

# MTT: The Maude Termination Tool (System Description)\*

Francisco Durán<sup>1</sup>, Salvador Lucas<sup>2</sup>, and José Meseguer<sup>3</sup>

<sup>1</sup> LCC, Universidad de Málaga, Spain

<sup>2</sup> DSIC, Universidad Politécnica de Valencia, Spain

<sup>3</sup> CS Dept., University of Illinois at Urbana-Champaign, USA

## 1 Introduction

Despite the remarkable development of the theory of termination of rewriting, its application to high-level programming languages is far from being optimal. This is due to the need for features such as conditional equations and rules, types and subtypes, (possibly programmable) strategies for controlling the execution, matching modulo axioms, and so on, that are used in many programs and tend to place such programs outside the scope of current termination tools. The operational meaning of such features is often formalized in a proof-theoretic manner by means of an inference system (see, e.g., [2, 3, 17]) rather than just by a rewriting relation. In particular, Generalized Rewrite Theories (GRT) [3] are a recent generalization of rewrite theories at the heart of the most recent formulation of MAUDE [4]. The corresponding termination notions can also differ from the standard ones, see [14]. For these reasons, although there is a good number of tools which can be used to automatically prove termination of rewriting (e.g., [9], [13], ...) they cannot directly prove termination of, e.g., Elan [1], Maude [4] or CafeOBJ [8] programs. As an illustrative example, consider the Maude module MARKS-LISTS in the MTT snapshot in Figure 1. Given a list representation of a multiset of natural numbers it (nondeterministically) computes its submultisets of size 2. A mark ‘#’ is *introduced* into a given List of numbers (of sort Nat) to yield a *marked* list of sort MList (supersort of List). The matching condition  $\langle N1 ; N2 ; N3 ; L' \rangle := \langle L \rangle$  in the conditional rule ensures that ‘#’ is introduced into lists of at least three elements. Symbol # is intended to mark a number to be *removed* by using the third rule (thus producing a *sublist* of the original one). The mark is *propagated* inside the structure of the list until it is finally *removed* (together with its companion number) to produce a list of sort List on which we can restart the process. Objects from both List and MList can be built by using a single *overloaded* constructor  $_;_$ .

Modeling MARKS-LISTS as a Conditional Term Rewriting System (CTRS, see [18]) by translating the matching condition into a rewriting condition as:

$$\langle L \rangle \rightarrow \langle \# ; L \rangle \text{ if } \langle L \rangle \rightarrow \langle N1 ; N2 ; N3 ; L' \rangle$$

---

\* Work partially supported by the EU (FEDER) and Spanish MEC under grants TIN 2005-09405-C02-01 and TIN 2007-68093-C02-02; José Meseguer was partially supported by ONR grant N00014-02-1-0715 and NSF Grant CCR-0234524.

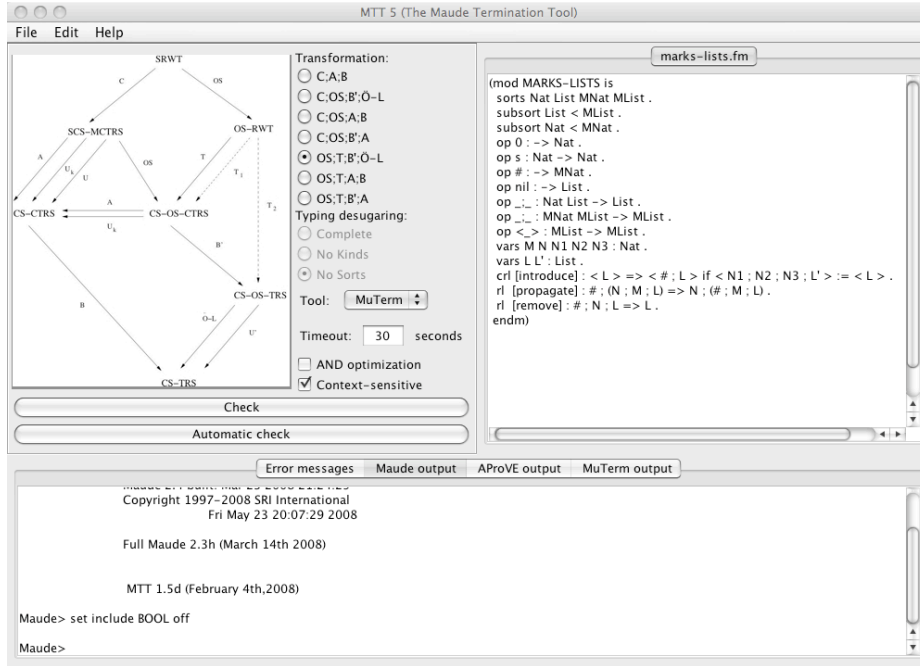


Fig. 1. MTT snapshot with module MARKS-LISTS

yields a *nonterminating* system. The application of this rule requires the reduction of (an instance of)  $\langle L \rangle$  into (an instance of)  $\langle N1 ; N2 ; N3 ; L' \rangle$  to satisfy the condition. Since the left-hand side  $\langle L \rangle$  of the conditional rule itself can also be considered in any attempt to satisfy the conditional part of the rule, we run into a nonterminating computation, see [14] for a deeper discussion of this issue.

However, viewed as a rewrite theory  $\mathcal{R} = (\Sigma, E, R)$  and executed as a Maude program, MARKS-LISTS is terminating! The key points here are the sort information and that solving the matching condition involves *no rewriting step*. Matching conditions are evaluated in Maude with respect to the set  $E$  of *equations* which is different from the rules  $R$  in  $\mathcal{R}$ . A *matching-modulo-E* semantics is given for solving matching conditions. In our MARKS-LISTS example,  $E$  is empty and the matching condition becomes *syntactic pattern matching*. *No reduction* is allowed! Indeed, only when the *two* kinds of  $E$ - and  $R$ -computations which are implicit in the specification are (separately!) taken into account, are we able to prove this program terminating. Other features like sort information (including both the existence of a sort hierarchy and also the association of sorts to function symbols in module MARKS-LISTS), memberships [2, 17], context-sensitivity [11, 12], rewriting modulo A-/AC-axioms, etc., can play a crucial role in the termination

behavior and hence in any attempt to provide an automatic proof of it, see, e.g., [5, 6, 15].

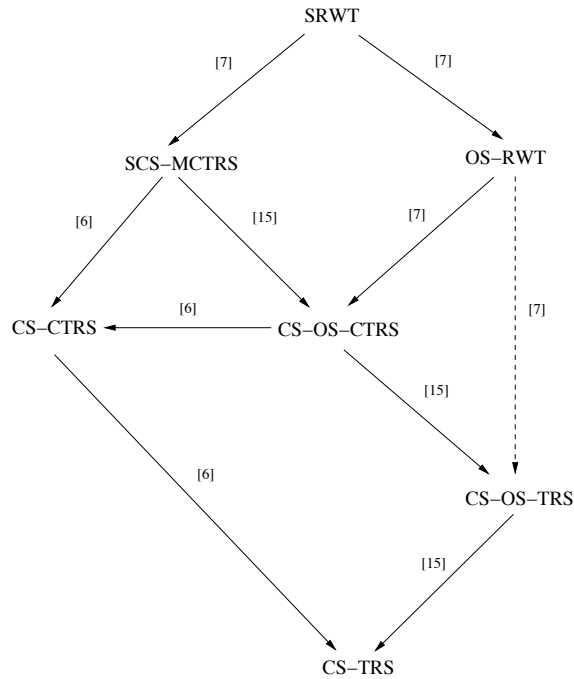
This paper emphasizes the techniques needed to go from standard termination methods and tools to termination tools for programs in rule-based languages with expressive features. Specifically, we focus on the implementation of MTT, a new tool for proving termination of Maude programs like MARKS-LISTS; see:

<http://www.lcc.uma.es/~duran/MTT>

Due to lack of space, we can only give a high-level overview of the techniques involved. Detailed accounts of all the transformations used can be found in [5, 6, 7, 14, 15].

## 2 Proving Termination by Transformation

In [5, 6, 15], different theory transformations associating a CS-TRS, i.e., a TRS (Term Rewriting System) together with a *replacement map*  $\mu$  [11, 12], to a *membership rewrite theory* [2, 17] were presented. In [7] we have generalized this methodology to rewriting logic theories  $\mathcal{R} = (\Sigma, E, R)$ , as the MARKS-LISTS one[3]. As discussed above, from the point of view of termination, the main problem with such theories is that there are two different rewrite relations (with  $E$  and with  $R$ ), which cannot always be just joined, because equational conditions (for instance, matching conditions) are solved using just  $E$ .



In MTT we take advantage of all previous theoretical developments and use *sequences of transformations* which are applied in a kind of pipeline to finally obtain a CS-TRS whose termination can be proved by using existing tools such as AProVE [9] or MU-TERM [13]. In this section, we briefly discuss such theory transformations whose hierarchy is depicted in the above diagram. All the time non-termination is preserved under the transformations in such a way that a proof of termination of a system which is downwards the diagram implies termination of the system which originated it upwards. Each arc is labelled with the reference where the corresponding transformation is described. Before describing these transformations, we recall the notion of rewrite theory.

## 2.1 Rewrite theories

A rewriting logic specification is called a *rewrite theory* (RWT) [3]. It is a tuple  $\mathcal{R} = (\Sigma, E \cup Ax, \mu, R, \phi)$ , where:

- $(\Sigma, E \cup Ax)$  is a membership equational (MEL) theory:  $\Sigma$  is an order-sorted signature [10],  $Ax$  is a set of (equational) axioms, and  $E$  is a set of sentences

$$t = t' \text{ if } A_1, \dots, A_n \quad \text{or} \quad t : s \text{ if } A_1, \dots, A_n$$

where the  $A_i$  are atomic equations or memberships  $t : s$  establishing that term  $t$  has sort  $s$  [2, 17].

- $\mu$  is a mapping specifying for each  $f \in \Sigma$  the argument positions under which subterms can be simplified with the equations in  $E$  [11, 12].
- $R$  is a set of *labeled conditional rewrite rules* of the general form

$$r : (\forall X) q \longrightarrow q' \text{ if } \left( \bigwedge_i u_i = u'_i \right) \wedge \left( \bigwedge_j v_j : s_j \right) \wedge \left( \bigwedge_l w_l \longrightarrow w'_l \right).$$

- $\phi : \Sigma \longrightarrow \mathbb{N}$  is a mapping assigning to each function symbol  $f \in \Sigma$  (with, say,  $n$  arguments) a set  $\phi(f) \subseteq \{1, \dots, n\}$  of *frozen positions* under which it is forbidden to perform any rewrites with rules in  $R$ .

Intuitively,  $\mathcal{R}$  specifies a *concurrent system*, whose states are elements of the initial algebra  $T_{\Sigma/E \cup Ax}$  and whose *concurrent transitions* are specified by the rules  $R$ , subject to the frozenness constraints imposed by  $\phi$ . Therefore, mathematically each state is modeled as an  $E \cup Ax$ -equivalence class  $[t]_{E \cup Ax}$  of ground terms, and rewriting happens *modulo*  $E \cup Ax$ , that is,  $R$  rewrites not just terms  $t$  but rather  $E \cup Ax$ -equivalence classes  $[t]_{E \cup Ax}$  representing states.

## 2.2 Proving Termination of Rewrite Theories with MTT

According to the diagram above, the most general kind of systems we can deal with are (sugared) rewrite theories (SRWTs). This means that (following the usual practice), we consider rewrite theories as presented by keeping the intuitive order-sorted subtyping features intact as helpful *syntactic sugar*, and adding

membership axioms only when they are strictly needed (see [15, 7] for a more detailed discussion). This *order-sorted way* allows us to take advantage of existing techniques for proving termination of order-sorted context-sensitive term rewriting systems (OS-CS-TRSs) [15]. On the other hand, we can still make use of the (older) *MEL-way* where we start from a (context-sensitive) membership equational specification and then apply the methods developed in [5, 6] to obtain a proof of termination. Thus, given an SRWT, we can either [7]:

1. translate it into an order-sorted rewrite theory (OS-RWT) where memberships have been handled by using membership predicates which express different membership conditions defined by (possibly conditional) rules, or
2. translate it into a sugared CS-CTRS with conditional rewrite and membership rules, i.e., an SCS-MCTRS.

Then, we can further transform a SCS-MCTRS into either a conditional context-sensitive TRS (CS-CTRS) or a conditional order-sorted CS-TRS (OS-CS-CTRS). In the first case, the transformation is accomplished by introducing membership predicate symbols (and rules) [6]. In the second case, the transformation tries to keep the sort structure of the SCS-MCTRS untouched and membership predicate symbols are introduced only if necessary [15]. After treating membership information, the next step is dealing with conditional rules. The classical transformation from CTRSs into TRSs (see, e.g., [18]) has been generalized to deal with context-sensitivity (so that a CS-CTRS is transformed into a CS-TRS) [6] and with sorts (then, an OS-CS-CTRS is transformed into an OS-CS-TRS) [15]. Finally, we can transform an OS-CS-TRS into a CS-TRS by using the transformation discussed in [15], which is a straightforward adaptation of Ölveczky and Lysne's transformation [19]. Nowadays, proofs of termination of CS-TRSs can be achieved by using AProVE [9] or MU-TERM [13]. However, since termination of a TRS  $\mathcal{R}$  implies that of the CS-TRS  $(\mathcal{R}, \mu)$  for any replacement map  $\mu$ , other tools for proving termination of rewriting could be used as well.

### 3 Implementation of the Tool

Our current tool MTT 1.5<sup>4</sup> takes Maude programs as inputs and tries to prove them terminating by using existing termination tools as back-ends. MTT 1.5 can use as back-end tool any termination tool supporting the TPDB syntax and following the rules for the Termination Competition [16].<sup>5</sup> This allows us to interact with the different tools in a uniform way, and not restricting ourselves to a specific set of tools. Thus, tools that have participated in the competition, like AProVE, MU-TERM, TTT, etc., or that accommodate to the syntax and form of

<sup>4</sup> Previous versions of MTT were able to handle only membership equational programs in the Maude syntax. Only two of the currently supported transformations were available, and it was able to interact, in an *ad hoc* way, only with AProVE, MU-TERM, and CiME.

<sup>5</sup> The Termination Competition rules and the TPDB syntax can be found at <http://www.lri.fr/~marche/termination-competition/>.

interaction, can be used as back-ends of MTT. In the MTT environment, Maude specifications can be proved terminating by using (any of these) distinct formal tools, allowing the user to choose the most appropriate one for each particular case, a combination of them, when a particular tool cannot find a proof.

Two main components can be distinguished: (1) a Maude specification that implements the theory transformations described in the diagram above, and (2) a Java application that connects Maude, and the back-end tools, 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, one transformation or another will be accomplished. The resulting TRS may be proved terminating by using any of the available back-end tools. Notice that such resulting TRS may have associative or associative-commutative operators, context sensitive information, etc., which are expected to be appropriately handled by the selected back-end tool.

Alternatively, the MTT *expert* can be used to try an appropriate sequence of them automatically. The current expert just tries different paths in the transformations graphs sequentially.

The interaction between MTT and the back-end tools can be done in two ways:

- If the external tool is installed in the same machine, they can interact via pipes. This is the more efficient form of interaction available.
- Interaction based on web services is also possible. This is the most flexible of the possibilities offered by MTT (no local configuration is required).

MTT has a graphical user interface where one can introduce the specifications to be checked. MTT gives the output of the back-end tool used in the check. All intermediate transformed specifications can also be obtained. Some benchmarks are here: <http://www.lcc.uma.es/~duran/MTT/mtt15/Samples>.

*Acknowledgements.* Special thanks are due to Claude Marché and Xavier Urbain for their collaboration in the development of the first versions of MTT.

## 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] A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theoretical Computer Science*, 236:35–132, 2000.
- [3] R. Bruni and J. Meseguer. Semantic foundations for generalized rewrite theories. *Theoretical Computer Science* 351(1):386-414, 2006.
- [4] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. All About Maude – A High-Performance Logical Framework. Volume 4350 of Lecture Notes in Computer Science, Springer, 2007.
- [5] F. Durán, S. Lucas, C. Marché, J. Meseguer, and X. Urbain. Proving Termination of Membership Equational Programs. In P. Sestoft and N. Heintze, editors, *Proc. of ACM SIGPLAN 2004 Symposium on Partial Evaluation and Program Manipulation, PEPM'04*, pages 147-158, ACM Press, 2004.

- [6] F. Durán, S. Lucas, C. Marché, J. Meseguer, and X. Urbain. Proving Operational Termination of Membership Equational Programs. *Higher-Order and Symbolic Computation*. To appear. Published online: April 2008.
- [7] F. Durán, S. Lucas, and J. Meseguer. Operational Termination in Rewriting Logic. Technical Report 2008. <http://www.dsic.upv.es/~slucas/tr08.pdf>
- [8] K. Futatsugi and R. Diaconescu. *CafeOBJ Report*. World Scientific, AMAST Series, 1998.
- [9] 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 Third International Joint Conference on Automated Reasoning, IJCAR'06*, LNAI 4130:281-286, Springer, 2006. Available at <http://www-i2.informatik.rwth-aachen.de/AProVE>.
- [10] 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.
- [11] S. Lucas. Context-sensitive computations in functional and functional logic programs. *Journal of Functional and Logic Programming*, 1998(1):1-61, January 1998.
- [12] S. Lucas. Context-sensitive rewriting strategies. *Information and Computation*, 178(1):294-343, 2002.
- [13] S. Lucas. MU-TERM: A Tool for Proving Termination of Context-Sensitive Rewriting. In V. van Oostrom, editor, *Proc. of the 15th International Conference on Rewriting Techniques and Applications, RTA'04*, LNCS 3091:200-209, Springer, 2004. Available at <http://www.dsic.upv.es/~slucas/csr/termination/muterm>.
- [14] S. Lucas, C. Marché, and J. Meseguer. Operational termination of conditional term rewriting systems. *Information Processing Letters*, 95(4):446-453, 2005.
- [15] S. Lucas and J. Meseguer. Operational Termination of Membership Equational Programs: the Order-Sorted Way. In G. Rosu, editor, *Proc. of the 7th International Workshop on Rewriting Logic and its Applications, WRLA'08, Electronic Notes in Theoretical Computer Science*, to appear, 2008.
- [16] Claude Marché and Hans Zantema. The termination competition. In F. Baader, editor, *Proc. of RTA'07*, LNCS 4533:303-313. Springer, 2007.
- [17] J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Proc. of WADT'97*, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61, Springer, 1998.
- [18] E. Ohlebusch. *Advanced Topics in Term Rewriting*. Springer, 2002.
- [19] P.C. Ölveczky and O. Lysne. Order-Sorted Termination: The Unsorted Way. In M. Hanus and M. Rodríguez-Artalejo, editors, *Proc. of the 5th International Conference on Algebraic and Logic Programming, ALP'96*, LNCS 1139:92-106, Springer, 1996.