cc$ by the definition of computability.
  - (C_2) $t \in \mathcal{A}cc$ by the same argument as in (C_1). $t \in \mathcal{A}cc$ and $t \rhd u$ hence $u \in \mathcal{A}cc$. By subject reduction for $\rhd$ (P_1), $u : \delta$, so $u \in \mathbb{C}_\delta$ by the definition of computability.
  - (C_3) $t : \delta$ so to show $t \in \mathbb{C}_\delta$ we need to show $t \in \mathcal{A}cc$. But for every w such that $t \rhd w$ we have $w \in \mathbb{C}_\delta$ by assumption and hence $w \in \mathcal{A}cc$ by the definition of computability and thus $t \in \mathcal{A}cc$.
- $\delta = \alpha \to \beta$
  - (C_1) Take variable $x : \alpha$ which is computable by induction hypothesis (C_3), as variables are not reducible (P_4). Now consider application $@(t, x)$ which is computable by the definition of computability. So $@(t, x) \in \mathcal{A}cc$ by induction hypothesis (C_1) and $t \in \mathcal{A}cc$ by monotonicity (P_7).
  - (C_2) By the definition of computability $u \in \mathbb{C}_{\alpha\to\beta}$ if for every $s \in \mathbb{C}_\alpha$, $@(u, s) \in \mathbb{C}_\beta$. $@(t, s) \in \mathbb{C}_\beta$ by the definition of computability and $@(t, s) \rhd @(u, s)$ by monotonicity assumption (P_7). Finally we conclude $@(u, s) \in \mathbb{C}_\beta$ by induction hypothesis (C_2).
  - (C_3) By the definition of computability $t \in \mathbb{C}_{\alpha\to\beta}$ if for every $s \in \mathbb{C}_\alpha$, $@(t, s) \in \mathbb{C}_\beta$. By induction hypothesis for (C_1), $s \in \mathcal{A}cc$ so we continue by well-founded inner induction on s with respect to $\rhd$.

    $@(t, s) : \beta$ is neutral so we can apply induction hypothesis for (C_3) and we are left to show that all reducts of $@(t, s)$ are computable. We do case analysis using (P_9). Since t is neutral and hence is not abstraction, we can exclude the β-reduction case and we are left with the following cases:

- $@(t, s) \rhd @(t', s)$ with $t \rhd t'$. Then t' is computable as so is every reduct of t and application of two computable terms is computable by the definition of computability.
- $@(t, s) \rhd @(t, s')$ with $s \rhd s'$. We observe that $s' \in \mathbb{C}$ by induction hypothesis for (C_2) and since $s \rhd s'$ we apply inner induction hypothesis to conclude $(t, s') \in \mathbb{C}_\beta$.
- $@(t, s) \rhd @(t', s')$ with $t \rhd t'$ and $s \rhd s'$. Every reduct of t is computable so $t' \in \mathbb{C}_{\alpha \to \beta}$. By induction hypothesis for (C_2) $s' \in \mathbb{C}_\alpha$. Again application of two computable terms is computable.

A simple consequence of (C_3) and (P_4) is $(\mathbf{C_4})$: all variables are computable. The last computability property involves abstractions.

Lemma 2 $(\mathbf{C_5})$. *Consider abstraction $(\lambda x : \alpha.t) : \alpha \to \beta$. If for every $u \in \mathbb{C}_\alpha$, $t[x/u] \in \mathbb{C}_\beta$ then $(\lambda x : \alpha.t) \in \mathbb{C}_{\alpha \to \beta}$.*

Proof. By the definition of computability $\lambda x : \alpha.t$ is computable if for every $s \in \mathbb{C}_\alpha$, $@(\lambda x : \alpha.t, s) \in \mathbb{C}_\beta$. Note that $t \in \mathbb{C}$ by assumption because $t = t[x/x]$ and variables are computable by (C_4). So by (C_1) both $t \in \mathcal{A}cc$ and $s \in \mathcal{A}cc$ and we proceed by induction on a pair of computable terms (t, s) with respect to the ordering $\gg = (\rhd, \rhd)_{lex}$. Now, since $@(\lambda x : \alpha.t, s)$ is neutral, by (C_3) we are left to show that all its reducts are computable. Let us continue by considering possible reducts of this application using (P_9). So we have $@(\lambda x : \alpha.t, s) \rhd u$ and the following cases to consider:

– $u = t[x/s]$. $u \in \mathbb{C}$ by the assumption.
– $u = @(\lambda x : \alpha.t, s')$ with $s \rhd s'$. $u \in \mathbb{C}$ by induction hypothesis for $(t, s') \ll (t, s)$.
– $u = @(w, s)$ with $\lambda x : \alpha.t \rhd w$. By (P_8) we know that the reduction is in the abstraction body of $\lambda x : \alpha.t$ so in fact $w = \lambda x : \alpha.t'$ with $t \rhd t'$. We conclude computability of u by induction hypothesis for $(t', s) \ll (t, s)$.
– $u = @(w, s')$ with $\lambda x : \alpha.t \rhd w$ and $s \rhd s'$. As in above case, by (P_8) we observe that $w = \lambda x : \alpha.t'$ with $t \rhd t'$ and we conclude computability of u by induction hypothesis for $(t', s') \ll (t, s)$.

6.3 Computability in Coq

Computability turned out to be by far the most difficult part of the development. In its first version ([10]) computability properties were stated as axioms. Making development axiom-free and proving all computability properties turned out to be a very laborious task after which the size of Coq scripts tripled.

Strictly speaking in terms of script size, the part of the formalization dealing with computability accounts for only slightly more than 5%. However, as those properties are at the heart of proofs concerning HORPO relation, providing proofs for them triggered many other developments.

This difficulty can be partially explained by the real complexity of the computability predicate proof method. Other factors that contributed to making this task difficult include:

– the fact that algebraic terms were encoded using pure $\lambda^{\rightarrow}$ terms,
– the necessity of defining computability modulo equivalence relation on terms.

For the clarity of presentation those issues are left implicit in the computability proofs presented in this section but in Coq proofs all of them had to be taken care of. Another difference is the use of de Bruijn indices [4] in the formalization to represent terms.

7 HORPO

In this section we present the core of this work: the results concerning the higher-order recursive path ordering (HORPO). We begin by presenting the definition of HORPO in 7.1, then some of its properties in 7.2 and its main property – well-foundedness – in 7.3. We conclude this section in 7.4 where we make some remarks about the formalization of HORPO.

7.1 Definition of HORPO

As indicated in the introduction the subject of our formalization is a slight variant of HORPO as presented in [8]. We begin by first presenting our version of the definition and then we discuss the differences comparing to the original definition by Jouannaud and Rubio.

Definition 2 (The higher-order recursive path ordering, $\succ$). *Assume a well-founded order $\rhd$ on the set of function symbols, called a precedence. We define HORPO relation $\succ$ on terms and in this definition by $\succeq$ we denote reflexive closure of HORPO (that is $\succeq \;\equiv\; \succ \cup =$) and by $\succ_{mul}$ its multiset extension.*

$\Gamma \vdash t : \delta \succ \Gamma \vdash u : \delta$ iff one of the following holds:

($\mathbf{H_1}$) $t = f(t_1, \ldots, t_n)$, $\exists i \in \{1, \ldots, n\} \;.\; t_i \succeq u$,
($\mathbf{H_2}$) $t = f(t_1, \ldots, t_n)$, $u = g(u_1, \ldots, u_k)$, $f \rhd g$, $t \succ\!\succ \{u_1, \ldots u_k\}$,
($\mathbf{H_3}$) $t = f(t_1, \ldots, t_n)$, $u = f(u_1, \ldots, u_k)$, $\{\{t_1, \ldots t_n\}\} \succ_{mul} \{\{u_1, \ldots, u_k\}\}$,
($\mathbf{H_4}$) $@(u_1, \ldots, u_k)$ *is a partial-left flattening of* u, $t \succ\!\succ \{u_1, \ldots u_k\}$,
($\mathbf{H_5}$) $t = @(t_l, t_r)$, $u = @(u_l, u_r)$, $\{\{t_l, t_r\}\} \succ_{mul} \{\{u_l, u_r\}\}$,
($\mathbf{H_6}$) $t = \lambda x \!:\! \alpha.t'$, $u = \lambda x \!:\! \alpha.u'$, $t' \succ u'$

where $\succ\!\succ$ is a relation between a term and a set of terms, defined as:
$t = f(t_1, \ldots, t_k) \succ\!\succ \{u_1, \ldots, u_n\}$ *iff* $\forall i \in \{1, \ldots, n\} \;.\; t \succ u_i \vee (\exists j \;.\; t_j \succeq u_i)$.

Note that, following Jouannaud and Rubio, we do not prove HORPO to be an ordering. In the following sections we will prove its well-foundedness and thus its transitive closure will be a well-founded ordering. There are three major differences between our definition and the definition from [8].

First let us note that in our variant only terms of equal types can be compared whereas in the original definition this restriction is weaker and it is possible to compare terms of equivalent types, where equivalence of types is a congruence generated by equating all sorts (in other words two types are equivalent if they have the same arrow structure). The reason for strengthening this assumption is that allowing to reduce between different sorts poses some technical difficulties. In [8] this problem was solved by extending the typing rules with the congruence rule which presence is basically equivalent to collapsing all sorts and which allows typing terms that normally would be ill-typed due to a sort clash. Our goal was to use $\lambda^{\rightarrow}$ in its purest form as a meta-language and hence we decided not to do that. Note however that this remark is relevant only for many-sorted signatures as for one sorted signatures type equality and type equivalence coincide.

The second difference is that the original definition of HORPO uses status and allows arguments of function symbols to be compared either lexicographically or as multisets, depending on the status, whereas we allow only for comparing arguments of functions as multisets. This choice was made simply to avoid dealing with status. Multiset comparison has been chosen as posing more difficulties. An extension with status and possibility of comparing arguments lexicographically should be relatively easy.

Finally we use different definition of multiset ordering. The property (M_1) will be crucial in lemmas preceding proof of well-foundedness of HORPO and for alternative definition of multiset extension only its weaker variant holds. This was the source of flaw in the approach of Jouannaud and Rubio we mentioned in Section 2. For more details see [11].

7.2 Properties of HORPO

In this section we will prove some properties of HORPO.

Lemma 3. *HORPO is stable under substitution, that is:* $t \succ u \implies t\gamma \succ u\gamma$

Proof. Induction on pair (t, u) ordered by $(\sqsubset, \sqsubset)_{lex}$ followed by case analysis on $t \succ u$.

- (H_1) $t = f(t_1, \dots, t_n)$ and $t_i \succeq u$ for some $i \in \{1, \dots, n\}$. But then $t\gamma = f(t_1\gamma, \dots, t_n\gamma) \succ u\gamma$ by (H_1) since $t_i\gamma \succeq u\gamma$ by the induction hypothesis.
- (H_2) $t = f(t_1, \dots, t_n)$, $u = g(u_1, \dots, u_k)$, $f \rhd g$ and $t \succ\!\succ \{u_1, \dots, u_k\}$. But then to get $t\gamma \succ u\gamma$ by (H_2) we only need to show $t\gamma \succ\!\succ \{u_1\gamma, \dots, u_k\gamma\}$. For every $i \in \{1, \dots, k\}$ we have $t \succ u_i \vee (\exists j \,.\, t_j \succeq u_i)$. In either case we have $t\gamma \succ u_i\gamma$ or $t_j\gamma \succeq u_i\gamma$ by the induction hypothesis.
- (H_3) $t = f(t_1, \dots, t_n)$, $u = f(u_1, \dots, u_k)$ and $\{\{t_1, \dots t_n\}\} \succ_{mul} \{\{u_1, \dots, u_k\}\}$ but then $\{\{t_1\gamma, \dots t_n\gamma\}\} \succ_{mul} \{\{u_1\gamma, \dots, u_k\gamma\}\}$ since for all $i \in \{1, \dots, n\}$, $j \in \{1, \dots, k\}$, $t_i \succ u_j$ implies $t_i\gamma \succ u_j\gamma$ by the induction hypothesis. So we get $t\gamma \succ u\gamma$ by (H_3).
- (H_4) $@(u_1, \dots, u_k)$ is a partial flattening of u and $t \succ\!\succ \{u_1, \dots u_k\}$. We use the same partial flattening for $u\gamma$ and get $t\gamma \succ\!\succ \{u_1\gamma, \dots, u_k\gamma\}$ with the same argument as in case (H_2). We conclude $t\gamma \succ u\gamma$ by (H_4).
- (H_5) $t = @(t_l, t_r)$, $u = @(u_l, u_r)$ and $\{\{t_l, t_r\}\} \succ_{mul} \{\{u_l, u_r\}\}$. Type considerations show that $t_l \succeq u_l$, $t_r \succeq u_r$ and $t_l \succ u_l \vee t_r \succ u_r$. By induction hypothesis on $(t_l\gamma, u_l\gamma)$ and $(t_r\gamma, u_r\gamma)$ we conclude $\{\{t_l\gamma, t_r\gamma\}\} \succ_{mul} \{\{u_l\gamma, u_r\gamma\}\}$ and hence $t\gamma \succ u\gamma$ by (H_5).

(H_6) $t = \lambda x : \alpha.t'$, $u = \lambda x : \alpha.u'$ and $t' \succ u'$. But then $t\gamma = \lambda x : \alpha.t'\gamma$, $u\gamma = \lambda x : \alpha.u'\gamma$ and $t'\gamma \succ u'\gamma$ by the induction hypothesis. So $t\gamma \succ u\gamma$ by (H_6).

Lemma 4. *HORPO is monotonous, that is:* $u \succ u' \implies t[u]_p \succ t[u']_p$.

Proof. The proof proceeds by induction on p and essentially uses the following observations:

– if $w_r \succ w'_r$ then $@(w_l, w_r) \succ @(w_l, w'_r)$ by (H_5).
– if $w_l \succ w'_l$ then $@(w_l, w_r) \succ @(w'_l, w_r)$ by (H_5).
– if $w \succ w'$ then $f(\ldots, w, \ldots) \succ f(\ldots, w', \ldots)$ by (H_3).
– if $w \succ w'$ then $\lambda x : \alpha.w \succ \lambda x : \alpha.w'$ by (H_6).

So we presented proofs for computability properties (P_6) and (P_7). Properties (P_1), (P_2), (P_4), (P_8) and (P_9) are direct from the definition of HORPO. (P_3) is easy using induction on pair of terms ordered by $(\sqsubset, \sqsubset)_{lex}$. For ($P_5$) more detailed presentation of equivalence relation $\sim$ is needed and we refer the interested reader to [11] for details.

We conclude this section with a result that is not present in [8], namely a proof of the fact that $\succ$ is decidable.

Theorem 1. *Given terms t and u the problem whether $t \succ u$ is decidable.*

Proof. Induction on the pair (t, u) ordered by $(\sqsubset, \sqsubset)_{lex}$ followed by a case analysis on t.

– $t = x$. Variables are in normal forms with respect to $\succ$ so we cannot have $x \succ u$.
– $t = @(t_l, t_r)$. Only (H_5) is applicable if $u = @(u_l, u_r)$ and for that, taking typing consideration into account, it is required that $t_l \succeq u_l$, $t_r \succeq u_r$ and $t_l \succ u_l \vee u_l \succ u_r$ all of which is decidable by induction hypothesis.
– $t = \lambda x : \alpha.t_b$. Only ($H_6$) is applicable for $u = \lambda x : \alpha.u_b$ and it is required that $t_b \succ u_b$ which we can decide by induction hypothesis.
– $t = f(t_1, \ldots, t_n)$. We have several cases to consider corresponding to application of different clauses of HORPO:
  - (H_1): for every $i \in \{1, \ldots, n\}$ we check whether $t_i \succeq u$ by application of induction hypothesis.
  - (H_2): u needs to be of the shape $u = g(u_1, \ldots, u_k)$ with $f \rhd g$ (we assume precedence to be decidable). We need to check whether $t \succ\!\succ \{u_1, \ldots, u_k\}$. So for every $i \in \{1, \ldots, k\}$ we check whether $t \succ u_i$ or $t_j \succ u_i$ for some $j \in \{1, \ldots, n\}$. Typing consideration are helpful in immediately discarding of many cases.
  - (H_3): comparison between all arguments of t and u is decidable by induction hypothesis so to conclude whether multisets of arguments can be compared we use (M_3).
  - (H_4): we consider all the possible partial flattenings $@(u_1, \ldots, u_k)$ of u (bounded by the size of u) and for each of them we check whether $t \succ\!\succ \{u_1, \ldots, u_k\}$ in the same way as in the (H_2) case.

7.3 Well-foundedness of HORPO

In this section we present the proof of well-foundedness of $\succ \cup \rightarrow_\beta$. This relation will play important role in this section so let us abbreviate it by $\rightsquigarrow \equiv \succ \cup \rightarrow_\beta$. For the proof we will use the computability predicate proof method due to Tait and Girard (as in [8]) which was discussed in Section 6.

Note that we will use computability with respect to $\rightsquigarrow$ and for that we need to prove the properties (P_1)-(P_9) for $\rightsquigarrow$. We proved (P_6) and (P_7) for $\succ$ in Section 7.2. For the remaining properties of $\succ$, all properties for $\rightarrow_\beta$ (which are easy and standard) we refer to [11]. All those properties easily generalize to the union if they hold for the components.

The crucial lemma states that if function arguments are computable then so is the function application. We first need an auxiliary lemma for which the proof is easy and can be found in [11].

Lemma 5. *For any* $t = f(t_1, \dots, t_n)$ *and* $u = g(u_1, \dots, u_k)$ *if* $t \succ\!\succ \{u_1, \dots u_k\}$ *and* $\forall i \in \{1, \dots, n\}$. $t_i \in \mathbb{C}$ *and* $\forall j \in \{1, \dots, k\}$. $t \rightsquigarrow u_j \implies u_j \in \mathbb{C}$ *then* $\forall j \in \{1, \dots, k\}$. $u_j \in \mathbb{C}$.

Lemma 6. *If* $t_1, \dots, t_n \in \mathbb{C}$ *then* $t = f(t_1, \dots, t_n) \in \mathbb{C}$.

Proof. The proof proceeds by well-founded induction on the pair of a function symbol and a multiset of computable terms, $(f, \{\{t_1, \dots, t_n\}\})$, ordered lexicographically by $(\triangleright, \rightsquigarrow_{mul})_{lex}$. Note that all terms in the multiset are computable and hence, by (C_1), strongly normalizable. So $(\triangleright, \rightsquigarrow_{mul})_{lex}$ is well-founded by (M_2) and (L_1) which justifies the induction argument.

Since t is neutral we apply (C_3) and we are left to show that for arbitrary u, such that $t \rightsquigarrow u$, $u \in \mathbb{C}$. We will show that by inner induction on the structure of u. We continue by case analysis on $t \rightsquigarrow u$. The first case corresponds to a beta-reduction step and the following ones to applications of the clauses (H_1), (H_2), (H_3) and (H_4) of the HORPO definition. Note that the clauses (H_5) and (H_6) are not applicable.

- (β) Let $t \rightarrow_\beta u$. The $\rightarrow_\beta$ step is in one of the arguments, so for some j we have $u = f(t_1, \dots t'_j, \dots t_n)$ with $t_j \rightarrow_\beta t'_j$. For every i, $t_i \in \mathbb{C}$ by assumption and $t'_j \in \mathbb{C}$ by (C_2) so we conclude $u \in \mathbb{C}$ by the outer induction hypothesis.
- (H_1) $t_i \succeq u$ for some $i \in \{1, \dots, n\}$. By assumption $t_i \in \mathbb{C}$ so $u \in \mathbb{C}$ by (C_2).
- (H_2) $u = g(u_1, \dots u_k)$ with $f \triangleright g$. All $u_i \in \mathbb{C}$ for $1 \le i \le k$ by Lemma 5 and since $(f, \{\{t_1, \dots t_n\}\})$ $(\triangleright, \rightsquigarrow_{mul})_{lex}$ $(g, \{\{u_1, \dots, u_k\}\})$ we conclude that $u \in \mathbb{C}$ by the outer induction hypothesis.
- (H_3) $u = f(u_1, \dots u_k)$ with $\{\{t_1, \dots, t_n\}\} \succ_{mul} \{\{u_1, \dots, u_k\}\}$. We can conclude $u \in \mathbb{C}$ by the outer induction hypothesis if we can prove that $u_i \in \mathbb{C}$ for $1 \le i \le k$. For arbitrary i, by (M_1) we get $t_j \succeq u_i$ for some j and since $t_j \in \mathbb{C}$ by assumption we conclude $u_i \in \mathbb{C}$ by (C_2).
- (H_4) $@(u_1, \dots, u_k)$ is some left-partial flattening of u and $t \succ\!\succ \{u_1, \dots, u_k\}$. By Lemma 5 we get $u_i \in \mathbb{C}$ for $1 \le i \le k$ and hence $u \in \mathbb{C}$.

The next step is to show that the application of a computable substitution gives computable term, where we define computable substitution as a substitution containing in its domain only computable terms.

Lemma 7. *We say that $\gamma = [x_1/u_1, \ldots, x_n/u_n]$ is a* computable substitution *if for every $i \in \{1, \ldots, n\}$, $u_i \in \mathbb{C}$. Let γ be computable substitution. Then for any term t, $t\gamma \in \mathbb{C}$.*

Proof. We proceed by induction on the structure of term t.

- $t = x$. If $x \in \mathit{Dom}(\gamma)$ then $\gamma = [\ldots, x/u, \ldots]$ and $t\gamma = u$ but $u \in \mathbb{C}$ since γ is computable. Otherwise $t\gamma = x \in \mathbb{C}$ as variables are computable (C_4).
- $t = f(t_1, \ldots, t_n)$ so $t\gamma = f(t_1\gamma, \ldots, t_n\gamma)$. We apply Lemma 6 and we are left to show that for $i \in \{1, \ldots, n\}$, $t_i\gamma \in \mathbb{C}$ which easily follows from the induction hypothesis.
- $t = @(t_l, t_r)$ and $t\gamma = @(t_l\gamma, t_r\gamma)$. Both $t_l\gamma$ and $t_r\gamma$ are computable by the induction hypothesis so $t\gamma \in \mathbb{C}$ by the definition of computability.
- $t = \lambda x : \alpha . t_b$ so $t\gamma = \lambda x : \alpha . t_b\gamma$. By application of (C_5) we are left to show that $t_b\gamma[x/u] \in \mathbb{C}$ for any $u \in \mathbb{C}_\alpha$. But $t_b\gamma[x/u] = t_b(\gamma \cup [x/u])$ since $x \notin \mathit{Dom}(\gamma)$. Since $\gamma \cup [x/u]$ is a computable substitution as so is γ and $u \in \mathbb{C}$, we can conclude $t\gamma \in \mathbb{C}$ by the induction hypothesis.

Now we are ready to present the main theorem stating that the union of HORPO and THE β-reduction relation of THE simply typed λ-calculus, is a well-founded relation on terms.

Theorem 2. *The relation $\rightsquigarrow$ is well-founded.*

Proof. We need to show that $t \in \mathcal{A}cc$ for arbitrary t. Consider an empty substitution ϵ, which is computable by definition. We also have $t = t\epsilon$ so we conclude $t \in \mathbb{C}$ by Lemma 7 and then $t \in \mathcal{A}cc$ by (C_1).

7.4 HORPO in Coq

The definition of HORPO and proof of its well-foundedness are main subjects of this work, however only less than 10% of Coqscripts is devoted to them. The development of this part was going rather smoothly. Mostly problematic were the computability properties but once they were there the proof of well-foundedness and other properties of HORPO could be accomplished with relative ease following closely the proofs from [8, 9].

The definition of HORPO is slightly complicated and that is because $\succ$, $\succ_{mul}$, $\succeq$ and $\succ\!\succ$ all need to be combined in one mutually inductive definition as all of them refer to each other.

8 Conclusions

We presented a formalization of the higher-order recursive path ordering in the theorem prover Coq. This development took two years and resulted in a rather big, complete, axiom-free Coq formalization that is a part of the CoLoR project.

The development contains three parts that are completely autonomous: simply typed lambda calculus, multisets and the multiset ordering. These parts will be submitted as Coq contributions so that hopefully they will be used in other developments. In fact the formalization of multisets and multiset ordering already have been used by Coupet-Grimal and Delobel in their recent formalization of RPO [5]. As for the main results concerning HORPO they are integral part of CoLoR library and hopefully will stimulate more developments in the area of higher-order rewriting.

Acknowledgements. The author would like to thank the anonymous reviewers for their useful comments.

References

1. H. P. Barendregt. Lambda calculi with types. *Handbook of logic in computer science (vol. II)*, pages 117–309, 1992.
2. U. Berger, S. Berghofer, P. Letouzey, and H. Schwichtenberg. Program extraction from normalization proofs. *Studia Logica*, 2005. Special issue, to appear.
3. S. Berghofer. A constructive proof of Higman's lemma in Isabelle. 3085:66–82, 2004.
4. N. G. de Bruijn. Lambda-calculus notation with nameless dummies: a tool for automatic formula manipulation with application to the Church-Rosser theorem. *Indag. Math.*, 34(5):381–392, 1972.
5. Solange Coupet-Grimal and William Delobel. A Constructive Axiomatization of the Recursive Path Ordering. Research report 28-2006, LIF, France, 2006.
6. N. Dershowitz. Orderings for term-rewriting systems. *Theor. Comput. Sci.*, 17:279–301, 1982.
7. J.-P. Jouannaud and M. Okada. Executable higher order algebraic specification languages. In *LICS '91*, 350–361, 1991.
8. J.-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *LICS '99*, 402–411, Italy, 1999.
9. J.-P. Jouannaud and A. Rubio. Higher-order recursive path orderings 'à la carte'. 2001.
10. A. Koprowski. Well-foundedness of the higher-order recursive path ordering in Coq. TI-IR-004, Vrije Universiteit, The Netherlands, 2004. Master's Thesis.
11. A. Koprowski. Coq formalization of the higher-order recursive path ordering. Technical report in CS-Report series, Eindhoven Univ. of Tech.
12. F. Leclerc. Termination proof of term rewriting systems with the multiset path ordering: A complete development in the system Coq. In TLCA '95, 902:312–327, 1995.
13. C. Murthy. Extracting constructive content from classical proofs. 1990. PhD Thesis.
14. H. Persson. Type theory and the integrated logic of programs. 1999. PhD Thesis.
15. F. van Raamsdonk. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in TCS*, chapter 11, pages 588–668. Cambridge University Press, 2003.