Proving Operational Termination of Membership Equational Programs

Francisco Durán, Salvador Lucas, Claude Marché, José Meseguer, Xavier Urbain

HOSC manuscript No.
(will be inserted by the editor)

Proving Operational Termination of Membership Equational Programs

Francisco Durán · **Salvador Lucas** ·
Claude Marché · **José Meseguer** · **Xavier Urbain**

the date of receipt and acceptance should be inserted later

Abstract Reasoning about the termination of equational programs in sophisticated equational languages such as ELAN, MAUDE, OBJ, CAFEOBJ, HASKELL, and so on, requires support for advanced features such as evaluation strategies, rewriting modulo, use of extra variables in conditions, partiality, and expressive type systems (possibly including polymorphism and higher-order). However, many of those features are, at best, only partially supported by current term rewriting termination tools (for instance MU-TERM, C*i*ME, APROVE, TTT, TERMPTATION, etc.) while they may be essential to ensure termination. We present a sequence of theory transformations that can be used to bridge the gap between expressive membership equational programs and such termination tools, and prove the correctness of such transformations. We also discuss a prototype tool performing the transformations on MAUDE equational programs and sending the resulting transformed theories to some of the aforementioned standard termination tools.

This research was partly supported by bilateral CNRS-DSTIC/UIUC research project "Rewriting calculi, logic and behavior", and by ONR Grant N00014-02-1-0715 and NSF Grant CCR-0234524; Francisco Durán was partially supported by the EU (FEDER) and the Spanish MEC, under grant TIN2005-09405-C02-01; Salvador Lucas was partially supported by the EU (FEDER) and the Spanish MEC, under grant TIN 2004-7943-C04-02, the Generalitat Valenciana under grant GV06/285, and the ICT for EU-India Cross-Cultural Dissemination ALA/95/23/2003/077-054 project.

F. Durán
LCC, Universidad de Málaga, Spain

S. Lucas
DSIC, Universidad Politécnica de Valencia, Spain

C. Marché
PCRI, LRI (CNRS UMR 8623), INRIA Futurs, Université Paris-Sud, France

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

X. Urbain
CEDRIC, IIE, Conservatoire National des Arts et Métiers, France

```
fmod LengthOfFiniteLists 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 length : NatList -> Nat .
   vars M N : Nat .
   var  IL : NatIList .
   var  L : NatList .
   eq zeros = cons(0,zeros) .
   eq length(nil) = 0 .
   eq length(cons(N, L)) = s(length(L)) .
endfm
```

Fig. 1 A MAUDE program example

1 Introduction

The goal of this work is to study transformational techniques that can help to bridge the gap between programs in expressive rule-based equational languages such as ASF+SDF [39], OBJ [19], MAUDE [5], CAFEOBJ [12], HASKELL [21], and modules in suitable equational subsets of ELAN [1] and CASL [7] on one hand, and termination tools assuming considerably more restrictive specifications (untyped, unconditional term rewriting systems) on the other. There is a clear tension between the goals of expressiveness and efficiency when using equational theories as *programs*, and the considerably simpler assumptions of standard reasoning techniques for rewrite systems and their associated tools. For example, many equational programs do not terminate in the usual sense, but do so when evaluated with suitable *types* and/or *strategies*.

Example 1 Consider the MAUDE specification in Figure 1, 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 `length` computes the length of a *finite* list. Note the *overloaded* operator `cons`, which can be used for building both finite and infinite lists of natural numbers and which is declared with evaluation *strategy* $(1\ 0)$. 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`.

This system is terminating (i.e., all reduction sequences, for any initial term, are finite), but both the evaluation *strategy* $(1\ 0)$ for `cons` and the use of sorts and subsorts (especially for `length`) are crucial to achieve this terminating behavior. In fact, by removing either the strategy annotation or the sort information we would get a non-terminating program: on the one hand, if reductions were allowed on the second argument of `cons`, then the evaluation of `zeros` would never terminate. On the other hand, an attempt to evaluate `length(xs)` will not terminate if `length` 'accepts' infinite lists `xs` like, e.g., `zeros`; this is forbidden by specifying that `length` only accepts lists of sort `NatList`, i.e., finite lists.

Current termination tools are not able to deal directly with programs like that in Example 1. This is because the programs make use of either types or strategies, or because of

other features such as conditional equations that are not handled by a given tool's input language. As illustrated by Example 1, these features are often essential to prove termination. Expressive features not handled by some current termination tools include:

1. Sorts, subsorts, overloading, and memberships (see [33, 3]);
2. Conditions, which may introduce extra variables;
3. Fixed evaluation strategies (e.g., leftmost innermost or leftmost outermost);
4. Programmable evaluation strategies, which permit annotating each function symbol with local strategy information on what arguments to evaluate or not (e.g., context sensitive rewriting strategies [24], E-strategies [19, 5], etc.);
5. Rewriting modulo axioms like associativity (A), commutativity (C), identity (I), AC, ACI, and so on.

For example, APROVE [17] supports some form of conditional equations (2), innermost rewriting (3), context-sensitive rewriting annotations (4), and AC symbols (5); C*i*ME [8] directly supports part of (5); and MU-TERM [26] directly supports (4). In all cases (and this is the main focus of this paper), these tools do not support the *combination* of these features.

1.1 Membership Equational Logic and Operational Termination

Equational languages with expressive features are supported by expressive *logics*, that typically include less expressive ones as sublogics. In this regard, membership equational logic (MEL) [33, 3] has proved to be a very expressive *logical framework*, in which a wide range of partial and total equational logics can be faithfully embedded [33]. This makes it an attractive framework logic for our main goal, which is developing termination techniques applicable to equational languages with expressive features. Specifically, modules in equational programming languages such as OBJ, CAFEOBJ, the equational sublanguage of ELAN, and a suitable executable fragment of CASL can all be faithfully represented as membership equational theories. Similarly, MAUDE's equational sublanguage, whose modules are membership equational theories, has itself a trivial, identity representation into this framework. As a consequence, our termination techniques are not only applicable to MAUDE, but also to all the above-mentioned languages.

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.

Operationally, and assuming good executability properties such as the Church-Rosser property [3] and admissibility in the sense explained in Section 3, equalities $t = t'$ can be treated as rewrite rules $t \longrightarrow t'$. 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 [23, 24]. 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. Note that in a CS-MRT rewriting and computation of memberships $t : s$ are *recursively intertwined*, because application of a conditional equation may require satisfying memberships in its conditions, and application of a conditional memberships may likewise require satisfying equalities in its condition. In particular, some useful programs may now only involve memberships, without

involving any rewriting. Consider, for example, the following *palindrome recognizer program* PALINDROME, which is a membership equational program expressible in MAUDE as follows:

```
fmod PALINDROME is
  protecting QID .  -- Imports sort Qid (quoted identifiers)
  sorts List Pal .
  subsorts Qid < Pal < List .
  op nil : -> Pal .
  op __ : List List -> List [assoc id: nil] .
  var I : Qid .
  var P : Pal .
  mb I P I : Pal .  -- membership axiom
endfm
```

This program –where list concatenation is expressed with empty syntax and satisfies associativity (`assoc`) and identity (`id` for `nil`) axioms– is terminating, that is, given a list of quoted identifiers the specification can always be used to compute in a finite number of steps whether it is a palindrome, i.e., has sort `Pal`, or not. But note that no rewriting at all is involved. Similarly, the program

```
fmod INF is
  protecting NAT .
  sort Inf .
  subsort Inf < Nat .
  var N : Nat .
  cmb s(N) : Inf if s(s(N)) : Inf .
               -- a conditional membership
endfm
```

is nonterminating, but again no rewriting is involved in its nontermination. This means that the standard theoretical framework of term rewriting, and the termination notions that have been developed for it, including those for Conditional Term Rewriting Systems (CTRSs), are insufficient for dealing with termination of MEL programs. For this reason, we use in this paper a proof-theoretic termination notion, called *operational termination* [29]. 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. Intuitively, an CS-MRT program is operationally terminating if all its well-formed proof trees are finite. For example, the nontermination of the `INF` program is witnessed by the infinite proof tree,

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

The following MAUDE program, involving both equations and memberships, shows how the recursive interaction between rewriting and membership computations can lead to subtle nontermination problems

```
fmod INF2 is
   sorts S .
   op a : -> [S] .
   op f : [S] -> [S] [strat (0)] .
   ceq a = f(a) if a : S .
endfm
```

Note that both `a` and `f` do not have a sort, and are only defined at the *kind* level, using the kind `[S]` associated to the sort `S` (see Section 2.2). Note also that `f` has a strategy `(0)`, forbidding reductions in the argument of `f`. MAUDE fails to terminate when trying to reduce the term `a`. The problem is that, to compute the sort of `a`, MAUDE tries to reduce `a` to canonical form. This is of course a correct proof attempt in membership rewriting logic that leads to the infinite proof tree

$$\dfrac{\dfrac{\dfrac{\dfrac{\dots}{\texttt{a}\rightarrow\texttt{f(a)}} \quad \texttt{f(a):s}}{\texttt{a:s}}}{\texttt{a}\rightarrow\texttt{f(a)}}}{\texttt{a:s}}$$

showing that `INF2` fails to be operationally terminating.

What these examples show, most strikingly the `PALINDROME` and `INF` specifications, is that termination of a declarative program may not involve rewriting at all, or, as in the case of `INF2`, may involve *both* rewriting and other computational relations. As we further explain in Section 2.3, one key advantage of the notion of operational termination is that it is parametric on the logic underlying the given programming language. In particular, it is useful to clarify termination issues for *conditional* specifications, even for the special case of rewriting specifications [29]. Intuitively, and this is for example illustrated by `INF2` above, the problem is that a conditional specification may have a terminating rewriting relation (`INF2` does, since it is the empty relation) and still be nonterminating by "looping" in evaluating a condition. Where some notions of conditional termination run aground, for example that of "effective termination" (see [29]), is in failing to give a proper account of such looping. In operational termination terms, any nonterminating behavior, either in the rewrite relation, or in a condition, or in any other computational relation, is both detected and characterized by the existence of an infinite proof tree.

1.2 Proving Termination of CS-MRTs by Program Transformation

In proving termination of a CS-MRT, an important goal is to exploit a wide range of standard termination tools. We achieve this goal by using a sequence of *theory transformations* that map the original program into increasingly simpler ones —each having the property that termination of the transformed program at each step ensures termination of the input program— until we reach a transformed program that we can enter into a tool. A CS-MRT may exhibit all the features (1)–(5) mentioned above. We transform it by applying two transformation steps eliminating, successively, features (1) and (2). In this paper we ignore (3), because indeed innermost rewriting with a conditional TRS is not clearly defined at present (see Section 6 for further discussion).

The endpoint of this transformation process is a TRS (Term Rewriting System) together with a replacement map μ (*modulo* a set of axioms). A substantial amount of research has already been devoted to the definition and implementation of techniques for proving termination of such context-sensitive TRSs (CS-TRSs) [2, 10, 15, 22, 27, 28, 42].

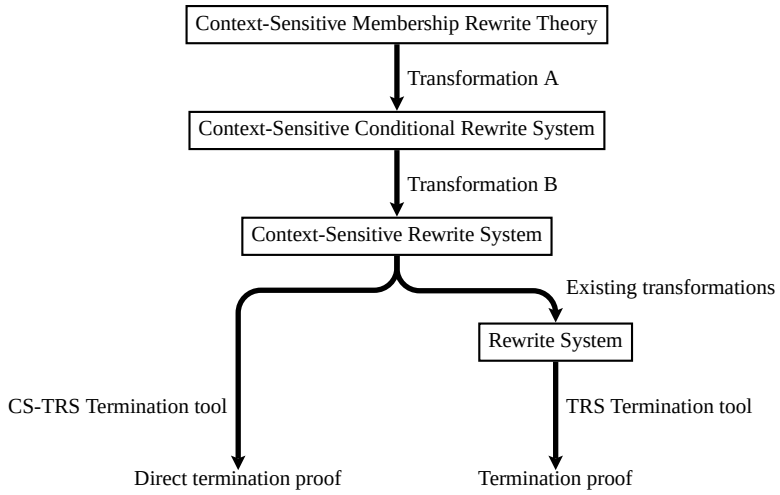

Fig. 2 Overview of the methodology

The sequence of theory transformations is summarized in Figure 2. Transformation A eliminates memberships and sorts (feature 1) resulting in an *unsorted*, *context-sensitive* and *conditional* rewrite theory. Transformation B eliminates conditions, possibly with extra variables (feature 2); it generalizes a known transformation from CTRSs to TRSs [35] in two ways: (i) by making it aware of context-sensitive rewriting information; and (ii) by allowing rewriting modulo axioms *Ax*. In this way we obtain an *unsorted* and *unconditional context-sensitive* rewrite theory.

We have implemented transformations A and B in the MAUDE Termination Tool (MTT, `http://www.lcc.uma.es/~duran/MTT/`). At this point, two options are available, leading to the forking in Figure 2. On the one hand, we can use a termination tool (such as MU-TERM) that can directly prove termination of *CSR* [2,27] (left branch). On the other hand, we can use several existing theory transformations, including those proposed by Lucas [22], Zantema [42], Ferreira and Ribeiro [10], and Giesl and Middeldorp [15] (see also [28]), to pass from a context-sensitive rewrite theory to an ordinary rewrite theory whose termination ensures that of the context-sensitive theory. These transformations are also implemented in MU-TERM and implicitly used in APROVE. The resulting theory can then be sent by MTT to a number of termination tools (namely C*i*ME, MU-TERM, and all tools supporting the TPDB syntax `http://www.lri.fr/~marche/termination-competition/`: APROVE, TTT [20], etc.)

This paper is organized as follows: in Section 2, we recall basics of CTRSs, Membership Equational Logic, and operational termination. Section 3 introduces Membership Rewrite Theories, and their operational semantics. In Section 4 we describe our theory transformations and prove their soundness w.r.t termination: transformation A is defined in Section 4.1, transformation B in Section 4.3. The example in Figure 1 is used as a running example for these transformations. In Section 5, we discuss implementation issues and experiments. We conclude with Section 6.

2 Preliminaries

2.1 Conditional Term Rewriting Systems

We refer the reader to [35] to recall the usual notions and notations regarding term rewriting and CTRSs. In general, a conditional rule is as follows:

$$l \to r \text{ if } s_1 = t_1, \cdots, s_n = t_n$$

where $l, r, s_1, t_1, \cdots, s_n, t_n$ are terms. l and r are called the left- and right-hand sides of the rule, and the sequence $s_1 = t_1, \cdots, s_n = t_n$ (often denoted c) is the *conditional part* of the rule. Rewrite rules $l \to r$ if c are classified according to the distribution of variables among l, r, and c, as follows: type 1, if $\mathrm{Var}(r) \cup \mathrm{Var}(c) \subseteq \mathrm{Var}(l)$; type 2, if $\mathrm{Var}(r) \subseteq \mathrm{Var}(l)$; type 3, if $\mathrm{Var}(r) \subseteq \mathrm{Var}(l) \cup \mathrm{Var}(c)$; type 4, if no restriction is given. An n-CTRS contains rewrite rules of type at most n.

It is well-known that the conditions $s_i = t_i$ for $1 \leq i \leq n$ can be interpreted in a number of different ways. *Join* CTRSs (often called *standard* CTRSs) interpret the equality symbol $=$ as joinability ($\downarrow_{\mathscr{R}}$). We are mainly concerned with *oriented* CTRSs [35], i.e., those whose (conditional) rules are written as follows:

$$l \to r \text{ if } s_1 \to t_1, \cdots, s_n \to t_n$$

indicating that the conditions $s_i \to t_i$ for $1 \leq i \leq n$ are intended to express the reachability, in arbitrary many steps, of (instances of) t_i from (instances of) s_i. A *normal* CTRS $\mathscr{R}$ is an oriented CTRS such that every t_i is a ground normal form (w.r.t. the unconditional TRS obtained by removing the conditional part from each conditional rule of $\mathscr{R}$) for $1 \leq i \leq n$. It is well-known that a join CTRS can be easily simulated by a normal CTRS by introducing new symbols *equal* and tt, adding the rule $equal(x,x) \to tt$, and encoding a condition $s = t$ into $equal(s,t) \to tt$ [31]. An oriented 3-CTRS $\mathscr{R}$ is called *deterministic* if for each $l \to r$ if $s_1 \to t_1, \ldots, s_n \to t_n$ in $\mathscr{R}$ and each $1 \leq i \leq n$, we have $\mathrm{Var}(s_i) \subseteq \mathrm{Var}(l) \cup \bigcup_{j=1}^{i-1} \mathrm{Var}(t_j)$.

Let $\mathscr{R}$ be a CTRS. We inductively define unconditional TRSs $\mathscr{R}_n$ for $n \in \mathbb{N}$ by $\mathscr{R}_0 = \emptyset$ and

$$\mathscr{R}_{n+1} = \{l\sigma \to r\sigma \mid l \to r \text{ if } s_1 \to t_1, \ldots, s_n \to t_n \in \mathscr{R} \wedge \forall i, s_i\sigma \to^*_{\mathscr{R}_n} t_i\sigma\}$$

The rewrite relation $\to_{\mathscr{R}}$ associated with a CTRS $\mathscr{R}$ is then $\to_{\mathscr{R}} = \bigcup_{n \in \mathbb{N}} \to_{\mathscr{R}_n}$.

In what follows we will need two further generalizations of the CTRS notion. First, we want to allow rewriting *modulo* a set Ax of equational axioms, so that matching of rules is performed with an Ax-matching algorithm. We therefore view such a CTRS as a triple $\mathscr{R} = (\Sigma, Ax, R)$ with Σ the signature of function symbols, Ax the equational axioms we rewrite modulo, and R the set of conditional rewrite rules. A second generalization is making rewriting *context-sensitive* [23,24] so that only certain function arguments are rewritten, whereas other arguments remain "frozen". For example, it is natural to restrict the evaluation of an **if-then-else** operator so that rewriting is only allowed on the first argument. In this way, we can express that the evaluation of the conditions only makes sense after evaluating the guard of the conditional expression. The simplest way of specifying requirements of this kind is to assume that there is a *replacement map* [23], i.e., a function $\mu : \Sigma \longrightarrow \mathscr{P}(\mathbf{N})$ 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. For example, $\mu(\textbf{if-then-else}) = \{1\}$, and in Example 1 $\mu(\mathrm{cons}) = \{1\}$. We then arrive at our most general CTRS notion, namely a context-sensitive CTRS (CS-CTRS) defined as a pair $(\mathscr{R}, \mu)$, with $\mathscr{R}$ a CTRS that may involve axioms Ax, and μ a replacement map.

An important advantage of context-sensitive rewriting is that rewrite systems that are nonterminating if rewriting is allowed on all term positions can often become terminating, and can also allow one to handle infinite data structures, such as in the example in Figure 1.

2.2 Membership Equational Theories

The simplest typed equational logic is many-sorted equational logic [34], in which function symbols are typed and each term has a sort. Order-sorted equational logic [18] generalizes this by allowing a subsort inclusion relation $s < s'$ between sorts, interpreted as subset inclusion in the models. In this way, some partial functions, hard to handle in a many-sorted setting, can become total. Membership equational logic [33, 3] further generalizes order-sorted equational logic, by allowing sorts and subsorts that are not defined just syntactically, as in the order-sorted setting, but whose domains of definition can be characterized by semantic conditions (see for example the definition of the `Pal` sort in the `PALINDROME` example in Section 1.1). This provides a general way of dealing with partial functions in equational specifications (which become total on appropriate sorts) and yields a logical framework into which many other equational formalisms, both partial and total, can be faithfully embedded [33]. As we explain below, by introducing a distinction between kinds and sorts, partiality can be achieved within a simple total setting.

We now explain in detail the syntax, models, and axioms of membership equational logic. A membership signature is a triple, $\Omega = (K, \Sigma, S)$, where (K, Σ) is a K-sorted signature, that is, K is a set, and Σ is an indexed family of sets $\Sigma = \{\Sigma_{w,k}\}_{(w,k)\in K^* \times K}$ —that we call "many-kinded" because the elements of K are called *kinds* so as to avoid confusion with the sorts S that are instead treated as predicates—and $S = \{S_k\}_{k\in K}$ is a disjoint family of unary predicates. Each $s \in S_k$ is called a *sort*, and is understood as a unary predicate on k, written $\_ : s$, so that elements satisfying the predicate determine the extension of the sort s in k. Intuitively, elements having some sort s are well-defined elements, whereas elements having a kind k but no sort are understood as error elements. For example, the term `f(a)` in the module `INF2` in Section 1.1 has kind `[S]` but has no sort; it should therefore be understood as an error or undefined element. Similarly, in a number hierarchy including sorts `Nat`, `Int`, and `Rat`, if we denote by `[Rat]` the corresponding kind to which all the above sorts belong, the term `7/0` has kind `[Rat]`, but has no sort and should therefore be understood as an error or undefined element.

Note that if in $\Omega = (K, \Sigma, S)$ the sets S_k are all empty for each kind $k \in K$, Ω becomes a standard many-sorted signature, and we obtain many-sorted equational logic as a special, degenerate case. However, since in this setting we wish to sharply distinguish between kinds and sorts, instead of calling a (K, Σ)-algebra a "many-sorted" algebra, we will now call it a many-kinded algebra. A model of Ω, called a membership algebra B is a (K, Σ)-algebra B together with an interpretation of each unary predicate $s \in S_k$ as a subset $B_s \subseteq B_k$. Ω-sentences are then universally quantified Horn clauses whose atomic predicates are either equalities $t = t'$ between two Σ-terms of the same kind, or unary membership predicates $t : s$ with t a Σ-term of kind k and $s \in S_k$. Therefore, such Horn clauses are either *conditional equations* (1) or conditional *memberships* (2):

$$t = t' \quad \text{if} \quad A_1, \ldots, A_n \tag{1}$$

$$t : s \quad \text{if} \quad A_1, \ldots, A_n \tag{2}$$

where the A_i are atomic equalities or memberships. In other words, membership equational logic is just the sublogic of many-sorted (although we see it here as "many-kinded")

Horn clause logic with equality in which all the predicates other than equality are unary. A membership equational *theory* is just a pair $T = (\Omega, E)$ with E a set of Ω-sentences. T-algebras are then Ω-algebras satisfying the clauses of T, according to the usual notion of satisfaction in many-sorted (again, seen as "many-kinded") first-order logic with equality. Given a membership equational theory T, there are free and initial T-algebras, and sound and complete inference rules [33]. Order-sorted notation $s_1 < s_2$ for subsorts can be used to abbreviate the conditional membership $(\forall x : k)\ x : s_2$ if $x : s_1$. Similarly, an operator declaration $f : s_1 \times \cdots \times s_n \to s$ corresponds to declaring f at the kind level and giving the membership axiom $(\forall x_1 : k_1, \ldots, x_n : k_n)\ f(x_1, \ldots, x_n) : s$ if $\bigwedge_{1 \leq i \leq n} x_i : s_i$. We write $(\forall x_1 : s_1, \ldots, x_n : s_n)\ t = t'$ in place of $(\forall x_1 : k_1, \ldots, x_n : k_n)\ t = t'$ if $\bigwedge_{1 \leq i \leq n} x_i : s_i$. The above abbreviations make it easy to embed order-sorted specifications as a special case of the more general membership equational specifications [33]. Specifically, an *order-sorted specification* is one in which: (1) the only memberships are subsort declarations $s_1 < s_2$ and operator declarations $f : s_1 \times \cdots \times s_n \to s$; and (2) the only other clauses in E are conditional equations of the form $(\forall x_1 : s_1, \ldots, x_n : s_n)\ t = t'$ if $\bigwedge_{1 \leq i \leq n} u_i = v_i$. The Maude language [5] supports all the order-sorted abbreviations just mentioned; furthermore, kinds do not have to be declared explicitly by the user: they are inferred by the system, that associates a kind to each connected component of sorts in the subsort ordering graph. For example, the specification in Figure 1 is order-sorted and has two kinds, corresponding to the connected components $\{\texttt{Nat}\}$ and $\{\texttt{NatList}, \texttt{NatIList}\}$. The first kind is denoted `[Nat]`, and the second kind can be equivalently denoted by either `[NatList]` or `[NatIList]`; that is, we represent kinds as equivalence classes of their corresponding sorts.

Admissible membership equational theories [5] provide a very general class of equational theories that are executable by equational rewriting. Their sentences are a union $E \cup Ax$, where Ax is a collection of equational axioms such as, for example, associativity, commutativity, and identity of some operators in Σ, for which a matching algorithm modulo Ax exists; we furthermore assume that the axioms Ax are unconditional and are defined *at the kind level*, that is, the variables in such axioms have kinds and do not involve any restrictions to sorts. The set E consists of conditional equations (1) and conditional memberships (2), where in (1) the variables in t' are among those in t or in some A_i, and where, in both (1) and (2) each A_i is either a membership $w_i : s_i$, or an equation $u_i = v_i$ such that any new variable not in t or in some A_j with $j < i$ must occur only in u_i or in some A_j with $j > i$; furthermore, if u_i introduces any new variables, then u_i must be a nonvariable term; we then call $u_i = v_i$ a *matching equation*. In MAUDE such matching equations are distinguished syntactically with the notation $u_i := v_i$.

2.3 Operational termination

We consider a logic $\mathscr{L}$ defined by inference rules, parameterized by a *theory* $\mathscr{S}$. That is, we focus on provability, and assume the axiomatic framework of general logics [32], in which what we call a *logic* becomes a particular style of presenting an *entailment system*. We refer to [4] for a more detailed account of the axiomatic metalogical background that we assume in what follows. The notion of *operational termination* [29] is *parametric* on the inference system. We briefly recall the notions we need for our purpose.

Definition 1 The set of (finite) proof trees for a theory $\mathscr{S}$ in a logic $\mathscr{L}$ and the head of a proof tree are defined inductively as follows. A *proof tree* is

– either an *open goal*, simply denoted as φ, where φ is a formula for $\mathscr{S}$; then, we define $head(\varphi) = \varphi$.

– or a *non-atomic* tree with φ as its head, denoted as

$$\frac{T_1 \quad \cdots \quad T_n}{\varphi} \quad (\Delta)$$

where φ is a formula for $\mathscr{S}$, Δ is an inference rule in $\mathscr{L}$, and T_1,...,T_n are proof trees such that

$$\frac{head(T_1) \quad \cdots \quad head(T_n)}{\varphi}$$

is an instance of Δ for the theory $\mathscr{S}$.

We say that a proof tree is *closed* whenever it is finite and contains no open goals.[1]

Notice the difference between φ, an open goal, and $\overline{\varphi}$, a goal closed by a rule without premises.

Definition 2 A proof tree T is a *proper prefix* of a proof tree T' if there are one or more open goals $\varphi_1,\ldots,\varphi_n$ in T such that T' is obtained from T by replacing each φ_i by a non-atomic proof tree T_i having φ_i as its head. We denote this as $T \subset T'$.

An *infinite proof tree* is an infinite increasing chain of finite trees, that is, a sequence $\{T_i\}_{i\in\mathbb{N}}$ such that for all i, $T_i \subset T_{i+1}$.

We characterize the proof trees with computational meaning (those which are computed by an *interpreter* [29]), by means of the notion of well-formed proof tree.

Definition 3 We say that a proof tree T is *well-formed* if it is either an open goal, or a closed proof tree, or a proof tree of the form

$$\frac{T_1 \quad \cdots \quad T_n}{\varphi} \quad (\Delta)$$

where for each j T_j is itself well-formed, and there is $i \leq n$ such that T_i is not closed, for any $j < i$ T_j is closed, and each of the T_{i+1} ,...,T_n is an open goal. An infinite proof tree is *well-formed* if it is an ascending chain of well-formed finite proof trees. $\mathscr{S}$ is called *operationally terminating* if no infinite well-formed tree for $\mathscr{S}$ exists.

So operational termination intuitively means that, given an initial goal, an interpreter that solves goals from left to right will either succeed in finite time in producing a closed proof tree, or will fail in finite time, not being able to close or extend further any of the possible proof trees, after exhaustively searching all such proof trees.

3 Rewriting with Membership Equational Theories

In the spirit of [3], we can associate to an admissible membership equational theory[2] $T = (\Omega, E \cup Ax)$ a corresponding (conditional) *membership rewrite theory* $\mathscr{R}_T = (\Omega', Ax, R_T)$ defined as follows. The signature of Ω' adds a fresh new kind Truth with a constant tt to

[1] Open goals appear at the leaves of a proof tree; but they can be *closed* by the application of inference rules with no premises. For example, an open goal $t \rightarrow t$ can be closed by applying a Reflexivity inference rule.

[2] As in [3], admissible theories $T = (\Omega, E \cup Ax)$ will always be assumed to have *non-empty kinds*, that is, for each kind k in Ω there is always a ground term of kind k.

```
fmod LengthOfFiniteListsMRT is
   kind [Nat].
   kind [NatIList] .
   op 0 : -> [Nat] .
   op s : [Nat] -> [Nat] .
   op zeros : -> [NatIList] .
   op nil : -> [NatList] .
   op cons : [Nat] [NatIList] -> [NatIList] [strat (1)] .
   op length : [NatIList] -> [Nat] .
   cmb L : NatIList if L : NatList .
   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 length(L) : Nat if L : NatList .
   eq zeros = cons(0,zeros) .
   eq length(nil) = 0 .
   ceq length(cons(N,L)) = s(length(L))
      if N : Nat /\ L : NatList .
endfm
```

Fig. 3 CS-MRT (in MAUDE syntax) for the program `LengthOfFiniteLists`

Ω, and for each kind k in T an operator equal : $k\,k \longrightarrow$ Truth. $\mathscr{R}_T$ has the same equational axioms Ax as T, so that rewriting is performed modulo Ax, and contains rules of the form $\mathrm{equal}(x,x) \rightarrow \mathrm{tt}$ for each kind k in T. Furthermore, for each admissible conditional equation of the form (1) in E the set R_T has a conditional rule of the form

$$t \rightarrow t' \quad \text{if} \quad A_1^\bullet, \ldots, A_n^\bullet \tag{3}$$

where if A_i is a membership then $A_i^\bullet = A_i$, if A_i is a matching equation $u_i = v_i$, then $A_i^\bullet$ is the rewrite condition $v_i \rightarrow^* u_i$, and if A_i is an ordinary equation $u_i = v_i$, then $A_i^\bullet$ is the rewrite condition $\mathrm{equal}(u_i, v_i) \rightarrow^* \mathrm{tt}$. Similarly, for each conditional membership in T of the form (2) we associate a conditional membership of the form,

$$t : s \quad \text{if} \quad A_1^\bullet, \ldots, A_n^\bullet \tag{4}$$

with the $A_i^\bullet$ defined exactly as before.

The point of associating to an admissible membership equational theory T a corresponding rewrite theory $\mathscr{R}_T$ is that we can perform equational reasoning by rewriting. Of course, unless $\mathscr{R}_T$ satisfies additional properties such as confluence, sort-decreasingness [3], Ax-coherence [41], and so on, equational reasoning by rewriting will only be sound but not necessarily complete.

Equational reasoning in a membership equational theory T by rewriting with the rules in $\mathscr{R}_T$ modulo the axioms Ax can be made more expressive by making the rewriting *context-sensitive* in the sense explained in Section 2.1. Therefore, we define a *context-sensitive membership rewrite theory* (CS-MRT) as a pair $(\mathscr{R}_T, \mu)$, where $\mathscr{R}_T$ is a membership rewrite theory, say, $\mathscr{R}_T = (\Omega', Ax, R_T)$, and the context information is provided by a replacement map μ. For instance, the CS-MRT specification (also given in MAUDE-like notation) which corresponds to the MAUDE program in Figure 1 is given in Figure 3. Here, `[Nat]` denotes the kind of sort `Nat`, and `[NatIList]` denotes the kind of both sorts `NatList` and `NatIList`. The profile of the operators is given in terms of these kinds. We omit the operator `equal` as no conditional rule includes equations in its conditional part. Note also the

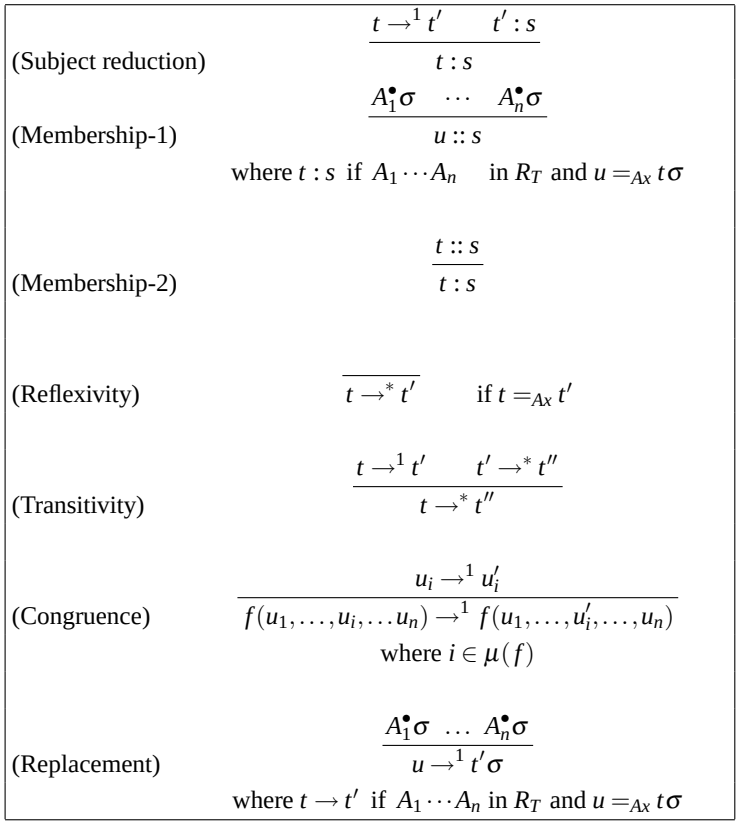

Fig. 4 Inference rules for context-sensitive membership rewriting

first conditional membership (with keyword `cmb`) which expresses that `NatList` is a subsort of `NatIList`. The sort profile for the arguments and result of each operator in the MAUDE program `LengthOfFiniteLists` are desugared here as memberships in the CS-MRT specification. In particular, viewing the sort profile of a function symbol as a shorthand for a kind profile together with a membership, such as for `cons` above, allows us to cleanly handle operator overloading: to each different sort profile corresponds a different membership. We also allow *ad-hoc* overloading, that is, operators with same name and different kind profile, although in that case we require that if f has kind profiles $k_1 \cdots k_n \to k$ and $k_1 \cdots k_n \to k'$, then $k = k'$.

We can define the rewriting relation associated to a CS-MRT by means of the inference rules of Figure 4, which[3] generalize to rewriting modulo Ax and adapt to the context-sensitive case those in Figure 7 in [3]. Note that inferences can now happen *modulo* the equational axioms Ax in the theory: matching with a conditional equation in the Replace-

[3] 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 by assuming that either: (i) the implementation performs *Ax-matching with extension* (as done, e.g., in Maude: see [6], Section 4.8); or (ii) the rules in our CS-MRT have been completed to be *coherent* with the axioms Ax (see [41]). Under either of these two assumptions, the simpler inference rules in Figure 4 are complete.

ment inference rule, and with a conditional membership in the Membership-1 rule, is performed *modulo Ax*; and Reflexivity also includes equality modulo *Ax*. Note also that the relation $t : s$ has a subrelation $t :: s$, corresponding to the special case of a membership in which the term t is not further rewritten before computing its sort. For each atom A appearing in a condition of a conditional rule or a conditional membership in $(\mathscr{R}_T, \mu)$ we extend our previous meta-notation $A^\bullet$ to memberships as follows: (1) if A is of the form $x : s$ with x a variable, then $A^\bullet = x :: s$; and if A is of the form $w : s$ with w a nonvariable term, then $A^\bullet = w : s$. The obtained inference system is context sensitive in a quite detailed way. The most obvious case is the restriction on the Congruence rule, which blocks rewriting in frozen argument positions; further context sensitivity is achieved through the $A^\bullet$ conjuncts in the conditions of the Membership-1 and Replacement rules. The point is that, if unrestricted, these inference rules could easily undermine context-sensitivity by evaluating subterms that are supposed to be frozen, thus easily leading to nontermination (see [9] for an example). This is prevented by the case when $A = x : s$, since then $A^\bullet = x :: s$. This means that if x matches a subterm of the term whose sort we are computing with the Membership-1 rule —or that we are trying to rewrite with the Replacement rule— then that subterm will not be further rewritten in the process of checking its sort.

We can use the notion of operational termination to explain the behavior of the nonterminating examples in Section 1.1. Note that, because of the distinction between $t : s$ and $t :: s$, the infinite proof tree for the `INF` module has to be expanded, alternating applications of (Membership-1) and (Membership-2) rules. In this way, we indeed obtain a well-formed infinite proof tree. Note that this example does not involve any rewriting. The given infinite proof tree for module `INF2` does not require any modification and is indeed well-formed.

The notion of CS-MRT just defined and its associated inference rules capture in particular the case of MAUDE functional modules. Indeed, a MAUDE functional module, after explicitly importing all its submodules and desugaring its subsort declarations and operator declarations as explicit conditional memberships (see Section 2.2 and the example in Figure 3), defines a membership equational theory $T = (\Sigma, E \cup Ax)$ which is required and checked to be admissible. In MAUDE, the axioms Ax are not explicitly declared by equations; they are instead declared as *operator attributes* of associativity, commutativity, and identity with the `assoc`, `comm`, and `id:` keywords. Furthermore, a MAUDE functional module defines a replacement map μ by means of the `strat` operator attribute, where if f has been declared with the strategy $(i_1 \ldots i_k)$ then $\mu(f) = \{i_1, \ldots, i_k)\} - \{0\}$, and if no such strategy has been declared for an n-ary f, then $\mu(f) = \{1, \ldots, n\}$. By the general transformation defined in this section, the admissible MEL theory T and the replacement map μ defined by the MAUDE functional module are then transformed into the CS-MRT $(\mathscr{R}_T, \mu)$. The MAUDE interpreter then provides a sequential strategy to apply the inference rules in Figure 4 in a specific order. Therefore, MAUDE's computations for a functional module of the form (T, μ) are a subset of those permitted by the inference system in Figure 4 for the CS-MRT $(\mathscr{R}_T, \mu)$.

4 Operational termination of CS-MRT programs

To close the gap between MEL programs and current termination tools, we define and prove correct two theory transformations, namely transformation A, mapping a CS-MRT to a corresponding CS-CTRS in such a way that operational termination of the CS-CTRS implies that of the original CS-MRT, and transformation B, mapping CS-CTRSs to CS-TRSs such that termination of the CS-TRS implies operational termination of the CS-CTRSs.

4.1 Transformation A: From CS-MRT modulo to CS-CTRS modulo

Given a CS-MRT $(\mathscr{R}_T, \mu)$, say with $\mathscr{R}_T = (K, \Sigma, S, Ax, R_T)$, we associate to it a CS-CTRS $(\widetilde{\mathscr{R}}_T, \widetilde{\mu})$ over a signature $\widetilde{\Sigma}$, modulo axioms $\widetilde{Ax}$ as follows: $\widetilde{\Sigma}$ contains, for each operator $f : w \to k$ where $w = k_1 \ldots k_n$ in Σ, an operator f^w of arity n. We furthermore add a truth-value constant tt, plus unary operators $is_k \in \widetilde{\Sigma}$ for each $k \in K$, and $is_s, is'_s \in \widetilde{\Sigma}$ for each $s \in S$, where $is_s(t)$ encodes $t :: s$ and $is'_s(t)$ encodes $t : s$. The role of operators f^w is to disambiguate ad-hoc overloading: for each Σ-term t, the $\widetilde{\Sigma}$-term $\widetilde{t}$ is obtained by making its variables unsorted, and by replacing each $f : w \to k$ by f^w. We assume that there is only one k for each w, so this operation is well-defined. The axioms $\widetilde{Ax}$ are just the equations $\widetilde{t} = \widetilde{t'}$ for each $t = t'$ in Ax. The set of rules $\widetilde{R}$ is given by $\widetilde{R} = R_K \cup R_S \cup R_C \cup R_M$, where R_K contains rules of the form

$$is_k(f^w(x_1, \ldots, x_n)) \to \text{tt} \quad \text{if} \quad \{is_{k_i}(x_i) \to \text{tt}\}_{1 \le i \le n} \tag{5}$$

for each $f : w \longrightarrow k$ in Σ, with $w = k_1 \ldots k_n$. The set R_S contains rewrite rules of the form $is'_s(x) \to is_s(x)$ for each sort $s \in S$ (to encode the (Membership-2) rule). The set R_C contains a conditional rule of the form,

$$\widetilde{t} \to \widetilde{t'} \quad \text{if} \quad \{is_{k_i}(x_i) \to \text{tt}\}_{1 \le i \le m}, \widetilde{A}_1, \ldots, \widetilde{A}_n \tag{6}$$

for each conditional rule $t \to t'$ if $A_1, \ldots, A_n$ in R_T involving variables $x_1 : k_1, \ldots, x_m : k_m$; here if A_i is a membership $u_i : s_i$, then: (i) if u_i is a nonvariable term, then $\widetilde{A}_i$ is the rewrite condition $is'_{s_i}(\widetilde{u_i}) \to \text{tt}$, and (ii) if $u_i \equiv x$ is a variable, then $\widetilde{A}_i$ is the rewrite condition $is_{s_i}(x) \to$ tt; otherwise, if A_i is a rewrite condition $u_i \to v_i$, then $\widetilde{A}_i$ is the rewrite condition $\widetilde{u_i} \to \widetilde{v_i}$. Finally, R_M contains a conditional rule of the form,

$$is_s(\widetilde{t}) \to \text{tt} \ \text{ if } \ \{is_{k_i}(x_i) \to \text{tt}\}_{1 \le i \le m}, \widetilde{A}_1, \ldots, \widetilde{A}_n. \tag{7}$$

for each conditional membership $t : s$ if $A_1, \ldots, A_n$ in R_T involving variables $x_1 : k_1, \ldots x_m : k_m$.

Regarding the replacement map $\widetilde{\mu}$, we define $\widetilde{\mu}(f^w) = \mu(f)$ for each $f : w \longrightarrow k$ in Σ, and for each $k \in K$ and each $s \in S$ we define $\widetilde{\mu}(is_k) = \emptyset$, $\widetilde{\mu}(is_s) = \emptyset$, and $\widetilde{\mu}(is'_s) = \{1\}$ (because one may reduce t in $t : s$ but not in $t :: s$).

The following theorem connects operational termination in the CS-MRT logic, given by the inference rules in Figure 4, and operational termination in the CS-CTRS logic, whose inference system consists of the inference rules of Reflexivity, Transitivity, Congruence and Replacement in Figure 4 above.

Theorem 1 *If the CS-CTRS $(\widetilde{\mathscr{R}}_T, \widetilde{\mu})$ is operationally terminating, then the CS-MRT $(\mathscr{R}_T, \mu)$ is operationally terminating.*

The proof is as follows: we show that any well-formed infinite ground[4] proof tree for $(\mathscr{R}_T, \mu)$ can be transformed into a well-formed infinite proof tree for $(\widetilde{\mathscr{R}}_T, \widetilde{\mu})$, using the following lemma.

Lemma 1 *For any well-formed ground proof tree Q for $(\mathscr{R}_T, \mu)$, there exists a well-formed ground proof tree $\alpha(Q)$ for $(\widetilde{\mathscr{R}}_T, \widetilde{\mu})$ whose head goal is*

[4] Since admissible membership equational theories have nonempty kinds, the operational termination of a CS-MRT is equivalent to the operational termination of its *ground* proofs. This is because, given an infinite proof tree T, we can always find a ground substituition σ yielding an infinite ground proof tree $\sigma(T)$ obtained by applying σ to all terms in T. Therefore, in what follows we reason in terms of ground proof trees.

- $is'_s(\widetilde{t}) \to^* \mathrm{tt}$ *if the head goal of* Q *was* $t : s$
- $is_s(\widetilde{t}) \to^1 \mathrm{tt}$ *if the head goal of* Q *was* $t :: s$
- $\widetilde{t} \to^* \widetilde{u}$ *if the head goal of* Q *was* $t \to^* u$
- $\widetilde{t} \to^1 \widetilde{u}$ *if the head goal of* Q *was* $t \to^1 u$

Moreover, if $Q \subset Q'$ *then* $\alpha(Q) \subset \alpha(Q')$*, so that for any infinite proof tree* Q*,* $\alpha(Q)$ *is infinite.*

So we are left to prove the lemma above. For this, we need an auxiliary lemma about substitutions, well-kinded terms, and equality modulo axioms. The first two statements in this lemma can be proved by straightforward structural induction, making use of rules of type (5). The proof of the last statement follows easily by induction on the length of proofs from the first statement, observing that, as pointed out in Section 2.2, the axioms Ax are given at the kind level.

Lemma 2 *For any term* t*, substitution* σ*, and condition* c*, we have* $\widetilde{t\sigma} = \widetilde{t}\ \widetilde{\sigma}$ *and* $\widetilde{(c\sigma)} = \widetilde{c}\ \widetilde{\sigma}$*. Furthermore, if* t *is a well-kinded ground term of kind* k *w.r.t* $(\mathscr{R}_T, \mu)$*, then* $(\widetilde{\mathscr{R}}_T, \widetilde{\mu}) \vdash is_k(\widetilde{t}) \to^1 \mathrm{tt}$*. Finally, for* t, t' *terms* $t =_{Ax} t'$ *implies* $\widetilde{t} =_{\widetilde{Ax}} \widetilde{t'}$*.*

Proof Obvious.

Transformation A is a map α defined by induction on the structure of well-formed ground trees. The base case corresponds to ground proof trees consisting of a single atom, for which α is defined according to the translation of head goals stated in Lemma 1. We now have to consider each of the inference rules and define α by cases. Suppose a well-formed ground proof tree Q where the fist inference step is an application of the Membership-1 inference rule, then this tree looks as follows:

$$\dfrac{\dfrac{T_1}{A_1^\bullet\sigma} \quad \cdots \quad \dfrac{T_n}{A_n^\bullet\sigma}}{u :: s}$$

where $t : s$ if $A_1 \cdots A_n$ in R_T and $u =_{Ax} t\sigma$. Therefore, in $(\widetilde{\mathscr{R}}_T, \widetilde{\mu})$ we have a conditional rewrite rule $is_s(\widetilde{t}) \to \mathrm{tt}$ if $\{is_{k_i}(x_i) \to \mathrm{tt}\}_{1 \le i \le m}, \widetilde{A}_1, \ldots, \widetilde{A}_n$. Then the ground proof tree $\alpha(Q)$ has the following form:

$$\dfrac{\dfrac{Q_1}{is_{k_1}(\widetilde{x_1\sigma}) \to^1 \mathrm{tt}} \quad \cdots \quad \dfrac{Q_m}{is_{k_m}(\widetilde{x_m\sigma}) \to^1 \mathrm{tt}} \quad \dfrac{\alpha(T_1)}{\alpha(A_1^\bullet\sigma)} \quad \cdots \quad \dfrac{\alpha(T_n)}{\alpha(A_n^\bullet\sigma)}}{is_s(\widetilde{u}) \to^1 \mathrm{tt}}$$

where the Replacement rule is correctly applied since $is_s(\widetilde{u}) =_{\widetilde{Ax}} is_s(\widetilde{t\sigma})$ by Lemma 2, the $Q_1, \ldots, Q_m$ are closed proof trees that exist by Lemma 2, and it is easy to show that they are unique, due to the assumptions making well-kinded terms unambiguous.

The case where the first inference step is the application of the Replacement inference rule is entirely analogous to the above case, except that the root goal $u \to^1 t'\sigma$ is now translated into the root goal $\widetilde{u} \to^1 \widetilde{t'}\widetilde{\sigma}$.

When the first inference step is the application of the Membership-2 inference rule, we have a well-formed ground proof tree Q of the form

$$\dfrac{\dfrac{T'}{t :: s}}{t : s}$$

and then $\alpha(Q)$ is of the form

$$\dfrac{\dfrac{}{is'_s(\widetilde{t}) \to^1 is_s(\widetilde{t})} \quad \dfrac{\dfrac{\alpha(T')}{is_s(\widetilde{t}) \to^1 \mathrm{tt}} \quad \mathrm{tt} \to^* \mathrm{tt}}{is_s(\widetilde{t}) \to^* \mathrm{tt}}}{is'_s(\widetilde{t}) \to^* \mathrm{tt}}$$

where we have applied the Transitivity rule to the root goal $is_s(\widetilde{t}) \to^* \mathrm{tt}$, and the Replacement rule to close the goal $is'_s(\widetilde{t}) \to^1 is_s(\widetilde{t})$.

When the first inference step is the application of the Subject Reduction inference rule, we have a well-formed ground proof tree Q of the form

$$\dfrac{\dfrac{T_1}{t \to^1 t'} \quad \dfrac{T_2}{t' : s}}{t : s}$$

and then $\alpha(Q)$ is of the form

$$\dfrac{\dfrac{\dfrac{\alpha(T_1)}{\widetilde{t} \to^1 \widetilde{t'}}}{is'_s(\widetilde{t}) \to^1 is'_s(\widetilde{t'})} \quad \dfrac{\alpha(T_2)}{is'_s(\widetilde{t'}) \to^* \mathrm{tt}}}{is'_s(\widetilde{t}) \to^* \mathrm{tt}}$$

where we have applied the Transitivity rule to the root goal $is'_s(\widetilde{t}) \to^* \mathrm{tt}$, and the Congruence rule to the goal $is'_s(\widetilde{t}) \to^1 is'_s(\widetilde{t'})$.

The translations $\alpha(Q)$ of a well-formed ground proof tree Q where the first inference step is the application of any of the remaining inference rules, namely, Reflexivity, Transitivity, or Congruence, all follow a very simple pattern, namely, if the proof tree Q is of the form

$$\dfrac{T_1 \quad \cdots \quad T_n}{G}$$

then $\alpha(Q)$ is of the form

$$\dfrac{\alpha(T_1) \quad \cdots \quad \alpha(T_n)}{\alpha(G)}$$

Note that in the case of Reflexivity, we use again Lemma 2 to replace equality modulo Ax by equality modulo $\widetilde{Ax}$.

It is also easy to check that α maps well-formed ground proof trees to well-formed ones, and that if $T \subset T'$, then $\alpha(T) \subset \alpha(T')$. To check this last property, we may assume without loss of generality, that $T \subseteq T'$ is the extension of T associated to the application of an inference rule. The result then follows easily by case analysis on the inference rule used and the definition of the corresponding tree extensions given above for each of the inference rules.

This ends the proof of Lemma 1 and therefore finishes the proof of Theorem 1. □

For purposes of proving termination, the implication in Theorem 1 is all we need. However, it is natural to ask whether Transformation A is *complete*, that is, is the implication in Theorem 1 actually an equivalence? We conjecture that it is an equivalence, and therefore that Transformation A is complete, but leave a detailed investigation of this problem for future research.

```
fmod LengthOfFiniteListsMRT_TA is
   sort S .
   op isKNat : S -> S [strat (0)] .
   op isKNatIList : S -> S [strat (0)] .
   op isNat : S -> S [strat (0)] .
   op isNatIList : S -> S [strat (0)] .
   op isNatList : S -> S [strat (0)] .
   op tt : -> S .
   op 0 : -> S .
   op s : S -> S .
   op zeros : -> S .
   op nil : -> S .
   op cons : S S -> S [strat (1 0)] .
   op length : S -> S .
   vars T M N IL L : S .
   eq isKNat(0) = tt .
   ceq isKNat(s(N)) = tt if isKNat(N) = tt .
   ceq isKNat(length(L)) = tt if isKNatIList(L) = tt .
   eq isKNatIList(nil) = tt .
   eq isKNatIList(zeros) = tt .
   ceq isKNatIList(cons(N,IL)) = tt
           if isKNat(N) = tt /\ isKNatIList(IL) = tt .
   ceq isNatIList(IL) = tt if isNatList(IL) = tt .
   eq isNat(0) = tt .
   ceq isNat(s(N)) = tt if isNat(N) = tt .
   ceq isNat(length(L)) = tt if isNatList(L) = tt .
   eq isNatIList(zeros) = tt .
   ceq isNatIList(cons(N,IL)) = tt
             if isNat(N) = tt /\ isNatIList(IL) = tt .
   eq isNatList(nil) = tt .
   ceq isNatList(cons(N,L)) = tt
              if isNat(N) = tt /\ isNatList(L) = tt .
   eq zeros = cons(0,zeros) .
   eq length(nil) = 0 .
   ceq length(cons(N,L)) = s(length(L))
            if isKNat(N) = tt /\ isKNatList(L) = tt /\
                  isNat(N) = tt /\ isNatList(L) = tt .
endfm
```

Fig. 5 Use of transformation A

Example 2 For our running example, we would get the transformed system in Figure 5. We have omitted the disambiguation of operators, since no ambiguity is involved in this example; also, equal has been omitted.

4.2 Variants of Transformation A

In order to provide the simplest input for the next transformation which removes conditions from rules (see Section 4.3), we can apply some obvious optimizations on the previous transformation which do not change the termination behavior of the program. The benefits of using these optimizations can be experimentally justified from the benchmarks discussed in Section 5.2 below.

4.2.1 Removal of kinds

In a first variant, the is_k predicates for kinds are omitted. This simplifies the resulting theory with minimal loss in its expressiveness, particularly for specifications in which, as it is usually the case, all variables of a conditional equation or rule are required to have a sort in the condition.

If all operator profiles involve only sorts, and all variables appearing in equations and memberships have a declared sort, then if k is the kind of a sort s, then $is_s(x) \to tt$ implies $is_k(x) \to tt$. Therefore, we can safely use $is_s(x) \to tt$ instead of $is_k(x) \to tt \wedge is_s(x) \to tt$ in the conditional part of the rules computed by the transformation.

4.2.2 Simplifying conditions

A conditional fragment *without extra variables* like

$$is_{s_1}(x_1) \to tt \wedge \cdots \wedge is_{s_k}(x_k) \to tt$$

in a conditional rule can be collapsed into a single expression

$$and(is_{s_1}(x_1), and(\cdots, is_{s_k}(x_k))\cdots) \to tt$$

by introducing a binary '`and`' operator defined by

```
op and : S S -> S .
eq and(tt,T) = T .
```

Moreover, if the right-hand side of the conditional rule is tt, we can use the previous expression with `and` as the new right hand-side of the rule: the conditional rule

$$l \to tt \text{ if } is_{s_1}(x_1) \to tt \wedge \cdots \wedge is_{s_k}(x_k) \to tt$$

eventually collapses into the unconditional one

$$l \to and(is_{s_1}(x_1), and(\cdots, is_{s_k}(x_k))\cdots)$$

This ends up with less symbols to be processed, and only one added rule instead of (potentially) several mutually recursive rules, thus easing the task of the termination tool.

For instance, with the two previous variants, the equations of the system in Figure 3 become the ones shown in Figure 6.

4.2.3 A variant for order-sorted theories

In this section we consider a much simpler variant of the transformation $(\mathscr{R}, \mu) \mapsto (\widetilde{\mathscr{R}}, \tilde{\mu})$ just defined. For order-sorted rewrite theories, which are the special case where the only memberships involved in conditions are variables, and the only membership axioms correspond to subsort and operator declarations (see Section 2.2), this variant drops also the is_s predicates for sorts. This variant is correct *only* for order-sorted theories, for example it would be invalid for the INF program of Section 1 which contains a membership but no rule, since one would get an empty TRS.

```
eq and(tt,T) = T .
eq isNatIList(IL) = isNatList(IL) .
eq isNat(0) = tt .
eq isNat(s(N)) = isNat(N) .
eq isNat(length(L)) = isNatList(L) .
eq isNatIList(zeros) = tt .
eq isNatIList(cons(N,IL)) = and(isNat(N),isNatIList(IL)) .
eq isNatList(nil) = tt .
eq isNatList(cons(N,L)) = and(isNat(N),isNatList(L)) .
eq zeros = cons(0,zeros) .
eq length(nil) = 0 .
ceq length(cons(N,L)) = s(length(L))
   if and(isNat(N),isNatList(L))) = tt .
```

Fig. 6 Optimized transformation A

4.2.4 Incompleteness

Obviously, since these simpler variants yield less restrictive conditions in the translated rules in $\widetilde{\mathscr{R}}$, these variants allow more rewrites and therefore our results apply to these simpler transformations, in the sense that a proof of operational termination for the transformed theory ensures operational termination of the original theory. But of course, these variants are incomplete. For instance, it is not possible to use variant 3 to prove termination of program `LengthOfFiniteLists` in Figure 1. In fact, the obtained CS-TRS:

```
zeros → cons(0,zeros)
length(nil) → 0
length(cons(N,L)) → s(length(L))
```

with $\mu(\texttt{cons}) = \{1\}$ is not (operationally) terminating

$$\texttt{length(}\underline{\texttt{zeros}}\texttt{)} \rightarrow \underline{\texttt{length(cons(0,zeros))}} \rightarrow \texttt{s(length(}\underline{\texttt{zeros}}\texttt{))} \rightarrow \cdots$$

4.3 Transformation B: From CS-CTRS modulo to CS-TRS modulo

To check operational termination with respect to the CS-CTRS logic, we propose a transformation associating to a CS-CTRS $(\mathscr{R},\mu)$ an unconditional CS-TRS $(\mathscr{U}(\mathscr{R}),\mathscr{U}(\mu))$. We generalize the classical transformation for proving operational termination of a 3-CTRS $\mathscr{R}$ as termination of a TRS $\mathscr{U}(\mathscr{R})$ [35, Definition 7.2.48], so as to handle both rewriting modulo axioms Ax, and the context-sensitive restrictions imposed by the replacement map μ. The classical transformation for proving termination of a deterministic 3-CTRS $\mathscr{R}$ yields a TRS $\mathscr{U}(\mathscr{R})$ given as follows: each conditional rule

$$l \rightarrow r \text{ if } s_1 \rightarrow t_1, \ldots, s_n \rightarrow t_n$$

is transformed into the $n+1$ unconditional rules

$$l \rightarrow U_1(s_1, \mathbf{x_1}) \tag{8}$$

$$U_{i-1}(t_{i-1}, \mathbf{x_{i-1}}) \rightarrow U_i(s_i, \mathbf{x_i}) \qquad 2 \leq i \leq n \tag{9}$$

$$U_n(t_n, \mathbf{x_n}) \rightarrow r \tag{10}$$

where the U_i are fresh new symbols added to the signature. The $\mathbf{x_i}$ are vectors of variables defined as follows: assume a given ordering on the set of variables $\mathscr{X}$. Then, $\mathbf{x_i}$ contains the

ordered sequence of the variables in the set $\mathrm{Var}(l) \cup \mathrm{Var}(t_1) \cup \cdots \cup \mathrm{Var}(t_{i-1})$ 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:

$$\begin{aligned} \mathbf{x_i} = \; & (\mathrm{Var}(l) \cup \mathrm{Var}(t_1) \cup \cdots \cup \mathrm{Var}(t_{i-1})) \\ \cap \quad & (\mathrm{Var}(t_i) \cup \mathrm{Var}(s_{i+1}) \cup \mathrm{Var}(t_{i+1}) \cup \cdots \\ \cup \quad & \mathrm{Var}(s_n) \cup \mathrm{Var}(t_n) \cup \mathrm{Var}(r)) \end{aligned}$$

In our approach, we allow rewriting modulo *Ax* and also transform the replacement map into a new replacement map $\mathscr{U}(\mu)$ as follows: $\mathscr{U}(\mu)(U) = \{1\}$ for all new symbols U that are introduced to deal with the equations in the conditional part of each rule in $\mathscr{R}$ (that is, only the first argument of U can be evaluated), and $\mathscr{U}(\mu)(f) = \mu(f)$ for all symbols $f \in \mathscr{F}$.

Example 3 For our running example (in the optimized version given in Figure 6), the corresponding unconditional translation of the only conditional rule consists of the rules:

```
length(cons(N,L)) -> uLength(and(isNat(N),isNatList(L)),L)
uLength(tt,L) -> s(length(L))
```

where we also have $\mathscr{U}(\mu)(\texttt{uLength}) = \{1\}$.

Theorem 2 *If $\mathscr{U}(\mathscr{R})$ is $\mathscr{U}(\mu)$-terminating modulo Ax, then $(\mathscr{R}, \mu)$ is operationally terminating modulo Ax.*

Note that in [29], we showed that for CTRSs, operational termination is equivalent to the so-called *quasi-decreasingness* property; and it is already known that for a standard CTRS $\mathscr{R}$, termination of $\mathscr{U}(\mathscr{R})$ implies quasi-decreasingness of $\mathscr{R}$ [35, Proposition 7.2.50 and Lemma 7.2.40]. So our theorem above is a generalization of this result to the case of context-sensitive rewriting modulo *Ax*. However, the proof we give below is completely different: the reason is that, although the proof of the result based on the quasi-decreasingness property can be extended to the context-sensitive case, it is however not clear how to further extend it to the modulo *Ax* case, because it would require the subterm modulo *Ax* relation to be well-founded, which is not necessarily the case, for example modulo identity.

Since $\mathscr{U}(\mathscr{R})$ is unconditional, $\mathscr{U}(\mu)$-termination modulo *Ax* of $\mathscr{U}(\mathscr{R})$ is equivalent to its operational termination. So, as for Theorem 1, our proof of Theorem 2 is done by proving that any infinite well-formed proof tree for $(\mathscr{R}, \mu)$ can be transformed into an infinite, well-formed proof tree for $(\mathscr{U}(\mathscr{R}), \mathscr{U}(\mu))$. This is a consequence of the following lemma.

Lemma 3 *For any well-formed proof tree T for $(\mathscr{R}, \mu)$ whose head goal is either $t \to^* u$ or $t \to^1 u$, there exists a well-formed proof tree $\beta(T)$ for $(\mathscr{U}(\mathscr{R}), \mathscr{U}(\mu))$ whose head goal is $t \to^* u$. Moreover, if $T \subset T'$ then $\beta(T) \subset \beta(T')$.*

Proof We start by two preliminary remarks. If a proof tree T for $(\mathscr{R}, \mu)$ has a head goal of the form $t \to^1 u$, then T has the shape

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{T_1}{v_1 \to^* w_1} \quad \ldots \quad \dfrac{T_k}{v_k \to^* w_k}}{t_n \to^1 u_n}\,(\text{Repl})}{\vdots}\,(\text{Congr})}{t_1 \to^1 u_1}\,(\text{Congr})}{t \to^1 u}\,(\text{Congr}) \tag{11}$$

If the head goal is $t \to^* u$, then T has the shape

$$\dfrac{\dfrac{T_1}{t_0 \to^1 t_1} \quad \dfrac{\dfrac{T_2}{t_1 \to^1 t_2} \quad \dfrac{\dfrac{T_n}{t_{n-1} \to^1 t_n} \quad \dfrac{}{t_n \to^* u}\,(\text{Refl})}{\vdots}\,(\text{Trans})}{t_1 \to^* u}\,(\text{Trans})}{t \to^* u}\,(\text{Trans}) \tag{12}$$

where $t_0 = t$ and $t_n =_{Ax} u$. So globally, a proof tree is made by alternation of the previous two shapes.

Second remark: if we have a proof tree T whose head goal is $t \to^* u$, then for any context C admissible for μ (that is the path to the hole follows only allowed positions) it is possible to build a proof tree for goal $C[t] \to^* C[u]$ by "pushing" the context into the transitivity and reflexivity steps:

$$\dfrac{\dfrac{\dfrac{T_1}{t_0 \to^1 t_1}}{\vdots\,(\text{Congr})}}{C[t_0] \to^1 C[t_1]} \quad \dfrac{\dfrac{\dfrac{T_2}{t_1 \to^1 t_2}}{\vdots\,(\text{Congr})}}{C[t_1] \to^1 C[t_2]} \quad \dfrac{\dfrac{\dfrac{T_n}{t_{n-1} \to^1 t_n}}{\vdots\,(\text{Congr})}}{C[t_{n-1}] \to^1 C[t_n]} \quad \dfrac{}{C[t_n] \to^* C[u]}\,(\text{Refl})\,(\text{Trans})\,(\text{Trans})\,(\text{Trans})$$

To prove the lemma, for each proof tree T we construct a corresponding $\beta(T)$ by induction on tree structure. We have two cases, depending on whether the head goal of T has the form $t \to^* u$ or $t \to^1 u$.

Case 1: the head goal is $t \to^* u$

Then T has the shape (12). By structural induction on trees, we may assume that each subtree

$$U_i = \dfrac{T_i}{t_{i-1} \to^1 t_i}$$

has a transformed tree $\beta(U_i)$ of the form

$$\dfrac{\dfrac{T_i^1}{t_{i-1} \to^1 t_i^1} \quad \dfrac{\dfrac{T_i^2}{t_i^1 \to^1 t_i^2} \quad \dfrac{\dfrac{T_i^{k_i}}{t_i^{k_i-1} \to^1 t_i^{k_i}} \quad \dfrac{}{t_i^{k_i} \to^* t_i}\,(\text{Refl})}{\vdots}\,(\text{Trans})}{t_i^1 \to^* t_i}\,(\text{Trans})}{t_{i-1} \to^* t_i}\,(\text{Trans})$$

then $\beta(T)$ is built as follows.

$$\dfrac{\dfrac{\dfrac{T_1^1}{t_0 \to^1 t_1^1} \quad \dfrac{\dfrac{T_1^2}{t_1^1 \to^1 t_1^2} \quad \vdots}{t_1^1 \to^* u}\,(\text{Trans}) \quad \dfrac{\dfrac{T_1^{k_1}}{t_1^{k_1-1} \to^1 t_1} \quad \dfrac{\dfrac{T_2^1}{t_1 \to^1 t_2^1} \quad \vdots}{t_1 \to^* u}\,(\text{Trans}) \quad \vdots \quad \dfrac{\dfrac{T_n^1}{t_{n-1} \to^1 t_n^1} \quad \dfrac{\dfrac{T_n^2}{t_n^1 \to^1 t_n^2} \quad \vdots}{t_n^1 \to^* u}\,(\text{Trans})}{t_{n-1} \to^* u}\,(\text{Trans}) \quad \vdots \quad \dfrac{\dfrac{T_n^{k_n}}{t_n^{k_n-1} \to^1 t_n} \quad \dfrac{}{t_n \to^* u}\,(\text{Refl})}{t_n^{k_n-1} \to^* u}\,(\text{Trans})}{t_1^{k_1-1} \to^* u}\,(\text{Trans})}{t_0 \to^* u}\,(\text{Trans})$$

The transformed tree above assumes that T is closed. If T is not closed, because some leftmost T_i^j is not closed, then $\beta(T)$ has to be "cut" at the level of T_i^j. In both cases, $\beta(T)$ is a well-formed tree if T is well-formed.

Case 2: the head goal is $t \to^1 u$

Then T has the shape (11).

Case 2.1: if there is at least one (Congruence) step

Then T has the shape

$$\dfrac{T'}{t \to^1 u}\,(\text{Congr})$$

By induction on tree structure, we have a transformed tree $\beta(T')$ for T', so we can build $\beta(T)$ by "pushing" the congruence step into $\beta(T')$, as described above.

Case 2.2: if there is no congruence step

then T has the shape

$$\dfrac{\dfrac{T_1}{s_1\sigma \to^* t_1\sigma} \quad \ldots \quad \dfrac{T_n}{s_n\sigma \to^* t_n\sigma}}{u \to^1 r\sigma}\,(\text{Repl})$$

for some conditional rule $l \to r$ if $s_1 \to t_1, \ldots, s_n \to t_n$ with $u =_{Ax} l\sigma$. In the transformed TRS, we have the rules (8), (9), (10), from which we are now going to build successively

trees for goals

$$\begin{array}{rl}
U_n(t_n,\mathbf{x_n})\sigma \to^* r\sigma & (G_n)\\
U_n(s_n,\mathbf{x_n})\sigma \to^* r\sigma & (H_n)\\
U_{n-1}(t_{n-1},\mathbf{x_{n-1}})\sigma \to^* r\sigma & (G_{n-1})\\
U_{n-1}(s_{n-1},\mathbf{x_{n-1}})\sigma \to^* r\sigma & (H_{n-1})\\
\vdots & \\
U_1(t_1,\mathbf{x_1})\sigma \to^* r\sigma & (G_1)\\
U_1(s_1,\mathbf{x_1})\sigma \to^* r\sigma & (H_1)\\
u \to^* r\sigma & (K)
\end{array}$$

Indeed, we need to be slightly more general, in order to take care of the axioms Ax: goals (G_k) are $v \to^* r\sigma$ for any $v =_{Ax} U_k(t_k,\mathbf{x_k})\sigma$.

1. Tree for goal (G_n): in the transformed TRS, we have the proof tree

$$\dfrac{\dfrac{}{v \to^1 r\sigma}\,(\text{Repl}) \qquad \dfrac{}{r\sigma \to^* r\sigma}\,(\text{Refl})}{v \to^* r\sigma}\,(\text{Trans})$$

   for any term v such that $v =_{Ax} U_n(t_n,\mathbf{x_n})\sigma$, using rule (10).
2. Tree for goal (H_k) from tree for goal (G_k): if we assume that, for $1 \le k \le n$ we have a proof tree T_k for any goal $v \to^* r\sigma$ with $v =_{Ax} U_k(t_k,\mathbf{x_k})\sigma$. By induction, we may assume that the subtree

$$\dfrac{T'}{s_k\sigma \to^* t_k\sigma}$$

   has a transformed tree, of the form

$$\dfrac{\dfrac{}{u_0 \to^1 u_1} \qquad \dfrac{\dfrac{}{u_1 \to^1 u_2} \qquad \dfrac{\dfrac{\dfrac{T'_{i-1}}{u_{i-1} \to^1 u_i} \qquad \dfrac{}{u_i \to^* t_k\sigma}\,(\text{Refl})}{\vdots}\,(\text{Trans})}{u_1 \to^* t_k\sigma}\,(\text{Trans})}{u_0 \to^* t_k\sigma}\,(\text{Trans})$$

   with $u_0 = s_k\sigma$ and $u_i =_{Ax} t_k\sigma$. Then we build a proof tree for the goal $U_k(s_k\sigma,\mathbf{x_k}\sigma) \to^* r\sigma$ as:

$$\dfrac{\dfrac{}{u_0 \to^1 u_1}\,(\text{Congr})}{U_k(u_0,\mathbf{x_k}\sigma) \to^1 U_k(u_1,\mathbf{x_k}\sigma)} \qquad \dfrac{\dfrac{}{u_1 \to^1 u_2}\,(\text{Congr}) \qquad \dfrac{\dfrac{T_k}{U(u_i,\mathbf{x_k}\sigma) \to r\sigma}}{\vdots}}{U_k(u_1,\mathbf{x_k}\sigma) \to^1 U_k(u_2,\mathbf{x_k}\sigma)}\,(\text{Trans})$$
$$\dfrac{}{U_k(u_1,\mathbf{x_k}\sigma) \to^* r\sigma}\,(\text{Trans})$$
$$U_k(u_0,\mathbf{x_k}\sigma) \to^* r\sigma$$

   where the proof tree T_k exists since $U(u_i,\mathbf{x_k}\sigma) =_{Ax} U_k(t_k,\mathbf{x_k})\sigma$. Note that the congruence steps above are valid with respect to the replacement map $\mu(U_k) = \{1\}$.

3. Tree for goal (G_{k-1}) from tree for goal (H_k): if we assume that, for $2 \leq k \leq n$ we have a proof tree T'_k for the goal $U_k(s_k, \mathbf{x_k})\sigma \to^* r\sigma$, then we build a proof tree for any goal $v \to^* r\sigma$ with $v =_{Ax} U_{k-1}(t_{k-1}, \mathbf{x_{k-1}})\sigma$ as:

$$\dfrac{\dfrac{}{v \to^1 U_k(s_k\sigma, \mathbf{x_k}\sigma)}\,(\text{Repl}) \qquad T'_k}{v \to^* r\sigma}\,(\text{Trans})$$

   using rule (9) in the application of (Replacement), and the fact that $v =_{Ax} U_{k-1}(t_{k-1}, \mathbf{x_{k-1}})\sigma$.
4. Tree for goal (K) from tree for goal (H_1): we have a proof tree T'_1 for the goal $U(s_1\sigma, \mathbf{x_1}\sigma) \to^* r\sigma$, and then we build the proof tree

$$\dfrac{\dfrac{}{u \to^1 U_1(s_1, \mathbf{x_1})\sigma}\,(\text{Repl}) \qquad T'_1}{u \to^* r\sigma}\,(\text{Trans})$$

   using rule (8) in the application of (Replacement), and the fact that $u =_{Ax} l\sigma$.

As for case 1, if the original proof tree is not closed, then some cut must be done in the transformed tree. In either cases, $\beta(T)$ is well-formed if T is so. In all cases, $\beta(T) \subset \beta(T')$ if $T \subset T'$. $\square$

Example 4 According to Theorems 2 and 1, termination of program `LengthOfFiniteLists` in Example 1 can be guaranteed by proving the μ-termination of the following TRS:

```
and(tt,T) -> T
isNatIList(IL) -> isNatList(IL)
isNat(0) -> tt
isNat(s(N)) -> isNat(N)
isNat(length(L)) -> isNatList(L)
isNatIList(zeros) -> tt
isNatIList(cons(N,IL)) -> and(isNat(N),isNatIList(IL))
isNatList(nil) -> tt
isNatList(cons(N,L)) -> and(isNat(N),isNatList(L))
zeros -> cons(0,zeros)
length(nil) -> 0
length(cons(N,L)) -> uLength(and(isNat(N),isNatList(L)),L)
uLength(tt,L) -> s(length(L))
```

where $\mu(\texttt{isNat}) = \mu(\texttt{isNatList}) = \mu(\texttt{isNatIList}) = \emptyset$, $\mu(\texttt{and}) = \mu(\texttt{cons}) = \mu(\texttt{uLength}) = \{1\}$ and $\mu(f) = \{1, \ldots, ar(f)\}$ for all other symbols f.

The μ-termination of this system can be automatically proved with AProVE, see Section 5.2 below for further details about the proof.

4.4 Improvements on the classical transformation

The following example shows that the use of replacement restrictions makes our transformation simulate more faithfully the original CTRS than the classical transformation does.

Example 5 Consider the following CTRS $\mathscr{R}$ in [13, Section 3]:

$$a \to b \qquad f(a) \to b \qquad g(X) \to g(a) \text{ if } f(X) \to X$$

As noticed by Giesl and Arts [13], this CTRS is quasi-decreasing, hence operationally terminating ([29, Theorem 2]). However, the classical transformation yields a TRS $\mathscr{U}(\mathscr{R})$:

$$a \to b \qquad f(a) \to b \qquad g(X) \to U(f(X),X) \qquad U(X,X) \to g(a)$$

which is *not* terminating:

$$\underline{g(a)} \to U(\underline{f(a)},a) \to U(b,\underline{a}) \to \underline{U(b,b)} \to \underline{g(a)} \to \cdots$$

In our version of the transformation, we consider $\mathscr{R}$ given with the *top* replacement map $\mu_\top(f) = \{1,\ldots,k\}$ for all k-ary symbols $f \in \mathscr{F}$. In this case, *CSR* and ordinary rewriting coincide. In our version of the classical transformation, $\mathscr{U}(\mu_\top)(U) = \{1\}$. It is not difficult to see that $\mathscr{U}(\mathscr{R})$ is $\mathscr{U}(\mu_\top)$-terminating. By Theorem 2, $(\mathscr{R},\mu_\top)$ (equivalently the CTRS $\mathscr{R}$) is operationally terminating.

Unfortunately, the use of replacement maps for the auxiliary symbols U improves but does *not* make the classical transformation complete for proving operational termination of deterministic 3-CTRS. The following example illustrates this point:

Example 6 Consider the following CTRS $\mathscr{R}$ [35, Example 7.2.51]:

$$\begin{aligned} h(d) &\to c(a) \\ h(d) &\to c(b) \\ f(k(a),k(b),X) &\to f(X,X,X) \\ g(X) &\to k(Y) \text{ if } h(X) \to d, h(X) \to c(Y) \end{aligned}$$

As shown by Ohlebusch, this CTRS is quasi-decreasing hence operationally terminating. However, the transformed TRS $\mathscr{U}(\mathscr{R})$:

$$\begin{aligned} g(X) &\to U_1(h(X),X) & f(k(a),k(b),X) &\to f(X,X,X) \\ U_1(d,X) &\to U_2(h(X),X) & h(d) &\to c(a) \\ U_2(c(Y),X) &\to k(Y) & h(d) &\to c(b) \end{aligned}$$

is not μ-terminating (where $\mu(U_1) = \mu(U_2) = \{1\}$ and $\mu(f) = \{1,\ldots,ar(f)\}$ for any other symbols f):

$$\begin{aligned} \underline{f(k(a),k(b),U_2(h(d),d))} &\to f(U_2(\underline{h(d)},d),U_2(\underline{h(d)},d),U_2(h(d),d)) \\ &\to^+ f(\underline{U_2(c(a),d)},\underline{U_2(c(b),d)},U_2(h(d),d)) \\ &\to^+ f(k(a),k(b),U_2(h(d),d)) \end{aligned}$$

It is interesting to note that the counter-example given above is not *Collapse-Extended*-terminating, that is, its termination is lost whenever one adds projection rules $\pi(x,y) \to x$ and $\pi(x,y) \to y$ for some new symbol π. CE-termination is known to be a nice notion of termination, because in practice terminating systems are indeed CE-terminating, and in contrast to standard termination it enjoys better modularity properties [38]. So an interesting open question is whether the $\mathscr{U}$ transformation is complete for CE-termination.

Further note that, regarding the classical transformation and *innermost* termination, Ohlebusch proved that quasi-decreasingness of a 3-CTRS implied innermost termination of the transformed unconditional TRS [35, Definition 7.2.52]. We conjecture that this holds for innermost-CS-termination, where innermost-CS-rewriting is the relation allowing rewriting steps only when the subterms at non-frozen positions are in μ-innermost normal form.

5 From theory to practice

As remarked in the introduction, once we have obtained a CS-TRS (i.e., a TRS $\mathscr{R}$ together with a replacement map μ), we can just try a proof of μ-termination of $\mathscr{R}$ (i.e., termination of *CSR* for $\mathscr{R}$ and the replacement map μ). Fortunately, several methods have been developed for this purpose. In the following section, we describe a tool which is able to deal automatically with CS-MRT specifications given as MAUDE programs.

5.1 MTT: A Prototype Implementation

Our current MAUDE Termination Tool (MTT) prototype is freely available for experimentation at `http://www.lcc.uma.es/~duran/MTT/`). It has a graphical interface which allows the user to input membership equational programs in the MAUDE syntax. The user may select different variants of transformations A and B, ask for the transformed program, and finally try to prove its termination by calling existing termination tools. Currently, it interacts with C*i*ME, MU-TERM and APROVE, but indeed it supports the TPDB syntax as output (`http://www.lri.fr/~marche/termination-competition`) and therefore hence any other tool supporting this syntax could be used as well. In the future, we plan to develop translations from other equational languages into MTT making these techniques available for those languages as well.

The tool implementation clearly distinguishes two parts: (1) a reflective MAUDE specification implements the theory transformations A and B (including optimized variants) described in Section 4, and (2) a Java application connects MAUDE, C*i*ME, MU-TERM and APROVE; and provides a graphical user interface. The Java application is in charge of sending the MAUDE specification introduced by the user to MAUDE to perform transformations; depending on the selections made by the user, one transformation or another will be accomplished. The resulting unsorted unconditional rewriting system may be proved terminating by using either C*i*ME, APROVE or MU-TERM. It is also possible to ask MU-TERM to perform a transformation from a CS-TRS to a TRS, and ask for a termination proof of the resulting TRS to the other back-end tools, as explained in Figure 2.

5.2 Experiments

In order to validate our approach in practice, we have used our implementation to (try to) prove termination of a number of (small) MAUDE programs. For these experiments we performed a fully automated proof search, attempting all possible transformations on each example, and all possible back-end tools. The results are presented on the web page:

`http://www.lri.fr/~marche/MTT/`.

which is currently under continuous development as part of the development of the MTT tool itself. We have observed that:

1. For a majority of the programs we have tried (around 80%), there is at least one back-end tool that leads to a termination proof on the CS-TRS obtained by some combination of the transformations described in Sections 4.1 and 4.3 above (possibly involving the refinements in Section 4.2).

2. The 'and' optimization of Section 4.2.2 is clearly helpful: the proof *always* takes less time when the optimization is activated; furthermore, it often avoids timeouts (e.g., when dealing with *bags* of natural numbers or *booleans*, in the setting of AC theories).
3. As expected, dealing with large programs is difficult. This clearly shows that there is a scaling-up issue in proving termination of programs. This means that modular techniques should be investigated further (see below).

6 Conclusions and further work

Proving termination of equational programs having expressive features such as conditions, typing, memberships, and evaluation strategies is important but nontrivial, because some of those features may not be supported by standard termination methods and tools. Yet, use of such features may be essential to ensure termination.

Sometimes a crucial issue may even be how to define the reduction relation. For example, with the two rules $f(a) \to f(b)$ and $a \to b$ if $f(a) \to f(b)$, do we have $f(a) \to^* f(b)$ with innermost strategy? In an interpreter like MAUDE, asking normalization of a loops forever, because it tries to apply the second rule, hence tries to reduce $f(a)$ with innermost strategy, hence tries to normalize a again. Therefore, we have focused on the recently introduced notion of operational termination [29] which closely corresponds to the termination of an interpreter. In fact, in this paper we have shown that, as claimed in [29], such a notion is flexible enough to provide a suitable notion of termination for languages and systems whose operational semantics is described by means of inference systems involving a variety of relations which are not necessarily rewrite relations (e.g., the memberships in the CS-MRT logic). In this sense, this paper provides a more satisfactory termination framework than the earlier version [9].

We have presented theory transformations that can be used to bridge the gap between equational programs and termination tools, have proved their correctness, and have discussed a prototype implementation in a tool taking MAUDE functional modules as inputs, performing the transformations, and mapping the resulting transformed theories to MU-TERM and from there to C*i*ME, AProVE, and other termination tools. Moreover, we have proposed variants and optimizations of our theory transformations that are also well suited for proving operational termination of MAUDE programs. Much work remains ahead, both in theoretical aspects and in experimentation. Theoretical issues that need to be further investigated include the following.

Firstly, our methods could be extended to prove termination of equational programs with *innermost* context-sensitive rewriting in the case of *unconditional rules*. For unconditional specifications, methods for such termination already exist and have been shown useful for proving termination of programs with elementary E-strategies in the OBJ sense [25]. There are also tools like AProVE or TERMPTATION which permit proving termination of innermost rewriting; and there are also tools like CARIBOO [11] which are specialized to deal with termination of rewriting under strategies (in particular, a class of innermost context-sensitive strategies for unconditional systems).

Secondly, our methods should be extended to take advantage of existing modular/incremental termination proof techniques [14, 30, 35–38] in our setting. Since MAUDE programs are built by composition of modules, termination should be proven incrementally: each time a new module is added, a proof of termination should be obtained by using the knowledge of termination of previous ones. However, further investigation is required, since MAUDE module hierarchies do not necessarily respect the usual hierarchical property

required for hierarchical TRSs, namely that for each rule added, the left-hand side's root symbol is a new symbol. Furthermore, even if this were to hold for some MAUDE programs, the transformations we have defined do not preserve that property, in particular because of sort elimination: if a new symbol f declares an old sort S as its codomain, then a new rule $is_S(f(\ldots)) \to \ldots$ has to be added, whereas is_S is an old symbol. A closely related topic is the development of techniques for proving termination of *parameterized modules*, as those definable in Full MAUDE and in MAUDE 2.2. This is the first-order analogue of termination techniques for polymorphic higher-order functions [16]. This problem is closely related to modularity, because one wants to investigate conditions under which a terminating parameterized module, when instantiated by a view to a terminating target instance module, results in an instantiation that can be guaranteed to be terminating.

Thirdly, completeness issues should be further investigated. We have shown in Section 4.4 that transformation B is not complete. We conjecture that transformation A is complete, and that transformation B is also complete if the transformed theory is evaluated with an innermost context-sensitive strategy. Completeness of transformation B when termination of the transformed theory is replaced by CE-termination should also be investigated.

Fourthly, *intrinsic proof methods* directly based on operational termination, and not requiring transformational approaches such as those presented in this work, should be investigated. In this regard, a future investigation of how operational termination and ordering-based termination approaches can be combined together, leading to intrinsic proof methods, for example for CS-MRTs, seems very worthwhile. The relationship between operational termination and quasi-decreasingness studied in detail in [29] can serve as a basis for a more general investigation of this kind.

More experimentation is needed to further extend and refine our methods. The current prototype provides a first basis for such experimentation; it should be extended and improved in several directions, including adding interfaces to other equational languages and termination tools, and adding support for the theoretical extensions mentioned above.

Acknowledgements We gratefully thank the referees for their very helpful and detailed comments and criticism on an earlier version and for example `INF2` that have substantially improved the presentation. We also thank Joe Hendrix and Ralf Sasse for their careful reading of the manuscript and their suggestions for improving it.

References

1. P. Borovanský, C. Kirchner, H. Kirchner, and P.-E. Moreau. ELAN from a rewriting logic point of view. *Theoretical Computer Science*, 285:155–185, 2002.
2. C. Borralleras, S. Lucas, and A. Rubio. Recursive path orderings can be context-sensitive. In A. Voronkov, editor, *Proc. of 18th International Conference on Automated Deduction*, volume 2392 of *Lecture Notes in Artificial Intelligence*, pages 314–331. Springer-Verlag, 2002.
3. A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theoretical Comput. Sci.*, 236:35–132, 2000.
4. R. Bruni and J. Meseguer. Generalized rewrite theories. In J. Baeten, J. Lenstra, J. Parrow, and G. Woeginger, editors, *Proceedings of the 30th International Colloquium on Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 252–266, 2003.
5. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Comput. Sci.*, 285(2):187–243, 2002.
6. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. Maude Manual (Version 2.2). December 2005, `http://maude.cs.uiuc.edu`.
7. CoFI Task Group on Semantics. CASL—The common algebraic specification language, version 1.0, Semantics. `http://www.brics.dk/Projects/CoFI/Documents/CASL/Semantics/index.html`, 1999.

8. E. Contejean, C. Marché, B. Monate, and X. Urbain. Proving termination of rewriting with C*i*ME. In A. Rubio, editor, *Proc. WST'03*, 2003. `http://cime.lri.fr`.
9. 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 PEPM'04*, pages 147–158. ACM Press, 2004.
10. M. C. F. Ferreira and A. L. Ribeiro. Context-sensitive AC-rewriting. In P. Narendran and M. Rusinowitch, editors, *Proc. RTA'99*, volume 1631 of *Lecture Notes in Computer Science*, pages 286–300, Trento, Italy, 1999. Springer-Verlag.
11. O. Fissore, I. Gnaedig, and H. Kirchner. Cariboo: An induction based proof tool for termination with strategies. In C. Kirchner, editor, *Proc. PPDP'02*, Pittsburgh, USA, 2002. ACM Press.
12. K. Futatsugi and R. Diaconescu. *CafeOBJ Report*. World Scientific, AMAST Series, 1998.
13. J. Giesl and T. Arts. Verification of Erlang Processes by Dependency Pairs. *Applicable Algebra in Engineering, Communications and Computing*, 12:39–72, 2001.
14. J. Giesl, T. Arts, and E. Ohlebusch. Modular termination proofs for rewriting using dependency pairs. *Journal of Symbolic Computation*, 34(2):21–58, 2002.
15. J. Giesl and A. Middeldorp. Transformation techniques for context-sensitive rewrite systems. *Journal of Functional Programming*, 14:379–427, 2004.
16. J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and disproving termination of higher-order functions. In B. Gramlich, editor, *Proc. of 5th International Workshop on Frontiers of Combining Systems, FroCoS'05*, volume 3717, pages 216–231, Vienna, Austria, 2005. Springer-Verlag.
17. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. AProVE: A system for proving termination. In van Oostrom [40]. `http://www-i2.informatik.rwth-aachen.de/AProVE`.
18. 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.
19. 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.
20. N. Hirokawa and A. Middeldorp. Tyrolean termination tool. In J. Giesl, editor, *Proc. RTA'05*, volume 3467 of *Lecture Notes in Computer Science*, pages 175–184, Nara, Japan, 2005. Springer-Verlag.
21. P. Hudak, S. Peyton-Jones, and P. Wadler. Report on the Functional Programming Language Haskell: a non–strict, purely functional language. *Sigplan Notices*, 27:1–164, 1992.
22. S. Lucas. Termination of context-sensitive rewriting by rewriting. In F. M. auf der Heide and B. Monien, editors, *Proc. of ICALP'96*, volume 1099 of *Lecture Notes in Computer Science*, pages 122–133. Springer-Verlag, 1996.
23. S. Lucas. Context-sensitive computations in functional and functional logic programs. *Journal of Functional and Logic Programming*, 1998(1), 1998.
24. S. Lucas. Context-sensitive rewriting strategies. *Information and Computation*, 178(1):294–343, 2002.
25. S. Lucas. Termination of programs with strategy annotations. Technical Report DSIC-II/20/03, DSIC, Universidad Politécnica de Valencia, 2003.
26. S. Lucas. MU-TERM, a tool for proving termination of context-sensitive rewriting. In van Oostrom [40]. `http://www.dsic.upv.es/~slucas/csr/termination/muterm/`.
27. S. Lucas. Polynomials for proving termination of context-sensitive rewriting. In I. Walukiewicz, editor, *Proc. FOSSACS'04*, volume 2987 of *Lecture Notes in Computer Science*, pages 318–332. Springer-Verlag, 2004.
28. S. Lucas. Proving termination of context-sensitive rewriting by transformation. *Information and Computation*, to appear 2006.
29. S. Lucas, C. Marché, and J. Meseguer. Operational termination of conditional term rewriting systems. *Inf. Process. Lett.*, 95:446–453, 2005.
30. C. Marché and X. Urbain. Modular and incremental proofs of AC-termination. *Journal of Symbolic Computation*, 38:873–897, 2004.
31. M. Marchiori. Unravelings and ultra-properties. In M. Hanus and M. Rodríguez-Artalejo, editors, *Proc. of ALP'96*, volume 1039 of *Lecture Notes in Computer Science*, pages 107–121. Springer-Verlag, 1996.
32. J. Meseguer. General logics. In *Logic Colloquium'87*, pages 275–329. North-Holland, 1989.
33. J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Proceedings WADT'97*, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61. Springer-Verlag, 1998.
34. 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.
35. E. Ohlebusch. *Advanced Topics in Term Rewriting*. Springer-Verlag, 2002.
36. E. Ohlebusch. Hierarchical termination revisited. *Inf. Process. Lett.*, 84(4):207–214, 2002.
37. X. Urbain. Automated incremental termination proofs for hierarchically defined term rewriting systems. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proc. IJCAR'01*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 485–498, Siena, Italy, 2001. Springer-Verlag.

38. X. Urbain. Modular and incremental automated termination proofs. *Journal of Automated Reasoning*, 32:315–355, 2004.
39. A. van Deursen, J. Heering, and P. Klint. *Language Prototyping: An Algebraic Specification Approach*. World Scientific, 1996.
40. V. van Oostrom, editor. *Rewriting Techniques and Applications*, Lecture Notes in Computer Science, Aachen, Germany, 2004. Springer-Verlag.
41. P. Viry. Equational rules for rewriting logic. *Theoretical Computer Science*, 285:487–517, 2002.
42. H. Zantema. Termination of context-sensitive rewriting. In H. Comon, editor, *Proc. RTA'97*, volume 1232 of *Lecture Notes in Computer Science*, pages 172–186, Sitges, Spain, 1997. Springer-Verlag.