Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 238 (2009) 207–225
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Operational Termination of Membership Equational Programs: the Order-Sorted Way

Salvador Lucas

DSIC, Universidad Politécnica de Valencia, Spain

José Meseguer

CS Dept., University of Illinois at Urbana-Champaign, USA

Abstract

Our main goal is automating termination proofs for programs in rewriting-based languages with features such as: (i) expressive type structures, (ii) conditional rules, (iii) matching modulo axioms, and (iv) context-sensitive rewriting. Specifically, we present a new operational termination method for membership equational programs with features (i)-(iv) that can be applied to programs in membership equational logic (MEL). The method first transforms a MEL program into a simpler, yet semantically equivalent, conditional *order-sorted* (OS) program. Subsequent trasformations make the OS-program unconditional, and, finally, unsorted. In particular, we extend and generalize to this richer setting an order-sorted termination technique for unconditional OS programs proposed by Ölveczky and Lysne. An important advantage of our method is that it minimizes the use of conditional rules and produces simpler transformed programs whose termination is often easier to prove automatically.

Keywords: Term rewriting, program analysis, operational termination, membership equational logic, order-sorted equational logic, rewriting logic.

1 Introduction

Our main goal is automating termination proofs for programs in rewriting-based languages with features such as: (i) expressive type structures, (ii) conditional rules, (iii) matching modulo axioms, and (iv) context-sensitive rewriting. As discussed and exemplified in [21,8] the execution of a declarative program, besides requiring in general the evaluation of *conditions*, may not involve rewriting at all, or may involve *both* rewriting and other computational relations. This is particularly relevant for *Membership Equational Logic* (MEL) [22] and the corresponding programs that, in general, need to evaluate both equations and memberships. For this reason, we use in this paper a proof-theoretic termination notion, called *operational termination* [21]. This notion is *parametric* on the logic: it can be defined not just for MEL, but for many other logics, that may or may not involve rewriting in their computations.

doi:10.1016/j.entcs.2009.05.021

In this setting, a program in a given *logic* is *operationally terminating* if all its well-formed proof trees are finite, see [21] and also Example 1.2 below.

Two expressive type structures shared by several rewriting-based languages are: (1) an *order-sorted* (OS) type structure, with sorts and subsorts; and (2) the type structure of MEL, which supports sorts, subsorts, and kinds, and where sort memberships can be conditional. In MEL the two basic types of atomic predicates are equalities $t = t'$, and memberships $t : s$ stating that a term t has sort s. The axioms of a MEL theory are then Horn clauses, whose head can be either an equation or a membership. There is a basic level of typing by *kinds*; and a more sophisticated one by *sorts*, which is achieved by deduction using theory axioms (the Horn clauses). Typing by sorts provides a general way to deal with *partiality*, in that a term having a kind but lacking a sort is regarded as an *undefined* or *error* element. For example, OBJ [15], CafeOBJ [11], and a subset of CASL [5] all support order-sorted specifications, while Maude [6] supports both OS and MEL specifications. The operational termination method in [21] applies to all these languages, and to programs with all the expressive features (i)-(iv) mentioned above. Since it is well-known that OS equational logic is a less expressive sublogic of MEL (see [22]), our method works by first transforming a MEL program into a semantically equivalent OS one. In a language like Maude, the way in which both OS and MEL programs are seamlessly supported is by keeping the intuitive order-sorted features intact as helpful *syntactic sugar*, and adding membership axioms only when they are strictly needed. These *sugared* MEL specifications make the first transformation simpler and easier to understand, and therefore we use them in this paper. Subsequent transformations eliminate conditions, and finally also sorts. However, in contrast to a previous transformational method of automating the operational termination of MEL programs proposed in [7,8], doing it *the order-sorted way* that we advocate here has the important advantage of minimizing the use of conditional rules, so that it produces simpler transformed programs that are often easier to prove automatically. We transform order-sorted programs into unsorted ones by extending and generalizing a method proposed by Ölveczky and Lysne [25].

To illustrate the challenges of automating termination proofs for expressive rewriting-based programs with features (i)-(iv) we use some Maude programs. This has the advantage of familiarizing the user with the *sugared* notation that makes the generalization from OS to MEL so seamless. We refer to [3, Section 3.3] and [22] for more theoretical studies that also use and justify this sugared notation.

Example 1.1 Consider the following Maude functional module [7]:

```
fmod LengthOfFiniteListsAndTake is
   sorts Nat NatList NatIList . subsort NatList < NatIList .
   op 0 : -> Nat .
   op s : Nat -> Nat .
   op zeros : -> NatIList .
   op nil : -> NatList .
   op cons : Nat NatIList -> NatIList [strat (1 0)] .
   op cons : Nat NatList -> NatList [strat (1 0)] .
   op take : Nat NatIList -> NatList .
   op length : NatList -> Nat .
   vars M N : Nat .
   var  IL : NatIList .
```

```
        var  L : NatList .
        eq zeros = cons(0,zeros) .
        eq take(0, IL) = nil .
        eq take(s(M), cons(N, IL)) = cons(N, take(M, IL)) .
        eq length(nil) = 0 .
        eq length(cons(N, L)) = s(length(L)) .
    endfm
```

where sorts NatList and NatIList are intended to classify finite and infinite lists of natural numbers, respectively. The function zeros generates an infinite list of zeros, and take can be used to obtain an initial (finite) segment of a (possibly infinite) list by giving the number of items we want to extract. Finally, length computes the length of a *finite* list. Note the *overloaded* operator cons, which can be used both for building finite and infinite lists of natural numbers and is declared with evaluation *strategy*[1] (1 0). The interpretation of this strategy annotation is as follows: the evaluation of an expression cons(h,t) proceeds by first evaluating h and then trying a reduction step at the top position (represented by 0). No evaluation is allowed on the second argument t because index 2 is missing in the annotation. Note also that NatList is a subsort of NatIList, thus allowing the use of take to extract items both from finite and infinite lists.

Operationally, and assuming good executability properties such as the *Church-Rosser property* and *admissibility*, equalities $t = t'$ can be treated as rewrite rules $t \longrightarrow t'$ [3]. Rewriting with equations as rules can furthermore be made *context-sensitive* by providing a replacement map μ that indicates which argument positions of a function symbol f must be reduced before equations for f are applied [18,19]. In this way we arrive at the notion of a *Context-Sensitive Membership Rewrite Theory* (CS-MRT), which is the operational form of a membership equational program [8]. In [7,8], (operational) termination of Maude functional modules as the one in Example 1.1 is proved by considering the underlying CS-MRT. For instance, Figure 1 shows the CS-MRT which corresponds to the module in Example 1.1. Note that the "order-sorted" syntactic sugar has been eliminated. This desugaring means that: (i) subsort declarations and operator declarations are desugared into conditional memberships; for example, the subsort declaration NatList < NatIList and the operator declaration op s : Nat -> Nat become, respectively, the conditional memberships L : NatIList if L : NatList and s(N) : Nat if N : Nat; and (ii) all quantification by sorted variables becomes an explicit condition that the kinded variable has that sort. For example, the equation take(0, IL) = nil becomes the conditional equation take(0,IL) = nil if IL : NatIList. The faithfull embedding of order-sorted logic into membership equational logic proved correct in [22] *is* precisely this desugaring. However, it is in some ways more intuitive to keep the order-sorted notation *whenever possible*, only using explicit memberships when a given axiom does not have an equivalent order-sorted formulation. For instance, in Example 1.2 (see below) the axiom s(N) : Inf if s(s(N)) : Inf cannot be expressed in order-sorted logic and is explicitly stated as such. The transformations

[1] Actually, the final 0 could be removed from the strategy annotation for cons because no rule applies on top of terms having cons as root symbol. However, since zero-ended strategy annotations are usually assumed/required in OBJ/Maude programs (see, e.g., [10]), we keep it in our example.

```
fmod LengthOfFiniteListsAndTake is
   kind [Truth] .
   kind [Nat].
   kind [NatIList] .
   op tt : -> [Truth] .
   op 0 : -> [Nat] .
   op s : [Nat] -> [Nat] .
   op zeros : -> [NatIList] .
   op nil : -> [NatList] .
   op cons : [Nat] [NatIList] -> [NatIList] [strat (1 0)] .
   op take : [Nat] [NatIList] -> [NatIList] .
   op length : [NatIList] -> [Nat] .
   vars M N : Nat .
   var  IL : NatIList .
   var  L : NatList .
   cmb L : NatIList if L : NatList .
   mb tt : Truth .
   mb 0 : Nat .
   cmb s(N) : Nat if N : Nat .
   mb zeros : NatIList .
   mb nil : NatList .
   cmb cons(N,IL) : NatIList if N : Nat /\ IL : NatIList .
   cmb cons(N,L) : NatList if N : Nat /\ L : NatList .
   cmb take(N,IL) : NatList if N : Nat /\ IL : NatIList .
   cmb length(L) : Nat if L : NatList .
   eq zeros = cons(0,zeros) .
   ceq take(0,IL) = nil if IL : NatIList .
   ceq take(s(M),cons(N,IL)) = cons(N,take(M,IL)) if M : Nat /\ N : Nat /\ IL : NatIList .
   eq length(nil) = 0 .
   ceq length(cons(N,L)) = s(length(L)) if N : Nat /\ L : NatList .
endfm
```

Fig. 1. CS-MRT for the program LengthOfFiniteListsAndTake

described in [7,8] allow us to prove (operational) termination of the original module as termination of a *Context-Sensitive Term Rewriting System* (CS-TRS), i.e., a TRS together with some replacement restrictions associated to the symbols in the signature, see [18,19]. The obtained CS-TRS can be proved terminating by using existing termination tools like AProVE [12] or MU-TERM [1,20] which are able to deal with such kind of termination problems. The whole proof process (transformations and calls to the external tools) can be managed by using the Maude Termination Tool (MTT [9]). Termination of the Maude program in Example 1.1 turns out to be difficult to prove in that way. The main problem is that (as exemplified above) most *sort information* is managed in MEL/CS-MRT by means of *conditional* equations including memberships in the conditions. In this way, virtually all programs are translated into conditional rewrite systems having many conditional rules with quite big conditional parts. Rules of this kind are harder to manage in termination proofs.

In contrast, the program in Example 1.1, viewed as an ordinary order-sorted specification, can easily be proved terminating by using a *context-sensitive* version of Ölveczky and Lysne's transformation [25] *without* introducing any conditional rule (see Example 6.2 below). This suggests that sugared MEL/CS-MRT specifications (see Section 3), where the 'order-sorted' components (in the sense of [3, Definition 6]) remain untouched and memberships are only used when a semantically equivalent order-sorted formulation is impossible, can provide a more effective way of dealing with termination of MEL programs. Of course, as illustrated by the following example (from [8]), some MEL programs, while still using OS syntactic sugar, may have essential MEL features that cannot be sugared away.

Example 1.2 The following Maude module

```
fmod INF is
  sorts Nat Inf .
  subsort Inf < Nat .
  op 0 : -> Nat .
  op s : Nat -> Nat .
  var N : Nat .
  cmb s(N) : Inf if s(s(N)) : Inf .
endfm
```

provides an interesting example of a nonterminating program involving no rewrite rule (borrowed from [8, Introduction]). Here, a conditional membership establishes that terms `s(N)` (for terms `N` of sort `Nat`) have sort `Inf` provided that `s(s(N))` has sort `Inf` too. Note that no rewriting step is involved here. However, the nontermination of the `INF` program is witnessed by the infinite proof tree,

$$\cfrac{\cfrac{\cfrac{\cdots}{\texttt{s(s(s(N))):Inf}}}{\texttt{s(s(N)):Inf}}}{\texttt{s(N):Inf}}$$

(Figure 3 shows the inference system for computations with CS-MRT programs)

In order to cover arbitrary MEL programs, in Section 4 we show how to appropriately transform arbitrary CS-MRTs into Order-Sorted Context-Sensitive Rewrite Theories (OS-CS-RTs). First, we deal with the conditional part of OS-CS-RTs. In Section 5, we extend Ohlebusch's transformation from CTRSs into TRSs [24] to transform an OS-CS-RT into an OS-CS-TRS. Then, we slightly generalize Ölveczky and Lysne's transformation from OS-TRSs into TRSs to deal with context-sensitivity information. In Section 6 we adapt their transformation to deal with OS-CS-TRSs and yield a CS-TRS whose termination can be proved by using existing tools. Section 7 discusses the possible use of concepts and results coming from the area of *many-sorted rewriting* to improve our proofs of termination. Section 8 concludes.

2 Preliminaries

We summarize here material from [14,22,17] on order-sorted rewriting. An *order-sorted equational specification* is a 4-tuple $(\Sigma, S, \leq, E)$ with $(\Sigma, S, \leq)$ an *order-sorted signature*, and E a set of (possibly conditional) Σ-equations. An order-sorted signature $(\Sigma, S, \leq)$ consists of a partially ordered set $(S, \leq)$ of *sorts*, where $s \leq s'$ is interpreted as *subsort inclusion*, and with Σ a $S^* \times S$-indexed family of sets $\Sigma = \{\Sigma_{w,s}\}_{(w,s)\in S^*\times S}$, which are *function symbols* with given string of argument sorts and result sort. The *connected components* of $(S, \leq)$ are the equivalence classes corresponding to the least equivalence relation containing $\leq$. If $f \in \Sigma_{s_1\ldots s_n,s}$, then we display the function symbol f as $f : s_1 \cdots s_n \longrightarrow s$. Some of these symbols f can be *subsort-overloaded*. For example, we can have a subsort inclusion $Nat \leq Int$ and two subsort-overloaded declarations $+ : Nat\ Nat \longrightarrow Nat$, and $+ : Int\ Int \longrightarrow Int$. By an order-sorted substitution we mean an S-indexed substitution σ such that for all sort $s \in S$ and all variable $x \in \mathcal{X}_s$, the sort s' of $\sigma(x)$ satisfies $s' \leq s$.

A simple syntactic condition on $(\Sigma, S, \leq)$ called *preregularity* [14] ensures that each term t has always a *least-sort* $LS(t)$ possible among all sorts in S.

The possibly conditional equations E can sometimes be decomposed as a disjoint union $Ax \uplus E'$, with Ax a set of axioms such as associativity, and/or commutativity, (Reflex) $$\frac{}{t \to^* t'} \quad \text{if } t =_{Ax} t'$$

(Trans) $$\frac{t \to^1 t' \qquad t' \to^* t''}{t \to^* t''}$$

(Congr) $$\frac{u_i \to^1 u_i'}{f(u_1, \ldots, u_i, \ldots, u_n) \to^1 f(u_1, \ldots, u_i', \ldots, u_n)} \quad \text{where } f \in \Sigma \text{ and } i \in \mu(f)$$

(Replacement) $$\frac{\sigma(u_1) \to^* \sigma(v_1) \quad \ldots \quad \sigma(u_n) \to^* \sigma(v_n)}{u \to^1 \sigma(t')}$$
where $t \to t'$ if $u_1 \to v_1 \cdots u_n \to v_n \in \mathcal{R}$, σ is an OS-substitution, and $u =_{Ax} \sigma(t)$

Fig. 2. Inference rules for order-sorted conditional rewriting

and/or identity for which an Ax-matching algorithm exists, and a set E' of equations that can be *oriented* as conditional rewrite rules R, that are applied modulo Ax. Furthermore, as in [4,8], we also consider a *replacement map* [18], i.e., a function $\mu : \Sigma \longrightarrow \mathcal{P}_{fin}(\mathbf{N})$ associating to each operator f of n arguments a set of argument positions $\mu(f) = \{i_1, \ldots, i_m\}$, with $1 \leq i_j \leq n$, which are those under which rewriting is allowed. This gives rise to the notion of an *order-sorted context-sensitive rewrite theory (OS-CS-RT)* as a 6-tuple $(\Sigma, S, \leq, \mu, Ax, R)$. Figure 2 gives a detailed inference system for conditional order-sorted rewriting, including also the case of contextual rewriting. Here we just point to the earlier work [17] on order-sorted rewriting as an operational semantics for order-sorted equational languages. In the following, we will use the notion of *sort-decreasingness*. A conditional rewrite rule $t \longrightarrow t'$ *if cond* is called *sort-decreasing* (resp. *sort-preserving*) if for each substitution θ such that $\theta(cond)$ holds, the least sort of $\theta(t')$ is smaller or equal than (resp. equal to) the least sort of $\theta(t)$: $LS(\theta(t)) \geq LS(\theta(t'))$ (resp. $LS(\theta(t)) = LS(\theta(t'))$). For a simple way to check these properties in the unconditional case see [17]. The conditional case has been studied in [3] in a more general MEL version.

3 Sugared context-sensitive membership rewrite theories

By a *sugared* context-sensitive membership rewrite theory (SCS-MRT) we understand a tuple $\mathfrak{T} = (\Sigma, S, \leq, \mu, Ax, R, M)$ where

(i) S is a set of *sorts* and $(S, \leq)$ is a partial oder.

(ii) $\Sigma = \Sigma_0 \uplus \Sigma_1$, where

  (a) $(\Sigma_0, S, \leq)$ is an order-sorted signature that we assume *preregular* modulo Ax.

  (b) Σ_1 is a collection of operator declarations of the form $f : [s_1] \cdots [s_n] \to [s]$, where $[s_1], \ldots, [s_n], [s] \in K(S, \leq)$ and we let $K(S, \leq) = S/{\equiv_{\leq}} \uplus \{[Truth]\}$ (we assume that $Truth \notin S$). Here, $\equiv_{\leq}$ is the smallest equivalence relation containing the order $\leq$. Hence, $K(S, \leq)$ contains a new kind for each connected component in $(S, \leq)$ plus a new kind $[Truth]$.

Roughly speaking, Σ_0 contains the symbols which are given an explicit sort in the SCS-MRT specification, whereas Σ_1 contains symbols that do not admit a profile based only on 'proper' sorts but rather require the use of *kinds* (corresponding to the connected components in $(S, \leq)$ as a whole). Such use of kinds is typically needed for functions that are *intrinsically partial.* For example, given a sort `Path` of paths in a graph, a binary path concatenation function has to be declared as the kind level as $\_;\_$: `[Path] [Path] -> [Path]`, because it is intrinsically partial on pairs of paths: it is undefined unless the target node of the first path coincides with the source node of the second path.

(iii) $\mu : \Sigma \to \mathcal{P}_{fin}(\mathbb{N})$ is a mapping sending each $f : s_1 \cdots s_n \to s$ in Σ_0 to a subset $\mu(f) \subseteq \{1, \ldots, n\}$, and likewise each $f : [s_1] \cdots [s_n] \to [s]$ in Σ_1 to a subset $\mu(f) \subseteq \{1, \ldots, n\}$ and such that if f is *subsort overloaded* in Σ_0, or there exists $f : s_1 \cdots s_n \to s$ in Σ_0 and $f : [s_1] \cdots [s_n] \to [s]$ in Σ_1, then $\mu(f)$ is the *same* for all such subsort overloaded versions of f.

(iv) Ax is a collection of axioms such as associativity, commutativity, and identity such that *all* variables in such axioms are of the form $x : [s]$, for $[s] \in K(S, \leq)$.

(v) R is a set of *conditional rewrite rules* of the form

$$\forall x_1 : s_1, \ldots, x_n : s_n, x_{n+1} : [s_{n+1}], \ldots, x_{n+m} : [s_{n+m}], t \to t' \text{ if } A_1 \wedge \ldots \wedge A_k \quad (\dagger)$$

with $s_1, \ldots, s_n \in S$ and $[s_{n+1}], \ldots, [s_{n+m}] \in K(S, \leq)$, and where the A_i are either rewrite conditions $u \to v$, or memberships $w : s'$. As usual, we view unconditional rules as a special case of conditional rule with empty condition.

(vi) M is a set of *conditional memberships* of the form

$$\forall x_1 : s_1, \ldots, x_n : s_n, x_{n+1} : [s_{n+1}], \ldots, x_{n+m} : [s_{n+m}], t : s \text{ if } A_1 \wedge \ldots \wedge A_k \quad (\dagger\dagger)$$

with $s_1, \ldots, s_n \in S$ and $[s_{n+1}], \ldots, [s_{n+m}] \in K(S, \leq)$, and the A_i as before. Again, unconditional memberships are a special case of conditional ones.

Note that an SCS-MRT can be *desugared* into a CS-MRT by:

(i) Taking $K(S, \leq)$ as the set of kinds.

(ii) Taking as signature $\Sigma_1 \cup \{f : [s_1] \cdots [s_n] \to [s] \mid f : s_1 \cdots s_n \to s \in \Sigma_0\}$.

(iii) Taking S as the set of sorts.

(iv) Transforming each $s \leq s'$ in $\leq$ into a membership axiom $x : s'$ if $x : s$.

(v) Defining μ exactly as before, but for operators now lifted to the kinds.

(vi) Adding for each $f : s_1 \cdots s_n \to s$ in Σ_0 a conditional membership

$$\forall x_1 : [s_1], \ldots, x_n : [s_n], f(x_1, \ldots, x_n) : s \text{ if } x_1 :: s_1 \wedge \cdots \wedge x_n :: s_n$$

where $t :: s$ is a subrelation of the relation $t : s$, corresponding to the special case of a membership in which the term t is not further rewritten before computing its sort (see Figure 3 and [8]).

(vii) Transforming each conditional rule $(\dagger)$ into a rule

$$\forall x_1 : [s_1], \ldots, x_{n+m} : [s_{n+m}], \quad t \to t' \text{ if } \bigwedge_{1 \leq \ell \leq n} x_\ell :: s_\ell \wedge A_1 \wedge \ldots \wedge A_k$$

(Subject reduction) $$\frac{t \to^1 t' \qquad t' : s}{t : s}$$

(Membership-1) $$\frac{A_1^\bullet\sigma \quad \cdots \quad A_n^\bullet\sigma}{u :: s}$$
where $t : s$ if $A_1 \cdots A_n$ in R and $u =_{Ax} t\sigma$

(Membership-2) $$\frac{t :: s}{t : s}$$

(Reflexivity) $$\frac{}{t \to^* t'} \qquad \text{if } t =_{Ax} t'$$

(Transitivity) $$\frac{t \to^1 t' \qquad t' \to^* t''}{t \to^* t''}$$

(Congruence) $$\frac{u_i \to^1 u_i'}{f(u_1, \ldots, u_i, \ldots u_n) \to^1 f(u_1, \ldots, u_i', \ldots, u_n)}$$
where $i \in \mu(f)$

(Replacement) $$\frac{A_1^\bullet\sigma \quad \ldots \quad A_n^\bullet\sigma}{u \to^1 t'\sigma}$$
where $t \to t'$ if $A_1 \cdots A_n$ in R and $u =_{Ax} t\sigma$

Fig. 3. Inference rules for context-sensitive membership rewriting

(viii) Transforming each conditional membership (††) into a membership

$$\forall x_1 : [s_1], \ldots, x_{n+m} : [s_{n+m}], \qquad t : s \;\text{ if }\; \textstyle\bigwedge_{1 \leq \ell \leq n} x_\ell :: s_\ell \;\wedge\; A_1 \;\wedge\; \ldots \;\wedge\; A_k$$

Note that each Maude specification is given as a *sugared* MEL theory, which, from the operational point of view, is understood as a SCS-MRT. The above desugaring into a CS-MRT is the operational analogue of the already-mentioned semantics-preserving embedding from OS logic to MEL logic defined in [22].

We can define the computations associated to a CS-MRT by means of the inference rules of Figure 3, where $A_i^\bullet = A_i$ whenever A_i is a membership $w : s$ or $x :: s$, and $A_i^\bullet$ is $u \to^* v$ if A_i is a rewrite condition $u \to v$. Note that inferences can now happen *modulo* the equational axioms Ax in the theory: matching with a conditional equation in the Replacement inference rule, and with a conditional membership in the Membership-1 rule, is performed *modulo* Ax; and Reflexivity also includes equality modulo[2] Ax.

4 Transforming SCS-MRTs into OS-CS-RTs

Given an SCS-MRT $\mathfrak{T} = (\Sigma, S, \leq, \mu, Ax, R, M)$, we first define the set $MB_{\mathfrak{T}}(S) \subseteq S$ (or just $MB(S)$ if $\mathfrak{T}$ is clear from the context) of its *membership sorts* as the smallest subset of S such that

(i) if a membership $\forall x_1 : s_1, \ldots, x_n : s_n, t : s$ if C belongs to M, then $s \in MB(S)$.

(ii) if $s \in MB(S)$ and $s \leq s'$, then $s' \in MB(S)$.

[2] Strictly speaking, the (Congruence) rule should be generalized, as done for (Replacement), to allow one-step rewrites from any term u such that $u =_{Ax} f(u_1, \ldots, u_i, \ldots, u_n)$. However, this extra generality can be avoided, see [8].

(iii) if $f : s_1 \cdots s_n \to s \in \Sigma_0$ and $s_i \in MB(S)$ for some i, $1 \leq i \leq n$, then $s \in MB(S)$.

Intuitively, the set of *membership sorts* contains those sorts s which are defined by either explicitly using memberships, or, indirecly, being supersorts of a membership sort, or having symbols of sort s whose arity involves membership sorts. Note that $MB(S) = \emptyset$ whenever $M = \emptyset$. We also define $OS(S) = S - MB(S)$. Intuitively, $OS(S)$ is the set of sorts *not affected by any intrinsic membership axiom*, so that *order-sorted computation* with the rules in Figure 2 (which uses implictly order-sorted parsing as the only inference system for membership in a sort) is *complete* for inferring membership in a sort in $OS(S)$. Instead, because of the presence of an intrinsic membership in either the given sort, or a subsort, or an argument sort of an operator, inferring membership in a sort in $MB(S)$ cannot be done by order-sorted inference *directly*. It can however be done *indirectly*, in the transformed theory $\mathfrak{T}^\bullet$ defined below, by an order-sorted encoding of membership in a sort in $MB(S)$ as the truth of an equationally-defined predicate. Note that the partition $S = OS(S) \uplus MB(S)$ induces a partition of the rules in $\mathcal{R}$ in two disjoint sets: $R = OS(R) \uplus MB(R)$, where $OS(R) \subseteq R$ is the set of conditional rules of the form (†) such that $s_1, \ldots, s_n \in OS(S)$, whereas $MB(R)$ is its complement, that is, those rules of the form (†) such that there exists $i, 1 \leq i \leq n$, with $s_i \in MB(S)$.

The theory transformation $\mathfrak{T} \mapsto \mathfrak{T}^\bullet$ we are looking for sends the SCS-MRT $\mathfrak{T}$ to the order-sorted context-sensitive rewrite theory (OS-CS-RT)

$$\mathfrak{T}^\bullet = (\Sigma^\bullet, S^\bullet, \leq^\bullet, \mu^\bullet, Ax, OS(R) \cup MB(R)^\bullet \cup M^\bullet \cup MB(\Sigma, S)^\bullet)$$

where

(i) $S^\bullet = OS(S) \uplus K(S, \leq)$.

(ii) $\leq^\bullet$ is the reflexive-transitive closure of the following relation
  (a) For all $s, s' \in OS(S), s \leq^\bullet s'$ iff $s \leq s'$.
  (b) For all $s \in OS(S), s \leq^\bullet [s]$.

(iii) $\Sigma^\bullet$ is the following set of operations:

$$\begin{aligned}\Sigma^\bullet = \; & \{f : \boldsymbol{w} \to s \in \Sigma_0 \mid ws \in OS(S)^+\} \cup \{f : [\boldsymbol{w}] \to [s] \mid f : \boldsymbol{w} \to s \in \Sigma_0\} \\ & \cup \{is_s : [s] \to [Truth] \mid s \in MB(S)\} \cup \{is'_s : [s] \to [Truth] \mid s \in S\} \\ & \cup \{tt : \to [Truth]\}\end{aligned}$$

(iv) $\mu^\bullet$ agrees with μ on the first two sets of operators above: maps $\mu(f : [\boldsymbol{w}] \to [s])$ to $\mu(f : \boldsymbol{w} \to s)$ and maps $\mu(is_s) = \emptyset$ and $\mu(is'_s) = \{1\}$. This choice for the replacement restrictions associated to these symbols is consistent with the intended use of is_s and is'_s: as discussed in [8, Section 4], predicates is_s deal with sort declarations for variables like $x : s$ where x is a variable and *no reduction* below is_s in an instance of $is_s(x)$ is required to check the membership (hence $\mu(is_s) = \emptyset$). On the other hand, predicates is'_s are intended to deal with sort conditions $is'_s(w)$ coming from 'proper' memberships $w : s$, where w is a nonvariable term (or s is a *membership sort*) and some *subject reduction* could be necessary to check the membership (thus $\mu(is'_s) = \{1\}$).

(v) The set $MB(R)^\bullet$ contains, for each rule of the form (†) in $MB(R)$ a corresponding rule of the form

$$\begin{array}{l}\forall x_1 : s_1, \ldots, x_p : s_p, x_{p+1} : [s_{p+1}], \ldots, x_n : [s_n], \ldots, x_{n+m} : [s_{n+m}], \\ \quad t \to t' \text{ if } \bigwedge_{p<\ell\leq n} is_{s_\ell}(x_\ell) \to tt \wedge A_1^\flat \wedge \ldots \wedge A_k^\flat\end{array}$$

where, without loss of generality, we assume that $s_1, \ldots, s_p \in OS(S)$, and $s_{p+1}, \ldots, s_n \in MB(S)$, and where if A_i is a rewrite condition $u \to v$, then $A_i^\flat = A_i$, and if A_i is a membership $w : s'$, then $A_i^\flat = is'_{s'}(w) \to tt$.

(vi) The set $M^\bullet$ contains, for each conditional membership of the form (††) in M, a conditional rule of the form

$$\begin{array}{l}\forall x_1 : s_1, \ldots, x_p : s_p, x_{p+1} : [s_{p+1}], \ldots, x_n : [s_n], \ldots, x_{n+m} : [s_{n+m}], \\ \quad is_s(t) \to tt \text{ if } \bigwedge_{p<\ell\leq n} is_{s_\ell}(x_\ell) \to tt \wedge A_1^\flat \wedge \ldots \wedge A_k^\flat\end{array}$$

where, again, $s_1, \ldots, s_p \in OS(S)$, and $s_{p+1}, \ldots, s_n \in MB(S)$, and the $A_i^\flat$ are as before.

(vii) Finally, $MB(\Sigma, S)^\bullet$ consists of the following additional rules:

(a) For each $s \in MB(S)$, we add a rule $\forall x : [s], is'_s(x) \to is_s(x)$.

(b) For each $s \in OS(S)$, we add a rule $\forall x : s, is'_s(x) \to tt$. Note that this rule is needed because memberships $w_j : s'_j$ with $s'_j \in OS(S)$, could appear in the conditional part of either a rule (†) or a membership (††). So, we need to express the membership predicate $w_j : s'_j$ as the truth condition $is'_{s'_j}(w_j) \to tt$. Exactly for this reason, we have defined the predicates is'_s for any $s \in S$. Instead, predicates is_s are *only defined for* $s \in MB(S)$. They are not needed for $s \in OS(S)$ because we have instead the rule $\forall x : s, is'_s(x) \to tt$ so that the sort checking is in this case performed by the order-sorted type structure.

(c) For each $s \in OS(S)$, $s' \in MB(S)$ with $s \leq s'$, we add rules

$$\forall x : s, is'_{s'}(x) \to tt \qquad \forall x : s, is_{s'}(x) \to tt$$

(d) For each constant symbol c of sort s, we add a rule $is_s(c) \to tt$.

(e) For each $f : s_1 \cdots s_n \to s \in \Sigma_0$ such that the sorts among the $s_1, \ldots, s_n$ in $MB(S)$ are $s_{i_1}, \ldots, s_{i_k}$ with $k \geq 1$, we add the conditional rule:

$$\begin{array}{l}\forall x_1 : s_1, \ldots, x_{i_1} : [s_{i_1}], \ldots, x_{i_k} : [s_{i_k}], \ldots, x_n : s_n, \\ \quad is_s(f(x_1, \ldots, x_n)) \to tt \text{ if } is_{s_{i_1}}(x_{i_1}) \to tt \wedge \cdots \wedge is_{s_{i_k}}(x_{i_k}) \to tt\end{array}$$

(f) For each $s, s' \in MB(S)$ such that $s \leq s'$, we add the rule

$$\forall x : [s'], is_{s'}(x) \to tt \text{ if } is_s(x) \to tt$$

Note that the previous transformation becomes quite simple when $MB(S)$ is empty. Then, no symbol is_s is necessary (see item (iii) above) and, hence, no rule (or condition) involving them is introduced; in particular, $MB(\mathcal{R}) = \emptyset$ (hence $MB(\mathcal{R})^\bullet = \emptyset$).

The following theorem expresses the main property of this transformation.

Theorem 4.1 *Let $\mathfrak{T}$ be an SCS-MRT, $\mathfrak{T}_0$ be its* unsugared *version (as a CS-MRT), and $\mathfrak{T}^\bullet$ be the corresponding transformed OS-CS-RT. Then, for all ground terms $t, t' \in \mathcal{T}(\Sigma)$ of the same kind and all sorts s of the kind, we have*

(i) $\mathfrak{T}_0 \vdash t \to^* t'$ *if and only if* $\mathfrak{T}^\bullet \vdash t \to^* t'$

(ii) $\mathfrak{T}_0 \vdash t \to^1 t'$ *if and only if* $\mathfrak{T}^\bullet \vdash t \to^1 t'$

(iii) $\mathfrak{T}_0 \vdash t : s$ *if and only if* $\mathfrak{T}^\bullet \vdash is'_s(t) \to^* tt$

(iv) *(for $s \in MB(S)$)* $\mathfrak{T}_0 \vdash t :: s$ *if and only if* $\mathfrak{T}^\bullet \vdash is_s(t) \to^* tt$

The following theorem is an easy consequence of Theorem 4.1 above. It connects operational ground termination (i.e., operational termination of ground terms) in the CS-MRT logic, given by the inference rules in Figure 3, and operational termination in the OS-CS-RT logic, whose inference system is showed in Figure 2.

Theorem 4.2 *An SCS-MRT $\mathfrak{T}$ is operationally ground terminating if and only if the OS-CS-RT $\mathfrak{T}^\bullet$ is operationally ground terminating.*

The restriction to ground terms is harmless by assuming (see [3]), that *all sorts are nonempty* (i.e., for each sort s there is a ground term of sort s). In this case, we can always instantiate a nonground nonterminating sequence into a ground one.

4.1 Examples

For the module in Example 1.1, $OS(S) = \{$`Nat`, `NatList`, `NatIList`$\}$ and $MB(S) = \emptyset$. The connected components in $(S, \leq)$ are `Nat` and `NatList` $<$ `NatIList`. Thus, $K(S, \leq) = \{$`[Nat]`, `[NatIList]`, `[Truth]`$\}$. The resulting OS-CS-RT module is:

```
fmod ExLengthFListsTake-OS is
  sorts Nat NatList NatIList [Nat] [NatIList] [Truth] .
  subsort NatList < NatIList < [NatIList] .
  subsort Nat < [Nat] .
  op tt : -> [Truth] .
  op 0 : -> Nat . op 0 : -> [Nat] .
  op s : Nat -> Nat .   op s : [Nat] -> [Nat] .
  op zeros : -> NatIList . op zeros : -> [NatIList] .
  op nil : -> NatList . op nil : -> [NatIList] .
  op cons : [Nat] [NatIList] -> [NatIList] [strat (1 0)] .
  op cons : Nat NatIList -> NatIList [strat (1 0)] .
  op cons : Nat NatList -> NatList [strat (1 0)] .
  op take : [Nat] [NatIList] -> [NatList] .
  op take : Nat NatIList -> NatList .
  op length : [NatList] -> [Nat] .
  op length : NatList -> Nat .
  op is'-Nat : [Nat] -> [Truth] .
  op is'-NatList : [NatIList] -> [Truth] .
  op is'-NatIList : [NatIList] -> [Truth] .
  vars M N : Nat .
  var IL : NatIList .
  var L : NatList .
  eq zeros = cons(0,zeros) .
  eq take(0,IL) = nil .
  eq take(s(M),cons(N,IL)) = cons(N,take(M,IL)) .
```

```
    eq length(nil) = 0 .
    eq length(cons(N,L)) = s(length(L)) .
    eq is'-Nat(N) = tt .
    eq is'-NatList(L) = tt .
    eq is'-NatIList(IL) = tt .
  endfm
```

For the Maude functional module in Example 1.2, we have $MB(S) = S = \{\texttt{Inf}, \texttt{Nat}\}$ and $OS(S) = \emptyset$. The only connected component in $(S, \leq)$ is $\texttt{Inf} < \texttt{Nat}$. Thus, $K(S, \leq) = \{\,\texttt{[Nat]}, \texttt{[Truth]}\,\}$. Note that *no order among these sorts remains in the transformed system.* The resulting OS-CS-RT module is:

```
  fmod INF-OS is
    sorts [Nat] [Truth] .
    op tt : -> [Truth] .
    op 0 : -> [Nat] .
    op s : [Nat] -> [Nat] .
    op is'-Inf : [Nat] -> [Truth] .
    op is'-Nat : [Nat] -> [Truth] .
    op is-Inf : [Nat] -> [Truth]  [strat (0)] .
    op is-Nat : [Nat] -> [Truth] [strat (0)] .
    var N : [Nat] .
    ceq is-Inf(s(N)) = tt if is-Nat(N) = tt /\ is'-Inf(s(s(N))) = tt .
    eq is'-Inf(N) = is-Inf(N) .
    eq is'-Nat(N) = is-Nat(N) .
    eq is-Nat(0) = tt .
    ceq is-Nat(s(N)) = tt if is-Nat(N) = tt .
    ceq is-Nat(N) = tt if is-Inf(N) = tt .
  endfm
```

5 Removing conditions from order-sorted specifications

In order to check operational termination with respect to OS-CS-RT logic, we propose a transformation associating an unconditional OS-CS-TRS $\mathcal{U}(\mathcal{R}) = (\mathcal{U}(\Sigma), S, \leq, \mathcal{U}(\mu), Ax, \mathcal{U}(R))$ to an OS-CS-RT $\mathcal{R} = (\Sigma, S, \leq, \mu, Ax, R)$. In [7,8], the classical transformation for proving operational termination of a *deterministic* 3-*CTRS*[3] $\mathcal{R}$ as termination of a TRS $\mathcal{U}(\mathcal{R})$ (see [24, Definition 7.2.48]), was generalized to handle rewriting modulo axioms Ax and the context-sensitive restrictions imposed by the replacement map μ. In our approach, we additionally deal with order-sorted rules. In particular, Maude functional modules are required to be *admissible* ([6, Chapter 4.6]); admissibility generalizes to MEL/SCS-MRT modules the notion of determinism for 3-CTRSs. Furthermore, it is not difficult to see that the transformation in Section 4 maps admissible modules into deterministic OS-CS-RTs (more precisely, the 'underlying' 3-CTRSs are deterministic). The adaptation of the transformation in [7,8] is simple. The new signature $\mathcal{U}(\Sigma)$ consists of the symbols in Σ together with new symbols U as described below. Regarding the rules, given an order-sorted conditional rule

$$\forall x_1 : s_1, \ldots, x_m : s_m, l \to r \ \text{ if } \ u_1 \to v_1, \ldots, u_n \to v_n$$

we obtain $n + 1$ (sort-decreasing) unconditional rules

[3] A 3-CTRS only contains rules $l \to r$ if c such that $\mathrm{Var}(r) \subseteq \mathrm{Var}(l) \cup \mathrm{Var}(c)$. A 3-CTRS $\mathcal{R}$ is called *deterministic* if for each $l \to r$ if $s_1 \to t_1, \ldots, s_n \to t_n$ in $\mathcal{R}$ and each $1 \leq i \leq n$, we have $\mathrm{Var}(s_i) \subseteq \mathrm{Var}(l) \cup \bigcup_{j=1}^{i-1} \mathrm{Var}(t_j)$, see [24].

$$\forall x_1 : s_1, \ldots, x_m : s_m, l \to U_1(u_1, \boldsymbol{x_1}) \tag{1}$$

$$\forall x_1 : s_1, \ldots, x_m : s_m, U_{i-1}(v_{i-1}, \boldsymbol{x_{i-1}}) \to U_i(u_i, \boldsymbol{x_i}) \qquad 2 \leq i \leq n \tag{2}$$

$$\forall x_1 : s_1, \ldots, x_m : s_m, U_n(v_n, \boldsymbol{x_n}) \to r \tag{3}$$

where the $\boldsymbol{x_i}$ are vectors of variables defined as follows: assume a given ordering on the set of variables $\mathcal{X}$. Then, $\boldsymbol{x_i}$ contains the ordered sequence of the variables in the set $\mathrm{Var}(l) \cup \mathrm{Var}(v_1) \cup \cdots \cup \mathrm{Var}(v_{i-1})$ for $1 \leq i \leq n$, which, by determinism, ensures that in the above rules each right-hand side variable occurs in the left-hand side; or, in a clever way so as to avoid keeping track of unused variables:

$$\begin{aligned} \boldsymbol{x_i} = & (\mathrm{Var}(l) \cup \mathrm{Var}(v_1) \cup \cdots \cup \mathrm{Var}(v_{i-1})) \\ & \cap \quad (\mathrm{Var}(v_i) \cup \mathrm{Var}(u_{i+1}) \cup \mathrm{Var}(v_{i+1}) \cup \cdots \\ & \qquad \cup \quad \mathrm{Var}(u_n) \cup \mathrm{Var}(v_n) \cup \mathrm{Var}(r)) \end{aligned}$$

Let $\boldsymbol{s_i}$ be the sorts corresponding to variables in $\boldsymbol{x_i}$. Now, we just need to set that the fresh symbols added to the signature are as follows: $U_i : [LS(u_i)] \times \boldsymbol{s_i} \to LS(l)$. Rewriting modulo Ax is allowed. The replacement map is transformed into a new replacement map $\mathcal{U}(\mu)$ as follows: $\mathcal{U}(\mu)(U) = \{1\}$ for all new symbols U that are introduced to deal with the equations in the conditional part of each rule in $\mathcal{R}$ (that is, only the first argument of U can be evaluated), and $\mathcal{U}(\mu)(f) = \mu(f)$ for all symbols $f \in \mathcal{F}$.

Theorem 5.1 *The OS-CS-RT $\mathcal{R}$ is terminating if the OS-CS-TRS $\mathcal{U}(\mathcal{R})$ is terminating.*

6 Termination of order-sorted rewriting

Termination of an OS-TRS $\mathcal{R}$ is obviously guaranteed if the *underlying* TRS $\Theta(\mathcal{R})$ (i.e., the TRS which is obtained after removing all sort information from the signature and the rules) is terminating. This is often called the *trivial* transformation for proving termination of OS-TRSs. For many years, this was the only way to deal with termination of OS-TRSs. As far as the authors know, the first work envisaging nontrivial methods for proving termination of order-sorted rewriting is [13]. In her paper, Gnaedig proposes an extension of the lexicographic path ordering which can prove termination of OS-TRSs whose 'underlying' TRS is nonterminating. In [25], Ölveczky and Lysne introduce a transformation from OS-TRSs into ordinary TRSs which can be used to prove termination of OS-TRSs. Ölveczky and Lysne showed that their transformation strictly subsumes Gnaedig's technique [25].

Ölveczky and Lysne's transformation is as follows: an order sorted signature Σ is translated into an unsorted signature $\mathcal{F} = \{f_{\boldsymbol{s}} \mid f : \boldsymbol{s'} \to s \in \Sigma$ for some s and $\boldsymbol{s} \leq \boldsymbol{s'}\}$ where the arity of $f_{\boldsymbol{s}}$ is $|\boldsymbol{s}|$ and $\boldsymbol{s} \leq \boldsymbol{s'}$ means that $s_i \leq s'_i$ for each i, $1 \leq i \leq |\boldsymbol{s}|$. Now, many sorted terms $t \in \mathcal{T}(\Sigma, \mathcal{X})$ are also disambiguated by using a transformation $\lfloor\ \rfloor : \mathcal{T}(\Sigma, \mathcal{X}) \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ given by [25, Definition 3]:

$$\lfloor x : s \rfloor = x \text{ for a variable } x \in \mathcal{X}_s$$

$$\lfloor f(t_1, \ldots, t_n) \rfloor = f_{LS(t_1) \cdots LS(t_n)}(\lfloor t_1 \rfloor, \ldots, \lfloor t_n \rfloor)$$

where $LS(t)$ is the *least sort* associated to the term t. Now, the OS-TRS $\mathcal{R}$ is transformed into a TRS $\lfloor\mathcal{R}\rfloor$ which consists of the signature $\mathcal{F}$ plus the following rules [25, Theorem 2]:

$$\{\lfloor\nu(l)\rfloor \to \lfloor\nu(r)\rfloor \mid l \to r \in \mathcal{R},\ \nu \text{ a specialization}\} \cup$$
$$\{f_{\boldsymbol{s}}(x_1,\ldots,x_n) \to f_{\boldsymbol{s}'}(x_1,\ldots,x_n) \mid f_{\boldsymbol{s}} \in \mathcal{F}, \boldsymbol{s}' \lessdot \boldsymbol{s}\}$$

Here, a specialization is a substitution ν which maps a variable of sort s into *another variable* x' of sort s' such that $s' \leq s$. We assume that the set S of sorts is finite and that specializations which are equivalent up to renaming are not used, so that we cannot obtain an infinite TRS from a finite OS-TRS due to the use of infinitely many specializations. Also, $\boldsymbol{s}' \lessdot \boldsymbol{s}$ means that $\boldsymbol{s} \leq \boldsymbol{s}'$ and there is i, $1 \leq i \leq n$ such that $s'_i < s_i$. According to [25, Theorem 2], this transformation is correct for *sort-decreasing* OS-TRSs. Furthermore, as showed by Ölveczky and Lysne, the second set of rules can be avoided when *sort-preserving* OS-TRSs are considered.

It is not difficult to see that Ölveczky and Lysne's transformation could also be used to prove termination of OS-CS-TRSs (i.e., OS-TRSs supplied with a replacement map μ for the many sorted function symbols $f \in \Sigma$). Provided that $\mu(f : \boldsymbol{s} \to s) = \mu(f : \boldsymbol{s} \to s')$ holds for all symbols f, we let the transformed replacement map $\lfloor\mu\rfloor$ be $\lfloor\mu\rfloor(f_{\boldsymbol{s}}) = \mu(f : \boldsymbol{s} \to s)$. We say that a TRS $\mathcal{R}$ is μ-terminating (or that the CS-TRS $(\mathcal{R},\mu)$ is terminating) if the context-sensitive rewrite relation associated to $\mathcal{R}$ and μ (written $\hookrightarrow_{\mathcal{R},\mu}$, see [18,19]) is terminating. The following result easily follows from the proof of [25, Theorem 2]:

Theorem 6.1 *The OS-CS-TRS $\mathcal{R}$ is terminating if $\lfloor\mathcal{R}\rfloor$ is $\lfloor\mu\rfloor$-terminating.*

Example 6.2 When the transformation in Section 4 is applied to the program in Example 1.1, we obtain the OS-CS-RT `LengthOfFiniteListsAndTake-OS` in Section 4.1. After applying the (context-sensitive) version of Ölveczky and Lysne's transformation to this OS-CS-TRS $\mathcal{R}$, we obtain a CS-TRS $\lfloor\mathcal{R}\rfloor$ given by:

```
zeros -> cons-N-IL(0,zeros)
take-N-IL(0,L) -> nil
take-N-FL(0,L) -> nil
take-N-IL(s-N(M),cons-N-IL(N,L)) -> cons-N-FL(N,take-N-IL(M,L))
take-N-FL(s-N(M),cons-N-FL(N,L)) -> cons-N-FL(N,take-N-FL(M,L))
length-FL(nil) -> 0
length-FL(cons-N-FL(N,L)) -> s-N(length-FL(L))
is'-Nat-N(N) -> tt
is'-NatList-FL(L) -> tt
is'-NatIList-IL(L) -> tt
is'-NatIList-FL(L) -> tt
```

(where the names of the symbols have been conveniently given suffixes expressing the appropriate sorts, according to the definition of $\lfloor\ \rfloor$) and the new replacement map is $\mu($`cons-N-IL`$) = \mu($`consN-FL`$) = \{1\}$ and $\mu(f) = \{1,\ldots,k\}$ for all other k-ary symbols $f \in \Sigma$. Since all rules in the program in Example 1.1 are *sort-preserving* we do not need the second set of rules of Ölveczky and Lysne's transformation. Termination of this CS-TRS can be easily proved by AProVE or mu-term.

Remark 6.3 Note that symbols `tt`, `is-Nat-N`, `is-NatList-FL`, `is-NatIList-IL`, and `is-NatIList-FL` do not occur in the first seven rules $\mathcal{S}$ of the system in Example

6.2, i.e., $\lfloor\mathcal{R}\rfloor$ is the *disjoint* union of $\mathcal{S}$ plus the last four rules $\mathcal{S}'$: $\lfloor\mathcal{R}\rfloor = \mathcal{S} \uplus \mathcal{S}'$ (see [16] for results about modularity of termination of CS-TRSs). The CS-TRS $\mathcal{S}'$ is easily seen to be terminating, and it is both *noncollapsing* and *nonduplicating*, by [16, Theorem 3], we can concentrate the attention in proving termination of $\mathcal{S}$. This kind of 'modular reasoning' applies to the CS-TRS obtained from any SCS-MRT having no conditional memberships by using the transformations in this paper.

7 Termination of many-sorted TRSs

The role of sorts in termination of rewriting has also been considered in the *many-sorted* setting. Although Many-Sorted Term Rewriting Systems (MS-TRSs) are a particular case of OS-TRSs, some interesting results have been formulated for MS-TRSs only. Furthermore, it is not difficult to see that Ölveczky and Lysne's transformation behaves as the *trivial* transformation Θ when applied to an MS-TRS:

Proposition 7.1 *For any MS-TRS $\mathcal{R}$, $\lfloor\mathcal{R}\rfloor$ terminates if and only if $\Theta(\mathcal{R})$ terminates.*

Example 7.2 The following many-sorted module, which corresponds to the famous Toyama's example, is borrowed from [26, Section 3.3]

```
fmod Toyama-OS is
  sorts S1 S2 .
  op 0 : -> S1 .
  op 1 : -> S1 .
  op f : S1 S1 S1 -> S1 .
  op g : S2 S2 -> S2 .
  vars x : S1 .
  vars y z : S2 .
  eq f(0,1,x) = f(x,x,x) .
  eq g(y,z) = y .
  eq g(y,z) = z .
endfm
```

As noticed by Zantema, this OS-TRS is terminating. However, according to Proposition 7.1, $\lfloor\mathcal{R}\rfloor = \Theta(\mathcal{R})$, which is well-known to be nonterminating.

Since MS-TRSs are OS-TRSs, Example 7.2 also shows that Ölveczky and Lysne's transformation is *incomplete* regarding proofs of termination of OS-TRSs.

Proposition 7.1 suggests that making use of some proper results for proving termination of MS-TRSs can be useful for our purposes. Termination of MS-TRSs was investigated by Zantema [26] as an auxiliary technique for proving termination of (unsorted) rewriting (although he mentioned no systematic method for proving termination of MS-TRSs). Zantema defined the *persistency* of a computational property of many-sorted TRSs as a property which is *not* affected by the removal of sort information from the MS-TRS. Of course, termination is a persistent property of *one-sorted* MS-TRSs. Zantema showed that, in general, termination is not a persistent property of MS-TRSs, but he gave conditions for persistency of termination for MS-TRSs: termination is persistent for MS-TRSs not containing both duplicating and collapsing rules [26, Theorem 14]. Aoto solved a conjecture (posed by Zantema) by showing that termination is persistent for MS-TRSs that contain only variables of the same sort [2, Theorem 4.23].

Persistence is an interesting property for our purposes: identifying subclasses of OS-TRSs for which termination is persistent allows us to remove all sort information to prove termination by using the *underlying* TRS without worrying about losing termination proofs due to missing sort information. When no ordering among sorts is given in an unconditional SCS-MRT, we actually start from a *many-sorted* TRS. Then, we can use these results to find a simpler proof of termination.

Example 7.3 Consider the following (many-sorted) Maude module:

```
mod MYNAT is
  sorts Bool Nat .
  op False : -> Bool .
  op True : -> Bool .
  op 0 : -> Nat .
  op s : Nat -> Nat .
  op plus : Nat Nat -> Nat .
  op prod : Nat Nat -> Nat .
  vars N M : Nat .
  eq plus(N, 0) = N .
  eq plus(N, s(M)) = s(plus(N, M)) .
  eq prod(N, 0) = 0 .
  eq prod(N, s(M)) = plus(prod(N, M), N) .
  eq even(0) = True .
  eq even(s(0)) = False .
  eq even(s(s(N))) = even(N) .
 endm
```

which specifies the addition and product of natural numbers together with a predicate to check whether a number is even or odd. Although this is a *many-sorted* specification, it only involves variables of sort `Nat`. Thus, according to Aoto's results, termination of $\mathcal{R}$ is equivalent to termination of $\Theta(\mathcal{R})$.

Unfortunately, Zantema and Aoto's results regarding persistence in MS-TRSs do not immediately generalize to OS-TRSs as the following example shows.

Example 7.4 Consider the following terminating OS-TRS $\mathcal{R}$ having no collapsing or duplicating rule, and containing only variables of the same sort [25, Example 4]:

```
fmod Example4-OL96 is
  sorts S S' .
  subsort S' < S .
  op f : S -> S .
  op g : S -> S .
  op a : -> S .
  op b : -> S' .
  var x : S' .
  eq f(x) = g(x) .
  eq g(a) = f(a) .
endfm
```

Since $\Theta(\mathcal{R})$ is not terminating (e.g., $\texttt{f(a)} \to_{\Theta(\mathcal{R})} \texttt{g(a)} \to_{\Theta(\mathcal{R})} \texttt{f(a)} \to_{\Theta(\mathcal{R})} \cdots$), we conclude that termination is not 'persistent' for OS-TRSs satisfying the conditions in [2,26] for persistence of termination in MS-TRSs.

8 Conclusions and future work

In the previous sections we have discussed a number of techniques which can be combined in order to prove termination of SCS-MRTs, which is the main goal of this paper. First of all, we stress that the transformation from CS-MRTs (hence from SCS-MRTs) into (unsorted!) CS-CTRSs given by Durán et al. [8, Section 4], was already able to deal with the sort information in CS-MRTs for the purpose of proving termination of the corresponding programs. In this paper, we have introduced an alternative (hopefully more efficient) way to deal with the sort information in SCS-MRTs by keeping the 'order-sorted' flavor as much as possible and then using Ölveczky and Lysne's transformation. We have implemented these transformations as part of the *Maude Termination Tool* (MTT [9]) to test their practical performance. Furthermore, we have experimental evidence that the examples considered in the benchmarks reported in [8, Section 5.2] (whose code is available at `http://www.lri.fr/~marche/MTT`) can be more efficiently handled by using the results and considerations in this paper (see the benchmarks at `http://www.lcc.uma.es/~duran/MTT`). However, the transformation in [8, Section 4] is conjectured to be *complete*, whereas Ölveczky and Lysne's transformation is not (see Example 7.2). Of course, since OS-CS-RTs can also be seen as CS-MRTs, we can still use Durán et al.'s transformation to deal with the OS-CS-RTs obtained after the transformation described in Section 4. For instance, after applying Durán et al.'s transformation to the OS-CS-RT in Example 7.2 we obtain a CS-TRS which is not difficult to see is terminating (but no automatic proof has been obtained yet).

Furthermore, in Section 7 we have showed that, whenever MS-TRSs $\mathcal{R}$ are considered, we have (in the persistent case) the opportunity to use the 'underlying' (simpler) TRS $\Theta(\mathcal{R})$ to prove termination without losing anything. The transformation from OS-CS-RTs into OS-CS-TRSs given in Section 5 does not modify the sorts in the original system. Thus, it can be used to obtain a MS-CS-TRS from a MS-CS-RT (i.e., an OS-CS-RT without any ordering among the sorts). However, the transformation in Section 4 may introduce new subsort relations due to the introduction of 'kind' sorts. In order to avoid this, when possible, it would be better to give a specialized version of the transformation which applies when considering SCS-MRTs which are actually *many-sorted*.

The aforementioned issues should be clarified and the resulting techniques appropriately combined in the near future.

Acknowledgement

We thank Paco Durán and the referees for many useful remarks. Salvador Lucas was supported by the EU (FEDER) and the Spanish MEC/MICINN (grant TIN2007-68093-C02-02) and by Generalitat Valenciana under grant GV06/285. José Meseguer was supported in part by ONR Grant N00014-02-1-0715.

References

[1] B. Alarcón, R. Gutiérrez, J. Iborra, and S. Lucas. Proving Termination of Context-Sensitive Rewriting with MU-TERM. *Electronic Notes in Theoretical Computer Science*, 188:105-115, 2007.

[2] T. Aoto. Solution to the Problem of Zantema on a Persistent Property of Term Rewriting Systems. *Journal of Functional and Logic Programming*, 2001(11):1-20, 2001.

[3] A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theoretical Computer Science*, 236:35–132, 2000.

[4] R. Bruni and J. Meseguer. Semantic foundations for generalized rewrite theories. *Theoretical Computer Science* 351(1):386-414, 2006.

[5] M. Bidoit and P.D. Mosses. Casl User Manual - Introduction to Using the Common Algebraic Specification Language. Lecture Notes in Computer Science 2900, 2004.

[6] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. All About Maude – A High-Performance Logical Framework. Lecture Notes in Computer Science 4350, 2007.

[7] F. Durán, S. Lucas, C. Marché, J. Meseguer, and X. Urbain. Proving Termination of Membership Equational Programs. In P. Sestoft and N. Heintze, editors, *Proc. of ACM SIGPLAN 2004 Symposium on Partial Evaluation and Program Manipulation, PEPM'04*, pages 147–158. ACM Press, 2004.

[8] F. Durán, S. Lucas, C. Marché, J. Meseguer, and X. Urbain. Proving Operational Termination of Membership Equational Programs. *Higher-Order and Symbolic Computation*, 21(1-2):59-88 2008.

[9] F. Durán, S. Lucas, and J. Meseguer. MTT: The Maude Termination Tool. In A. Armando, P. Baumgartner, and G. Dowek, editors, *Proc. of the 4th International Joint Conference on Automated Reasoning, IJCAR'08*, LNCS 5195:313-319, Springer-Verlag, Berlin, 2008. Available at http://www.lcc.uma.es/~duran/MTT.

[10] S. Eker. Term Rewriting with Operator Evaluation Strategies. In C. Kirchner and H. Kirchner, editors, *Proc. of the 2nd International Workshop on Rewriting Logic and its Applications, WRLA'98*, Electronic Notes in Computer Science, 15:1-20, 1998.

[11] K. Futatsugi and R. Diaconescu. *CafeOBJ Report*. World Scientific, AMAST Series, 1998.

[12] J. Giesl, P. Schneider-Kamp, and R. Thiemann. APROVE 1.2: Automatic Termination Proofs in the Dependency Pair Framework. In U. Furbach and N. Shankar, editors, *Proc. of the 3rd International Joint Conference on Automated Reasoning, IJCAR'06*, LNAI 4130:281-286, Springer-Verlag, Berlin, 2006. Available at http://www-i2.informatik.rwth-aachen.de/AProVE.

[13] I. Gnaedig. Termination of Order-sorted Rewriting. In H. Kirchner and G. Levi, editors, *Proc. of the 3rd International Conference on Algebraic and Logic Programming, ALP'92*, LNCS 632:37-52, Springer-Verlag, Berlin, 1992.

[14] J. Goguen and J. Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105:217–273, 1992.

[15] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducing OBJ. In *Software Engineering with OBJ: Algebraic Specification in Action*. Kluwer, 2000.

[16] B. Gramlich and S. Lucas. Modular termination of context-sensitive rewriting. In C. Kirchner, editor, *Proc. of the 4th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, PPDP'02*, pages 50-61, ACM Press, New York, 2002.

[17] C. Kirchner, H. Kirchner, and J. Meseguer. Operational Semantics of OBJ3. In T. Lepistö and A. Salomaa, editors, *Proc. of 15th International Colloquium on Automata, Languages and Programming, ICALP'88*, LNCS 317:287-301, Springer-Verlag, Berlin, 1988.

[18] S. Lucas. Context-sensitive computations in functional and functional logic programs. *Journal of Functional and Logic Programming*, 1998(1):1-61, 1998.

[19] S. Lucas. Context-sensitive rewriting strategies. *Information and Computation*, 178(1):294–343, 2002.

[20] S. Lucas. MU-TERM: A Tool for Proving Termination of Context-Sensitive Rewriting In V. van Oostrom, editor, *Proc. of 15th International Conference on Rewriting Techniques and Applications, RTA'04*, LNCS 3091:200-209, Springer-Verlag, Berlin, 2004. Available at http://zenon.dsic.upv.es/muterm.

[21] S. Lucas, C. Marché, and J. Meseguer. Operational termination of conditional term rewriting systems. *Information Processing Letters*, 95:446–453, 2005.

[22] J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Proc. of the 12th International Workshop on Recent Trends in Algebraic Development Techniques, WADT'97*, LNCS 1376:18–61, Springer-Verlag, Berlin, 1998.

[23] J. Meseguer and J. Goguen. Initiality, induction and computability. In M. Nivat and J. Reynolds, editors, *Algebraic Methods in Semantics*, pages 459–541. Cambridge University Press, 1985.

[24] E. Ohlebusch. *Advanced Topics in Term Rewriting.* Springer-Verlag, 2002.

[25] P.C. Ölveczky and O. Lysne. Order-Sorted Termination: The Unsorted Way. In M. Hanus and M. Rodríguez-Artalejo, editors, *Proc. of the 5th International Conference on Algebraic and Logic Programming, ALP'96*, LNCS 1139:92-106, Springer-Verlag, Berlin, 1996.

[26] H. Zantema. Termination of term rewriting: interpretation and type elimination. *Journal of Symbolic Computation*, v17:23-50, 1994.