Well-foundedness of the recursive path ordering in Coq

Nicole de Kleijn, Adam Koprowski, and Femke van Raamsdonk
Department of Computer Science, Vrije Universiteit
Amsterdam, The Netherlands
ndkleijn,akoprow,femke@cs.vu.nl

June 2004

Abstract

The recursive path ordering due to Dershowitz is one of the important methods to prove termination of first-order term rewriting. The generalization to the higher-order case is due to Jouannaud and Rubio. This work is concerned with a formalization in Coq of the proof of well-foundedness of both the first- and the higher-order version of the recursive path ordering. The proof does not rely on Kruskal's tree theorem.

1 Introduction

Background. A term rewriting system is terminating if all its reduction sequences are finite. Termination of term rewriting is in general undecidable, but especially for first-order term rewriting many techniques have been developed to prove termination in particular cases. Obviously, a term rewriting system is terminating if it can be shown that for every rewrite step $s \to t$ we have $s > t$ in some well-founded ordering on the set of terms.

An important syntactical technique to prove termination of first-order term rewriting is the one using the recursive path ordering (RPO) defined by Dershowitz [2]. The starting point of the definition of RPO is a well-founded ordering on the set of function symbols. Then two terms are compared by first comparing their head-symbols, and then recursively comparing their arguments. We consider here only the case where the arguments are compared as multisets, but there are other possibilities, such as comparing them as lexicographically ordered lists. RPO turns out to be a reduction ordering, which means that in order to prove $s > t$ for every rewrite step $s \to t$, it is sufficient to prove $l > r$ for every rewrite rule $l \to r$. RPO is one of the important termination methods that is suitable for automation. A crucial point in the justification of the termination method using RPO is the proof of its well-foundedness. Originally this proof uses Kruskal's tree theorem.

For higher-order term rewriting, where terms are simply typed λ-terms, not so many termination methods exist. A bottleneck in the generalization of RPO to the higher-order case was the fact that there is no suitable extension of Kruskal's tree theorem for higher-order terms. A breakthrough was the introduction of the higher-order recursive path ordering (HORPO) by Jouannaud and Rubio [4]. A crucial point is that the proof of well-foundedness of HORPO does not rely on Kruskal's tree theorem, but uses instead the computability proof method due to Tait and Girard. The specialization to the first-order case yields a new proof of well-foundedness of RPO, that proceeds by induction on the structure of terms, which is independently also presented in the work by Persson [11].

Contribution. The present work is concerned with a formalization of well-foundedness of RPO and HORPO in Coq. On the one hand, this work can be seen as a contribution to the development of the theory for extending the proof assistant Coq with rewriting: if we want to add the possibility of calculating using term rewrite rules, then we need a formal proof of the termination of those rules. On the other hand, this work might contribute to the study of automatically proving termination of term rewriting: several tools for proving termination of term rewriting have been developed, and it would be interesting to be able to feed their output to a proof assistant such as Coq or Isabelle to verify the correctness of the output.

History and status of the work. This work is based on two master's thesis projects. The first one was carried out by Nicole de Kleijn [6] at the Vrije Universiteit in the period February 2003 - August 2003. This project is concerned with a formalization of the well-foundedness of the first-order recursive path ordering. The second project is carried out by Adam Koprowski [7], also at the Vrije Universiteit, in the period February 2004 - August 2004. This project is concerned with a formalization of the well-foundedness of the higher-order recursive path ordering, in the version with syntactic pattern matching (not modulo $\beta\eta$).

The work is still in progress. The current status of the formalization of well-foundedness of HORPO is that is consists of around 5000 lines of code in Coq version 8.0. The main hypotheses concern the computability properties. The formalization of well-foundedness of RPO is done in Coq version 7.3.1 and parts of it still need to be transferred to version 8. The present extended abstract is a first presentation of the combined projects and will be worked out in more detail later on.

Related work. Persson [11] introduces the notion of recursive path relation which generalizes the recursive path ordering by not requiring the underlying relations to be orderings. He presents a constructive proof of

well-foundedness of a general form of recursive path relations. This proof is very similar to, and independently obtained, of the specialization to the first-order case of the proof of well-foundedness of HORPO by Jouannaud and Rubio [4]. The proof in [11] is extracted from the classical proof using a minimal bad sequence argument by using open induction due to Raoult [13]. Persson also presents an abstract formalization of well-foundedness of recursive path relations in the proof-checker Agda. The main difference between the work by Persson and the current work is the level of abstraction: here we are much more concrete. Another difference is of course the use of Agda instead of Coq.

Leclerc [8] presents a formalization in Coq of well-foundedness of RPO with the multiset ordering. The Coq script consists of about 250 pages and hence is not presented in full detail in the reference. The focus is on giving upper bounds for descending sequences in RPO. There are quite some differences between the work by Leclerc and the current work. Most datatypes are represented differently, and also the current development is like the one by Persson substantially shorter.

Murthy [9] formalizes a classical proof of Higman's lemma, a specific instance of Kruskal's tree theorem, in a classical extension of Nuprl 3. The classical proof is due to Nash-Williams and uses a minimal bad sequence argument. The formalized classical proof was automatically translated into a constructive proof using Friedman's A-translation. The formalization is very big: around 10 megabytes.

Berghofer [1] presents a constructive proof of Higman's lemma in Isabelle. The constructive proof is due to Coquand and Fridlender.

Organization of the paper. In this paper we focus on the formalization of the well-foundedness of the higher-order version of the recursive path ordering, since the first-order case can be obtained as a specialization. Several variations of the higher-order version of the recursive path ordering exist, see [5, 12]. The formalization deals so far only with the most simple version, where HORPO is defined for algebraic-functional systems (AFSs) with a simple typing discipline. An important point here is that we do not work modulo β, and matching is syntactic. Further, we consider the version of HORPO without the computable closure.

To start with, the definition of algebraic-functional systems (AFSs) with a simple typing discipline is given. Along the way we comment on some aspects of the formalization. Then the definition of HORPO, and the proof of its well-foundedness are presented. The definition and the well-foundedness proof use a part of the theory of finite multisets, which is formalized in [7].

2 Higher-order term rewriting

There are several definitions of higher-order rewriting. They fall into two categories: the ones where rewriting is roughly defined modulo β, and the ones where plain rewriting is used, that is, matching is syntactic. Definitions in the first category are the combinatory reduction systems (CRSs) introduced by Klop, the similar class of expression reduction systems (ERSs) introduced independently by Khasidashvili, and the higher-order rewrite systems (HRSs) defined by Nipkow. In the second category are the algebraic-functional systems (AFSs) introduced by Jouannaud and Okada [3].

In the formalization we restrict attention to the case of AFSs with a simple type discipline, like in the first definition of HORPO. In this section we recall the main ingredients of the definition of these AFSs.

Types. To start with, we assume a non-empty set of *base types* with decidable equality. From the set of base types, simple types are built inductively according to the following grammar:

$$A, B ::= a \mid A \to B$$

with a a base type. Simple types, shortly called types, are written as $A, B, C, \ldots$.

For the definition of HORPO, we use the following equivalence relation on the set of simple types: A and B are equivalent if they have the same arrow structure. That is, the equivalence relation $\equiv$ on the set of types is defined as the smallest relation that satisfies the following two clauses:

1. we have $a \equiv b$ if a and b are both base types, and
2. we have $A \to A' \equiv B \to B'$ if $A \equiv A'$ and $B \equiv B'$.

For instance, $\mathsf{nat} \to \mathsf{bool} \equiv \mathsf{natlist} \to \mathsf{nat}$.

In the formalization, simple types are defined as an inductive type, and the equivalence on the set of types is a fixed point definition.

Pre-terms. We assume a countably infinite set of *variables* written as $x, y, z, \ldots$. Also, a non-empty set of *function symbols* is assumed, which is supposed to be disjoint from the set of variables. Every function symbol comes equipped with a unique type. We assume a decidable equality relation on the set of function symbols.

Pre-terms, also called raw terms, are built from variables, function symbols, abstraction and simple types, and application. The set of *pre-terms* is defined inductively according to the following grammar:

$$M, N ::= x \mid f \mid \lambda x : A.\, M \mid @(M, N)$$

The terminology 'pre-term' has nothing to do with being in β-normal form or not; it means that no typing relation is taken into account yet.

In the formalization, De Bruijn indices are used to avoid the α-conversion problems. So a variable is a natural number, and an abstraction is of the form $\lambda^A M$. Pre-terms are defined as an inductive type. In the paper, we use named variables for the sake of readability.

The typing relation. An *environment* is a set of declarations of the form $x : A$ with all variables distinct. A pre-term M is said to be *typable* if we can derive $\Gamma \vdash M : A$ for some environment Γ and some type A. A typable pre-term is called a *term*. The typing relation is defined by the following clauses:

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : A \to B}$$

$$\frac{f : A}{\Gamma \vdash f : A} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash @(M, N) : N}$$

Given an environment, a term has a unique type.

Because of the De Bruijn notation, an environment in the formalization is just a list of types. The typing relation

```
Typing : Env -> Preterm -> SimpleType -> Set
```

is defined inductively, taking as inputs an environment, a pre-term, and a type. Finally, a term is a record consisting of an environment Γ, a pre-term M, a type A and a typing derivation $\Gamma \vdash M : A$.

There is a slight difference between the definition of pre-terms and terms as used here, and the one in [4]. There a function symbol has besides a type also an arity, expressing the number of arguments it should get. A function symbol f with arity 3 can be used to build a term of the form $f(a, b, c)$. If this term has type $D \to E$, then it can be used to form the application $@(f(a, b, c), d)$ of type E. Here and in the formalization we do not use functional application but only the binary application written as $@$. So a function symbol $f : A \to B \to C \to D \to E$ can for instance be used to build the terms $@(f, a)$, $@(@(f, a), b)$, and $@(@(@(f, a), b), c)$.

Substitution. A *substitution* is a finite mapping from variables to terms written as $\{x_1 \mapsto M_1, \ldots, x_n \mapsto M_n\}$. In the formalization, the definition of substitution is quite involved. It deals for instance with the lifting of indices.

Rewriting. The focus in the formalization so far is on well-foundedness of the higher-order recursive path ordering. For the definition of the ordering and the proof of its well-foundedness it is in fact not necessary to know

the rewrite relation of AFSs. But of course HORPO is defined with the aim to prove termination of rewriting, so we continue here by explaining shortly the definition of the rewrite relation. This part is not present in the formalization so far.

The definition of rewriting in an AFS is as follows. First, the rewrite relation $\to_R$ is induced by a set of rewrite rules. A *rewrite rule* is of the form $\Gamma \vdash l \to r : A$, where both $\Gamma \vdash l : A$ and $\Gamma \vdash r : A$ are derivable. We have a *rewrite step* $M \to_R N$ if there is a position p in M such that $M|_p$ is of the form $l\sigma$. Further, $N = M[p \leftarrow r\sigma]$, that is, N is obtained from M by replacing the subterm $l\sigma$ at position p by the subterm $r\sigma$. We do not give all formal definitions here; they are mainly standard. Note that we do not mention typing compatibility issues. An important thing is that matching is *syntactic*.

Besides the algebraic rewrite rules that induce the rewrite relation $\to_R$, there is also the rule for β-reduction:

$$@(\lambda x : A.\, M, N) \to_\beta M\{x \mapsto N\}$$

The rewrite relation of interest is the union of $\to_R$ and the rewrite relation $\to_\beta$ induced by the β-reduction rule. This rewrite relation is denoted by $\to_{R\beta}$ or also shortly by $\to$.

We conclude the description of AFSs by giving some examples of rewriting systems. In the examples we will sometimes write $f(M_1, \ldots, M_n)$ instead of $@(\ldots @(f, M_1) \ldots M_n)$. This is just a notation meant to increase readability. We also use the convention that functional variables are written in upper case.

Example 2.1.

1. First we consider the example for map. We assume two base types: nat and $\mathsf{natlist}$. The following signature is used:

$$\begin{array}{lcl} \mathsf{nil} & : & \mathsf{natlist} \\ \mathsf{cons} & : & \mathsf{nat} \to \mathsf{natlist} \to \mathsf{natlist} \\ \mathsf{map} & : & \mathsf{natlist} \to (\mathsf{nat} \to \mathsf{nat}) \to \mathsf{natlist} \end{array}$$

   The rewrite rules for map are the following:

$$\begin{array}{lcl} \mathsf{map}(\mathsf{nil}, Z) & \to & \mathsf{nil} \\ \mathsf{map}(\mathsf{cons}(h, t), Z) & \to & \mathsf{cons}(@(Z, h), \mathsf{map}(t, Z)) \end{array}$$

   Here $Z : \mathsf{nat} \to \mathsf{nat}$ is a variable for a function on natural numbers, and $h : \mathsf{nat}$ and $t : \mathsf{natlist}$ are variables for the head and the tail.

   As we will see later, this rewrite system can be proved to be terminating using HORPO.

2. The second example is Gödel's recursor for natural numbers. We assume base types nat and A. The following signature is used:

$$\begin{array}{lcl} \mathsf{0} & : & \mathsf{nat} \\ \mathsf{s} & : & \mathsf{nat} \to \mathsf{nat} \\ \mathsf{rec} & : & \mathsf{nat} \to A \to (\mathsf{nat} \to A \to A) \to A \end{array}$$

The rewrite rules for rec are the following:

$$\begin{array}{lcl} \mathsf{rec}(\mathsf{0}, y, Z) & \to & y \\ \mathsf{rec}(\mathsf{s}(x), y, Z) & \to & @(Z, x, \mathsf{rec}(x, y, Z)) \end{array}$$

Here $Z : \mathsf{nat} \to A \to A$ is a functional variable.

Also this AFS can be shown to be terminating using HORPO.

3. Finally, we consider an example from [3]. We assume a base type A, and the following signature:

$$f : A \to A \to A$$

There is one rewrite rule for the symbol f:

$$f(@(Z, x), x) \to f(@(Z, x), @(Z, x))$$

The rewrite relation induced by this rewrite rule and the rule for β-reduction is not terminating:

$$\begin{array}{ll} f(@(\lambda x : A.\, x, y), y) & \to \\ f(@(\lambda x : A.\, x, y), @(\lambda x : A.\, x, y)) & \to \\ f(@(\lambda x : A.\, x, y), y) & \to \\ \ldots & \end{array}$$

3 The higher-order recursive path ordering

In this section we give the definition of HORPO for AFSs with simple types. This is the first definition of HORPO as presented in [4]. A more powerful version of HORPO using also the notion of computable closure is presented also in that paper. Later on, HORPO has been extended in several ways, as presented in [5, 12]. In this paper and in the formalization, we only consider the first, most simple version.

We assume a well-founded ordering on the set of function symbols, called a precedence, denoted by $\triangleright$. We use again the notation $f(M_1, \ldots, M_n)$. Also, we use the notation $@(M_1, \ldots, M_n)$ with just one instead of $n - 1$ application symbols. Given an environment Γ, the type of a term M is unique and denoted by $\mathsf{type}(M)$.

Definition 3.1. Suppose that $\Gamma \vdash M : A$ and $\Gamma \vdash N : A'$ for some environment Γ, and for types A and A' with $A \equiv A'$. Then we have $\Gamma \vdash M \succ N$ if this can be inferred using the following clauses:

1. $M = f(M_1, \ldots, M_k)$ for some $k \geq 0$
   there exists $i \in \{1, \ldots, k\}$ such that $M_i \succeq N$

2. $M = f(M_1, \ldots, M_k)$ for some $k \geq 0$
   $N = g(N_1, \ldots, N_l)$ for some $l \geq 0$
   $f \triangleright g$
   for all $j \in \{1, \ldots, l\}$ we have:
   if $\mathsf{type}(M) \equiv \mathsf{type}(N_j)$ then $M \succ N_j$,
   if $\mathsf{type}(M) \not\equiv \mathsf{type}(N_j)$, then
   there exists $i \in \{1, \ldots, k\}$ such that $M_i \succeq N_j$

3. $M = f(M_1, \ldots, M_k)$ for some $k \geq 1$
   $N = f(N_1, \ldots, N_k)$
   $\{\{M_1, \ldots, M_k\}\} \succ_{mul} \{\{N_1, \ldots, N_k\}\}$

4. $M = f(M_1, \ldots, M_k)$ for some $k \geq 0$
   $N = @(N_1, \ldots, N_l)$ for some $l \geq 2$
   for all $j \in \{1, \ldots, l\}$ we have:
   if $\mathsf{type}(M) \equiv \mathsf{type}(N_j)$ then $M \succ N_j$,
   if $\mathsf{type}(M) \not\equiv \mathsf{type}(N_j)$, then
   there exists $i \in \{1, \ldots, k\}$ such that $M_i \succeq N_j$

5. $M = @(M_1, M_2)$
   $N = @(N_1, N_2)$
   $\{\{M_1, M_2\}\} \succ_{mul} \{\{N_1, N_2\}\}$

6. $M = \lambda x : B.\, M_0$
   $N = \lambda x : B'.\, N_0$
   $M_0 \succ N_0$

$\diamond$

A few remarks concerning this definition:

- This is the definition that is formalized.
- Two terms can only be compared using HORPO if they have in the same environment equivalent types. We assume this, also if at most places we write only $M \succ N$.
- We use $\{\{\ldots\}\}$ to denote a multiset. The extension of $\succ$ to multisets is denoted by $\succ_{mul}$. More comments on this issue can be found below.

- A first difference with the definition as given in [4] is that we have one clause less: in case of two terms starting with the same head-symbol, we compare the arguments only by the multiset extension of $\succ$. In the original definition there is yet another clause, were the arguments are put in lists which are compared using the lexicographic extension of $\succ$. We do not consider this clause in the formalization. (In the original definition of RPO due to Dershowitz, the lexicographic clause is not present. The lexicographic version of RPO is introduced in an unpublished note by Kamin and Lévy.)

- The second (and last) difference with the definition as given in [4] is that the statement

  for all $j \in \{1, \ldots, l\}$ we have:
  if $\mathsf{type}(M) \equiv \mathsf{type}(N_j)$ then $M \succ N_j$,
  if $\mathsf{type}(M) \not\equiv \mathsf{type}(N_j)$, then
  there exists $i \in \{1, \ldots, k\}$ such that $M_i \succeq N_j$

  as in the definition above is deterministic. This is a slight change compared to the original definition, where a non-deterministic statement is given. It is remarked already there that this non-deterministic statement can be replaced by the deterministic statement that we use here. Clearly the deterministic version is more suitable for a formalization.

- Note that in clause 5 for application M_1 can only be compared with N_1, and M_2 can only be compared with N_2 for typing reasons. Hence there are three possibilities (in fact we need to know the definition of the multiset extension in order to see this): $M_1 \succ^* N_1$ and $M_2 = N_2$, or $M_1 = N_1$ and $M_2 \succ^* N_2$, or $M_1 \succ^* N_1$ and $M_2 \succ^* N_2$, with $\succ^*$ the transitive closure of $\succ$.

- In fact, for typing reasons in clause 4 we have $k \geq 1$.

- It can be shown that clause 5 for application is redundant in the case that M_1 is a function symbol. This is used in the formalization.

- It is crucial that only terms of equivalent types are compared using HORPO. However, note that the precedence does not take the types of the function symbols into account; it is possible that function symbols of different types are related in the precedence.

- The first three clauses of the definition together give the original first-order definition of the recursive path ordering as introduced by Dershowitz [2]. To make this slightly more precise we need to represent a first-order term rewriting system as an AFS which is not very hard:

take one base type a (representing the set of terms), and give a function symbol of arity n the type $a \to \ldots \to a \to a$ with in total $n+1$ times a.

In the following example we show that the rewrite rules for map and for the recursor rec can be oriented using HORPO.

Example 3.2.

1. First we consider the example for map. We use the precedence $\mathsf{map} \triangleright \mathsf{cons}$.

   We have $\mathsf{map}(\mathsf{nil}, Z) \succ \mathsf{nil}$ by an application of clause 1.

   In order to show $\mathsf{map}(\mathsf{cons}(h,t), Z) \succ \mathsf{cons}(@(Z,h), \mathsf{map}(t,Z))$, we apply clause 2. Then two things need to be shown: first we need to show that $\mathsf{map}(\mathsf{cons}(h,t), Z) \succ @(Z,h)$ (note that $\mathsf{natlist}$ and nat are equivalent types), and second that $\mathsf{map}(\mathsf{cons}(h,t), Z) \succ \mathsf{map}(t,Z)$. The first holds by an application of clause 4. The second holds by an application of clause 3 because $\mathsf{cons}(h,t) \succ t$.

2. Second we consider the example for rec.

   We have $\mathsf{rec}(\mathsf{0}, y, Z) \succ y$ by an application of clause 1.

   In order to show $\mathsf{rec}(\mathsf{s}(x), y, Z) \succ @(Z, x, \mathsf{rec}(x,y,Z))$ we apply clause 4. Then three things remain to be shown. First, we have $Z \succeq Z$. Second, we have $\mathsf{rec}(\mathsf{s}(x), y, Z) \succ x$ by an application of clause 1. Note that the types A and nat are equivalent. Third, we have $\mathsf{rec}(\mathsf{s}(x), y, Z) \succ \mathsf{rec}(x,y,Z)$ by an application of clause 3 because $\mathsf{s}(x) \succ x$.

In the formalization, the definition of HORPO is formalized by mutually dependent inductive types. To start with, there is the inductive definition of HORPO that takes into account the environment and the types of the terms to be compared. Depending on the definition of HORPO are the extension of HORPO to multisets of terms (used in clause 3 of the definition above) and the reflexive closure (in the definition above written as $\succeq$) of HORPO. There is a mutual dependency between the definition of HORPO and the definition of a relation that is called pre-HORPO in the formalization. The clauses in the definition of pre-HORPO correspond to the clauses of the definition given above. Pre-HORPO does not take the environments and types of the terms into account. It uses another inductive definition, which formalizes the statement concerning the arguments used both in clause 2 and in clause 4, which in its turn relies on the definition of HORPO.

HORPO is used to show well-foundedness of the algebraic rewrite rules of an AFS. Besides those rules, there is also the rule for β-reduction, and the rewrite relation we are interested in is the union of the algebraic rewrite

relation $\rightarrow_R$ and β-reduction $\rightarrow_\beta$. The well-foundedness proof is hence concerned with the union $\succ \cup \rightarrow_\beta$.

The main goal of the formalization so far is to prove well-foundedness of $\succ \cup \rightarrow_\beta$. It is not (yet) concerned with issues as proving that $\succ$ is transitive, or stable under application of substitutions and contexts. We do not consider those issues in this paper either.

3.1 Finite multisets

The definitions of HORPO and RPO use the extension of those relations to finite multisets. In the recursive call of the definitions, when the multiset extension is used, it is not yet known whether RPO and HORPO are orderings. As a matter of fact, many versions of HORPO are not transitive, and hence are not orderings. So the definitions of HORPO and RPO use the multiset extension of a relation which is not necessarily transitive. In this subsection we briefly comment on such multiset extensions, which form a substantial part of the formalization presented in [7].

We assume a set X and a relation $>$ on X. The intuition of a finite multiset over X is that it is a finite set with repeated elements, or a finite list where the order is irrelevant. More formally, a *finite multiset* over X is a mapping $m : X \rightarrow \mathbb{N}$ with $m(x) \neq 0$ for only finitely many elements of X. We denote a multiset m using $\{\{\ldots\}\}$ and repeating each element x exactly $m(x)$ times. For instance, $\{\{1, 1, 2\}\}$ is the multiset m over $\mathbb{N}$ with $m(1) = 2$, and $m(2) = 1$, and $m(n) = 0$ for all other n. The operations multiset union, denoted by $\cup$, element, denoted by $\in$, are defined as usual. Equality on multisets is denoted by $=$. The empty multiset is denoted by $\emptyset$.

In the formalization, finite multisets are first rendered as an abstract datatype. Then a representation using finite lists is given.

The definition of *multiset reduction*, denoted by $\Rightarrow$, is as follows: we have $m \Rightarrow n$ if

$$\begin{aligned} &m = m' \cup \{\{x\}\} \\ &n = m' \cup n' \\ &\forall y \in n' : x > y \end{aligned}$$

The multiset extension of $>$, denoted by $>_{mul}$, is defined as the transitive closure of $\Rightarrow$.

An alternative definition, here denoted by $>>$, of the multiset extension of $>$ is as follows: we have $m >> n$ if

$$\begin{aligned} &m = p \cup m' \text{ with } m' \neq \emptyset \\ &n = p \cup n' \\ &\forall y \in n' \, \exists x \in m' : x > y \end{aligned}$$

The two definitions can be proved to be equivalent if the underlying relation $>$ is transitive. This is done in the formalization. If the underlying relation $>$ is not transitive, then the two definitions are not necessarily the same.

Consider for instance the relation $>$ defined by $a > b$ and $b > c$, but where $a > c$ does not hold. We have $\{\{a\}\} \Rightarrow \{\{b\}\}$ and $\{\{b\}\} \Rightarrow \{\{c\}\}$ and hence $\{\{a\}\} >_{mul} \{\{c\}\}$. However, $\{\{a\}\} >> \{\{c\}\}$ does not hold.

Not all versions of HORPO are transitive. Therefore in the development transitivity is not assumed. Hence it is relevant to know which definition of the multiset extension is used: it is the first one, where the multiset extension is defined as the transitive closure of the multiset reduction. The version of HORPO used in the formalization is transitive; however, this is issue is not considered in the formalization.

If $>$ is not transitive, then from $m >_{mul} n$ we cannot conclude that for every $y \in n$ there is a $x \in m$ such that $x > y$. See for instance the example given above, where we have $\{\{a\}\} >_{mul} \{\{c\}\}$, but not $a > c$. We do have the weaker property that from $m >_{mul} n$ we can conclude that for every $y \in n$ there is a $x \in m$ such that $x >^* y$, with $>^*$ the transitive closure of $>$. This property is used in the proof of well-foundedness.

The formalization contains several properties of finite multisets. The most important one, which is used to justify the well-founded induction in the proof of the key lemma for the well-foundedness of $\succ \cup \rightarrow_\beta$, is the following: if all elements of a finite multiset m are well-founded with respect to $>$, then m is well-founded with respect to $>_{mul}$. Also: if $>$ is well-founded on X, then $>_{mul}$ is well-founded on the set of finite multisets over X. The proof that is formalized proceeds mainly by well-founded induction and is due to Buchholz [10]. It is already formalized in Isabelle and HOL.

4 Well-foundedness of HORPO

In this section we present the main steps of the proof of well-foundedness of $\succ \cup \rightarrow_\beta$ due to Jouannaud and Rubio [4]. The proof makes use of the computability predicate due to Tait and Girard with respect to the relation $\succ \cup \rightarrow_\beta$. Some standard properties of computability are used. In the formalization, they are assumed as hypotheses.

The specialization of the well-foundedness proof to the case of first-order RPO is also independently given by Persson [11]. There the proof is obtained from the classical proof using a minimal bad sequence argument by means of the principle of open induction due to Raoult [13]. The outline of the classical proof (for the first-order case, considering only $\succ$) is as follows: Suppose that it is not the case that all $\succ$-sequences are finite. Then there is an infinite $\succ$-sequence. Then there is an infinite *minimal* sequence in the sense that for every step $M \succ N$ in the infinite minimal sequence any alternative $M \succ N'$ with N' a subterm of N would yield only finite sequences. The constructive part of the classical proof consists of showing by well-founded induction that all minimal sequences are finite. Then from the contradiction we conclude that all $\succ$-sequences are finite. Actually it is not necessary

to consider the subterm relation in the definition of minimal sequence; any well-founded ordering works.

We continue by presenting the key lemma in the well-foundedness proof. We use the notion of computability with respect to $\succ \cup \to_\beta$, and some properties of computability which are standard and not mentioned here explicitly. For the first-order case, a similar lemma is proved. The difference in the statement is that it is concerned with well-founded, not computable, terms. The main difference in the proof is that it deals with fewer cases.

Lemma 4.1. Suppose that $M_1, \ldots, M_k$ are computable for $k \geq 0$. Then $f(M_1, \ldots, M_k)$ is computable.

Proof. The proof proceeds by well-founded induction on the pair $(f, \{\{M_1, \ldots, M_k\}\})$, lexicographically ordered by the precedence on function symbols, and the multiset extension of $\succ \cup \to_\beta$ to finite multisets of computable (and hence well-founded with respect to $\succ \cup \to_\beta$) elements. Because both components are well-founded, also the lexicographic product is well-founded. We denote this ordering in the proof by $>$.

We proceed by showing that all successors of $f(M_1, \ldots, M_k)$ with respect to $\succ \cup \to_\beta$ are computable. Because $f(M_1, \ldots, M_k)$ is neutral, this yields that it is computable itself. We suppose $f(M_1, \ldots, M_k) \succ \cup \to_\beta N$.

0. Suppose that N is obtained by a β-reduction step in one of the M_i: $M_i \to_\beta M_i'$. Then $N = f(\ldots, M_i', \ldots)$ and $(f, \{\{\ldots, M_i, \ldots\}\}) > (f, \{\{\ldots, M_i', \ldots\}\})$. By the induction hypothesis, N is computable.

1. Suppose that N is obtained by an application of clause 1 of HORPO. Then $M_i \succeq N$ for some $M_i \in \{M_1, \ldots, M_k\}$. Since M_i is by assumption computable, by a computability property also its successor N is computable.

2. Suppose that N is obtained by an application of clause 2. Then $N = g(N_1, \ldots, N_l)$ with the conditions of clause 2. The idea is to apply the induction hypothesis using $(f, \{\{M_1, \ldots, M_k\}\}) > (g, \{\{N_1, \ldots, N_l\}\})$. Therefore we need to show that all N_i are computable with respect to $\succ \cup \to_\beta$. For N_i with the same type as M, we have $M \succ N_i$. Then computability of N_i follows by adding an inner induction hypothesis on the size of N. For N_i with another type than M, we have $M_j \succeq N_i$ for some j, and then computability of N_i follows by a computability property, since a successor of a computable term is computable.

3. Suppose that N is obtained by an application of clause 3. Then $N = f(N_1, \ldots, N_k)$ with $\{\{M_1, \ldots, M_k\}\} \succ_{mul} \{\{N_1, \ldots, N_k\}\}$. The idea is to apply the induction hypothesis using $(f, \{\{M_1, \ldots, M_k\}\}) > (f, \{\{N_1, \ldots, N_k\}\})$. Therefore we need to show that all N_i are computable. For every N_i there are two possibilities. Either we have

$N_i = M_j$ for some j. In that case N_i is computable by assumption. Or we have $M_j \succ^* N_i$. In that case computability of N_i follows from the computability property stating that the successor of a computable term is computable.

4. Suppose that N is obtained by an application of clause 4. Then $N = @(N_1, \ldots, N_l)$. We need to show that all N_i are computable. This is done is the same way as for case 2.

We conclude that all successors with respect to $\succ \cup \rightarrow_\beta$ of M are computable. Because M is neutral this yields that M is computable. $\diamond$

The formalization of the key lemma is quite hard. Now the main theorem to be formalized is the following.

Theorem 4.3. Let M be a term and let γ be a computable substitution. Then M^γ is computable.

Proof. The proof proceeds by induction on the size of M.

1. If M is a variable, so $M = x$, then $M^\gamma = \gamma(x)$ and hence by assumption computable.

2. If M is a function symbol, so $M = f$, then $M^\gamma = f$ which is computable by Lemma 4.1.

3. If M is an abstraction, so $M = \lambda x : A.\, M_0$, then it is sufficient to show that $M_0^\gamma\{x \mapsto N\}$ is computable for any computable N. If N is computable, then $\delta = \gamma \cup \{x \mapsto N\}$ is also computable. By the induction hypothesis, M_0^δ is computable. This yields that M^γ is computable.

4. If M is an application, so $M = @(M_1, M_2)$, then by induction M_1^γ and M_2^γ are both computable. This yields that M^γ is computable.

$\diamond$

In the formalization, once the key lemma is there, the abstraction clause is the most difficult. For the moment a hypothesis is added stating that if a substitution is computable, then also its lifting is computable. From this result, computability and hence well-foundedness with respect to $\succ \cup \rightarrow_\beta$ follows by using it with the identity substitution.

5 Concluding remarks

There are several obvious possibilities for extending the formalization: consider properties of HORPO such as transitivity, consider also the computability properties (this would be quite some work), and consider a variant of HORPO that applies to rewriting modulo β as in HRSs.

References

[1] S. Berghofer. A constructive proof of Higman's lemma in Isabelle. In S. Berardi, M. Coppo, and F. Damiani, editors, *Proceedings of the International Workshop Types for Proofs and Programs (TYPES 2003)*, volume 3085 of *LNCS*, pages 66–82. Springer, 2004.

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

[3] J.-P. Jouannaud and M. Okada. A computation model for executable higher-order algebraic specification languages. In *Proceedings of the 6th annual IEEE Symposium on Logic in Computer Science (LICS '91)*, pages 350–361, Amsterdam, The Netherlands, July 1991.

[4] J.-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *Proceedings of the 14th annual IEEE Symposium on Logic in Computer Science (LICS '99)*, pages 402–411, Trento, Italy, July 1999.

[5] J.-P. Jouannaud and A. Rubio. Higher-order recursive path orderings 'à la carte'. `http://www.lix.polytechnique.fr/Labo/Jean-Pierre.Jouannaud/biblio.html`, 2003.

[6] N. de Kleijn. Well-foundedness of RPO in Coq. Master's thesis, Vrije Universiteit, Amsterdam, The Netherlands, August 2003.

[7] A. Koprowski. Well-foundedness of the higher-order recursive path ordering in Coq. Master's thesis, Vrije Universiteit, Amsterdam, The Netherlands, August 2004. To appear.

[8] F. Leclerc. Termination proof of term rewriting systems with the multiset path ordering: A complete development in the system Coq. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Proceedings of the 2nd International Conference on Typed Lambda Calculi and Applications (TLCA'95)*, volume 902 of *LNCS*, pages 312–327, Edinburgh, UK, April 1995. Springer.

[9] C. Murthy. *Extracting constructive content from classical proofs.* PhD thesis, Cornell University, New York, USA, 1990.

[10] T. Nipkow. An inductive proof of the wellfoundedness of the mulitset order. `http://www4.informatik.tu-muenchen.de/~nipkow/misc/index.html`, October 1998. A proof due to W. Buchholz.

[11] H. Persson. *Type Theory and the Integrated Logic of Programs.* PhD thesis, Göteborg University, Göteborg, Sweden, May 1999.

[12] F. van Raamsdonk. On termination of higher-order rewriting. In A. Middeldorp, editor, *Proceedings of the 12th International Conference on Rewriting Techniques and Applications (RTA '01)*, pages 261–275, Utrecht, The Netherlands, May 2001.

[13] J.-C. Raoult. Proving open properties by induction. *Information Processing Letters*, 29:19–23, 1988.