Executing E-LOTOS processes in Maude

Alberto Verdejo Narciso Martí-Oliet
alberto@sip.ucm.es narciso@sip.ucm.es
Dpto. Sistemas Informáticos y Programación
Universidad Complutense de Madrid. Spain

Abstract

Rewriting logic can be used as a semantic framework where the formal structural operational semantics of specification languages as E-LOTOS can be described in a easy way. E-LOTOS has been developed within ISO for the formal specification of open distributed concurrent systems in general. Its formal semantics has been already defined, but tools need to be developed in order to show the features of the language, and to check the correctness of the specifications. By using the rewriting logic language Maude, not only the E-LOTOS semantic rules can be described, but they can be executed. Furthermore, due to the rewriting logic reflective capabilities, properties about the E-LOTOS specifications can be proved in the same framework. For example, both an E-LOTOS protocol specification and the tool that checks it by exploring all its behaviours can be implemented by using the Maude language. This approach has been successfully applied to the Milner's toy language CCS, defining its semantics and verifying processes written in that language. The same ideas are being extended to a bigger language such as E-LOTOS.

Introduction

Rewriting logic was introduced in [5] as a unified model of concurrency in which several well known models of concurrent systems were represented in a common framework. This goal was further extended in [4] to the idea of rewriting logic as a logical and semantic framework. It was shown that many other logics, widely different in nature, can be represented inside rewriting logic in a natural and direct way. The general way in which such representations are achieved is by:

- Representing formulas or, more generally, proof-theoretic structures such as sequents, as terms in an order-sorted equational data type whose equations express structural axioms natural to the logic in question.
- Representing the rules of deduction of a logic as rewrite rules that transform certain patterns of formulas into other patterns modulo the given structural axioms.

Similar techniques can be used to naturally specify and prototype many languages and systems in rewriting logic. In particular, the similarities between rewriting logic and structural operational semantics were noted in [5] and further explored in [4]. As an illustrative example, the paper [4] completely develops a representation of Milner's CCS [7] in rewriting logic, extending ideas first introduced in [6].

We claim that these techniques can be applied to a bigger language such as E-LOTOS [8]. E-LOTOS has been developed within ISO for the formal specification of open distributed concurrent systems in general. Its formal semantics has been already defined, but tools need to

be developed in order to show the features of the language, and to check the correctness of the specifications.

By using the rewriting logic language Maude [2], a high-performance language and system supporting both equational and rewriting logic computation, not only the E-LOTOS semantic rules can be described, but they can be *executed*. Furthermore, due to the rewriting logic reflective capabilities, properties about the E-LOTOS specifications can be proved in the same framework. For example, both an E-LOTOS protocol specification and the tool that checks it by exploring all its behaviours can be implemented by using the Maude language.

The general idea, as stated in [4], for implementing in rewriting logic an operational semantics as the one defined for E-LOTOS is to translate each semantic rule into a rewriting rule where the premises are rewritten to the conclusion, or into a rewriting rule where the conclusion is rewritten to the premises. We follow the second approach because we want to be able to prove in a bottom-up way that a given transition is valid in E-LOTOS .

We need operations to build the different judgements in the E-LOTOS semantics. For example, for the judgement $B \xrightarrow{G(RN)} B'$, defining an E-LOTOS transition in its dynamic operational semantics, we have the operator

```
op _--_->_ : Process Act Process -> Judgement .
```

We deal with sets of judgements, so that a semantic rule is represented as a rewriting rule where the unitary set consisting of the judgement representing the conclusion is rewritten to the set consisting of the judgements representing the premises. For example,

```
rl [sel] :       E |- B1 [] B2 -- G(RN) -> B1'
           => ---------------------------------
                  E |- B1 -- G(RN) -> B1' .
```

In particular, in the representation of an axiom the conclusion is rewritten to the empty set of judgements. Thus, a transition is possible in the structural operational semantics of E-LOTOS if and only if the judgement representing it can be rewritten to the empty set of judgements.

However, we found two problems while working with this approach in the current version of Maude. The first one is that sometimes new variables appear in the premises which are not in the conclusion, such as `NEW1`:

```
rl [stasem] :        ( C |- P := E  ==> exit < RT > )
              => ------------------------------------------
                  ( C |- E ==> exit < $ 1 => ?(NEW1)T > )
                   ( C |- ( P =>PM ?(NEW1)T) -> < RT > )  .
```

Rewriting rules with new variables in the righthand side cannot be directly used by the current Maude system.

The second problem is that sometimes several rules can be applied to rewrite a judgement. For example, in the case of the selection operator, in addition to the previous rule, we also have the following one with an equivalent lefthand side

```
rl [sel] :       E |- B1 [] B2 -- G(RN) -> B2'
           => ---------------------------------
                  E |- B2 -- G(RN) -> B2' .
```

In general, not all of these possibilities are successful, so we have to deal with the whole tree of possible rewritings of a judgement, searching if one of the branches leads to the empty set of judgements.

In the following sections, we sketch how these problems can be solved in the current version of the Maude system by using its implementation of the reflective capabilities of rewriting logic, which allow to represent rewriting logic inside itself [3], and in particular to control the rewriting process.

Definition of E-LOTOS semantics in Maude

The problem of new variables in the righthand side of a rewriting rule is solved by using the concept of *explicit metavariables* presented in [9]. New variables in the righthand side of a rewriting rule represent "unknown" values when we are rewriting. By using metavariables we make explicit this lack of knowledge. The semantics with explicit metavariables has to bind them to concrete values when these values are known, and these bindings have to be propagated to other judgements. We will have different kinds of metavariables, one for each element of E-LOTOS which is represented by a new variable in a premise of a semantic rule. For example, we have metavariables for actions and an operator to represent bindings between a metavariable and its current value:

```
op `[_:=_`] : MetaVarAct Act -> MetaBinding .
```

These metabindings have to reach all the judgements, so we introduce an operation to enclose the set of judgements, and a rule to propagate a binding

```
op  {{_}} : JudgementSet -> Configuration .
var JS : JudgementSet .
rl [bind] : {{ [?A := A] JS }} => {{ <act ?A := A > JS }} .
```

We need *new* metavariables each time a rewriting rule is applied. This is achieved by building metavariables with an operator as follows

```
op ?`(_`)A : Qid -> MetaVarAct .
```

Rewriting has to be controlled by a *strategy* that instantiates the metavariables with a new (quoted) identifier each time one of the rules is applied, in order to build *new* metavariables. The strategy presented in the next section does this besides implementing the search in the tree of possible rewritings.

Making the semantics executable

In this section we show how the reflective properties of Maude [3] can be used to control the rewriting of a term, and the search in the tree of possible rewritings of a term. The depth-first strategy is based on the work in [1], although modified to deal with the substitution of metavariables explained in the previous section.

The search strategy is parameterized with respect to a constant `MOD` equal to the metarepresentation of the Maude module which we want to work with. Since we are defining a strategy to search a tree of possible rewritings, we need a notion of search goal. For the strategy to be general enough, we assume that the metarepresented module `MOD` has an operation `ok` (defined at the object level), which returns a value of sort `Answer` such that

- `ok(T) = solution` means that the term `T` is one of the terms we are looking for, that is, `T` denotes a solution;
- `ok(T) = no-solution` means that the term `T` is not a solution and no solution can be found below `T` in the search tree;

- `ok(T) = maybe-sol` means that `T` is not a solution, but we do not know if there are solutions below it.

The strategy controls by means of the operation `meta-apply` the possible rewritings of a term. `meta-apply` returns *one* of the possible rewritings at the top level of a given term. Our first step is to define an operation `allRew` that returns *all* the possible *one-step sequential* rewritings [5] of a given term by using rewriting rules with a given label. Since `meta-apply` only applies rules at the top level of a term, but we need also rewrite at the subterms (or arguments) of the given term, we use operations to get an argument of a term, and to replace an argument by all the possible rewritings of this argument. `meta-apply` has to be used with a substitution that assigns new identifiers to the new variables in the righthand side of the rewriting rule being applied. Hence, `allRew` receives the greatest number used to substitute variables in `T` and uses it to create new identifiers.

Then, we can define a strategy to search in the tree of all possible rewritings of a term `T` a term that satisfies the predicate `ok`. Each node of the search tree is a pair whose first component is a term and whose second component is a number representing the greatest number used as identifier for new variables in the process of rewriting the term.

Finally, following the ideas given in [2] to build strategy expressions, we define the operations

```
sorts Strategy StrategyExpression .
op idle : -> Strategy .
op rewWith : Term Strategy -> StrategyExpression .
op failure : -> StrategyExpression .
op depth : Qid -> Strategy .
```

in such a way that `rewWith(T, depth(L))` means that term `T` has to be rewritten using rules with label `L`, exploring all the possibilities in a depth-first way until a solution is found. The search is defined by using another operation `rewWithDF`

```
op rewWithDF : PairSequence TermSet Qid -> StrategyExpression .
eq rewWith(T, depth(L)) = rewWithDF(< T, 0 >, emptySet, L) .
```

which receives a sequence of pairs containing the terms that have not yet been checked, a set of already visited terms, and the label of the rules to be used. When there is no solution in the tree of rewritings of term `T`, it returns the strategy expression `failure`. If there are solutions then `rewWith(T',idle)` is returned, where `T'` is the first solution found in a depth-first way. The definition is as follows:

```
eq rewWithDF(nilPairSeq, TSe, L) = failure .
eq rewWithDF(seqPair(< T , N >, PL), TSe, L) =
    if isIn(T, TSe) then rewWithDF(PL, TSe, L)
    else (if meta-reduce(MOD, 'ok[T]) == {'solution}'Answer then rewWith(T, idle)
          else (if meta-reduce(MOD, 'ok[T]) == {'no-solution}'Answer then
                   rewWithDF(PL, set(T, TSe), L)
                else rewWithDF(seqPair(buildPairs(allRew(T, L, N), (N + 3)),
                                       PL), set(T, TSe), L)
                fi)
          fi)
    fi .
```

How to obtain information about a process from its semantics

By using the search strategy instantiated with the representation of the Maude module defining the E-LOTOS semantics, we can find out if a given transition $B \xrightarrow{G(RN)} B'$ is possible or not.

We can also take advantage of the metavariables, in order to obtain more information about the E-LOTOS processes, such as the actions that a process can perform or the successors of a process after performing an action. If we look for all the possible rewritings of the term `B -- G(RN) -> ?('proc)P` which lead to an empty set of judgements, each one of these rewritings will bind metavariable `?('proc)P` with the representation of a process `B'` such that $B \xrightarrow{G(RN)} B'$. In order to do that we have to keep the metabindings produced. When a metabinding is produced and it is propagated, it has to be saved in a local memory of produced metabindings,

```
rl [bind] : {{ [?P := P] JS | JS' }} =>
            {{ (<proc ?P := P > JS) | [?P := P] JS' }} .
```

When `{{emptyJudgementSet | JS'}}` is reached, `JS'` will contain all the produced metabindings. In particular, `?('proc)P` will be bound with a sucessor of `B`.

We also have to modify the strategy in order to get *all* the solutions, that is, to explore the whole tree of rewritings finding all the nodes that satisfy function `ok`.

Conclusion

We have sketched how the E-LOTOS structural operational semantics can be described in a framework where it can be executed, in such a way that we can prove whether a semantic judgement, such as a transition, is possible. Moreover, thanks to the use of metavariables, we can obtain information about the E-LOTOS processes, such as the actions that a process can perform or the successors of a process after performing an action.

This is still work in progress in the sense that all the techniques have been developed and successfully applied to Milner's CCS, and we are still in the process of writing all the semantic rules for E-LOTOS in Maude. However, we do not anticipate any new problems.

References

[1] R. Bruni. *Tile Logic for Synchronized Rewriting of Concurrent Systems*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 1999.

[2] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and José Quesada. *Maude: Specification and Programming in Rewriting Logic*. SRI International, March 1999. `http://maude.csl.sri.com`.

[3] M. Clavel and J. 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://www1.elsevier.nl/mcs/tcs/pc/volume4.htm`.

[4] N. Martí-Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework. Technical Report SRI-CSL-93-05, SRI International, Computer Science Laboratory, August 1993.

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

[6] José Meseguer, Kokichi Futatsugi, and Timothy Winkler. Using rewriting logic to specify, program, integrate, and reuse open concurrent systems of cooperating agents. In *Proc. Int. Symposium on New Models for Software Architecture, Tokyo, Japan, November 1992*, pages 61–106. Research Institute of Software Engineering, 1992.

[7] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[8] Juan Quemada, editor. Final committee draft on Enhancements to LOTOS. ISO/IEC JTC1/SC21/WG7 Project 1.21.20.2.3., May 1998.

[9] M.-O. Stehr and J. Meseguer. Pure type systems in rewriting logic. In *Proc. of LFM'99: Workshop on Logical Frameworks and Meta-Languages*, Paris, France, September 1999.