The Maude Formal Tool Environment

Manuel Clavel[1], Francisco Durán[2], Joe Hendrix[3], Salvador Lucas[4], José Meseguer[3], and Peter Ölveczky[5]

[1] Universidad Complutense de Madrid, Spain
[2] Universidad de Málaga, Spain
[3] University of Illinois at Urbana-Champaign, IL, USA
[4] Universidad Politécnica de Valencia, Spain
[5] University of Oslo, Norway

Abstract. This paper describes the main features of several tools concerned with the analysis of either Maude specifications, or of extensions of such specifications: the ITP, MTT, CRC, ChC, and SCC tools, and Real-Time Maude for real-time systems. These tools, together with Maude itself and its searching and model-checking capabilities constitute Maude's formal environment.

1 Introduction

Maude is a language and a system based on rewriting logic [1, 2]. Maude modules are rewrite theories, while computation with such modules corresponds to efficient deduction by rewriting. Because of its logical basis and its initial model semantics, a Maude module defines a precise mathematical model. This means that Maude and its formal tool environment can be used in three, mutually reinforcing ways: as a declarative programming language, as an executable formal specification language, and as a formal verification system. The Maude system, its documentation, and related papers and applications are available from the Maude website `http://maude.cs.uiuc.edu`.

Besides being able to use Maude's inductive theorem prover (ITP) to verify inductive properties of functional modules, and the built-in support for verifying invariants and LTL formulas, we can use the following Maude tools to formally verify other properties: (1) the Maude Termination Tool (MTT) [8] can be used to prove termination of functional modules (§3); (2) the Maude Church-Rosser Checker (CRC) [3] can be used to check the Church-Rosser property of functional modules (§4); (3) the Maude Coherence Checker (ChC) can be used to check the coherence (or ground coherence) of unconditional system modules (§5); and (4) the Maude Sufficient Completeness Checker (SCC) [11] can be used to check that defined functions have been fully defined in terms of constructors (§6). Furthermore, if we are dealing with rewriting logic specifications of real-time systems, we can use the Real-Time Maude tool (§7) to both simulate such specifications and to perform search and model-checking analysis of their LTL properties. Full Maude, an extension of Maude, written in Maude itself, has played a key role in the construction of some of these tools. Full Maude has become a common infrastructure on top of which tools like these can be built, but also environments for other languages, such as, e.g., the Real-Time Maude tool.

In the following sections we summarize the main features of these tools. For further details on them please check the given references or visit the indicated web sites.

2 The ITP: an Inductive Theorem Prover

The ITP tool [4] is a theorem-proving *assistant*. It can be used to interactively verify properties of membership equational specifications. An important feature of the ITP is that it supports proofs by structural induction and complete induction. Another interesting feature is that incompletely specified operations can be reasoned about so as to support incrementality. That is, operations do not have to be completely specified before inductive properties about them can be verified mechanically. The ITP tool is a Maude program. It comprises over 8000 lines of Maude code that make extensive use of the reflective capabilities of the system. In fact, rewriting-based proof simplification steps are directly executed by the powerful underlying Maude rewriting engine. The ITP tool is currently available as a web-based application that includes a *module editor*, a *formula editor*, and a *command editor*. These editors allow users to create and modify their specifications, to formalize properties about them, and to guide the proofs by filling in and submitting web forms. The web application also offers a goal viewer, a script viewer, and a log viewer. They generate web pages that allow the user to check, print, and save the current state of a proof, the commands that have guided it, and the logs generated in the process by the Maude system. The ITP web-based application can be accessed at `http://maude.sip.ucm.es:8080/webitp/`. The ITP is still an experimental tool, but the results obtained so far are quite encouraging. It is the only theorem prover at present that supports reasoning about membership equational logic specifications. The powerful integration of term rewriting with a decision procedure for linear arithmetic with uninterpreted function symbols [5], while also available in other rewriting-based theorem provers like RRL [14], has been easily and efficiently implemented in the ITP by exploiting the reflective design of the tool and the reflective capabilities of the Maude system. This fact has encouraged us to plan to add other decision procedures to our tool in the near future. Another interesting extension of the tool is the implementation of the cover set induction method, a feature already available in RRL [15].

3 The Maude Termination Tool

The Maude Termination Tool (MTT) checks the termination of Maude specifications. Maude, as other equational and rule-based programming languages, has expressive features such as: advanced typing constructs including sorts, subsorts, kinds, and memberships; matching modulo axioms; evaluation strategies; and very general conditional rules. Proving termination of programs having such features is nontrivial, since some of these features are not supported by standard termination methods and tools. Yet, the use of such features may be essential to ensure termination. MTT uses several theory transformations, some of which are described in [7, 8], to bridge the gap between expressive equational programs and conventional termination tools for (variants of) term rewriting systems, which are used as backends. Currently, MU-TERM [16] and AProVE [9] provide the most accurate termination proofs for MTT and they can be used as backends. Tools which implement less specific (but still valid) proofs like C*i*ME [6] can be used as well. The transformed termination problems are given to the back-end tools in TPDB syntax. This makes MTT extensible, so that new tools supporting TPDB syntax can be added as back-ends. The tool implementation distinguishes two parts: a reflective Maude specification implements the theory transformations described in [7, 8], and a Java application connects Maude to the back-end termination tools and provides a graphical user interface. The Java application is in charge of sending the Maude specification provided by the user to Maude to perform transformations. Depending on the selections, one transformation or another will be accomplished. The resulting unsorted unconditional (context-sensitive) rewriting system obtained from such transformations is proved terminating by using the above-mentioned tools as backends. To alleviate the installation requirements on external tools, the application includes support for connecting to the external tools remotely via different alternatives, including sockets, RMI, and web services. This feature is particularly attractive for those platforms for which there is no version available of some of the tools. The tool and all the related information is available from `http://www.lcc.uma.es/~duran/MTT`.

4 The Church-Rosser Checker

For order-sorted specifications, being Church-Rosser and terminating means not only confluence—so that a unique normal form will be reached—but also a *sort decreasingness* property, namely that the normal form will have the least possible sort among those of all other equivalent terms. The *Church-Rosser Checker* (CRC) [3] is a tool to help checking whether a Maude order-sorted conditional equational specification satisfies the Church-Rosser property.

A specification with an initial algebra semantics can often be ground Church-Rosser even though some of its critical pairs may not be joinable. That is, the specification can often be ground Church-Rosser without being Church-Rosser for arbitrary terms with variables. The CRC can be used to check specifications with an initial algebra semantics that have already been proved terminating and now need to be checked to be ground Church-Rosser. If the specification cannot be shown to be ground Church-Rosser by the tool, proof obligations consisting of a set of critical pairs and a set of membership assertions that must be shown, respectively, ground-joinable, and ground-rewritable to a term with the required sort are generated and are given back to the user as a useful guide in the attempt to establish the ground Church-Rosser property. Since this property is in fact inductive, in some cases the ITP (§2) can be enlisted to prove some of these proof obligations. In other cases, the user may in fact have to modify the original specification by carefully considering the information conveyed by the proof obligations. The tool is written entirely in Maude, and is in fact an *executable specification* of the formal inference system that it implements. A complete execution environment for the tool has been built in Maude, and it has been integrated within Full Maude. The tool, together with its documentation and some examples, is available from `http://www.lcc.uma.es/~duran/CRC`.

5 The Maude Coherence Checker

Coherence is a key executability requirement for rewrite theories. It allows reducing the, in general undecidable, problem of computing rewrites of the form $[t]_{E \cup A} \longrightarrow [t']_{E \cup A}$, with A a set of equational attributes (associativity, commutativity, identity) for which matching algorithms exist and E a set of equations, to the much simpler and decidable problem of computing rewrites of the form $[t]_A \longrightarrow [t']_A$. The *Maude Coherence Checker* (ChC), which is written in Maude using a reflective design as an extension of Full Maude, provides a *decision procedure* for order-sorted system modules whose equations and rules are unconditional. The tool generates a set of *critical pairs*, whose coherence guarantees that of the entire system module. It then checks whether each of these pairs is coherent. The system module given as input to the tool is always assumed to be ground Church-Rosser and terminating. The CRC (§4) and the MTT (§3) can be used to try to prove such properties. For Maude system modules, which have an initial model semantics, the weaker requirement of *ground coherence*, that is, coherence for ground terms, is enough. When the ChC tool cannot prove coherence—either because this fails, or because the input specification falls outside the class of decidable theories—it outputs a set of *proof obligations* associated with the critical pairs that it could not prove coherent. The user can then interact with the ChC tool to try to prove the ground coherence of the input system module by a constructor-based process of reasoning by cases. In the end, either: (1) all proof obligations are discharged and the module is shown to be ground coherent; or (2) proving ground coherence can be reduced to proving that the inductive validity of a set of equations follows from the equational part of the input system module, for which the ITP can be used (§2); or (3) it is not possible to reduce some of the proof obligations to inductively proving some equations. Case (3) may be a clear indication that the specification is not ground coherent, so that a new specification should be developed. The tool, together with its documentation and some examples, is available from `http://www.lcc.uma.es/~duran/ChC`.

6 The Sufficient Completeness Checker

The *Maude Sufficient Completeness Checker* (SCC) [11] is a tool for checking that each operation in a equational Maude specification is defined on all valid inputs. The SCC verifies that the constructor operator declarations are annotated with the `ctor` attribute, and that enough equations have been given so that the remaining operations reduce to constructor terms. Specifications may import any of the built-in Maude modules. The tool uses the characterization of sufficient completeness given in [12] which allows for operations to be intentionally partial by declaring these operations at the kind level rather than the sort level. This allows the SCC to successfully analyze such specifications without raising false warnings. The tool is designed for unparameterized, order-sorted, left-linear, and unconditional Maude specifications that are ground terminating and Church-Rosser. It is a decision procedure for this class when every associative symbol is commutative, but for associative symbols that are not commutative it uses an algorithm from [13] based on machine learning techniques that works well in practice. If the specification is not sufficiently-complete, the SCC returns a counterexample to aid the user in identifying errors. The tool is not complete for specifications with non-linear or conditional axioms, but nevertheless has proven useful in identifying errors. The SCC accepts interactive commands to check the sufficient completeness of a Maude module, and internally constructs a *Propositional Tree Automaton* [13] whose language is empty iff the Maude module is sufficiently complete. The emptiness check is performed by a C++ tree automata library named CETA. Recently, the tool has been extended to check several important completeness problems of context-sensitive specifications [10]. It requires an extended version of Maude 2.3 linked to CETA, and may be downloaded from the SCC website at `http://maude.cs.uiuc.edu/tools/scc`.

7 The Real-Time Maude Tool

The Real-Time Maude tool [18] extends Maude to support the formal specification and analysis of real-time systems. The system's state space and its *instantaneous* transitions are defined, as in Maude, by, respectively, a membership equational logic theory and a set of rewrite rules. Time elapse is modeled by *tick rewrite rules* of the form `{`t`} => {`t'`} in time` u `if` *cond*, where `{_}` is an operator that encloses the state. Real-Time Maude extends Maude's efficient rewriting, search, and LTL model checking capabilities to the timed setting by: (i) analyzing behaviors up to a given time duration; and (ii) by having a *time sampling* treatment of *dense* time, in which only some moments in time are visited. Real-Time Maude is implemented in Maude, and achieves high performance by *simultaneously* transforming a real-time module *and* a query into a semantically equivalent Maude rewrite theory and a Maude query.

Real-Time Maude has proved useful to model real-time systems in an object-oriented way. In particular, the ease and flexibility with which an appropriate form of communication can be defined has been exploited in state-of-the-art applications including: (i) The AER/NCA protocol suite for multicast in active networks [19]—we were able to find all known bugs in AER/NCA, as well as some previously unknown bugs not discovered by traditional testing and simulation—; (ii) the OGDC wireless sensor network algorithm [20]—we have shown that Real-Time Maude simulations provide more reliable estimates of the *performance* of OGDC than the simulation tool used by the OGDC developers. On the theoretical level, we have given simple and easily checkable conditions for object-oriented specifications that ensure that Real-Time Maude analyses are sound and complete also for dense time [17]. The useful class of systems satisfying these criteria include AER/NCA and OGDC, that are clearly beyond the pale of timed automata. The Real-Time Maude tool is a mature tool, which is available, together with its documentation and several case studies, from `http://www.ifi.uio.no/RealTimeMaude`.

References

1. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: specification and programming in rewriting logic. *Th. Comp. Sci.*, 285(2):187–243, 2002.
2. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott. *All About Maude, A High-Performance Logical Framework*, vol. 4350 of *LNCS*.
3. M. Clavel, F. Durán, S. Eker, and J. Meseguer. Building equational proving tools by reflection in rewriting logic. In *CAFE: An Industrial-Strength Algebraic Formal Method*. Elsevier, 2000.
4. M. Clavel, M. Palomino, and A. Riesco. Introducing the ITP tool: a tutorial. *J. of Universal Computer Science*, 2007. To appear.

5. M. Clavel, M. Palomino, and J. Santa-Cruz. Integrating decision procedures in reflective rewriting-based theorem provers. In S. Antoy and Y. Toyama, eds., *Procs. WRS'04*.
6. E. Contejean, C. Marché, B. Monate, and X. Urbain. Proving termination of rewriting with C*i*ME. In A. Rubio, ed., *Procs. of WST'03*, pp. 71–73, 2003.
7. F. Durán, S. Lucas, J. Meseguer, C. Marché, and X. Urbain. Proving termination of membership equational programs. In P. Sestoft and N. Heintze, eds., *Procs. of PEPM'04*, pp. 147–158, 2004.
8. F. Durán, S. Lucas, J. Meseguer, C. Marché, and X. Urbain. Proving operational termination of membership equational programs. *Higher-Order and Symb. Comp.*, 2007. To appear.
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, eds., *Procs. of IJCAR'06*, vol. 4130 of *LNAI*, pp. 281–286, 2006.
10. J. Hendrix and J. Meseguer. On the completeness of context-sensitive order-sorted specifications. Tech. Report UIUCDCS-R-2007-2812, U. of Illinois, 2007.
11. J. Hendrix, J. Meseguer, and H. Ohsaki. A sufficient completeness checker for linear order-sorted specifications modulo axioms. In U. Furbach and N. Shankar, eds., *Procs. of IJCAR'06*, vol. 4130 of *LNAI*, pages 151–155, 2006.
12. J. Hendrix, H. Ohsaki, and J. Meseguer. Sufficient completeness checking with propositional tree automata. Tech. Report UIUCDCS-R-2005-2635, U. of Illinois, 2005.
13. J. Hendrix, H. Ohsaki, and M. Viswanathan. Propositional tree automata. In F. Pfenning, ed., *Procs. of RTA'06*, vol. 4098 of *LNCS*, pp, 50–65, 2006.
14. D. Kapur and M. Subramaniam. New uses of linear arithmetic in automated theorem proving by induction. *J. of Automated Reasoning*, 16(1-2):39–78, 1996.
15. D. Kapur and H. Zhang. An overview of rewrite rule laboratory (RRL). *J. Computer and Mathematics with Applications*, 29(2):91–114, 1995.
16. S. Lucas. MU-TERM: A tool for proving termination of context-sensitive rewriting. In V. van Oostrom, ed., *Procs. of RTA'04*, vol. 3091 of *LNCS*, pp. 200–209, 2004.
17. P. C. Ölveczky and J. Meseguer. Abstraction and completeness for Real-Time Maude. In *Procs. WRLA'06*, 2006.
18. P. C. Ölveczky and J. Meseguer. Semantics and pragmatics of Real-Time Maude. *Higher-Order and Symb. Comp.*, 20(1/2), 2007. To appear.
19. P. C. Ölveczky, J. Meseguer, and C. L. Talcott. Specification and analysis of the AER/NCA active network protocol suite in Real-Time Maude. *Formal Methods in System Design*, 29:253–293, 2006.
20. P. C. Ölveczky and S. Thorvaldsen. Formal modeling and analysis of the OGDC wireless sensor network algorithm in Real-Time Maude. Submitted for publication.