Maude's Internal Strategies

Francisco Durán

DLCC, Universidad de Málaga, Campus de Teatinos, Málaga, Spain
`duran@lcc.uma.es`

Abstract. Maude is a reflective language supporting both rewriting logic and membership equational logic. Reflection is systematically exploited in Maude, endowing the language with powerful metaprogramming capabilities, including declarative strategies to guide the deduction process.

1 Introduction

Maude [3,4] is a high-level language and high-performance system supporting both equational and rewriting logic computation for a wide range of applications. Rewriting logic [12] is a logic of change that can naturally deal with state and with highly nondeterministic concurrent computations. In rewriting logic, the state space of a distributed system is specified as an algebraic data type in terms of an equational specification (Σ, E), where Σ is a signature of sorts (types) and operations, and E is a set of (conditional) equational axioms. In Maude, the underlying equational logic is *membership equational logic* [13], a Horn logic whose atomic sentences are equalities $t = t'$ and *membership assertions* of the form $t : s$, stating that a term t has sort s.

Maude's functional and system modules are, respectively, membership equational theories and rewrite theories. The equations in functional modules, considered as rules in the left to right direction, are assumed to be Church-Rosser and terminating. Therefore, canonical forms are reached by canonical simplification regardless of the order of application. Although the (equational) reductions in Maude are basically *innermost* (or eager), Maude is able to exhibit an outermost (or lazy) behavior on particular operator arguments by using *strategy annotations* [10].

A rewrite theory is a pair (T, R), with T a membership equational theory, and R a collection of (labelled and possibly conditional) *rewrite rules* involving terms in the signature of T. Rewriting in (T, R) happens *modulo* the equational axioms in T.[1] The rules in R need not

[1] Maude supports rewriting modulo all combinations of associativity, commutativity, and identity.

be Church-Rosser and need not be terminating, opening up in this way a whole world of new applications. This generality needs some control when the specifications become executable, because the user needs to make sure that the rewriting process does not go in undesired directions.

In those cases in which we just want to test for executability, or consider the evolution of the system with no specific interest in a concrete execution path, Maude provides two built-in strategies: The *rewrite* command follows a top-down lazy rule-fair strategy, and the *frewrite* command follows a position-fair bottom-up strategy. Maude also provides a *search* command, for those cases in which we are interested in exploring all possible execution paths from the starting term for states satisfying some property. The *search* command does a breadth-first exploration of the tree of possible rewrites.

In general however we may be interested in other forms of execution, and the choice of appropriate *strategies* is crucial for executing rewrite theories. In the Maude system, this need for providing strategies for controlling the rewriting process has been satisfied by developing strategies at the metalevel. Strategies are defined in extensions of the predefined module `META-LEVEL` by using predefined functions in it, like `metaReduce`, `metaApply`, `metaXapply`, etc. as building blocks. It is in this way possible to define at the metalevel a whole variety of internal strategy languages [2,5], that is, the strategy language is defined inside the same rewriting logic framework, instead of being defined as an add-on extralogical feature.

2 Reflection and the `META-LEVEL` module

Informally, a reflective logic is a logic in which important aspects of its metatheory can be represented at the object level in a consistent way, so that the object-level representation correctly simulates the relevant metatheoretic aspects. In particular, rewriting logic is reflective [2], and Maude's language design and implementation make systematic use of the fact that rewriting logic is reflective. The predefined functional module `META-LEVEL` efficiently implements key functionality of the universal theory $\mathcal{U}$.

`META-LEVEL` has sorts `Term` and `Module`, so that the representations of a term t and of a module $\mathcal{R}$ are, respectively, a term $\overline{t}$ of sort `Term` and a term $\overline{\mathcal{R}}$ of sort `Module`. The module `META-LEVEL` also provides key metalevel functions for rewriting and evaluating terms at the metalevel, namely, `metaApply`, `metaXapply`, `metaRewrite`, `metaReduce`, etc. For example, the function `metaReduce` takes as arguments the representation of a module $\mathcal{R}$ and the representation of a term t in that module, and returns the representation of the fully reduced form of the term t using the equations in $\mathcal{R}$, together with its corresponding sort or kind:

```
op metaReduce : Module Term -> ResultPair [special ...] .
op {_,_} : Term Type -> ResultPair [ctor] .
```

The operation `metaXapply` applies a rule on a term in any possible position. The first four arguments are the metarepresentation of a module $\mathcal{R}$, the metarepresentation of a term t in $\mathcal{R}$, a label l of some rules in $\mathcal{R}$, and a set of assignments (possibly empty) defining a partial substitution σ for the variables in those rules. The last natural number enumerates the solutions, since there can be different such rewrites with different substitutions and at different positions. The other two numeric arguments indicate the minimum and maximum depth in the term where the application of the rule can take place.

```
op metaXapply : Module Term Qid Substitution Nat Bound Nat
     ~> Result4Tuple? [special ...] .
op {_,_,_,_} : Term Type Substitution Context
     -> Result4Tuple [ctor] .
```

`metaXapply` returns a tuple of sort `Result4Tuple` consisting of a term, with the corresponding sort or kind, a substitution, and the context inside the given term where the rewriting has taken place.

3 Internal Strategies

There is great freedom for defining many different types of strategies, or even many different strategy languages inside Maude. This can be done in a completely user-definable way, so that users are not limited by a fixed and closed particular strategy language.

Rewriting logic has very good properties as a logical and semantic framework, in which many other logics and many semantic formalisms can be naturally represented [11,14]. In Maude, the meta-theory of rewriting logic is accessible to the user in a clear and principled way, giving to Maude very good properties as a logical and semantic framework, in which many different logics and formalisms can be expressed and executed. In fact, some of the most interesting applications of Maude are *metalanguage* applications, in which Maude is used to create executable environments for different logics, theorem provers, languages, and models of computation.

Reflection allows a complete control of the rewriting of a given term using the rewrite rules in a theory. This expressive power has been used in different applications. For example, in Real Time Maude [16] modules there is a distinction between eager and lazy rules, and only rewriting paths that satisfy the requirement that lazy rules are only applied when no eager rule can be applied make sense for this kind of modules; a object-fair strategy was used in Mobile Maude [6] a long time before such an strategy was available in Maude (such an internal strategy was in fact a prototype specification of the `frewrite` object-fair strategy currently available in Maude); Durán, Escobar and Lucas have proposed in [7] an extension of Full Maude which includes commands that compute (constructor) normal forms of initial expressions even when the use of strategy annotations together with the built-in computation strategy of Maude is not able to obtain them; the same authors have proposed in [8] another extension furnishing Maude with the ability of dealing with on-demand strategy annotations; Braga [1] extended Full Maude to support rewrites in the conditions of rules some time before it was available in Maude to be able to represent Action Semantics [15]; Pita and Martí-Oliet proposed in [17] the use of a meta-object to control the execution of a set of rules, which had to be applied following a specific order; etc.

Although very powerful, there are many applications in which simpler strategies are enough, for which it would be desirable to provide ways of avoiding the conceptual complexity of going to the metalevel. In this line, Martí-Oliet, Meseguer, and Verdejo have proposed in [**?**] an object-level basic strategy language for Maude very close to the ELAN strategies, and Durán, Roldán and Vallecillo have proposed generic invariant-driven strategies that control the execution of systems by guaranteeing that the given invariants are satisfied [9]. Both proposals has been implemented in Maude (the first one as an extension of Full Maude), which shows the expressiveness of the reflective capabilities of Maude for defining strategies.

The Maude system, its documentation, a collection of examples and case studies, and a list of related papers are available (free of charge) at `http://maude.cs.uiuc.edu`.

References

1. C. O. Braga. *Rewriting Logic as a Semantic Framework forModular Structural Operational Semantics*. PhD thesis, Departamento de Informática, Pontificia Universidade Católica de Rio de Janeiro, Brasil, 2001.
2. M. Clavel. *Reflection in General Logics and in Rewriting Logic with Applications to the Maude Language*. PhD thesis, Universidad de Navarra, 1998.
3. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: Specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187–243, 2002.
4. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. Maude 2.0 manual. Available in `http://maude.cs.uiuc.edu`., June 2003.
5. M. Clavel and J. Meseguer. Reflection in conditional rewriting logic. *Theoretical Computer Science*, 285, 2002.
6. F. Durán, S. Eker, P. Lincoln, and J. Meseguer. Principles of Mobile Maude. In *Proceedings of Joint Symposium ASA/MA 2000*, volume 1882 of *Lecture Notes in Computer Science*. Springer, 2000.
7. F. Durán, S. Escobar, and S. Lucas. New evaluation commands for maude within full maude. In *Proceedings of 5th International Workshop on Rewriting Logic and its Applications (WRLA'04)*, 2004.
8. F. Durán, S. Escobar, and S. Lucas. On-demand evaluation for maude. In *Proceedings of RULE'04*, 2004.
9. F. Durán, M. Roldán, and A. Vallecillo. Invariant-driven strategies for maude. In *Proceedings of WRS'04*, 2004.
10. S. Eker. Term rewriting with operator evaluation strategy. In C. Kirchner and H. Kirchner, editors, *Proceedings of 2nd International Workshop on Rewriting Logic and its Applications (WRLA'98)*, volume 15 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1998. Available at `http://www.elsevier.nl/locate/entcs/volume15.html`.
11. N. Martí-Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework. volume 9, pages 1–87. Kluwer Academic Publishers, second edition, 2002.
12. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96:73–155, 1992.
13. J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Recent Trends in Algebraic Development Techniques*, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61. Springer, 1998.
14. J. Meseguer. Research directions in rewriting logic. In U. Berger and H. Schwichtenberg, editors, *Computational Logic*. Springer, 1999.
15. P. Mosses. *Action Semantics*. Cambridge University Press, 1992.
16. P. C. Ölveczky and J. Meseguer. Specification of real-time and hybrid systems in rewriting logic. *Theoretical Computer Science*, 285, 2002.
17. I. Pita and N. Martí-Oliet. A Maude specification of an object-oriented model for telecommunication networks. *Theoretical Computer Science*, 285, 2002.