Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 248 (2009) 83–92
www.elsevier.com/locate/entcs

Web Services and Interoperability for the Maude Termination Tool

Francisco Durán [1,4]

LCC, Universidad de Málaga, Málaga, Spain

Salvador Lucas [2,5]

DSIC, Universidad Politécnica de Valencia, Valencia, Spain

José Meseguer [3,6]

CSD, University of Illinois at Urbana-Champaign, Urbana, IL, USA

Francisco Gutiérrez [1,7]

LCC, Universidad de Málaga, Málaga, Spain

Abstract

This paper presents the Maude Termination Tool (MTT) version 1.5. MTT takes Maude programs as inputs and tries to prove them terminating by applying different transformation techniques and by using existing termination tools as back-ends. MTT can use as back-end tool any termination tool supporting the TPDB syntax, either locally if it follows the rules for the Termination Competition, or remotely as web services. 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 others that accommodate to the syntax and form of 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, or trying different alternatives in the case of a particular tool cannot find a proof.

Keywords: Maude, rewriting logic, termination

[1] Partially supported by the EU (FEDER) and Spanish MEC under grants TIN 2005-09405-C02-01 and TIN 2008-03107.

[2] Partially supported by the EU (FEDER) and Spanish MEC under grant TIN 2007-68093-C02-02.

[3] Partially supported by ONR grant N00014-02-1-0715 and NSF Grant CCR-0234524.

[4] Email: duran@lcc.uma.es

[5] Email: slucas@dsic.upv.es

[6] Email: meseguer@cs.uiuc.edu

[7] Email: pacog@lcc.uma.es

doi:10.1016/j.entcs.2009.07.061

1 Introduction

Although the theory of termination has had a remarkable development in the last years, and there is today a good number of tools which can be used to automatically prove termination of rewriting (e.g., AProVE [11], C*i*ME [5], MU-TERM [17], Torpa [25], TTT [14], etc.), they cannot directly prove termination of programs in high-level equational languages as, e.g., ASF+SDF [24], CafeOBJ [6], ELAN [1], Haskell [15], Maude [4,3], OBJ [13], or Prolog [23]. This is due to the use in these languages of advanced features such as conditional equations and rules, types and subtypes, (possibly programmable) strategies for controlling the execution, matching modulo axioms, and so on. Programs using these features are placed outside the scope of current termination tools, which assume considerably more restrictive specifications (untyped, unconditional term rewriting systems).

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, e.g., with suitable types, memberships, strategies, etc. This situation has been studied recently for the case of the programming languages Haskell [12] and Maude [10,7,8,18,19].

Consider the Maude functional module `FINITE-LISTS` below, 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`.

```
fmod FINITE-LISTS 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 .
  var N : Nat .
  var L : NatList .
  eq zeros = cons(0, zeros) .
  eq length(nil) = 0 .
```

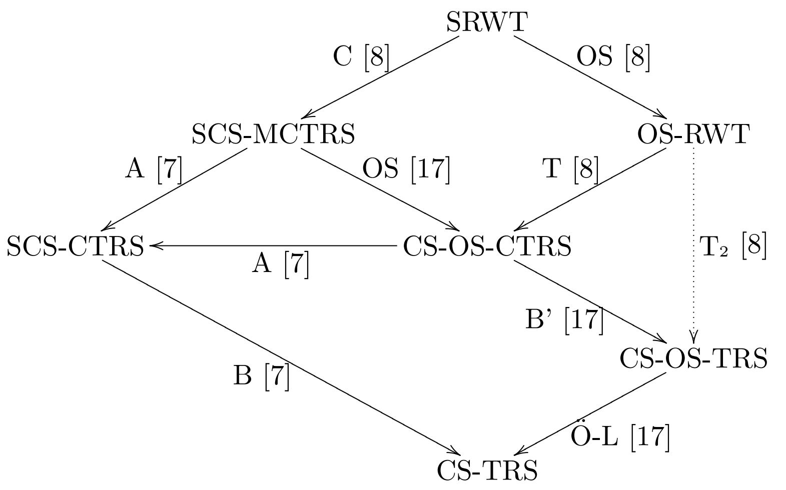

Fig. 1. Transformations available in MTT.

```
  eq length(cons(N, L)) = s(length(L)) .
endfm
```

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.

In this paper we present the Maude Termination Tool (MTT), which takes Maude programs as inputs and tries to prove them terminating by applying different transformation techniques and by using existing termination tools as back-ends.

2 A transformational approach

In recent years, a number of non-termination preserving theory transformations associating a context-sensitive term rewriting system (CS-TRS, a TRS together with a replacement map μ) [16] to a membership rewrite theory [2,22] and to a rewriting logic theory [21] have been presented. In MTT, we take advantage of these previous 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. The complete set of transformations available is shown in Figure 1, where each transformation comes with a reference where it is described. 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. See [10,7,9,8,18,19] for details.

MTT 5 provides these sequences of transformations as alternative transforma-

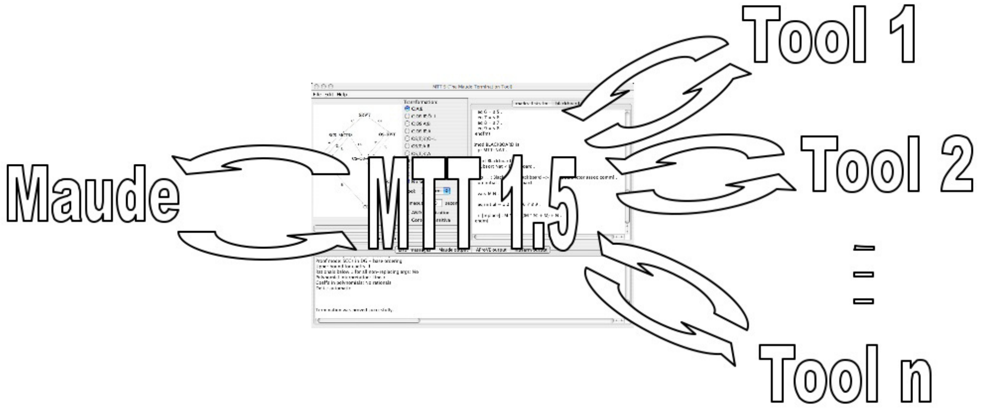

Fig. 2. Interactions between the tools.

tions, between which one can choose the more appropriate for each particular case. Thus, one can proof the termination of a Maude specification by transforming a sugared rewrite theory (SRWT) into a context-sensitive term rewrite theory (CS-TRS), which can be handled by tools like AProVE and MU-TERM combining different transformations. But one can follow several alternative paths. E.g., a context-sensitive membership conditional term rewriting system (CS-MCTRS) can be transformed into a context-sensitive conditional term rewriting system (CS-CTRS) by using transformation A (which can be either complete, discarding information on kinds, or discarding all information on sorts), or into a context-sensitive order-sorted conditional term rewriting system (CS-OSCTRS) by using the transformation OS.

3 Architecture of the tool

The implementation of MTT clearly distinguishes two parts: (1) a Maude specification that implements the theory transformations described in the diagram in Figure 1, and (2) a Java application that connects Maude and the back-end tools, and provides a graphical user interface. Figure 2 shows the current interactions between the tools.

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 (alternatively, the MTT *expert* can be used to try an appropriate sequence of them automatically). The resulting unsorted unconditional rewriting system obtained from such transformations may be proved terminating by using any of the available back-end tools. Notice that such resulting CS-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. The tool's output is given as result. Optionally, the intermediate specifications can be shown.

Once a specification is open in one of the editors of the tool, it can be used to check its termination. MTT does such a check by sending a termination-preserving transformation of the user's specification to some external termination tool.

MTT can use as back-end tool any termination tool supporting the TPDB syntax[8] and following the rules for the Termination Competition [20], celebrated every year as part of the Woskshop on Termination (WST). The basic rules in the competition for term rewriting systems are:

- Each tool must be available as an executable that takes as argument the name of a file describing a termination problem, and an integer giving the maximal CPU time in seconds allowed to give an answer.
- The tool must run without any user interaction, and the answer must be printed on standard output.
- The input file will be in the common format of the TPDB.
- The answer must start by either YES or NO, meaning that the given rewrite system is terminating or not terminating, respectively. The output should include a proof trace of the claimed result, thus providing enough information for the termination being checked by a third party.

This procedure provides a uniform way of interacting with the different tools. In previous versions of the tool, MTT was able to interact with C*i*ME, AProVE, and MU-TERM, but now other formal tools, such as TTT, Jambox, etc. can be considered as well for "the same price".

Moreover, the interaction between MTT and each of the tools can be done in two ways:

- If the external tool is installed in the same machine, they can interact locally (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 installaton is required).

The use of web services frees us from the burden of installing and configuring the external tools locally. And not only that, it can happen that some tool is not distributed for our platform. Even more, in the future we may be using tools that are not available today.

In the MTT environment, Maude specifications can be proved terminating by using (any of these) distinct formal tools, either installed locally or remotely, allowing the user to choose the most appropriate tool for each particular case, a combination of them, or trying different alternatives in the case of a particular tool not being able to find a proof.

4 MTT in use

MTT has a graphical user interface where one can introduce or load and edit the specifications to be checked. Figure 3 shows a snapshot of the GUI of the application with a `nat-list.fm` file in its editing window. This file contains several modules and

[8] The TPDB format is described at `http://www.lri.fr/~marche/tpdb/format.html`.

views, with the `NAT-LIST` module at the top of the structured specification, which defines lists of natural numbers. In this figure we can see how we can attempt the termination check either using the transformation options we select or an automatic check using the *expert*. When the *Check* button is clicked, the transformation selected on the graph is used to attempt the termination checking of the specification in the active editing panel. The path in the graph can be selected just by clicking on the edges of the graph. When the mouse passes on a node or arrow of the graph, a When selected, edges turn orange. If the selected edges form a valid path, they turn green. In Figure 3, the transformation C;A;B is selected in the graph. It was selected when the *Check* button was pressed, and therefore the panel at the bottom of the figure shows the result of the termination check as given by the selected tool, AProVE, for the `NAT-LIST` module, using the C;A;B transformation. MTT gives the output of the back-end tool used in the check in the pane at the bottom of the GUI. In this case we see only the last lines of such proof, but we can scroll up to see it entirely. All intermediate transformed specifications obtained in the transformation process can also be shown. This is optional, as we will see below. The use of the AND-optimization (see [7]) and the inclusion of the context-sensitive information are given as options. A timeout for the proofs can also be given. Proofs can be interrupted at any time by clicking on the red cross button in the corner at the right top of the proof pane.

Alternatively, we can choose to use the expert by pressing the *Automatic check* button. The expert just attempts the termination check using the selected tool with a sequence of paths in the graph. The order in which the transformations are considered is given by their cost. The more complexity introduced in the transformation, the more cases covered. But this complexity in the specifications passed to the back-end tools make it harder for them to find a proof. If a proof can be found using a simpler transformation, it should be attempted first, since a more complex transformation can be unable to find a proof in the same case.

Multiple proofs can be carried out simultaneously, either for the same module or for different ones. Each time a *Check* or *Automatic check* is pressed, a new pane is added, in which the result of the proof attempts for the options selected when the button was clicked are shown. Each tab shows the name of the tool it is interacting with as its label, with a colored diamond on its side showing its state. While waiting for the transformed specification from Maude, the diamond is blue; if Maude fails, it turns red. Once Maude returns a valid transformed specification, the diamond turns yellow, and it is like that while the chosen termination tool is active looking for a proof; when the interaction with the corresponding tool finishes, the diamond turns either green o red depending on whether the proof succeeded or not. Additionally, the path being used, together with the selection for and-optimization and context-sensitivity, is shown at the top of each pane. Figure 4 shows a snapshot of the tool with several files open and different proofs attempted. Notice the different tools used and the colors indicating the different results.

At any time, we can add a new back-end tool, or a new configuration for an existing tool. We can, for example, set up MU-TERM and AProVE , with AProVE

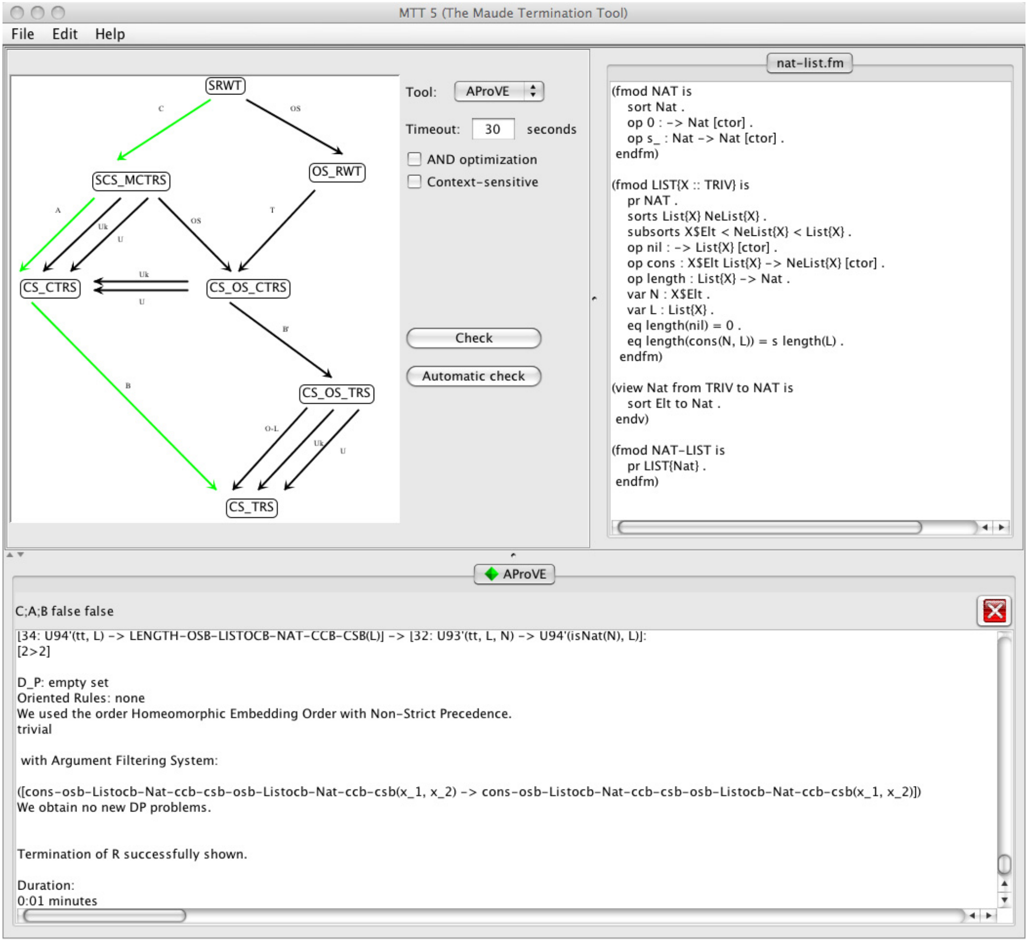

Fig. 3. Snapshot of the tool.

both to be used locally and remotely, and MU-TERM to be used with TPDB and Maude syntax (MU-TERM accepts Maude syntax). We can edit the preferences to configure the different tools available, or to add or remove tools at any time, by using the *Preferences...* option of the *File* menu. Figure 5 shows a snapshot of the preferences window of MTT. In this case Maude, AProVE and MU-TERM are configured, and the pane shown corresponds to AProVE, which is settled to be used locally. Since we assume that the interaction is the same for every tool, we can add a new tool just by clicking on the *Add tool* button and providing the path of the binary, if to be used locally, or the URL of the web service, if to be used remotely. Notice that we can also decide whether we wish to see the intermediate specifications or not, just by clicking on the corresponding mark box. We can also remove the configuration information of any tool by clicking on the *Remove tool* button, or mark the *Not available* to indicate that the corresponding back-end tool is not available, in which case the configuration information is kept by MTT but it is not offered to the user for its proofs.

MTT is available at `http://maude.lcc.uma.es/MTT`. The binary, its documen-

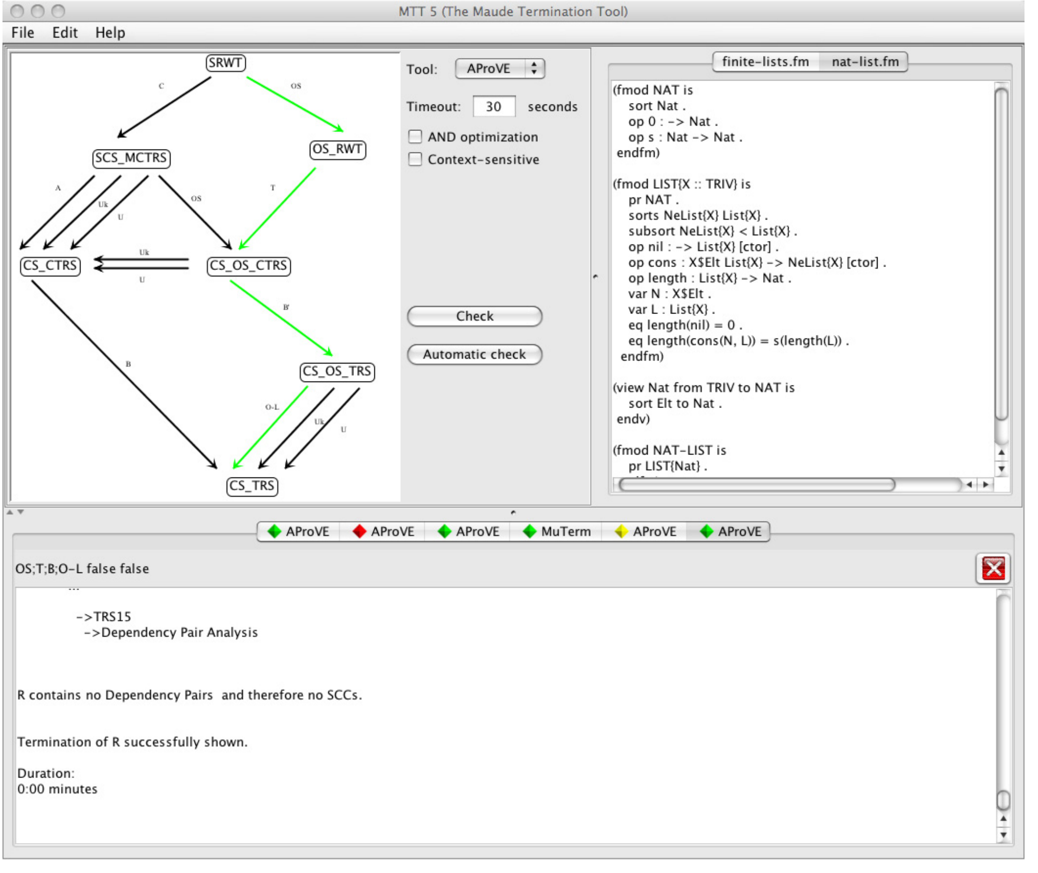

Fig. 4. Snapshot of the tool.

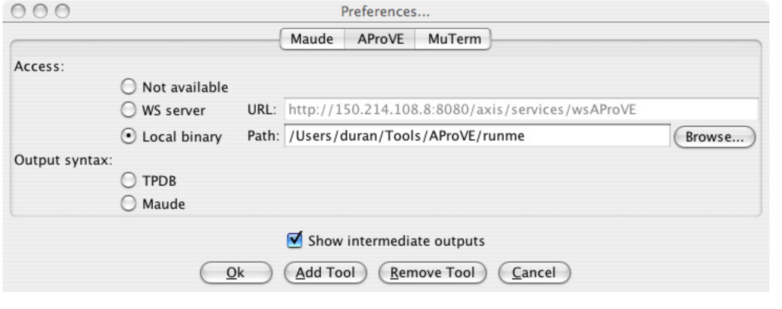

Fig. 5. Snapshot of the tool preferences window.

tation, and some benchmarks are also available.

5 Conclusions and future work

MTT 5 can check the termination of Maude programs (both rewrite theories and membership equational theories).

The techniques needed to go from standard termination methods and tools to termination tools for programs in rule-based languages with expresive features has been discussed in different works (see e.g. [10,7,8,18,19]). Here, we have focussed on the implementation of MTT, a new tool for proving termination of Maude programs like the previous one, and more specifically in its interaction facilities. MTT uses the Maude system and a number of termination tools as back-ends with which it must interact. All the interactions happens transparently to the user, hiding it behind a user-friendly graphical interface that helps and guides the user in the proofs. Moreover, although the interaction with Maude is mandatory, the back-end tools used are completely configurable. In fact, the number and characteristics of these tools can change dynamically as the requirements of the user change.

There are still several restrictions on the Maude specifications to be checked

- built-ins cannot be used,
- attributes owise and identity elements are not supported,
- the specification must be in Full Maude notation (each module must be enclosed in parentheses and operator declarations must be in their single-token equivalent form, see the Maude official web site for further information), etc.

Acknowledgement

Special thanks are due to Claude Marché and Xavier Urbain for their collaboration in the development of the first versions of MTT. We would also like to thank José Luis Luque and Ahmed Behiya for their help in the implementation of MTT.

References

[1] Peter Borovanský, Claude Kirchner, Hélène Kirchner, and Pierre-Etienne Moreau. ELAN from the rewriting logic point of view. *Theoretical Computer Science*, 285(2):155–185, August 2002.

[2] Adel Bouhoula, Jean-Pierre Jouannaud, and José Meseguer. Specification and proof in membership equational logic. *Theoretical Computer Science*, 236(1):35–132, 2000.

[3] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José Quesada. Maude: Specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187–243, 2002.

[4] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. *All About Maude - A High-Performance Logical Framework: How to Specify, Program, and Verify Systems in Rewriting Logic.* Lecture Notes in Computer Science. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[5] Evelyne Contejean, Claude Marché, Benjamin Monate, and Xavier Urbain. Proving termination of rewriting with CiME. Technical Report II/15/03, DSIC, Universidad Politécnica de Valencia, Spain, 2003.

[6] Răzvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report. The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*, volume 6 of *AMAST Series in Computing.* World Scientific, 1998.

[7] Francisco Durán, Salvador Lucas, Claude Marché, José Meseguer, and Xavier Urbain. Proving operational termination of membership equational programs. *Higher-Order and Symbolic Computation*, 21(1-2):59–88, 2008.

[8] Francisco Durán, Salvador Lucas, and José Meseguer. Operational termination in rewriting logic. Available at `http://www.dsic.upv.es/~slucas/tr08.pdf`, 2008.

[9] Francisco Durán, Salvador Lucas, and José Meseguer. A survey on proving termination of rewriting-based programming languages by transformation. *Electronic Notes in Theoretical Computer Science*, 2009. This volume.

[10] Francisco Durán, Salvador Lucas, José Meseguer, Claude Marché, and Xavier Urbain. Termination of membership equational programs. In *ACM SIGPLAN 2004 Symposium on Partial Evaluation and Program Manipulation (PEPM'04)*, pages 147–158. ACM Press, 2004.

[11] Jürgen Giesl, Peter Schneider-Kamp, and René Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In U. Furbach and N. Shankar, editors, *Proceedings of the Third International Joint Conference on Automated Reasoning (IJCAR'06)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 281–286. Springer, 2006.

[12] Jürgen Giesl, Stephan Swiderski, Peter Schneider-Kamp, and René Thiemann. Automated termination analysis for haskell: From term rewriting to programming languages. In Frank Pfenning, editor, *Term Rewriting and Applications. 17th International Conference, RTA 2006 Seattle, WA, USA, August 12-14, 2006 Proceedings*, volume 4098 of *Lecture Notes in Computer Science*, pages 297–312. Springer, 2006.

[13] Joseph Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouannaud. Introducing OBJ. In Joseph Goguen and Grant Malcolm, editors, *Software Engineering with OBJ: Algebraic Specification in Action*. Kluwer, 2000.

[14] Nao Hirokawa and Aar Middeldorp. Tyrolean termination tool: Techniques and features. *Information and Computation*, 205(4):474–511, April 2007.

[15] Paul Hudak, Simon Peyton-Jones, and Philip Wadler. Report on the functional programming language Haskell: a non–strict, purely functional language. *SIGPLAN Notices*, 27:1–164, 1992.

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

[17] Salvador Lucas. MU-TERM: A tool for proving termination of context-sensitive rewriting. In V. van Oostrom, editor, *Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA'04)*, volume 3091 of *Lecture Notes in Computer Science*, pages 200–209. Springer, 2004.

[18] Salvador Lucas, Claude Marché, and José Meseguer. Operational termination of conditional term rewriting systems. *Information Processing Letters*, 95(4):446–453, 2005.

[19] Salvador Lucas and José Meseguer. Operational termination of membership equational programs: the order-sorted way. To appear in Electronic Notes in Theoretical Computer Science, 2009.

[20] Claude Marché and Hans Zantema. The termination competition. In Franz Baader, editor, *Term Rewriting and Applications, 18th International Conference, RTA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4533 of *Lecture Notes in Computer Science*, pages 303–313. Springer, 2007.

[21] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[22] José Meseguer. Membership algebra as a logical framework for equational specification. In Francesco Parisi-Presicce, editor, *Recent Trends in Algebraic Development Techniques*, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61. Springer, 1998.

[23] Ulf Nilsson and Jan Maluszynski. *Logic, Programming and Prolog*. John Wiley & Sons, second edition, 1995.

[24] Arie van Deursen, Jan Heering, and Paul Klint. *Language Prototyping: An Algebraic Specification Approach*. World Scientific, 1996.

[25] Hans Zantema. Torpa: Termination of string rewriting systems. *Journal of Automated Reasoning*, 34(105-139), 2006.