Using Rewriting Logic to implement FULL

[Extended Abstract]

J. Bryans[1], A. Verdejo[2] and C. Shankland[1]

[1]Department of Computing Science and Mathematics,
University of Stirling, Stirling, FK9 4LA, UK
email: {jwb, ces}@cs.stir.ac.uk

[2]Universidad Complutense de Madrid, 28040 Madrid, Spain
email: alberto@sip.ucm.es

1 Introduction

A common problem of automated tools for Formal Methods is their difficulty in dealing with infinite systems. Such systems may arise through the use of infinite data types. For example, in the LOTOS [7] formal description technique the simple process $g?x : Nat; exit$ results in an infinite choice, one for each member of Nat. Rewriting logic [8] is a possible approach to dealing with such systems.

Rewriting logic is a logic of *becoming* or *change* that can naturally deal with state and with highly nondeterministic concurrent computations. It can be used as a semantic framework for a wide range of languages and models of concurrency. Maude [4] is an executable specification language based on rewriting logic and supporting both equational and rewriting logic computation. Maude can be used as a *metalanguage* [5] in which executable environments for different logics, languages, and models of computation are created. Maude has powerful metaprogramming capabilities, with which the user can define strategies to guide the deduction process.

We present here work on the use of rewriting logic and Maude to implement a model checker for the modal logic FULL [1, 2]. FULL is used to describe properties over data and processes, and is verified with respect to *symbolic transition systems* (STS).

2 Symbolic Transition Systems and FULL

Symbolic Transition Systems are essentially transition systems whose transitions can have free variables in the data label and are additionally labelled with a *transition condition* representing the conditions under which that transition is available. This approach was first introduced by Hennessy and Lin [6] who gave a symbolic semantics for value passing CCS.

Basically, an STS is a directed graph whose nodes are tagged with sets of free variables, and whose branches are labelled with a boolean condition and an event. For example, $g?x : Nat[x > 5]; P \xrightarrow{x > 5 \quad gx} P$ is the transition from the process "$g?x : Nat[x > 5]$ then P" via action gx, where g is a gate name and x is a symbolic value, with condition $x > 5$ to the process "P". This transition may not be taken if the condition is not satisfied. When reasoning about processes we simply collect such conditions and pass them to a separate verification system for the data type.

Although Symbolic Transition Systems can be used independently, our work has been focused on using them to give a new finitely branching semantics [3] for LOTOS. LOTOS is a popular process description language that has been used in a variety of applications. The language includes a rich set of operators for describing both process control *and* data, which may in turn affect control. However, much of the foundational work, and subsequently the verification tools, has ignored all, or parts, of the data aspect of the language because of the problem of infinite branching.

A set of rules presented in [3] define how a symbolic transition system may be constructed from a LOTOS process expression. The resulting transition system is typically a cyclic graph (if recursive processes are involved) and is always of finite width (since only a finite number of branches may be described in a LOTOS process).

In order to describe properties of both the actions and the data in symbolic transition systems we developed a modal logic called Full LOTOS Logic (FULL) Earlier work [1] describes the logic in more detail; here we give an overview.

FULL is based on HML as presented in [9]. The usual modal operators $\langle\ \rangle$ and $[\ ]$ have been supplemented by quantifiers over data. For example, $\langle \exists x\ g\rangle\Phi$ expresses that there is some action gv such that Φ with v substituted for x holds subsequently. The value v may be a specific value in the transition system, or another symbolic one. The other operators are $[\exists x\ g]$, $\langle\forall x\ g\rangle$ and $[\forall x\ g]$. These express different path computations through the STS.

3 Implementation of STSs and FULL using Maude

Following the ideas presented in [10] we have been working on representing the LOTOS symbolic semantics and the FULL logic in Maude.

Symbolic transitions are represented as terms of sort `Transition`, and the semantic rules are translated into rewrite rules where the conclusion is rewritten to the premisses. In this way, we start with a transition to be proved valid and work backwards using the rewriting process, maintaining a set of transitions that have to be fulfilled in order to prove the correctness of the first transition. This transition can be rewritten to the empty set if and only if it is a valid transition in the LOTOS symbolic semantics.

Examples of semantic rules and its translation as rewrite rules are

$$a; P \xrightarrow{\mathrm{tt}\quad a} P$$

```
rl [sym] :
     A ; P -- ?b -- ?a --> ?P
  => --------------------------------
        [ ?b := true ]
        [ ?a := A ]
        [ ?P := P ] .
```

which says that the transition `A ; P -- ?b -- ?a --> ?P` is a valid transition that produces three bindings: metavariable `?b` is bound to the value `true`, `?a` to action `A`, and `?P` to process `P`; and

$$\frac{P_1 \xrightarrow{b\quad \alpha} P_1'}{P_1\,[>P_2 \xrightarrow{b\quad \alpha} P_1'\,[>P_2} \quad \text{if } \mathbf{name}(\alpha) \neq \delta$$

```
rl [sym] :
     P1 [> P2 -- ?b -- ?a --> ?P
  => --------------------------------
        P1 -- ?b -- ?a --> ?(NEW1)P
        [ name(?a) =/= delta ]
        [ ?P := ?(NEW1)P [> P2 ] .
```

There are two problems in this approach: the existence of new variables in the righthand side of the rewrite rules and the nondeterministic application of the semantic rules, arising from, for example, rules for choice. To solve the first problem we use *explicit metavariables*, that will be bound to concrete values in the process of rewriting and these bindings will be propagated to the rest of transitions. For example, in the rule above `?(NEW1)P` is an explicit metavariable.

To solve the second problem of nondeterministic application of rewrite rules, we need a strategy to control the rewriting of a term and the search in the tree of all possible rewrites of a term. This will also help us deal with metavariables.

We extend `META-LEVEL` with operations that generate all the one-step rewritings of a term, and operations that search in a depth-first way the tree of possible rewritings looking for the empty set of transitions. `META-LEVEL` is a predefined Maude module which implements the rewriting logic reflective capabilities and allows us to deal with modules as normal terms and control the rewriting process. In this controlled process of rewriting, new metavariables are created and substituted for the new variables in the righthand side of the rewrite rules.

While these operations work, bindings are produced. If we keep these bindings, we can obtain new information besides the fact that the transition is possible. For example, if we start with the transition where the final process is a metavariable, it will be bound to one of the possible successors of the first process. This is useful to represent the FULL logic.

Some of the operators of the modal logic FULL are equivalent to operators within the Hennessy Milner logic HML, and can be implemented in very much the same way. We begin by defing a sort `FullFormula` to represent the logic and its operators. (At the time of writing not all of the operators of FULL have been implemented.)

```
sort FullFormula .
ops tt ff : -> FullFormula .
op _/\_ : FullFormula FullFormula -> FullFormula .
op `[_`]_ : TermSet FullFormula -> FullFormula .
op forall : TermSet FullFormula -> JudgementSeq .
op _|=_ : Term FullFormula -> Judgement .
```

Then, we define rewrite rules that rewrite a judgement $P \models \Phi$ into the set of judgements that have to be fulfilled.

```
rl [and] : P |= Phi /\ Psi => (P |= Phi) (P |= Psi) .

rl [box] : P |= [ K ] Phi => forall(succ(P, K), Phi) .

eq forall({}, Phi) = emptyJS .
eq forall(P U PS, Phi) = (P |= Phi) forall(PS, Phi) .
```

So, just as the rules for the symbolic semantics rewrite a transition to be proved valid into a set of transitions to be fulfilled, here we rewrite a FULL formula to be proved valid into a set of judgements to be fulfilled. The goal is to produce the empty judgement set, indicating that the original FULL formula is true. For example, the base cases include the rule where the judgement `P |= true` is rewritten to `emptyJS`.

An advantage of using Maude is that the reasoning can be carried out completely automatically by incorporating the Full LOTOS data types in the same tool. This will be done by translating the ACT ONE data type specifications into Maude equational specifications that will be included automatically in the Maude module containing the semantic rules. In this way, rewriting will be perfomed modulo the ACT ONE equations defining the particular data types of the Full LOTOS specification we are working with.

Work is ongoing to implement the rest of the FULL operators, i.e. the quantified modal operators. Particular problems to be addressed are the implementation of substitution (crucial to the quantified modal operators), and the representation of infinite depth transition systems.

References

[1] M. Calder, S. Maharaj, and C. Shankland. A Modal Logic for Full LOTOS based on Symbolic Transition Systems. *The Computer Journal*, 2001. In press.

[2] M. Calder, S. Maharaj, and C. Shankland. An Adequate Logic for Full LOTOS. To appear in Proceedings of Formal Methods Europe'01, LNCS series, Springer-Verlag, 2001.

[3] M. Calder and C. Shankland. A Symbolic Semantics and Bisimulation for Full LOTOS. Technical Report CSM-159, University of Stirling, 2000.

[4] 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, Jan. 1999, revised Aug. 1999. `http://maude.csl.sri.com`.

[5] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José Quesada. Maude as a metalanguage. In Claude Kirchner and Hélène Kirchner, editors, *Proc. 2nd Intl. Workshop on Rewriting Logic and its Applications, Pont-à-Mousson, Nancy, France*, volume 15 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1998. `http://www.elsevier.nl/locate/entcs/volume15.html`.

[6] M. Hennessy and H. Lin. Symbolic Bisimulations. *Theoretical Computer Science*, 138:353–389, 1995.

[7] International Organisation for Standardisation. *Information Processing Systems — Open Systems Interconnection — LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, 1988.

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

[9] C. Stirling. Temporal Logics for CCS. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, LNCS 354, pages 660–672. Springer-Verlag, 1989. REX School/Workshop, Noordwijkerhout, The Netherlands, May/June 1988.

[10] Alberto Verdejo and Narciso Martí-Oliet. Implementing CCS in Maude. In Tommaso Bolognesi and Diego Latella, editors, *Formal Methods For Distributed System Development. FORTE/PSTV 2000 IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communications Protocols (FORTE XIII) and Protocol Specification, Testing and Verification (PSTV XX) October 10–13, 2000, Pisa, Italy*, pages 351–366. Kluwer Academic Publishers, 2000.