

Electronic Notes in Theoretical Computer Science 238 (2009) 207–225

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

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

Salvador Lucas

*DSIC, Universidad Polit´ecnica de Valencia, Spain*

Jos´e 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 equa- tional 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, condi- tional *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 O¨ lveczky 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.

# 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](#_bookmark40),[8](#_bookmark27)] 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](#_bookmark41)] 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](#_bookmark40)]. 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.

1571-0661/© 2009 Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

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](#_bookmark40)] and also Example [1.2](#_bookmark3) 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 mem- berships 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 ax- ioms 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 sophisti- cated 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](#_bookmark34)], CafeOBJ [[11](#_bookmark30)], and a subset of CASL [[5](#_bookmark21)] all support order- sorted specifications, while Maude [[6](#_bookmark22)] supports both OS and MEL specifications. The operational termination method in [[21](#_bookmark40)] 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](#_bookmark41)]), 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](#_bookmark26),[8](#_bookmark27)], 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 automat- ically. We transform order-sorted programs into unsorted ones by extending and generalizing a method proposed by O¨ lveczky and Lysne [[25](#_bookmark42)].

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](#_bookmark23), Section 3.3] and

[[22](#_bookmark41)] for more theoretical studies that also use and justify this sugared notation.

**Example 1.1** Consider the following Maude functional module [[7](#_bookmark26)]:

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](#_bookmark1) (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 −→ t'* [[3](#_bookmark23)]. 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](#_bookmark37),[19](#_bookmark38)]. 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](#_bookmark27)]. In [[7](#_bookmark26),[8](#_bookmark27)], (operational) termination of Maude functional modules as the one in Ex- ample [1.1](#_bookmark0) is proved by considering the underlying CS-MRT. For instance, Figure [1](#_bookmark2) shows the CS-MRT which corresponds to the module in Example [1.1](#_bookmark0). Note that the “order-sorted” syntactic sugar has been eliminated. This desugaring means that: (i) subsort declarations and operator declarations are desugared into conditional mem- berships; for example, the subsort declaration NatList < NatIList and the oper- ator declaration op s : Nat -> Nat become, respectively, the conditional mem- berships 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 embed- ding of order-sorted logic into membership equational logic proved correct in [[22](#_bookmark41)] *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](#_bookmark3) (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](#_bookmark29)]), 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](#_bookmark26),[8](#_bookmark27)] 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 signa- ture, see [[18](#_bookmark37),[19](#_bookmark38)]. The obtained CS-TRS can be proved terminating by using existing termination tools like AProVE [[12](#_bookmark31)] or mu-term [[1](#_bookmark20),[20](#_bookmark39)] 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](#_bookmark28)]). Termination of the Maude program in Example [1.1](#_bookmark0) 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* equa- tions 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](#_bookmark0), viewed as an ordinary order-sorted specification, can easily be proved terminating by using a *context-sensitive* version of O¨ lveczky and Lysne’s transformation [[25](#_bookmark42)] *without* introducing any conditional rule (see Example [6.2](#_bookmark15) below). This suggests that sugared MEL/CS-MRT specifications (see Section [3](#_bookmark5)), where the ‘order-sorted’ components (in the sense of [[3](#_bookmark23), 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](#_bookmark27)]), 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](#_bookmark27), 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,

*...*

s(s(s(N))):Inf

s(s(N)):Inf s(N):Inf

(Figure [3](#_bookmark6) shows the inference system for computations with CS-MRT programs) In order to cover arbitrary MEL programs, in Section [4](#_bookmark7) we show how to appro-

priately 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](#_bookmark12), we extend Ohlebusch’s transformation from CTRSs into TRSs [[24](#_bookmark43)] to

transform an OS-CS-RT into an OS-CS-TRS. Then, we slightly generalize O¨ lveczky

and Lysne’s transformation from OS-TRSs into TRSs to deal with context-sensitivity information. In Section [6](#_bookmark14) we adapt their transformation to deal with OS-CS-TRSs and yield a CS-TRS whose termination can be proved by using existing tools. Sec- tion [7](#_bookmark16) discusses the possible use of concepts and results coming from the area of

*many-sorted rewriting* to improve our proofs of termination. Section [8](#_bookmark19) concludes.

# Preliminaries

We summarize here material from [[14](#_bookmark33),[22](#_bookmark41),[17](#_bookmark36)] on order-sorted rewriting. An *order- sorted equational speciﬁcation* is a 4-tuple (Σ*, S, ≤, E*) with (Σ*, S, ≤*) an *order-sorted signature*, and *E* a set of (possibly conditional) Σ-equations. An order-sorted sig- nature (Σ*, S, ≤*) consists of a partially ordered set (*S, ≤*) of *sorts*, where *s ≤ s'* is interpreted as *subsort inclusion*, and with Σ a *S∗ × S*-indexed family of sets Σ = *{*Σ*w,s}*(*w,s*)*∈S∗×S* , which are *function symbols* with given string of argument sorts and result sort. The *connected components* of (*S, ≤*) are the equivalence classes corresponding to the least equivalence relation containing *≤*. If *f ∈* Σ*s*1*...sn,s*, then we display the function symbol *f* as *f* : *s*1 *··· sn −→ s*. Some of these symbols *f* can be *subsort-overloaded*. For example, we can have a subsort inclusion *Nat ≤ Int* and two subsort-overloaded declarations + : *Nat Nat −→ Nat* , and + : *Int Int −→ Int* . By an order-sorted substitution we mean an *S*-indexed substitution *σ* such that for all sort *s ∈ S* and all variable *x ∈ Xs*, the sort *s'* of *σ*(*x*) satisfies *s' ≤ s*.

A simple syntactic condition on (Σ*, S, ≤*) called *preregularity* [[14](#_bookmark33)] 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 E'*, with *Ax* a set of axioms such as associativity, and/or commutativity,

(Reflex)

*t →∗ t'*

if *t* =*Ax t'*

*t →*1 *t'*

*t' →∗ t''*

(Trans)

*t →∗ t''*

*ui →*1 *u'*

*i*

(Congr)

*f* (*u*1*,..., ui,..., un*) *→*1 *f* (*u*1*,..., u' ,..., un*)

*i*

where *f ∈* Σ and *i ∈ μ*(*f* )

*σ*(*u*1) *→∗ σ*(*v*1) *... σ*(*un*) *→∗ σ*(*vn*)

(Replacement) *u →*1 *σ*(*t'*)

where *t → t'* if *u*1 *→ v*1 *··· un → vn ∈ R*, *σ* is an OS-substitution, and *u* =*Ax σ*(*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](#_bookmark24),[8](#_bookmark27)], we also consider a *replacement map* [[18](#_bookmark37)], i.e., a function *μ* : Σ *−→ Pfin*(**N**) associating to each operator *f* of *n* arguments a set of argument positions *μ*(*f* ) = *{i*1*,..., im}*, with 1 *≤ ij ≤ 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 (Σ*, S, ≤, μ, Ax, R*). Figure [2](#_bookmark4) 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](#_bookmark36)] 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 −→ t' if cond* is called *sort-decreasing* (resp. *sort-preserving* ) if for each substitution *θ* such that *θ*(*cond* ) holds, the least sort of *θ*(*t'*) is smaller or equal than (resp. equal to) the least sort of *θ*(*t*): *LS*(*θ*(*t*)) *≥ LS*(*θ*(*t'*)) (resp. *LS*(*θ*(*t*)) = *LS*(*θ*(*t'*)) ). For a simple way to check these properties in the unconditional case see [[17](#_bookmark36)]. The conditional case has been studied in [[3](#_bookmark23)] in a more general MEL version.

# Sugared context-sensitive membership rewrite theo- ries

By a *sugared* context-sensitive membership rewrite theory (SCS-MRT) we under- stand a tuple T = (Σ*, S, ≤, μ, Ax, R, M* ) where

1. *S* is a set of *sorts* and (*S, ≤*) is a partial oder.
2. Σ = Σ0Σ1, where
   1. (Σ0*, S, ≤*) is an order-sorted signature that we assume *preregular* modulo

*Ax*.

* 1. Σ1 is a collection of operator declarations of the form *f* : [*s*1] *···* [*sn*] *→* [*s*], where [*s*1]*,...,* [*sn*]*,* [*s*] *∈ K*(*S, ≤*) and we let *K*(*S, ≤*) = *S/≡≤ {*[*T ruth*]*}* (we assume that *Truth /∈ S*). Here, *≡≤* is the smallest equivalence relation containing the order *≤*. Hence, *K*(*S, ≤*) contains a new kind for each connected component in (*S, ≤*) plus a new kind [*T ruth*].

Roughly speaking, Σ0 contains the symbols which are given an explicit sort in

the SCS-MRT specification, whereas Σ1 contains symbols that do not admit a profile based only on ‘proper’ sorts but rather require the use of *kinds* (cor- responding to the connected components in (*S, ≤*) as a whole). Such use of kinds is typically needed for functions that are *intrinsically partial*. For exam- ple, 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], be- cause 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.

1. *μ* : Σ *→ Pfin*(N) is a mapping sending each *f* : *s*1 *··· sn → s* in Σ0 to a subset *μ*(*f* ) *⊆ {*1*,..., n}*, and likewise each *f* : [*s*1] *···* [*sn*] *→* [*s*] in Σ1 to a subset *μ*(*f* ) *⊆ {*1*,..., n}* and such that if *f* is *subsort overloaded* in Σ0, or there exists *f* : *s*1 *··· sn → s* in Σ0 and *f* : [*s*1] *···* [*sn*] *→* [*s*] in Σ1, then *μ*(*f* ) is the *same* for all such subsort overloaded versions of *f* .
2. *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*] *∈ K*(*S, ≤*).
3. *R* is a set of *conditional rewrite rules* of the form

*∀x*1 : *s*1*,..., xn* : *sn, xn*+1 : [*sn*+1]*,..., xn*+*m* : [*sn*+*m*]*,t → t'* if *A*1 *∧ ... ∧ Ak* (*†*)

with *s*1*,..., sn ∈ S* and [*sn*+1]*,...,* [*sn*+*m*] *∈ K*(*S, ≤*), and where the *Ai* are either rewrite conditions *u → v*, or memberships *w* : *s'*. As usual, we view unconditional rules as a special case of conditional rule with empty condition.

1. *M* is a set of *conditional memberships* of the form

*∀x*1 : *s*1*,..., xn* : *sn, xn*+1 : [*sn*+1]*,..., xn*+*m* : [*sn*+*m*]*,t* : *s* if *A*1 *∧ ... ∧ Ak* (*††*)

with *s*1*,..., sn ∈ S* and [*sn*+1]*,...,* [*sn*+*m*] *∈ K*(*S, ≤*), and the *Ai* 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:

1. Taking *K*(*S, ≤*) as the set of kinds.
2. Taking as signature Σ1 *∪ {f* : [*s*1] *···* [*sn*] *→* [*s*] *| f* : *s*1 *··· sn → s ∈* Σ0*}*.
3. Taking *S* as the set of sorts.
4. Transforming each *s ≤ s'* in *≤* into a membership axiom *x* : *s'* if *x* : *s*.
5. Defining *μ* exactly as before, but for operators now lifted to the kinds.
6. Adding for each *f* : *s*1 *··· sn → s* in Σ0 a conditional membership

*∀x*1 : [*s*1]*,..., xn* : [*sn*]*,f* (*x*1*,..., xn*) : *s* if *x*1 :: *s*1 *∧· · · ∧ xn* :: *sn*

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](#_bookmark6) and [[8](#_bookmark27)]).

1. Transforming each conditional rule (*†*) into a rule

*∀x*1 : [*s*1]*,..., xn*+*m* : [*sn*+*m*]*, t → t'* if 1*≤l≤n xl* :: *sl ∧ A*1 *∧ ... ∧ Ak*

(Subject reduction)

*t →*1 *t' t'* : *s t* : *s*

*A•σ ··· A• σ*

(Membership-1)

1 *n*

*u* :: *s*

where *t* : *s* if *A*1 *··· An* in *R* and *u* =*Ax tσ*

(Membership-2)

*t* :: *s*

*t* : *s*

(Reflexivity)

*t →∗ t'*

*t →*1 *t'*

if *t* =*Ax t'*

*t' →∗ t''*

(Transitivity)

*t →∗ t''*

*ui →*1 *u'*

*i*

(Congruence)

*f* (*u*1*,..., ui,... un*) *→*1 *f* (*u*1*,..., u' ,..., un*)

*i*

where *i ∈ μ*(*f* )

*A•σ . . . A• σ*

1 *n*

(Replacement)

*u →*1 *t'σ*

where *t → t'* if *A*1 *··· An* in *R* and *u* =*Ax tσ*

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

1. Transforming each conditional membership (*††*) into a membership

*∀x*1 : [*s*1]*,..., xn*+*m* : [*sn*+*m*]*, t* : *s* if 1*≤l≤n xl* :: *sl ∧ A*1 *∧ ... ∧ Ak*

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](#_bookmark41)].

We can define the computations associated to a CS-MRT by means of the in- ference rules of Figure [3](#_bookmark6), where *A•* = *Ai* whenever *Ai* is a membership *w* : *s* or *x* :: *s*, and *A•* is *u →∗ v* if *Ai* is a rewrite condition *u → 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](#_bookmark8) *Ax*.

*i*

*i*

# Transforming SCS-MRTs into OS-CS-RTs

Given an SCS-MRT T = (Σ*, S, ≤, μ, Ax, R, M* ), we first define the set *MB* T(*S*) *⊆ S* (or just *MB* (*S*) if T is clear from the context) of its *membership sorts* as the smallest subset of *S* such that

1. if a membership *∀x*1 : *s*1*,..., xn* : *sn,t* : *s* if *C* belongs to *M* , then *s ∈ MB* (*S*).
2. if *s ∈ MB* (*S*) and *s ≤ s'*, then *s' ∈ 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*,..., ui,..., un*). However, this extra generality can be avoided, see [[8](#_bookmark27)].

1. if *f* : *s*1 *··· sn → s ∈* Σ0 and *si ∈ MB* (*S*) for some *i*, 1 *≤ i ≤ n*, then

*s ∈ 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*) = *∅* whenever *M* = *∅*. 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](#_bookmark4) (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 T*•* 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*) *MB* (*S*) induces a partition of the rules in *R* in two disjoint sets: *R* = *OS* (*R*) *MB* (*R*), where *OS* (*R*) *⊆ R* is the set of conditional rules of the form (*†*) such that *s*1*,..., sn ∈ OS* (*S*), whereas *MB* (*R*) is its complement, that is, those rules of the form (*†*) such that there exists *i,* 1 *≤ i ≤ n*, with *si ∈ MB* (*S*).

The theory transformation T *'→* T*•* we are looking for sends the SCS-MRT T to the order-sorted context-sensitive rewrite theory (OS-CS-RT)

T*•* = (Σ*•, S•, ≤•, μ•, Ax, OS* (*R*) *∪ MB* (*R*)*• ∪ M• ∪ MB* (Σ*, S*)*•*)

where

1. *S•* = *OS* (*S*) *K*(*S, ≤*).
2. *≤•* is the reflexive-transitive closure of the following relation
   1. For all *s, s' ∈ OS* (*S*)*, s≤•s'* iff *s ≤ s'*.
   2. For all *s ∈ OS* (*S*)*, s≤•*[*s*].
3. Σ*•* is the following set of operations:

Σ*•* = *{f* : ***w*** *→ s ∈* Σ0 *| ws ∈ OS* (*S*)+*} ∪ {f* : [***w***] *→* [*s*] *| f* : ***w*** *→ s ∈* Σ0*}*

*∪ {iss* : [*s*] *→* [*T ruth*] *| s ∈ MB* (*S*)*} ∪ {is'* : [*s*] *→* [*T ruth*] *| s ∈ S}*

*s*

*∪ {tt* : *→* [*T ruth*]*}*

1. *μ•* agrees with *μ* on the first two sets of operators above: maps *μ*(*f* : [***w***] *→* [*s*]) to *μ*(*f* : ***w*** *→ s*) and maps *μ*(*iss*) = *∅* and *μ*(*is'* ) = *{*1*}*. This choice for the replacement restrictions associated to these symbols is consistent with the intended use of *iss* and *is'* : as discussed in [[8](#_bookmark27), Section 4], predicates *iss* deal with sort declarations for variables like *x* : *s* where *x* is a variable and *no reduction* below *iss* in an instance of *iss*(*x*) is required to check the membership (hence *μ*(*iss*) = *∅*). On the other hand, predicates *is'* are intended to deal with sort conditions *is'* (*w*) coming from ‘proper’ memberships *w* : *s*, where *w* is a

*s*

*s*

*s*

*s*

nonvariable term (or *s* is a *membership sort* ) and some *subject reduction* could be necessary to check the membership (thus *μ*(*is'* ) = *{*1*}*).

*s*

1. The set *MB* (*R*)*•* contains, for each rule of the form (*†*) in *MB* (*R*) a correspond- ing rule of the form

*∀x*1 : *s*1*,..., xp* : *sp, xp*+1 : [*sp*+1]*,..., xn* : [*sn*]*,..., xn*+*m* : [*sn*+*m*]*,*

*t → t'* if *iss* (*xl*) *→ tt ∧ Ab ∧ ... ∧ Ab*

*p<l≤n Æ* 1 *k*

where, without loss of generality, we assume that *s*1*,..., sp ∈ OS* (*S*), and *sp*+1*,..., sn ∈ MB* (*S*), and where if *Ai* is a rewrite condition *u → v*, then *Ab* = *Ai*, and if *Ai* is a membership *w* : *s'*, then *Ab* = *is' '* (*w*) *→ tt*.

*i i s*

1. The set *M•* contains, for each conditional membership of the form (*††*) in *M* , a conditional rule of the form

*∀x*1 : *s*1*,..., xp* : *sp, xp*+1 : [*sp*+1]*,..., xn* : [*sn*]*,..., xn*+*m* : [*sn*+*m*]*,*

*iss*(*t*) *→ tt* if *iss* (*xl*) *→ tt ∧ Ab ∧ ... ∧ Ab*

*p<l≤n Æ* 1 *k*

where, again, *s*1*,..., sp ∈ OS* (*S*), and *sp*+1*,..., sn ∈ MB* (*S*), and the *Ab* are

*i*

as before.

1. Finally, *MB* (Σ*, S*)*•* consists of the following additional rules:
   1. For each *s ∈ MB* (*S*), we add a rule *∀x* : [*s*]*, is'* (*x*) *→ iss*(*x*).

*s*

* 1. For each *s ∈ OS* (*S*), we add a rule *∀x* : *s, is'* (*x*) *→ tt*. Note that this

*s*

rule is needed because memberships *wj* : *s'* with *s' ∈ OS* (*S*), could appear

*j*

*j*

in the conditional part of either a rule (*†*) or a membership (*††*). So, we

need to express the membership predicate *wj* : *s'* as the truth condition

*j*

*is' '* (*wj*) *→ tt*. Exactly for this reason, we have defined the predicates *is'*

*sj s*

for any *s ∈ S*. Instead, predicates *iss* are *only deﬁned for s ∈ MB* (*S*). They are not needed for *s ∈ OS* (*S*) because we have instead the rule

*∀x* : *s, is'* (*x*) *→ tt* so that the sort checking is in this case performed by the order-sorted type structure.

*s*

* 1. For each *s ∈ OS* (*S*), *s' ∈ MB* (*S*) with *s ≤ s'*, we add rules

*∀x* : *s, is' '* (*x*) *→ tt ∀x* : *s, iss'* (*x*) *→ tt*

*s*

* 1. For each constant symbol *c* of sort *s*, we add a rule *iss*(*c*) *→ tt*.
  2. For each *f* : *s*1 *··· sn → s ∈* Σ0 such that the sorts among the *s*1*,..., sn* in

*MB* (*S*) are *si*1 *,..., sik* with *k ≥* 1, we add the conditional rule:

*∀x*1 : *s*1*,..., xi*1 : [*si*1 ]*,..., xik* : [*sik* ]*,..., xn* : *sn,*

*iss*(*f* (*x*1*,..., xn*)) *→ tt* if *issi*1 (*xi*1 ) *→ tt ∧ ··· ∧ issik* (*xik* ) *→ tt*

* 1. For each *s, s' ∈ MB* (*S*) such that *s ≤ s'*, we add the rule

*∀x* : [*s'*]*, iss'* (*x*) *→ tt* if *iss*(*x*) *→ tt*

Note that the previous transformation becomes quite simple when *MB* (*S*) is empty. Then, no symbol *iss* is necessary (see item ([iii](#_bookmark9)) above) and, hence, no rule (or condi- tion) involving them is introduced; in particular, *MB* (*R*) = *∅* (hence *MB* (*R*)*•* = *∅*).

The following theorem expresses the main property of this transformation.

**Theorem 4.1** *Let* T *be an SCS-MRT,* T0 *be its* unsugared *version (as a CS-MRT), and* T*• be the corresponding transformed OS-CS-RT. Then, for all ground terms t, t' ∈T* (Σ) *of the same kind and all sorts s of the kind, we have*

1. T0 *▶ t →∗ t' if and only if* T*• ▶ t →∗ t'*
2. T0 *▶ t →*1 *t' if and only if* T*• ▶ t →*1 *t'*
3. T0 *▶ t* : *s if and only if* T*• ▶ is'* (*t*) *→∗ tt*

*s*

1. (*for s ∈ MB*(*S*)) T0 *▶ t* :: *s if and only if* T*• ▶ iss*(*t*) *→∗ tt*

The following theorem is an easy consequence of Theorem [4.1](#_bookmark10) 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](#_bookmark6), and operational termi- nation in the OS-CS-RT logic, whose inference system is showed in Figure [2](#_bookmark4).

**Theorem 4.2** *An SCS-MRT* T *is operationally ground terminating if and only if the OS-CS-RT* T*• is operationally ground terminating.*

The restriction to ground terms is harmless by assuming (see [[3](#_bookmark23)]), 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.

* 1. *Examples*

For the module in Example [1.1](#_bookmark0), *OS* (*S*) = *{*Nat*,* NatList*,* NatIList*}* and *MB* (*S*) =

*∅*. The connected components in (*S, ≤*) are Nat and NatList *<* NatIList. Thus,

*K*(*S, ≤*) = *{*[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](#_bookmark3), we have *MB* (*S*) = *S* =

*{*Inf*,* Nat*}* and *OS* (*S*) = *∅*. The only connected component in (*S, ≤*) is Inf *<* Nat. Thus, *K*(*S, ≤*) = *{*[Nat]*,* [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

# Removing conditions from order-sorted specifications

In order to check operational termination with respect to OS-CS-RT logic, we pro- pose a transformation associating an unconditional OS-CS-TRS *U* (*R*) = (*U* (Σ)*, S, ≤*

*, U* (*μ*)*, Ax, U* (*R*)) to an OS-CS-RT *R* = (Σ*, S, ≤, μ, Ax, R*). In [[7](#_bookmark26),[8](#_bookmark27)], the classical transformation for proving operational termination of a *deterministic* 3*-CTRS* [3](#_bookmark13) *R* as termination of a TRS *U* (*R*) (see [[24](#_bookmark43), Definition 7.2.48]), was generalized to han-

dle 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](#_bookmark22),

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 transfor- mation in Section [4](#_bookmark7) maps admissible modules into deterministic OS-CS-RTs (more precisely, the ‘underlying’ 3-CTRSs are deterministic). The adaptation of the trans- formation in [[7](#_bookmark26),[8](#_bookmark27)] is simple. The new signature *U* (Σ) consists of the symbols in Σ together with new symbols *U* as described below. Regarding the rules, given an order-sorted conditional rule

*∀x*1 : *s*1*,..., xm* : *sm,l → r* if *u*1 *→ v*1*,..., un → vn*

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

*j*=1

3 A 3-CTRS only contains rules *l → r* if *c* such that Var(*r*) *⊆* Var(*l*) *∪* Var(*c*). A 3-CTRS *R* is called *deterministic* if for each *l → r* if *s*1 *→ t*1*,..., sn → tn* in *R* and each 1 *≤ i ≤ n*, we have Var(*si*) *⊆* Var(*l*) *∪* S*i−*1 Var(*tj* ), see [[24](#_bookmark43)].

*∀x*1 : *s*1*,..., xm* : *sm,l → U*1(*u*1*,* ***x*1**) (1)

*∀x*1 : *s*1*,..., xm* : *sm, Ui−*1(*vi−*1*,* ***xi—*1**) *→ Ui*(*ui,* ***xi***) 2 *≤ i ≤ n* (2)

*∀x*1 : *s*1*,..., xm* : *sm, Un*(*vn,* ***xn***) *→ r* (3)

where the ***xi*** are vectors of variables defined as follows: assume a given ordering on the set of variables *X* . Then, ***xi*** contains the ordered sequence of the variables in the set Var(*l*) *∪* Var(*v*1) *∪· · ·∪* Var(*vi−*1) for 1 *≤ i ≤ 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:

***xi*** = (Var(*l*) *∪* Var(*v*1) *∪· · · ∪* Var(*vi−*1))

*∩* (Var(*vi*) *∪* Var(*ui*+1) *∪* Var(*vi*+1) *∪· · ·*

*∪* Var(*un*) *∪* Var(*vn*) *∪* Var(*r*))

Let ***si*** be the sorts corresponding to variables in ***xi***. Now, we just need to set that the fresh symbols added to the signature are as follows: *Ui* : [*LS*(*ui*)] *×* ***si*** *→ LS*(*l*). Rewriting modulo *Ax* is allowed. The replacement map is transformed into a new replacement map *U* (*μ*) as follows: *U* (*μ*)(*U* ) = *{*1*}* for all new symbols *U* that are introduced to deal with the equations in the conditional part of each rule in *R* (that is, only the first argument of *U* can be evaluated), and *U* (*μ*)(*f* ) = *μ*(*f* ) for all symbols *f ∈ F*.

**Theorem 5.1** *The OS-CS-RT R is terminating if the OS-CS-TRS U* (*R*) *is termi- nating.*

# Termination of order-sorted rewriting

Termination of an OS-TRS *R* is obviously guaranteed if the *underlying* TRS Θ(*R*) (i.e., the TRS which is obtained after removing all sort information from the sig- nature 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](#_bookmark32)]. 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](#_bookmark42)], O¨ lveczky and Lysne introduce a transformation from OS-TRSs into ordinary TRSs

which can be used to prove termination of OS-TRSs. O¨ lveczky and Lysne showed

that their transformation strictly subsumes Gnaedig’s technique [[25](#_bookmark42)].

O¨ lveczky and Lysne’s transformation is as follows: an order sorted signature Σ is translated into an unsorted signature *F* = *{f****s*** *| f* : ***ss*** *→ s ∈* Σ for some *s* and ***s*** *≤*

***ss****}* where the arity of *f****s*** is *|****s****|* and ***s*** *≤* ***ss*** means that *si ≤ s'* for each *i*, 1 *≤*

*i*

*i ≤ |****s****|*. Now, many sorted terms *t ∈ T* (Σ*, X* ) are also disambiguated by using a transformation *[ ♩* : *T* (Σ*, X* ) *→T* (*F, X* ) given by [[25](#_bookmark42), Definition 3]:

*[x* : *s♩* = *x* for a variable *x ∈ Xs*

*[f* (*t*1*,..., tn*)*♩* = *fLS*(*t*1)*···LS*(*tn*)(*[t*1*♩,..., [tn♩*)

where *LS*(*t*) is the *least sort* associated to the term *t*. Now, the OS-TRS *R* is transformed into a TRS *[R♩* which consists of the signature *F* plus the following rules [[25](#_bookmark42), Theorem 2]:

*{[ν*(*l*)*♩→ [ν*(*r*)*♩| l → r ∈ R, ν* a specialization*} ∪*

*{f****s***(*x*1*,..., xn*) *→ f****ss*** (*x*1*,..., xn*) *| f****s*** *∈ F,* ***ss*** « ***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' ≤ 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, ***ss*** « ***s*** means that ***s*** *≤* ***ss*** and there is *i*, 1 *≤ i ≤ n*

such that *s'*

*i*

*< si*. According to [[25](#_bookmark42), Theorem 2], this transformation is correct

¨

for *sort-decreasing* OS-TRSs. Furthermore, as showed by Olveczky and Lysne, the second set of rules can be avoided when *sort-preserving* OS-TRSs are considered.

It is not difficult to see that O¨ lveczky and Lysne’s transformation could also be used to prove termination of OS-CS-TRSs (i.e., OS-TRSs supplied with a re- placement map *μ* for the many sorted function symbols *f ∈* Σ). Provided that *μ*(*f* : ***s*** *→ s*) = *μ*(*f* : ***s*** *→ s'*) holds for all symbols *f* , we let the transformed replacement map *[μ♩* be *[μ♩*(*f****s***) = *μ*(*f* : ***s*** *→ s*). We say that a TRS *R* is *μ*- terminating (or that the CS-TRS (*R, μ*) is terminating) if the context-sensitive rewrite relation associated to *R* and *μ* (written *‹→R,μ*, see [[18](#_bookmark37),[19](#_bookmark38)]) is terminating. The following result easily follows from the proof of [[25](#_bookmark42), Theorem 2]:

**Theorem 6.1** *The OS-CS-TRS R is terminating if [R♩ is [μ♩-terminating.*

**Example 6.2** When the transformation in Section [4](#_bookmark7) is applied to the program in Example [1.1](#_bookmark0), we obtain the OS-CS-RT LengthOfFiniteListsAndTake-OS in Section [4.1](#_bookmark11). After applying the (context-sensitive) version of O¨ lveczky and Lysne’s transformation to this OS-CS-TRS *R*, we obtain a CS-TRS *[R♩* 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 *[ ♩*) and the new replacement map is *μ*(cons-N-IL) = *μ*(consN-FL) = *{*1*}* and *μ*(*f* ) = *{*1*,..., k}* for all other *k*-ary symbols *f ∈* Σ. Since all rules in the program in Example [1.1](#_bookmark0) are *sort-preserving* we do not need the second set of rules of O¨ lveczky 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 *S* of the system in Example

[6.2](#_bookmark15), i.e., *[R♩* is the *disjoint* union of *S* plus the last four rules *S'*: *[R♩* = *S S'* (see [[16](#_bookmark35)] for results about modularity of termination of CS-TRSs). The CS-TRS *S'* is easily seen to be terminating, and it is both *noncollapsing* and *nonduplicating*, by [[16](#_bookmark35), Theorem 3], we can concentrate the attention in proving termination of *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.

# 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 O¨ lveczky and Lysne’s transformation behaves as the *trivial* transformation Θ when applied to an MS-TRS:

**Proposition 7.1** *For any MS-TRS R, [R♩ terminates if and only if* Θ(*R*) *termi- nates.*

**Example 7.2** The following many-sorted module, which corresponds to the famous Toyama’s example, is borrowed from [[26](#_bookmark44), 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 Propo- sition [7.1](#_bookmark17), *[R♩* = Θ(*R*), which is well-known to be nonterminating.

Since MS-TRSs are OS-TRSs, Example [7.2](#_bookmark18) also shows that O¨ lveczky and Lysne’s transformation is *incomplete* regarding proofs of termination of OS-TRSs.

Proposition [7.1](#_bookmark17) suggests that making use of some proper results for proving ter- mination of MS-TRSs can be useful for our purposes. Termination of MS-TRSs was investigated by Zantema [[26](#_bookmark44)] 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 termi- nation for MS-TRSs: termination is persistent for MS-TRSs not containing both duplicating and collapsing rules [[26](#_bookmark44), 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](#_bookmark25), 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 pred- icate 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 re- sults, termination of *R* is equivalent to termination of Θ(*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 *R* having no collapsing or duplicating rule, and containing only variables of the same sort [[25](#_bookmark42), 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 Θ(*R*) is not terminating (e.g., f(a) *→*Θ(*R*) g(a) *→*Θ(*R*) f(a) *→*Θ(*R*) *···* ), we conclude that termination is not ‘persistent’ for OS-TRSs satisfying the condi- tions in [[2](#_bookmark25),[26](#_bookmark44)] for persistence of termination in MS-TRSs.

# 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 Dura´n et al. [[8](#_bookmark27), Section 4], was already able to deal with the sort information in CS-MRTs for the pur- pose 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 possi- ble and then using O¨ lveczky and Lysne’s transformation. We have implemented these transformations as part of the *Maude Termination Tool* (MTT [[9](#_bookmark28)]) to test their practical performance. Furthermore, we have experimental evidence that the examples considered in the benchmarks reported in [[8](#_bookmark27), Section 5.2] (whose code is available at <http://www.lri.fr/~marche/MTT>) can be more efficiently han- dled 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](#_bookmark27), Section 4] is conjectured to be *complete*, whereas O¨ lveczky and Lysne’s transformation is not (see Example [7.2](#_bookmark18)). Of course, since OS-CS-RTs can also be seen as CS-MRTs, we can still use Dur´an et al.’s transformation to deal with the OS-CS-RTs obtained after the transformation described in Section [4](#_bookmark7). For instance, after applying Dura´n et al.’s transformation to the OS-CS-RT in Example [7.2](#_bookmark18) 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](#_bookmark16) we have showed that, whenever MS-TRSs *R* are con- sidered, we have (in the persistent case) the opportunity to use the ‘underlying’ (simpler) TRS Θ(*R*) to prove termination without losing anything. The transfor- mation from OS-CS-RTs into OS-CS-TRSs given in Section [5](#_bookmark12) 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](#_bookmark7) may introduce new subsort relations due to the in- troduction 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 ap- propriately combined in the near future.

# Acknowledgement

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

# References

1. B. Alarc´on, R. Guti´errez, 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.

1. A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theoretical Computer Science*, 236:35–132, 2000.
2. R. Bruni and J. Meseguer. Semantic foundations for generalized rewrite theories. *Theoretical Computer Science* 351(1):386-414, 2006.
3. M. Bidoit and P.D. Mosses. Casl User Manual - Introduction to Using the Common Algebraic Specification Language. Lecture Notes in Computer Science 2900, 2004.
4. M. Clavel, F. Dur´an, S. Eker, P. Lincoln, N. Mart´ı-Oliet, J. Meseguer, and C. Talcott. All About Maude – A High-Performance Logical Framework. Lecture Notes in Computer Science 4350, 2007.
5. F. Dur´an, S. Lucas, C. March´e, 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.
6. F. Dur´an, S. Lucas, C. March´e, J. Meseguer, and X. Urbain. Proving Operational Termination of Membership Equational Programs. *Higher-Order and Symbolic Computation*, 21(1-2):59-88 2008.
7. F. Dur´an, 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.](http://www.lcc.uma.es/~duran/MTT) [lcc.uma.es/~duran/MTT](http://www.lcc.uma.es/~duran/MTT).
8. 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.
9. K. Futatsugi and R. Diaconescu. *CafeOBJ Report*. World Scientific, AMAST Series, 1998.
10. 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>.
11. 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.
12. 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.
13. 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.
14. 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.
15. C. Kirchner, H. Kirchner, and J. Meseguer. Operational Semantics of OBJ3. In T. Lepist¨o and A. Salomaa, editors, *Proc. of 15th International Colloquium on Automata, Languages and Programming,* *ICALP’88*, LNCS 317:287-301, Springer-Verlag, Berlin, 1988.
16. S. Lucas. Context-sensitive computations in functional and functional logic programs. *Journal of* *Functional and Logic Programming*, 1998(1):1-61, 1998.
17. S. Lucas. Context-sensitive rewriting strategies. *Information and Computation*, 178(1):294–343, 2002.
18. 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/](http://zenon.dsic.upv.es/muterm) [muterm](http://zenon.dsic.upv.es/muterm).
19. S. Lucas, C. March´e, and J. Meseguer. Operational termination of conditional term rewriting systems.

*Information Processing Letters*, 95:446–453, 2005.

1. 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.
2. 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.
3. E. Ohlebusch. *Advanced Topics in Term Rewriting*. Springer-Verlag, 2002.
4. P.C. O¨ lveczky and O. Lysne. Order-Sorted Termination: The Unsorted Way. In M. Hanus and

M. Rodr´ıguez-Artalejo, editors, *Proc. of the 5th International Conference on Algebraic and Logic Programming, ALP’96*, LNCS 1139:92-106, Springer-Verlag, Berlin, 1996.

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