Termination of order-sorted rewriting

Peter C. Ølveczky
Department of Informatics
University of Oslo
E-mail: peterol@ifi.uio.no

Abstract

We present a method for proving termination of order-sorted rewrite systems by transforming an order-sorted rewrite system into an unsorted one such that termination of the latter implies termination of the order-sorted system. The method is inspired by ideas of Gnaedig and Ganzinger and contains as special cases the method presented in [Gn92a] and the method that simply ignores sort information.

1 Introduction

Order-sorted specifications, i. e. many-sorted specifications where the set of sorts is partially ordered, has been introduced to provide for a more powerful type concept, allowing us to express partiality of functions, error handling and subtype inheritance [GM88]. In languages like OBJ3 [GW88], order-sorted specifications have an operational semantics based on *rewriting*.

One of the most important properties a rewrite system may have is *termination*, which means that no infinite computation can take place. Termination is especially important in completion procedures, and last but not least, proving termination of order-sorted rewrite systems ensures termination of programs written in languages like OBJ3.

In general it is undecidable whether a rewrite system terminates, but numerous techniques (e. g. based on the lexicographic path ordering [KL80, Der87]), have been developed that with varying degrees of success may be used to prove termination of *unsorted* rewrite systems. These techniques have been used to prove termination of order-sorted systems by simply ignoring sort information [CH93, Gan91, GKK90, Gn92b, Wal92]. However, this approach is not good enough since many order-sorted systems terminate where the corresponding unsorted system would not terminate. This is illustrated by the following example, taken from [Gn92a] where the problem of order-sorted termination was first addressed.

Example 1 This system defines a predicate is_even over the integers. The idea is to define it over the positive numbers first and to extend it to the negative numbers by using the operation opp.

sorts: $Zero$, $NzNeg$, Neg, $NzPos$, Pos, Int, $Bool$
$Zero \leq Neg \leq Int$
$Zero \leq Pos \leq Int$
$NzNeg \leq Neg$
$NzPos \leq Pos$
funcs: $0 : Zero$
$s : Pos \rightarrow NzPos$
$p : Neg \rightarrow NzNeg$
$true : Bool$
$false : Bool$
$is\_even : Int \rightarrow Bool$
$opp : NzNeg \rightarrow NzPos$
rules: $is\_even(0) \longrightarrow true$
$is\_even(s(0)) \longrightarrow false$
$(\forall x : Pos)\ is\_even(s(s(x))) \longrightarrow is\_even(x)$
$(\forall y : NzNeg)\ is\_even(y) \longrightarrow is\_even(opp(y))$
$opp(p(0)) \longrightarrow s(0)$
$(\forall y : NzNeg)\ opp(p(y)) \longrightarrow s(opp(y))$

The system is terminating, the normal form of any ground term $is\_even(t)$ being either $true$ or $false$. But if sorts were ignored, the system would not terminate since the fourth rule would generate the infinite rewrite sequence

$$is\_even(y) \longrightarrow is\_even(opp(y)) is\_even(opp(opp(y))) \longrightarrow \cdots$$

However, the system does terminate because $opp(y : NzNeg)$ is of sort $NzPos$, so y cannot be instantiated with $opp(t)$ for any term t.

In Ganzinger [Gan91], where an order-sorted rewrite system is transformed into a many-sorted equational rewrite system, each function declaration $f : w \rightarrow s$ induces a (many-sorted) function symbol $f_{w \rightarrow s}$. This idea was adopted by Gnaedig [Gn92a] where these labelled function symbols are used to prove termination of order-sorted rewrite systems where the signature is *minimal*, i. e. if $f : w \rightarrow s$ and $f : w' \rightarrow s'$ are two function declarations then w and w' must be incomparable with respect to $\leq$. Termination can then be proven if $l' \succ_{lpo} r'$ for every rule $l \longrightarrow r$, t' being t where each function symbol f is replaced with the "corresponding" symbol $f_{w \rightarrow s}$. Unfortunately, not many signatures are minimal, and even for those that are minimal, this approach is not very strong (e. g. this method cannot show termination of the above system even though the signature is minimal).

We will also pursue the idea of labelling each function symbol in a term, but for our purpose the best result is obtained when a function symbol is labelled with the smallest sorts of its arguments. In the previous example the fourth rule would be labelled

$$is\_even_{NzNeg}(y) \longrightarrow is\_even_{NzPos}(opp_{NzNeg}(y))$$

The symbols is_even_{NzNeg} and is_even_{NzPos} may then be treated as distinct symbols in e. g. the lexicographic path ordering making the inequality

$$is\_even_{NzNeg}(y) \succ_{lpo} is\_even_{NzPos}(opp_{NzNeg}(y))$$

valid with precedence $is\_even_{NzNeg} \succ_F is\_even_{NzPos}$ and $is\_even_{NzNeg} \succ_F opp_{NzNeg}$.

We will use such labelling to transform an order-sorted rewrite system $\mathcal{R}$ into an unsorted rewrite system R' such that $\mathcal{R}$ terminates whenever R' does. We can thus use unsorted termination orderings to prove that R', and hence $\mathcal{R}$ terminates. However, special features of termination orderings imply that the best result after applying such an ordering is not guaranteed when the transformation R' is the most accurate. We will keep this in mind and will therefore exemplify our method with the lexicographic path ordering.

This paper is an extended abstract of [Ølv94] where a more thorough investigation on the subject of order-sorted termination as well as the correctness proofs for claims in this paper can be found.

2 Basic notions

2.1 Order-sorted rewriting

We will only introduce the syntactic aspects of order-sorted algebra and refer to [GM88] and [Wal92] for semantics of order-sorted specifications. In this paper, notions and notations mainly follow [GM88] and [GKK90].

Order-sorted algebra is based on a subsort relation, a partial order $\leq$ on a set S of sorts. We write $s' < s$ when $s' \leq s$ and $s' \neq s$ and $\geq$ for the symmetric relation of $\leq$. The subsort relation $\leq$ can be extended to n-tuples of sorts (often denoted w) where $w = \langle s_1, \ldots, s_n \rangle \leq \langle s'_1, \ldots, s'_1 \rangle = w'$ iff $s_i \leq s'_i$ for $i = 1..n$ and $w < w'$ iff $w \leq w'$ and $w \neq w'$. We write $w \lessdot w'$ when $w < w'$ and w and w' only differs in ith position, i. e. $w = \langle s_1, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots, s_n \rangle$, $w' = \langle s_1, \ldots, s_{i-1}, s'_i, s_{i+1}, \ldots, s_n \rangle$ and $s_i < s'_i$.

A function declaration has the form $f : s_1, \ldots, s_n \rightarrow s$ where each s_i is a sort symbol. If $n = 0$ then f is called a constant and the declaration is written $f : s$.

An order-sorted signature $\langle S, \leq, \Sigma \rangle$, usually denoted just Σ, consists of a partially ordered sort set $\langle S, \leq \rangle$ and a set Σ of function declarations. In this paper, all signatures are assumed to be finite, i. e. to contain a finite number of sorts and declarations.

In the rest of this paper we assume that $\langle S, \leq, \Sigma \rangle$ is an order-sorted signature. Given a set $X = \{X_s \mid s \in S\}$ of disjoint sets, the set of terms of sort s, denoted $\mathcal{T}(\Sigma, X)_s$, is the least set such that $X_{s'} \subseteq \mathcal{T}(\Sigma, X)_s$ if $s' \leq s$, and $f(s_1, \ldots, s_n) \in \mathcal{T}(\Sigma, X)_s$ if there exists a declaration $f : s_1, \ldots, s_n \rightarrow s'$ such that t_i is a term of sort s_i for $i = 1..n$ and $s' \leq s$. The set $\mathcal{T}(\Sigma, X)$ is defined as $\mathcal{T}(\Sigma, X) = \{\mathcal{T}(\Sigma, X)_s \mid s \in S\}$ and an element of any set $\mathcal{T}(\Sigma, X)_s$ is called a (well-formed) term. Given a set X of variables that contains an infinite number of variables of each sort, we will assume that every term t in $\mathcal{T}(\Sigma, X)$ has a unique (and computable) least sort, denoted $LS(t)$. A ground term is a term that does not contain any variables, and we will assume that there exists a ground term of every sort s in the signature.

A term may be viewed as an ordered tree in the obvious way. A *position* in a term can be seen as a finite sequence of natural numbers, identifying a path from the root of the term. The node in the term where the path ends is the actual position. The subterm of t at position p, written $t|_p$, is the subterm of t which has its root symbol at position p. The result of replacing the subterm of t at position p by the term u is written $t[u]_p$. If p and q are positions we write $p.q$ to denote the concatenation of p and q.

A *substitution* $\sigma = \{\sigma_s : X_s \rightarrow \mathcal{T}(\Sigma, Y)_s \mid s \in S\}$ is a mapping from variables to terms. We shall use lowercase Greek letters to denote substitutions. The application of a substitution σ to a term t consists of simultaneously replacing the variables occurring in t by a term according to the mapping. The result of this application is written $t\sigma$.

A *specialization*, often written μ, is a substitution $\mu : X \rightarrow Y$ that maps a variable x of sort s to a variable x of sort s' less than or equal to s, i. e. $(x\!:\!s)\mu = x\!:\!s'$ where $s' \leq s$. In a finite signature, a finite set of variables has a finite set of specializations.

An order-sorted *rewrite rule* is a directed equation $(\forall X)\ \ l \longrightarrow r$ where l and r are terms in $\mathcal{T}(\Sigma, X)$. A major difference between unsorted and order-sorted rewriting lies in the fact that a term t cannot always be rewritten if the left-hand-side of a rule matches a subterm of t. The reason for this is either that rewriting could result in a non-well-formed term or because of the definition of congruence in order-sorted algebras where $f(a) \approx f(b)$ may not be valid even though $a \approx b$ and both $f(a)$ and $f(b)$ are well-formed terms. Therefore, given a *rewrite system*, i. e. a set of rewrite rules, $\mathcal{R}$, a term $t \in \mathcal{T}(\Sigma, X)$ *rewrites* to t' with a rule $(\forall Y)\ l \longrightarrow r$ in $\mathcal{R}$ at position p, denoted $t \xrightarrow[\mathcal{R}]{} t'$ if

1. there is a substitution $\sigma : Y \rightarrow \mathcal{T}(\Sigma, X)$ such that $l\sigma = t|_p$ and $t' = t[r\sigma]_p$ and
2. there is a sort s such that, for x a variable of sort s, $t[x]_p$ is a well-formed term and $l\sigma, r\sigma \in \mathcal{T}(\Sigma, X)_s$.

We write $t \xrightarrow[\mathcal{R}]{+} t'$ when t rewrites to t' in one or more steps. We will assume that every rule $l \longrightarrow r$ is *sort-decreasing*, i. e. $LS(l\sigma) \geq LS(r\sigma)$ for every substitution σ. Sort-decreasingness ensures 1) that condition 2 in the above definition of rewriting is satisfied when the first condition is fulfilled, 2) that the critical pair lemma is valid and 3) that Birkhoff's theorem is valid for confluent systems [GKK90]. Sort-decreasingness may however be too restrictive for many purposes; we will therefore indicate how the method presented can be extended to handle non-sort-decreasing systems as well.

A rewrite system $\mathcal{R}$ is *terminating* if there does not exist any infinite rewrite sequence

$$t_1 \xrightarrow[\mathcal{R}]{} t_2 \xrightarrow[\mathcal{R}]{} t_3 \xrightarrow[\mathcal{R}]{} \cdots$$

2.2 Termination of unsorted rewriting

Unsorted rewriting is the special case of order-sorted rewriting where the signature has only one sort. Since an (unsorted) rewrite system can model a Turing machine, termination of rewriting is in general undecidable (see for instance [Der87]). Many techniques, mostly so-called *reduction* or *termination orderings*, have been developed that sometimes may be used to prove termination of unsorted rewrite systems.

A *quasi-ordering* $\succsim$ on a set S is a reflexive and transitive relation on S. The symmetric relation to $\succsim$ is denoted $\precsim$. The strict part $\succ$ of $\succsim$ and the induced equivalence relation $\approx$ are defined $\succ = (\succsim \setminus \precsim)$ and $\approx = (\succsim \cap \precsim)$ respectively.

We will use the *lexicographic path ordering* to exemplify our method. Assume that F is a set of function symbols and X is a set of variables. Let $\succsim_F$ be a quasi-ordering on $F \cup X$ such that a variable (seen as a constant in the ordering) is incomparable in $\succ_F$ with any other symbol. The lexicographic ordering $\succ_{lpo}$ on the set of $T(F, X)$ of terms is then defined as follows:

$$
\begin{array}{lll}
 & \multicolumn{2}{c}{t = f(t_1, \ldots, t_m) \succ_{lpo} g(t'_1, \ldots, t'_n) = t'} \\
 & \multicolumn{2}{c}{\text{iff}} \\
\text{either} & & t_i \succ_{lpo} t' \quad \text{or} \quad t_i \approx_{lpo} t' \\
\text{or} & f \succ_F g & \text{and} \quad t \succ_{lpo} t'_i \text{ for all } i \in 1..n \\
\text{or} & f \approx_F g & \text{and} \quad \langle t_1, \ldots, t_m \rangle \succ_{lpo}^{lex} \langle t'_1, \ldots, t'_n \rangle \\
 & & \text{and} \quad t \succ_{lpo} t'_i \quad \text{for} \quad i \geq 2
\end{array}
$$

where $t \approx_{lpo} t'$ if t and t' are equal up to replacement of $\approx_F$ -equal symbols.

A finite rewrite system R is terminating if $l \succ_{lpo} r$ for each rule $l \longrightarrow r$ in R.

The reader is referred to [Der87] for an introduction to unsorted termination.

3 Termination of order-sorted rewriting

In this section we propose a technique for proving termination of an order-sorted rewrite system $\mathcal{R}$ by transforming it into an unsorted system R'.

Definition 1 *Let Σ be an order-sorted signature. Its* translation *into a corresponding unsorted signature (i. e. a set of function symbols) F is defined*

$$F = \{f_w \mid f : w' \rightarrow s \text{ declaration in } \Sigma \text{ for any } s \text{ and } w \leq w'\}$$

A term t in $\mathcal{T}(\Sigma, X)$ is represented in $T(F, X)$ by the term $\lfloor t \rfloor$ where each function symbol in t is labelled with the least sorts of its arguments:

Definition 2 *The mapping $\lfloor\ \rfloor : \mathcal{T}(\Sigma, X) \rightarrow T(F, X)$ where F is the translation of Σ is defined inductively as follows:*

- $\lfloor x : s \rfloor = x$ *for a variable* $x \in X_s$.
- $\lfloor f(t_1, \ldots, t_n) \rfloor = f_{LS(t_1),\ldots,LS(t_n)}(\lfloor t_1 \rfloor, \ldots, \lfloor t_n \rfloor)$

We will use $\lfloor\ \rfloor$ to make a *reflection* of an order-sorted rewrite system $\mathcal{R}$:

Definition 3 *An unsorted rewrite system R'* reflects *an order-sorted rewrite system $\mathcal{R}$ iff*

$$t \xrightarrow[\mathcal{R}]{} t' \Longrightarrow \lfloor t \rfloor \xrightarrow[R']{+} \lfloor t' \rfloor$$

for all terms $t, t' \in \mathcal{T}(\Sigma, X)$.

Clearly, $\mathcal{R}$ terminates whenever its reflection does.

Let us now define R'. Letting $R' = \{\lfloor l \rfloor \longrightarrow \lfloor r \rfloor \mid l \longrightarrow r \in \mathcal{R}\}$ is not sufficient, R' must at least contain a rule $\lfloor l\mu \rfloor \longrightarrow \lfloor r\mu \rfloor$ for every specialization μ of each rule $l \longrightarrow r$ in $\mathcal{R}$. Unfortunately, the following example shows that not even this definition is sufficient to make R' a reflection of an order-sorted system $\mathcal{R}$.

Example 2 Let a rewrite system $\mathcal{R}$ be given

sorts: s, s_1, s_2
$s_1 \leq s$
$s_2 \leq s$
funcs: $f : s \rightarrow s$
$g : s \rightarrow s$
$g : s_1 \rightarrow s_2$
$a : s$
$b : s_1$
rule: $a \longrightarrow b$

The term $f(g(a))$ rewrites to $f(g(b))$ in $\mathcal{R}$ but $\lfloor f(g(a)) \rfloor = f_s(g_s(a))$ and $\lfloor f(g(b)) \rfloor = f_{s_2}(g_{s_1}(b))$ so we need

$$f_s(g_s(a)) \xrightarrow[R']{+} f_{s_2}(g_{s_1}(b))$$

which is not true if R' only contains the rule $a \longrightarrow b$.

In the above example all the (unsorted) function symbols have changed even though the order-sorted rewriting occurred at position 1.1. We call such a "side effect" of a rewrite step a *function symbol jump* or just a *F-jump*:

Definition 4 *Assume that a term t is rewritten to t' in an order-sorted rewrite system and that the rule is applied at position p in t. There is a* function symbol jump*, often just called an F*-jump*, from f_w to $f_{w'}$ if there is a position q in t such that $\lfloor t \rfloor|_q = f_w(\ldots)$ and $\lfloor t' \rfloor|_q = f_{w'}(\ldots)$ and there is no position r such that $q = p.r$*

In the previous example we had two F-jumps, from f_s to f_{s_2} and from g_s to g_{s_1} respectively. Sort-decreasingness and monotonicity properties of order-sorted signatures imply that $w' \lessdot w$ when an F-jump from f_w to $f_{w'}$ occurs. To handle this, a so-called *extra rule*

$$f_w(x_1, \ldots, x_n) \longrightarrow f_{w'}(x_1, \ldots, x_n)$$

is added to R' for all symbols $f_w, f_{w'}$ in F when $w' \lessdot w$. In example 2, R' would have contained the extra rules $f_s(x) \longrightarrow f_{s_1}(x)$, $f_s(x) \longrightarrow f_{s_2}(x)$, $g_s(x) \longrightarrow g_{s_1}(x)$ and $g_s(x) \longrightarrow g_{s_2}(x)$. Together with a rule $a \longrightarrow b$, R' is a reflection of $\mathcal{R}$ where the rewrite sequence $f_s(g_s(a)) \xrightarrow[R']{} f_s(g_s(b)) \xrightarrow[R']{} f_s(g_{s_1}(b)) \xrightarrow[R']{} f_{s_2}(g_{s_1}(b))$ is the desired $\lfloor f(g(a)) \rfloor \xrightarrow[R']{+} \lfloor f(g(b)) \rfloor$.
A reflection R' can then be defined

$$\begin{aligned} R' \;=\; & \{ \lfloor l\mu \rfloor \longrightarrow \lfloor r\mu \rfloor \mid l \longrightarrow r \in \mathcal{R},\ \mu \text{ specialization } \} \bigcup \\ & \{ f_w(x_1, \ldots, x_n) \longrightarrow f_{w'}(x_1, \ldots, x_n) \mid f_w \in F, w' \lessdot w \} \end{aligned}$$

Using the lexicographic path ordering, the extra rules are decreasing iff $f_w \succ_F f_{w'}$ in the precedence when $w' \lessdot w$. The relation $\succ_F$ being transitive, this means that $f_w \succ_F f_{w'}$ for all w, w' where $w' < w$. However, we can allow a precedence $f_w \approx_F f_{w'}$ by replacing each function symbol f_w in R' with a canonical element of the class $[f_w]_{\approx_F}$ with respect to f and omitting the obvious extra rules.

The first version of our method is thus described by the following theorem:

Theorem 5 *A finite, sort-decreasing order-sorted rewrite system $\mathcal{R}$ terminates if*

$$\lfloor l\mu \rfloor \succ_{lpo} \lfloor r\mu \rfloor$$

for each rule $l \longrightarrow r$ in $\mathcal{R}$ and each specialization μ of the variables in the rule, provided that the precedence $\succsim_F$ satisfies

$$f_w \succsim_F f_{w'}$$

for all symbols $f_w, f_{w'} \in F$ where $w' < w$.

This method, which can be applied to other termination orderings in the same way, can be totally automated and contains as special cases the method published in [Gn92a] and the method that ignores sort information (by letting $f_w \approx_F f_{w'}$ in the precedence for all function symbols f in Σ and labels w and w'). The main disadvantages of the method are:

1. Loss in efficiency, even though investigating each specialization seems inherent to most problems in order-sorted rewriting.

2. R' contains a rule $f_w(x_1, \ldots, x_n) \longrightarrow f_{w'}(x_1, \ldots, x_n)$ whenever $w' \lessdot w$ even in systems where an F-jump from f_w to $f_{w'}$ cannot occur. If, for instance, all specializations of all rules are sort-preserving, *no* F-jump can take place, and therefore all restrictions on the precedence $\succsim_F$ are superfluous. This problem is addressed in section 4.2.

3. A possible F-jump form f_w to $f_{w'}$ can be handled in more elaborate ways than just adding a rule $f_w(x_1, \ldots, x_n) \longrightarrow f_{w'}(x_1, \ldots, x_n)$. This opportunity is dealt with in the next section.

4. The method requires sort-decreasing rules. In section 4.2 we will indicate how it can be extended to non-sort-decreasing systems as well.

An example

Let us see how the method can be applied to the following system, which cannot be shown to terminate by ignoring sort information:

Example 3 Let a rewrite system $\mathcal{R}$ be

sorts: s', s
$s' \leq s$
funcs: $f : s \to s$
$g : s \to s$
$a : s$
$b : s'$
rules: $(\forall x : s')\ f(x) \to g(x)$
$g(a) \to f(a)$

Since $s' < s$ the precedence $\succsim_F$ must satisfy $g_s \succsim_F g_{s'}$ and $f_s \succsim_F f_{s'}$ and

$$\begin{array}{rcl} \lfloor f(x : s') \rfloor = f_{s'}(x) & \succ_{lpo} & g_{s'}(x) = \lfloor g(x) \rfloor \\ \lfloor g(a) \rfloor = g_s(a) & \succ_{lpo} & f_s(a) = \lfloor f(a) \rfloor \end{array}$$

which are all satisfied with the precedence

$$g_s \succ_F f_s \succ_F f_{s'} \succ_F g_{s'}$$

4 Extensions of the method

4.1 A more elaborate treatment of possible F-jumps

A possible F-jump from f_w to $f_{w'}$ is handled with an extra rule $f_w(x_1, \ldots, x_n) \longrightarrow f_{w'}(x_1, \ldots, x_n)$ in R' (or by having $f_w \approx_F f_{w'}$), forcing the precedence to satisfy $f_w \succsim_F f_{w'}$. The following example shows a weakness of this approach:

Example 4 Let the signature Σ and the terminating rewrite system $\mathcal{R}$ be given

sorts: s', s
$s' \leq s$
funcs: $f : s \to s$
$g : s' \to s$
$h : s \to s$
$h : s' \to s'$
$a : s$
$b : s'$
rules: $(\forall x : s')\ f(x) \longrightarrow f(g(x))$
$a \longrightarrow b$

To have the desired inequality

$$\lfloor f(x:s') \rfloor = f_{s'}(x) \succ_{lpo} f_s(g_{s'}(x)) = \lfloor f(g(x)) \rfloor$$

we need $f_{s'} \succ_F f_s$, but our method requires $f_s \succsim_F f_{s'}$ since $s' \leq s$. Note also that an F-jump from f_s to $f_{s'}$ may occur because of the second rule.

We may have an F-jump from $f_{s_1,..,s_i,..,s_n}$ to $f_{s_1,..,s'_i,..,s_n}$ if f is "applied" to an instance $l\sigma \longrightarrow r\sigma$ of a rule where $LS(l\sigma) = s_i$ and $LS(r\sigma) = s'_i$ or if there is a function symbol g in Σ such that $LS(g(\ldots l\sigma \ldots)) = s_i$ and $LS(g(\ldots r\sigma \ldots)) = s'_i$. In example 4, we have F-jumps from f_s to $f_{s'}$ because f is "applied" to the second rule ($f(a) \xrightarrow[\mathcal{R}]{} f(b)$) and because h "changes" value sort (e. g. $f(h(a)) \xrightarrow[\mathcal{R}]{} f(h(b))$, $f(h(h(a))) \xrightarrow[\mathcal{R}]{} f(h(h(b)))$, ...).

An extra rule

$$f_{s_1,..,s_i,..,s_n}(x_1, \ldots, x_i, \ldots, x_n) \longrightarrow f_{s_1,..,s'_i,..,s_n}(x_1, \ldots, x_i, \ldots, x_n)$$

may therefore be replaced by a rule

$$f_{s_1,..,s_i,..,s_n}(x_1, \ldots, \lfloor l\mu \rfloor, \ldots, x_n) \longrightarrow f_{s_1,..,s'_i,..,s_n}(x_1, \ldots, \lfloor r\mu \rfloor, \ldots, x_n)$$

for every rule specialization $l\mu \longrightarrow r\mu$ where $LS(l\mu) = s_i$ and $LS(r\mu) = s'_i$ and with a rule

$$f_{s_1,..,s_i,..,s_n}(x_1, \ldots, g_w(y_1, \ldots, y_m), \ldots, x_n) \longrightarrow f_{s_1,..,s'_i,..,s_n}(x_1, \ldots, g_{w'}(y_1, \ldots, y_m), \ldots, x_n)$$

for each pair $g_w, g_{w'}$ of function symbols in F where $w' \lessdot w$, $LS(g(\overline{x} : w)) = s_i$ and $LS(g(\overline{y} : w')) = s'_i$.

Unfortunately, we do not have

$$f_{s_1,..,s_i,..,s_n}(x_1, \ldots, \lfloor l\mu \rfloor, \ldots, x_n) \succ_{lpo} f_{s_1,..,s'_i,..,s_n}(x_1, \ldots, \lfloor r\mu \rfloor, \ldots, x_n)$$

or

$$f_{s_1,..,s_i,..,s_n}(x_1, \ldots, g_w(y_1, \ldots, y_m), \ldots, x_n) \succ_{lpo} f_{s_1,..,s'_i,..,s_n}(x_1, \ldots, g_{w'}(y_1, \ldots, y_m), \ldots, x_n)$$

if the arity of f is greater than 1 and $f_{s_1,..,s_i,..,s_n} \not\succsim_F f_{s_1,..,s'_i,..,s_n}$. Using any path ordering, all function symbols that violate the restriction $w' < w \Longrightarrow f_w \succsim_F f_{w'}$ must be unary. Other orderings, such as the Knuth-Bendix-ordering [KB70], do not have this restriction. For the lexicographic path ordering, this method can be described by the following theorem:

Theorem 6 *Assume that Σ is a finite order-sorted signature and $\mathcal{R}$ a finite set of sort-decreasing rewrite rules. $\mathcal{R}$ terminates if the following conditions are satisfied:*

1. *For any non-unary function symbol* g *in* Σ

$$g_w \in F \wedge w' < w \Longrightarrow g_w \succsim_F g_{w'}$$

2. *For every rule* $l \longrightarrow r$ *in* $\mathcal{R}$ *and every specialization* μ

$$\lfloor l\mu \rfloor \succ_{lpo} \lfloor r\mu \rfloor$$

3. *For any pair* $f_s, f_{s'}$ *in* F *where* $s' < s$ *and* $f_s \not\succsim_F f_{s'}$ *we have*

   - $f_s(\lfloor l\mu \rfloor) \succ_{lpo} f_{s'}(\lfloor r\mu \rfloor)$, *that is* $\lfloor l\mu \rfloor \succ_{lpo} f_{s'}(\lfloor r\mu \rfloor)$, *for every rule specialization* $l\mu \longrightarrow r\mu$ *where* $LS(l\mu) = s$ *and* $LS(r\mu) = s'$.
   - $f_s(g_w(x_1, \ldots, x_n)) \succ_{lpo} f_{s'}(g_{w'}(x_1, \ldots, x_n))$, *that is* $g_w \succ_F f_{s'}$ *and* $g_w \succ_F g_{w'}$, *for each pair* $g_w, g_{w'}$ *in* F *where* $w' \lessdot w$, $LS(g(\overline{x} : w)) = s$ *and* $LS(g(\overline{y} : w')) = s'$.

Although the method becomes quite complicated with this version it can be fully automated and can be used to prove termination of the system in example 4.

Example 4 (cont.) We need $f_{s'} \succ_F f_s$ to obtain the inequality $f_{s'}(x) \succ_{lpo} f_s(g_s(x))$. Therefore, the inequalities $f_s(a) \succ_{lpo} f_{s'}(b)$ and $f_s(h_s(x)) \succ_{lpo} f_{s'}(h_{s'}(x))$ must also hold. The precedence

$$a \succsim_F h_s \succ_F f_{s'} \succ_F f_s \succsim_F g_s \succsim_F g_{s'} \succsim_F b \succsim_F h_{s'}$$

satisfies all the requirements of method 2 and hence termination of the system is established.

The process described above can then be repeated such that an extra rule

$$f_s(g_w(x)) \longrightarrow f_{s'}(g_{s'}(x))$$

in R' is replaced by a set of rules

$$f_s(g_w(\lfloor l\mu \rfloor)) \longrightarrow f_{s'}(g_{s'}(\lfloor r\mu \rfloor))$$

and a set of rules

$$f_s(g_w(h_{s_1}(x))) \longrightarrow f_{s'}(g_{s'}(h_{s_2}(x)))$$

for all necessary rule specializations and pairs of F-symbols. Then, the rule $f_s(g_w(h_{s_1}(x))) \longrightarrow f_{s'}(g_{s'}(h_{s_2}(x)))$ may be substituted in the same way, and this process can continue. However, the process may not terminate. Assume for instance that the system in example 4 is extended and that we need a precedence $h_{s'} \succ_F h_s$. Then, a (possible) extra rule $h_s(x) \longrightarrow h_{s'}(x)$ is replaced by $h_s(a) \longrightarrow h_{s'}(b)$ and $h_s(h_s(x)) \longrightarrow h_{s'}(h_{s'}(x))$. The latter is not decreasing in the ordering and is therefore replaced by $h_s(h_s(a)) \longrightarrow h_{s'}(h_{s'}(b))$ and $h_s(h_s(h_s(x))) \longrightarrow h_{s'}(h_{s'}(h_{s'}(x)))$ which is replaced by ... and so on.

It may happen (at least for the path orderings) that the number of iterations necessary to prove termination is bounded by the depth of the terms in the rules in $\mathcal{R}$.

4.2 Possible F-jumps

Until now, we have assumed (more or less) that an F-jump from $f_{s_1,..,s_i,..,s_n}$ to $f_{s_1,..,s'_i,..,s_n}$ can occur whenever $s'_i < s_i$ and have concentrated on how to handle such a possible jump. However, in some systems an F-jump from $f_{s_1,..,s_i,..,s_n}$ to $f_{s_1,..,s'_i,..,s_n}$ may not occur. Our method can be strengthened by computing the set of F-jumps that actually can take place. A more thorough analysis also has the advantage that the proposed method can be used on non-sort-decreasing systems as well, which is exemplified by the following system:

Example 5 Let a terminating rewrite system be given

sorts: s, s_1, s_2, s_3, s_4
$s_1 \leq s$
$s_2 \leq s$
$s_3 \leq s$
$s_4 \leq s$
funcs: $f : s \rightarrow s$
$g : s \rightarrow s$
$g : s_3 \rightarrow s_1$
$g : s_4 \rightarrow s_2$
$h : s_1 \rightarrow s_2$
$a : s_3$
$b : s_4$
rules: $a \longrightarrow b$
$(\forall x : s_1)\ f(x) \longrightarrow f(h(x))$

The second rule implies that an unsorted analysis cannot prove termination of the system which is non-sort-decreasing by the first rule. An analysis of the system gives the set of possible F-jumps as follows:

g_{s_3} to g_{s_4}	$(g(a) \xrightarrow[\mathcal{R}]{} g(b))$
g_{s_1} to g_{s_2}	$(g(g(a)) \xrightarrow[\mathcal{R}]{} g(g(b)))$
f_{s_3} to f_{s_4}	$(f(a) \xrightarrow[\mathcal{R}]{} f(b))$
f_{s_1} to f_{s_2}	$(f(g(a)) \xrightarrow[\mathcal{R}]{} f(g(b)))$

In a first approach, each possible F-jump from f_{w_1} to f_{w_2} imply a restriction $f_{w_1} \succsim_F f_{w_2}$ using the lexicographic path ordering. In this example we would have the restrictions $g_{s_3} \succsim_F g_{s_4}$, $g_{s_1} \succsim_F g_{s_2}$, $f_{s_3} \succsim_F f_{s_4}$, and $f_{s_1} \succsim_F f_{s_2}$ which combined with $f_{s_1} \succ_F f_{s_2}$, $f_{s_1} \succ_F h_{s_1}$ and $a \succ_F b$ proves termination of the system.

A complete analysis with respect to possible F-jumps is quite tedious but it *is* decidable whether a F-jump from f_{w_1} to f_{w_2} may take place. Any possible F-jump can then be handled in the way described in the previous section.

4.3 Other improvements

The method presented here can be improved in many ways, e. g. in finding more suitable orderings which tackle the specific problems with the method, making the method more efficient. Another improvement, also mentioned in [Gn92a], is to "preprocess" the rewrite system *before* applying the technique presented here. For instance, a rule $(\forall x : s)\ l \longrightarrow r$ can be replaced by two rules $(\forall x : s_1)\ l \longrightarrow r$ and $(\forall x : s_2)\ l \longrightarrow r$ provided that there is no ground term of sort s which is not a term of either sort s_1 or sort s_2.

5 Conclusion

The method presented is an extension of the works of Gnaedig [Gn92a] and Ganzinger [Gan91]. The latter is not concerned with termination when transforming an order-sorted system into a many-sorted one for completion purposes, but introduces the idea of labelling function symbols and adding equations/rules to handle F-jumps. Gnaedig [Gn92a] requires, in addition to sort-decreasingness, that the signatures are minimal. That approach has the advantage of being more efficient, but our method has been shown to be significantly stronger.

The technique presented is only one of many approaches to the problem of proving termination of order-sorted rewriting. A totally different idea we have been looking at, is to handle variables in better ways in the termination orderings, always achieving the crucial $t \succ t' \Longrightarrow t\sigma \succ t'\sigma$. In unsorted orderings, a variable is not comparable to any term not containing the same variable. This restriction can sometimes be loosened by letting a variable of sort s be comparable to some terms that are not of sort s. In [Ølv94] we propose an extension to the lexicographic path ordering based on this principle, thereby being able to prove termination of a system which cannot be shown to terminate with the method given in this paper.

Acknowledgements: I would like to thank Olav Lysne for comments on the full version [Ølv94] of this paper.

References

[CH93] H. Chen and J. Hsiang. *Order-Sorted Equational Specification and Completion.* Unpublished note, Department of Computer Science, State University of New York at Stony Brook, 1993.

[Der82] N. Dershowitz. *Orderings for term-rewriting systems.* Theoretical Computer Science 17(3):279-301, 1982.

[Der87] N. Dershowitz. *Termination of rewriting.* Journal of Symbolic Computation, 3:69-116, 1987.

[DJ90] N. Dershowitz and J.P. Jouannaud. *Rewrite systems.* Handbook of Theoretical Computer Science, chapter 15, North-Holland, 1990.

[Gan91] H. Ganzinger. *Order-sorted completion: the many-sorted way.* Theoretical Computer Science 89(1991) 3-32.

[GKK90] I. Gnaedig, C. Kirchner and H. Kirchner. *Equational Completion in Order-Sorted Algebras.* Theoretical Computer Science 72:169-202, 1990.

[GM88] J. A. Goguen and J. Meseguer. *Order-Sorted Algebra I.* Center for the Study of Language and Information, Stanford University, 1988.

[Gn92a] I. Gnaedig. *Termination of order-sorted rewriting.* Proc. Algebraic and Logic Programming. Third International Conference, Springer-Verlag, 1992.

[Gn92b] I. Gnaedig. *ELIOS-OBJ: theorem proving in a specification language.* Proc. ESOP '92, 4th European Symposion on Programming, Springer-Verlag, 1992.

[GW88] J. A. Goguen and T. Winkler. *Introducing OBJ3.* Tech. report, SRI International, Computer Science Lab., 1988.

[KB70] D. Knuth and P. Bendix. *Simple word problems in universal algebras.* J. Leech, editor, Computational Problems in Abstract Algebra, pages 263-297, Pergamon Press, Oxford, 1970.

[KL80] S. Kamin and J.-J. Lévy. *Two generalizations of the recursive path ordering.* Unpublished note, Dept. of Computer Science, University of Illinois, Urbana, IL.

[Wal92] U. Waldmann. *Semantics of order-sorted specifications.* Theoretical Computer Science 94(1992) 1-35.

[Ølv94] P. C. Ølveczky. *Terminering av typeordnet omskrivning.* Masters thesis, Dept. of Informatics, University of Oslo, 1994. In Norwegian.