# Implementing the modal logic FULL using Ergo

**[Extended Abstract]**

Peter Robinson
Software Verification Research Centre
Australia
pjr@csee.uq.edu.au

Carron Shankland
University of Stirling
United Kingdom
ces@cs.stir.ac.uk

## 1 Introduction

Although formal methods are becoming more popular, their uptake is at least partially governed by the success of associated tools. Theorem provers give the most flexibility, but are often difficult to use, while model checking tools are easy to use, but seldom deal effectively with large and infinite systems. The solution has often been to ignore such systems, but that is no longer a tenable position. Infinite state systems arise regularly and therefore must be dealt with satisfactorily.

We present a prototype model checker for the modal logic FULL [1]. Two strategies are used to deal with infinite systems. Firstly, the logic is defined over *Symbolic Transition Systems* (STSs), which remove infinite branching. Secondly, the model checker is implemented using the theorem prover Ergo [9], which can reason about those symbolic values.

## 2 LOTOS, Symbolic Transitions and FULL

LOTOS [5] is a popular process description language that has been in use for well over a decade in a number of domains, including protocols and services [7], and distributed systems [6]. The success of these applications has been amplified by the use of a number of mature verification tools.

LOTOS includes a rich set of operators for describing both process control *and* data. However, much of the foundational work on LOTOS, and subsequently the verification tools, has ignored all, or parts, of the data aspect of the language because of the problem of infinite state. In particular, the use of data introduces infinite branching into the underlying state transition systems. For example, the simple process `g?x:Nat; exit` results in an infinite choice, one for each member of `Nat`. This presents a serious obstacle to reasoning, particularly to approaches based on (finite) model-checking.

The focus of our work has been to provide a complete approach to data including semantics, abstract properties expressed using logic, equivalence relations, and tools to support reasoning about behaviours. At the heart of this framework is a *symbolic* treatment of data. A new *finitely branching* semantics of LOTOS was previously given in terms of symbolic state transition systems (STSs) [3].

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 in [4] which 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. A set of rules presented in [3] define how a symbolic transition system may be constructed from a LOTOS process behaviours. 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 behaviour).

In order to describe abstract properties of symbolic transition systems a modal logic called Full LOTOS Logic (FULL) was developed. Earlier work [1, 2] describes the logic in more detail; here we give an overview.

FULL is based on the classic modal logic for concurrent systems known as Hennessy Milner Logic (HML). We use the version as presented in [8]. In addition to the usual operators of $\langle\ \rangle$ and $[\ ]$, which can be thought of as existential and universal quantification over transitions, we add quantifiers over data which may be communicated by those transitions. We refer to these as the quantified modal operators. For example, if a process $P$ satisfies $\langle \exists x\ g \rangle \Phi$ then there is a transition labelled by $gv$ such that $\Phi$ with $v$ substituted for $x$ holds with respect to the new state. The value $v$ may be a specific value in the transition system, or a symbolic one. Further, the formula $[\exists x\ g]\Phi$ expresses that for some value $v$, every transition that can perform a $gv$ action must lead to a state in which $\Phi$ with $v$ substituted for $x$ holds. The other operators combine the universal quantifier over data with the modal operators: $\langle \forall x\ g \rangle$ and $[\forall x\ g]$.

# 3 ERGO

Ergo 5 [9] is an interactive proof tool, designed and implemented at the SVRC. Ergo 5 has many useful features, but the ones most relevant here are:

- A generalised sequent calculus proof style.
- User-definable heuristics (tactics) for automation of proof steps.
- Many (50+) supplied tactics that encode common proof patterns including support for transformational reasoning (window inference).
- An Emacs graphical user interface for proof browsing and construction.
- Support for storing, browsing and replaying proofs.

Ergo 5 is implemented in Qu-Prolog, which is a high-level language designed primarily for rapid prototyping of interactive theorem provers. The object-level terms of Qu-Prolog include quantified terms, substitutions and object variables. Qu-Prolog can also be used for writing user tactics and prover extensions.

# 4 Implementing STS and FULL in Ergo

We give a few examples of the representation of the modal logic FULL using Ergo. A complete description of the representation will be given in the full paper.

In Ergo we define a new theory for STSs and start by declaring the required constants and primitive inference rules. An example of a declaration of a constant is

```
declare models(_,_)
```

which declares `models` as a two place function symbol whose semantics are as follows.

```
models(State,STSForm)
```

is true if and only if the STS formula `STSForm` is true at `State`.

For example,

```
models(s1, dia(g, X, X > 0))
```

is true if and only if there is a `g` transition from state `s1` with data value `X` satisfying `X > 0`.

After the declarations of the required constants we declare the primitive inference rules (called rules in Ergo). The following three rules come from the STS theory. First, the obvious rule for logical "and".

```
rule models_and_iff ===
    models(State, Exp1 and Exp2)
    <=> models(State, Exp1) and models(State, Exp2).
```

Secondly, a two step rule which extracts first the data variable of the transition (which can be dealt with by standard rules for `ex`, existential quantification), and then deals with the existence of a suitable transition.

```
rule models_ex_dia_iff(x) ===
    models(State, ex x:Type dia(Trans, V, Form))
    <=>
    ex x:Type models(State, dia(Trans, V, Form))
       provided x not_free_in Type,
                x not_free_in State,
                x not_free_in Trans.

rule models_dia_iff ===
    models(State, dia(Trans, V, Form))
    <=>
    ex x is_trans(State, Trans, x)
            and formula(x, V) and models(to(x), Form)
       provided x not_free_in State,
                x not_free_in Trans,
                x not_free_in Form,
                x not_free_in V.
```

These rules are a relatively direct translation of the mathematical definition of FULL. For example, the rule for the last operator defined above $\langle \exists x\ g \rangle$, or `ex x:Type dia(Trans, V, Form)` is

$$
\begin{aligned}
t \models \langle \exists x\ g \rangle \Phi \;=\;\; & \text{for some value } v \text{, either for some } t',\ t \xrightarrow{\mathrm{tt}\quad gv} t' \text{ and } t' \models \Phi[v/x] \\
& \text{or for some } t',\ t \xrightarrow{b\quad gz} t' \text{ and } b[v/z] \equiv \mathrm{tt} \text{ and } t'_{[v/z]} \models \Phi[v/x]
\end{aligned}
$$

In fact, the Ergo representation is more compact since the two cases of data value and data variable do not need to be separated. Also, the representation of the operator (with data before gate) allows separate rules to be applied for dealing with the data and the gate, whereas in the FULL semantics we are forced to deal with them together.

All the STS rules are stated as equivalences so that they can be used as part of a rewrite system using transformational reasoning. After carrying out some example proofs by hand, tactics were developed that use the STS rules as well as inherited classic logic rules such as one-point rules and formula simplification rules. These tactics automatically transform `models` formulae into classical formulae involving the datatypes appearing on transitions.

The next step is to incorporate rules for the data type as sepcified by the ACT ONE equational specification part of the LOTOS behaviour, and develop tactics to simplify terms involving data types. Then the entire proof can be carried out automatically. This ability to incorporate theories for both the processses and the data is the advantage of using a theorem prover to prototype our model checker.

Further work includes refinement of tactics and use of Qu-Prolog to implement tactics more efficiently. A problem yet to be addressed is the representation of behaviours with infinite depth which depend on data parameters.

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

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

[6] C. Pecheur. Using LOTOS for specifying the CHORUS distributed operating system kernel. *Computer Communications*, 15(2):93–102, March 1992.

[7] M. Sighireanu and R. Mateescu. Verification of the Link Layer Protocol of the IEEE-1394 Serial Bus (FireWire): an Experiment with E-LOTOS. *Springer International Journal on Software Tools for Technology Transfer (STTT)*, 2(1):68–88, Dec. 1998.

[8] 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.

[9] M. Utting. The Ergo 5 Generic Proof Engine. Technical Report 97-44, Software Verification Research Centre, 1997.