Using Maude*

Manuel Clavel[1], Fransisco Durán[2], Steven Eker[2], Patrick Lincoln[2],
Narciso Martí-Oliet[3], Jose Meseguer[2], and Jose F. Quesada[4]

[1] Department of Philosophy, University of Navarre, Spain
[2] SRI International, Menlo Park, CA 94025, USA
[3] Facultad de Ciencias Matemáticas, Universidad Complutense, Madrid, Spain
[4] CICA (Centro de Informática Científica de Andalucía), Seville, Spain

Maude is a wide-spectrum reflective logical language based on rewriting logic [7] that can be used to specify, prototype, and formally analyze concurrent software systems, specification languages, logics, and theorem provers. Because of its efficient implementation, it can also be used as a programming language and as a meta-tool to generate other tools. This paper gives a brief introduction to the language and illustrates with examples some of the features of the current version, available free of charge together with examples, documentation, and papers from SRI: see `http://maude.csl.sri.com`. The key characteristics of Maude can be summarized as follows:

- *Based on rewriting logic.* This makes it particularly well suited to express concurrent and state-changing aspects of systems declaratively.
- *Wide-spectrum.* Rewriting logic is a logical and semantic framework for both specification and efficient execution.
- *Multiparadigm.* Since rewriting logic conservatively extends equational logic, an equational style of functional programming is naturally supported in a sublanguage. A declarative style of concurrent object-oriented programming is also supported with a simple logical semantics.
- *Reflective.* Rewriting logic is reflective [4, 1]. The design of Maude capitalizes on this fact to support a novel style of *metaprogramming* with very powerful module-combining and module-transforming operations that surpass those of traditional parameterized programming.
- *Internal Strategies.* The strategies controlling the rewriting process can be defined by rewrite rules and can be reasoned about inside the logic [4, 5, 1].

Maude's implementation has been designed with the explicit goals of supporting executable specification and formal methods applications, of being easily extensible, and of supporting reflective computations. Although it is an interpreter, its advanced semicompilation techniques support flexibility and traceability without sacrificing the performance of up to 1.665 million rewrites per second in the free theory and between 131 thousand, and 1 million rewrites per second if associativity and commutativity axioms are used, on a 500MHz Alpha.

* Supported through Rome Laboratories contract F30602-97-C-0312, by DARPA and NASA through Contract NAS2-98073, by Office of Naval Research Contract N00014-96-C-0114, and N00014-99-C-0198.

A Decision Procedure for Bands

Bands are idempotent semigroups. Deciding the word problem for bands is a subtle problem, since the naive approach of using the idempotency equation as a string rewriting rule yields a nonconfluent system. The Maude module below specifies the confluent and terminating equational system proposed by Siekmann and Szabo [8]. The subtle part is that, even though the rules are string rewriting rules applied *modulo* associativity, the added conditional rule has to compare *sets* of elements in appropriate substrings. Thus, in addition to a sort `List` with an associative concatenation operator, we also need an auxiliary sort `Set` with an associative and commutative union operation and an idempotency equation. This illustrates Maude's support for rewriting *modulo* equational axioms. All combinations of associativity, commutativity, and left and right identity are supported.

```
fmod ASSOC-IDP is protecting QID .
  sorts List Set .  subsorts Qid < List Set .
  op __ : List List -> List [assoc] .         *** list concatenation
  op _,_ : Set Set -> Set [assoc comm] .      *** set union
  op {_} : List -> Set .                      *** set of a list
  var I : Qid .  var S : Set .   vars L P Q : List .
  eq S,S = S .  eq L L = L .                  *** set and list idempotence
  ceq L P Q = L Q if {L} == {Q} and {L P} == {L} .
  eq {I} = I .  eq {I L} = I,{L} .
endfm
```

We can then decide the equality of two given words by equality, e.g.,

```
reduce 'a 'b 'c == 'a 'b 'c 'b 'a 'b 'c
```

It is not difficult to see that both words are reductions, using the idempotency equation as a rule, from the common ancestor `'a 'b 'a 'b 'c 'b 'a 'b 'c`.

Reflection and the `META-LEVEL`

Rewriting logic is reflective [4, 1] in the precise sense that there is a finitely presented rewrite theory U such that for any finitely presented rewrite theory T (including U itself) we have the following equivalence

$$T \vdash t \longrightarrow t' \iff U \vdash \langle \overline{T}, \overline{t} \rangle \longrightarrow \langle \overline{T}, \overline{t'} \rangle$$

where $\overline{T}$ and $\overline{t}$ are terms representing T and t as data elements of U. In Maude reflection is efficiently supported through its predefined `META-LEVEL` module, which provides key functionality for the universal theory U. In particular, `META-LEVEL` has sorts `Term` and `Module`, whose respective terms are metarepresentations $\overline{t}$ and $\overline{T}$, for t term and T a module (that is, a rewrite theory). For example, a term $t =$`f(a,g(b))`, in a module `FOO`, with `a`, `b` constants of sort `Foo` is metarepresented as $\overline{t} =$`'f[{'a}Foo,'g[{'b}Foo]]`. `META-LEVEL` has a number of functions for performing metalevel computations in the universal theory [2]. In particular, the `meta-apply` function applies at the metalevel a rule in a module to a term. Its operator declaration is

```
op meta-apply : Module Term Qid Substitution MachineInt -> ResultPair .
```

The first and second arguments are metarepresentations $\overline{T}$, and $\overline{t}$ for a module T and a term t; the third argument is the label of the rule, the fourth is a substitution instantiating some variables in the rule, and the fifth is a number indicating the match instance with which we want to rewrite. Since matching may be performed *modulo* axioms such as associativity, commutativity and/or identity, in general a rule may match a subject term in several different ways. The result of applying the function is a pair, consisting of the (metarepresentation of) the rewritten term and the matching substitution in case of success, or an error constant and the empty substitution in case of failure.

A Reflective Example

The following example demonstrates the use of the Maude metalevel. The example consists of defining a metalevel function `findAllRewrites` that, given a term t in a module T, will find (the representation of) all one-step rewrites from t. Note that `meta-apply` only applies a rule at the top of the subject term, whereas here we want all one-step rewrites at all term positions.

```
fmod META is protecting META-LEVEL .
  sort TermSet .  subsort Term < TermSet .
  var T : Term .  var S : Substitution . var L : Qid .
  vars TL Before After : TermList .  vars OP SORT : Qid .
  var N : MachineInt .   op first : ResultPair -> Term .
  op ~ : -> TermList .  eq ~, TL = TL .  eq TL, ~ = TL .  op {} : -> TermSet .
  op _|_ : TermSet TermSet -> TermSet [assoc comm id: {}] .
  op meta-apply1 : Term Qid MachineInt -> Term .
  op findAllRewrites : Term Qid -> TermSet .
  op findTopRewrites : Term Qid MachineInt -> TermSet .
  op findLowerRewrites : Term Qid -> TermSet .
  op rewriteArguments : Qid TermList TermList Qid -> TermSet .
  op rebuild : Qid TermList TermSet TermList -> TermSet .
  eq meta-apply1(T, L, N) = first(meta-apply(['FOO], T, L, none, N)) .
  eq T | T = T .  eq first({T, S}) = T .
  eq findAllRewrites(T, L) = findTopRewrites(T, L, 0) | findLowerRewrites(T, L) .
  eq findTopRewrites(T, L, N) = if meta-apply1(T, L, N) == error* then {}
       else meta-apply1(T, L, N) | findTopRewrites(T, L, N + 1) fi .
  eq findLowerRewrites({OP}SORT, L) = {} .
  eq findLowerRewrites(OP[TL], L) = rewriteArguments(OP, ~, TL, L) .
  eq rewriteArguments(OP, Before, T, L) = rebuild(OP, Before, findAllRewrites(T, L), ~) .
  eq rewriteArguments(OP, Before, (T, After), L) =
       rebuild(OP, Before, findAllRewrites(T, L), After) |
         rewriteArguments(OP, (Before, T), After, L) .
  eq rebuild(OP, Before, {}, After) = {} .
  eq rebuild(OP, Before, T, After) = OP[Before, T, After] .
  eq rebuild(OP, Before, (T | TS), After) = (OP[Before, T, After]) |
       rebuild(OP, Before, TS, After) .
endfm
```

Given a module `FOO` with only one rule labeled `'one`, the following finds all one step rewrites from `f(a, g(b))`.

```
reduce findAllRewrites('f[{'a}Foo,'g[{'b}Foo]], 'one).
```

Applications

Maude is a wide-spectrum language and an attractive formal meta-tool for building many advanced applications and formal tools. Substantial applications include: a module system for Maude implemented in Maude [6], an inductive theorem prover; a Church-Rosser checker (both part of a formal environment for Maude and for the CafeOBJ language [3]); an HOL to Nuprl translator; a proof assistant for the Open Calculus of Constructions (OCC); and a translator from J. Millen's CAPSL specification language to the CIL intermediate language. In addition, several language interpreters and strategy languages, several object-oriented specifications—including cryptographic protocols and network applications—and a variety of executable translations mapping logics, architectural description languages and models of computation into the rewriting logic reflective framework have been developed by different authors.

We thank everyone on the Maude team for their contributions to the system and this paper. We thank our colleagues working on similar systems such as CafeOBJ and ELAN for interesting discussions and helpful comments.

References

[1] Manuel Clavel. Reflection in general logics and in rewriting logic, with applications to the Maude language. Ph.D. Thesis, University of Navarre, 1998.

[2] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José Quesada. Maude: specification and programming in rewriting logic. SRI International, January 1999, `http://maude.csl.sri.com`.

[3] Manuel Clavel, Francisco Durán, Steven Eker, and José Meseguer. Building equational proving tools by reflection in rewriting logic. In *Proc. of the CafeOBJ Symposium '98, Numazu, Japan*. CafeOBJ Project, April 1998. `http://maude.csl.sri.com`.

[4] Manuel Clavel and José Meseguer. Reflection and strategies in rewriting logic. In J. Meseguer, editor, *Proc. First Intl. Workshop on Rewriting Logic and its Applications*, volume 4 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1996. `http://www.elsevier.nl/cas/tree/store/tcs/free/noncas/pc/volume4.htm`.

[5] Manuel Clavel and José Meseguer. Internal strategies in a reflective logic. In B. Gramlich and H. Kirchner, editors, *Proceedings of the CADE-14 Workshop on Strategies in Automated Deduction (Townsville, Australia, July 1997)*, pages 1–12, 1997.

[6] Francisco Durán. A reflective module algebra with applications to the Maude language. Ph.D. Thesis, University of Malaga, 1999.

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

[8] J. Siekmann and P. Szabo. A noetherian and confluent rewrite system for idempotent semigroups. *Semigroup Forum*, 25:83–110, 1982.