Specifying properties of Basic LOTOS processes using temporal logic

Carron Kirkwood
University of Glasgow
Department of Computing Science, University of Glasgow, Glasgow, U.K.
email: carron@dcs.gla.ac.uk

Abstract

Temporal logic can be used to describe desirable properties of a system in a more abstract, less constructive, manner than when using process algebra alone. This is a well researched area for other process algebras, but not so for LOTOS. This paper is an initial attempt to fill that gap by investigating the use of the modal μ-calculus with Basic LOTOS, laying the groundwork for current work on Full LOTOS, i.e. including data types, and logic.

Keywords

Basic LOTOS specification and verification, partial specification, modal μ-calculus

1 INTRODUCTION

In the process algebra literature a common approach to verification is to take two descriptions of a system and prove them equivalent in some sense. Usually we call one the *specification*; it is written at a high level of abstraction and is concerned with *what* the system does and how it reacts to input. The other we call the *implementation*; it is written at a concrete level, typically utilising smaller, simpler components, and describes the inner workings of the system in detail, i.e. *how* the system operates. These descriptions can be compared at different levels of abstraction using one of many behavioural equivalence or preorder relations defined in the literature. This exercise gives us confidence in the correctness of our system because it shows we can write two different but consistent descriptions of a system.

This form of verification has limitations; in particular, it is inappropriate to show the specification equivalent to the implementation if the specification is *partial*, i.e. describes some aspects of the system, ignoring others. The use of preorder relations rather than equivalence relations cannot always solve this problem. Indeed, the constructive nature of process algebra makes it unsuitable for partial specifications. For this sort of specification a *modal* or *temporal logic*, e.g. (Hennessy & Milner, 1985, Kozen, 1983, Manna & Pnueli, 1992), might be more suitable because logic is non-constructive, allowing us to write more abstract specifications. We may also evaluate logical formulae with respect to a model, expressed using process algebra, considering only relevant aspects of that model. This allows us to retain process algebra for the implementation. Another benefit of logic is that a logical formula may be used to capture the difference between two behaviourally inequivalent processes, see (Hennessy & Milner, 1985).

The use of logic as an additional specification language has been extensively researched for other process algebras; however, relatively little research in this area has been carried out specifically with respect to the ISO standard formal description technique LOTOS (ISO 8807, 1988). We are aware of: (Fantechi *et al.*, 1989) in which a compositional temporal logic semantics for Basic LOTOS is given, (Ernberg *et al.*, 1990) in which HML (Hennessy & Milner, 1985) is used, (De Nicola *et al.*, 1993) and (Fernandez *et al.*, 1992) which describe model checkers for Basic LOTOS, and (Ghribi & Logrippo, 1993) which describes a symbolic model checker for Full LOTOS. The logics used in all of these is rather weak; for example, it cannot express cyclical properties. Rather than pursuing these investigations, we prefer to identify a stronger, more suitable logic, and to then adapt it for use with LOTOS as necessary. This leads us to the modal μ-calculus (Kozen, 1983), a powerful and elegant logic commonly used with process algebras. This logic is defined over labelled transition systems (and is thus independent of particular process algebra syntax). In particular, this means we can easily utilise the literature on modal μ-calculus and CCS (for example) in investigating the use of modal μ-calculus with Basic LOTOS (since Basic LOTOS and CCS are both defined over labelled transition systems). This paper describes our investigations with the modal μ-calculus and Basic LOTOS only; current work (Brinksma *et al.*, 1995) deals with extending the modal μ-calculus for use with Full LOTOS.

We begin by giving a brief tutorial on the modal μ-calculus together with our suggestions for automated and manual proof techniques suitable for use with Basic LOTOS. We assume the reader is familiar with LOTOS or a similar process algebra. This is followed by some examples which illustrate the need for partial specifications and motivate our use of logic. We show how logic can be used to solve the problems presented by these examples. Finally, we discuss the advantages of this approach over the approach discussed at the beginning of this section, the problems of this approach, and further work.

2 A BRIEF OVERVIEW OF THE MODAL μ-CALCULUS

Since the modal μ-calculus has been presented extensively in the literature we give only an overview of the main points of the logic. A good introduction is (Stirling, 1994), which we follow in presenting the logic without negation. In the following syntax Z ranges over propositional variables and $K \subseteq \mathit{Act}$, where *Act* denotes the set of all actions (including **i**).

$$\Phi \ ::= \ Z \mid \Phi_1 \ \wedge \ \Phi_2 \mid \Phi_1 \ \vee \ \Phi_2 \mid [K]\Phi \mid \langle K \rangle \Phi \mid \nu Z.\Phi \mid \mu Z.\Phi$$

In addition to the usual propositional logic operators, we have modalities, $[K]$ and $\langle K \rangle$, expressing transitional change, and fixed point operators, νZ and μZ, expressing iterative properties. Informally, the modal operators are known as "always" and "possible" respectively. $[K]\Phi$ means that Φ holds after *every* performance of any action in K, and $\langle K \rangle\Phi$ means that Φ holds after *some* performance of any action in K. The fixed point operators may be taken to mean infinite and finite recursion respectively.

Abbreviations useful in formulae are: $[-]$ to denote $[\mathit{Act}]$ and $[-K]$ to denote $[\mathit{Act} - K]$. Similarly, $\langle - \rangle$ denotes $\langle \mathit{Act} \rangle$ and $\langle -K \rangle$ denotes $\langle \mathit{Act} - K \rangle$. Although K is a set, we usually omit the braces in the modalities.

We now consider automated and manual proof techniques for checking a modal μ-calculus formula is satisfied by a Basic LOTOS process, written $P \models \Phi$.

2.1 Proof techniques

Using tools

There are no tools which allow us to model check a modal μ-calculus formula against a Basic LOTOS model directly; however, we may use a combination of existing tools to perform different parts of the task.

The basic idea is that we first use a tool which takes a Basic LOTOS expression, P, and produces the corresponding labelled transition system. Since the modal μ-calculus is defined on the labelled transition system of a process rather than depending on the syntax of the particular process algebra involved, we can use model checking tools which are similarly language independent to take that labelled transition system and a modal μ-calculus formula, Φ, returning $P \models \Phi$. This procedure is enabled by the development of the FC2 format for representing labelled transition systems in ASCII which facilitates transfer between tools.

Although we are restricted to tools which accept Basic LOTOS for the first part, we can use any tool which accepts modal μ-calculus and FC2 for the second part. For example, we may use Auto (Madelaine & Vergamini, 1989) to produce the FC2 labelled transition system, followed by the Concurrency Workbench (Cleaveland *et al.*, 1989) for the model checking. An example of such use may be found in (Gnesi *et al.*, 1992).

A disadvantage of these tools (and this method in general) is that they rely on transition graphs and are therefore subject to the usual finiteness restrictions.

Tableau proof technique

Rather than using automated model checking techniques, we may instead prefer to use inference rules to construct a proof of $P \vdash \Phi$. An advantage of this technique is that, unlike the approach described above, we need not construct the whole labelled transition system. For small examples the saving may not be significant; however, it may be very important for larger examples and for infinite transition systems.

Several techniques exist which rely only on the structure of the labelled transition system, and not the structure of the process algebra expression, e.g. the goal directed tableau method of (Stirling & Walker, 1990). This is the proof technique we use in (Kirkwood, 1995).

Note that when we move to Full LOTOS neither of these techniques can be applied. Typically, Full LOTOS transition systems are infinite, therefore current tools will be ineffectual; however, the technique of (Stirling & Walker, 1990) has been extended to be applicable to infinite transition systems (Bradfield & Stirling, 1990), and may therefore be of use. This topic will be further explored in (Brinksma *et al.*, 1995).

3 MOTIVATING EXAMPLES

During our study of verification of properties of LOTOS specifications (Kirkwood, 1994) we encountered a number of problems characterised by the need to be able to express some notion of partial specification. Below we give a description of two of these problems and discuss the problems arising during the verification process; in particular, why the equivalence approach to verification broke down, and how logic can be used to solve these problems.

We present only the minimum of information required to give suitable motivation for this investigation; more details of these studies may be found in (Kirkwood, 1993) and (Thomas, 1994) respectively. Also, lack of space prevents us from presenting the verification proofs; these may be found in (Kirkwood, 1995).

3.1 The abstract communications case study

The first example, originally presented in (Kirkwood, 1993), concerns the specification and verification of a small communications protocol. We have two main descriptions of the system; an abstract specification in terms of three disjoint entities ("protocols") and a more detailed, "implementation" in terms of four communicating entities ("processes"). Figures 1 and 2 contain the Basic LOTOS descriptions. The action names have been abstracted, but it may be helpful to know that actions prefixed *m* are messages, and those prefixed *p* and *n* are positive and negative acknowledgements respectively.

```
process P1 := m1; (n1; P1 [] p1; P1) endproc
process P2 := m3; (n3; P2 [] p3; (P2 [] m6; p6; P2)) endproc
process P3 := P3 [] m4; (   n4; P3
                         [] p4; (   P3 [] m7; p7; P3
                                 [] m5; p5; (P3 [] m7; p7; P3))) endproc
```

Figure 1 The LOTOS Descriptions of the Login Case Study Protocols

```
process A := m1; (n1; A [] p1; A) endproc
process C := m3; (n3; C [] p3; m6; p6; C) endproc
process D := m4; (n4; D [] p4; (    m5; p5; m7; p7; D
                                [] m7; p7; D)) endproc
process B :=
  m1; m3; (   n3; n1; B
           [] p3; m4; (   n4; m6; p6; n1; B
                       [] p4; set; (  timeout; m6; p6; m7; p7; n1; B
                                   [] m5; tcancel; p5; m6; p6;
                                        m7; p7; p1; B))) endproc
```

Figure 2 The LOTOS Descriptions of the Login Case Study Processes

The verification task is to prove the implementation satisfies the specification, which we may interpret as showing them equivalent. Given that the specification is in three parts, a natural approach to the proof is to try to prove each protocol equivalent to the corresponding part of the implementation.

For example, consider the first protocol, `P1`. Our aim is to try to prove

hide `[m3,p3,n3,m4,p4,n4,m5,p5,m6,p6,m7,p7,set,timeout,tcancel]` **in**
`(A |[m1, p1, n1]| B)` *satisfies* `P1` (1)

holds for some interpretation of satisfies. Only A and B are involved because they are the only processes which deal with the P1 part of the system. Note that we must hide events in the left hand side which do not occur in the protocol P1 because of the way the behavioural relations of LOTOS are defined. Unfortunately, this introduces nondeterminism in the implementation which cannot be reconciled with the deterministic specification. We cannot show (1) holds for any interpretation of *satisfies*.

The problem arises because the specification is partial, i.e. the implementation contains lots of information which the specification does not (for example, about the timer, and about how events in different protocols relate). Although we can capture some aspects of partial specification by using preorder relations rather than equivalences, this is not sufficient for this example.

Another way to deal with partial specification is to add the context in which the specification operates. In (Kirkwood, 1993) the specification is strengthened by adding the missing information in order to demonstrate an equivalence between the two descriptions. Although this information can be added in a modular way, this is not a particularly satisfactory solution. We have assumed that failure in the verification indicates a specification error; however, it may indicate that we are trying to prove the wrong thing, or trying to prove it in the wrong way. In this case we can turn to logic, which allows us to give a partial specification of a system and show that the appropriate parts of an implementation satisfy that specification without having to say anything about the rest of the implementation, i.e. we can relate the specification and implementation in a way which automatically ignores events in the implementation not specified by the specification.

How do we specify P1 using logic? The main property is

$$[\mathtt{m1}]\mu Z.([-]Z \lor \langle \mathtt{p1}\rangle \mathrm{tt} \lor \langle \mathtt{n1}\rangle \mathrm{tt})$$

Informally: "After every occurrence of m1 we have a finite sequence of actions and eventually either p1 or n1 occurs."

This formula applies only to one iteration of the system; to say it holds repeatedly, for infinite occurrences of m1, we add a greatest fixed point

$$\textit{P1-logic} \stackrel{def}{=} \nu Y.([\mathtt{m1}]\mu Z.([-]Z \lor \langle \mathtt{p1}\rangle \mathrm{tt} \lor \langle \mathtt{n1}\rangle \mathrm{tt}) \land [-]Y)$$

and the correctness of the system is expressed by Processes $\models$ *P1-logic* where Processes is the parallel composition of A, B, C and D. It is straightforward to show this expression holds using the tableau method of (Stirling & Walker, 1990); see (Kirkwood, 1995).

3.2 The radiation machine

Our second example concerns the specification and verification of a radiation machine, originally presented in (Thomas, 1994). The general purpose of the radiation machine is to supply two sorts of radiation; a low strength electron beam and an xray beam, produced by raising the strength of the beam and scattering it through a shield. The process description models the changing position of the shield and strength of the beam and the actions of the operator, requesting either electron or xray dosage. Events in the systems are hb - high beam, lb - low beam, hs - high shield, `ls` - low shield, `xr` - xray dose, `el` - electron dose, and `fire` - fire! The specification of the machine may be found in (Thomas, 1994).

We wish to express the safety* requirement that a `fire` event can never occur when the shield is low and the beam is high, because this will result in a radiation overdose for the patient. In (Thomas, 1994) the machine (described using process algebra) was shown to be unsafe by demonstrating that it could execute an unsafe trace (also specified using process algebra), e.g. `lb;ls;xr;xr;hb;xr;hb;el;fire`.

While this is a good way to demonstrate a machine is unsafe, it is considerably more difficult to demonstrate a machine is safe because we cannot be sure that we have captured all the unsafe traces in our specification. In fact, the specification of (Thomas, 1994) does not capture all unsafe traces; this error was undetected for several months despite the example being presented to colleagues on many occasions.

Although we could try to specify the good traces of the machine using process algebra, effectively we have already done this; the machine itself is (supposed to be) a specification of the good traces. If we try to specify these again (using the same techniques) we run the risk of repeating design errors made in the original specification. We can minimize that risk by changing our view of the system; one way of doing this is to adopt logic to describe the good traces. A further advantage of logic is that we can say explicitly that a `fire` event can never occur when the shield is low and the beam is high.

The safety requirement of the radiation machine, given that the top-level process of the LOTOS description is called `STARTUP`, is fully expressed by `STARTUP` $\models$ *Good-Machine* where

$$
\begin{array}{lll}
\textit{Good-Machine} & \stackrel{\textit{def}}{=} & [\texttt{lb}][\texttt{ls}]\textit{LowLow} \\
\textit{LowLow} & \stackrel{\textit{def}}{=} & \nu Z.([-]Z \ \wedge \ ([\texttt{hs}]\textit{HighLow} \ \vee \ [\texttt{hb}]\textit{LowHigh})) \\
\textit{HighLow} & \stackrel{\textit{def}}{=} & \nu Z.([-]Z \ \wedge \ ([\texttt{ls}]\textit{LowLow} \ \vee \ [\texttt{hb}]\textit{HighHigh})) \\
\textit{HighHigh} & \stackrel{\textit{def}}{=} & \nu Z.([-]Z \ \wedge \ ([\texttt{ls}]\textit{LowHigh} \ \vee \ [\texttt{lb}]\textit{HighLow})) \\
\textit{LowHigh} & \stackrel{\textit{def}}{=} & \nu Z.([-]Z \ \wedge \ ([\texttt{hs}]\textit{HighHigh} \ \vee \ [\texttt{lb}]\textit{LowLow} \ \vee \ [\texttt{fire}]\mathrm{ff}))
\end{array}
$$

Each property models a state, and the formulae express how the state changes with the occurrence of beam and shield actions. The naming convention of these properties reflects the position of the beam and the shield, e.g. *LowHigh* means the machine is in the state in which the shield is low and the beam is high, i.e. the dangerous state in which a radiation overdose may be delivered. The logical formula $[\texttt{fire}]\mathrm{ff}$ explicitly says the `fire` event is forbidden.

Although we have fully specified the safety property, the specification is rather cumbersome; this is partly because of the way in which the problem is modelled. The logical specification above highlights that we are interested in the state of the beam and the shield rather than the events which lead to that state. A better way of modelling this is to use Full LOTOS with its abstract data types; however, we do not, as yet, have a logic capable of expressing the desired property (which must also include data).

---

*We use Lamport's classification (Lamport, 1977) of safety as nothing bad ever happens.

4 CONCLUSIONS AND FURTHER WORK

We have considered the use of logic as an alternative means of specifying a system. Verification of the system can be achieved by model checking the logical specification of the system with respect to the process algebra implementation. Specifically, we used the modal μ-calculus with Basic LOTOS.

In order to demonstrate the advantages of using logic we reconsidered the examples which had motivated our study of logic, namely the abstract communications case study and the radiation machine of (Kirkwood, 1993) and (Thomas, 1994). Using logic we were able to express satisfaction of a partial specification in the case study; this was not possible using process algebra alone. Similarly, it was straightforward to specify the good behaviour of the radiation machine directly, explicitly forbidding certain actions. Again, this was not possible using process algebra. Logic allows us to capture abstract system properties more effectively and more easily.

A problem of using logic is that sometimes it is difficult for the specifier to be sure that the desired property has been captured, especially when fixed point operators are used. A logical specification tends to be less obvious and easy to understand than a process algebra specification. However, we expect that skill in specifying properties will come with experience, and our experience has been that application of proof techniques such as (Stirling & Walker, 1990) by hand helps to improve understanding of the property specified.

To evaluate whether our approach to verification is a useful one we must answer the following questions: "is this sufficient to show the correctness of the system?" and "are the desired properties captured (effectively)?". While the first question is necesssarily subjective, we have at least shown that the use of logic improves our ability to capture desired properties.

The work described here has been a preliminary investigation; our aim is to adapt the logic for Full LOTOS. This is the topic of (Brinksma *et al.*, 1995) in which we define a late symbolic semantics for LOTOS and temporal logic which allows us to use data in the modalities and to interpret logical formulae with respect to a Full LOTOS specification.

Acknowledgements

Many thanks to Muffy Thomas for her support and patient reading of draft manuscripts and to the referees for their helpful comments. This work has been funded by EPSERC grant "Temporal Aspects of Verification of LOTOS Specifications".

REFERENCES

Bradfield, J., & Stirling, C. (1990). Verifying Temporal Properties of Processes. *Pages 115–125 of:* Baeten, J.C.M., & Klop, J.W. (eds), *Concur'90*. LNCS 458.

Brinksma, E., Kirkwood, C., & Thomas, M. (1995). *A Symbolic Semantics and a Temporal Logic for Full LOTOS Processes*. In preparation.

Cleaveland, R., Parrow, J., & Steffen, B. (1989). The Concurrency Workbench. *Pages 24–37 of:* Sifakis, J. (ed), *Automatic verification methods for finite state systems*. LNCS 407. Springer-Verlag.

De Nicola, R., Fantechi, A., Gnesi, S., & Ristori, G. (1993). An action based framework for verifying logical and behavioural properties of concurrent systems. *Computer Networks and*

ISDN Systems, **25**(7).

Ernberg, P., Fredlund, L., & Jonsson, B. (1990). *Specification and Validation of a Simple Overtaking Protocol using LOTOS*. Tech. rept. T9006. Swedish Institute of Computer Science.

Fantechi, A., Gnesi, S., & Laneve, C. (1989). An Expressive Temporal Logic for LOTOS. *Pages 261–276 of:* Vuong, S. (ed), *Formal Description Techniques, II*. Elsevier Science Publishers B.V. (North-Holland).

Fernandez, J-C., Garavel, H., Mounier, L., Rasse, A., Rodriguez, C., & Sifakis, J. (1992). A Toolbox for the Verification of LOTOS Programs. Clarke, L.A. (ed), *Proceedings of the 14th International Conference on Software Engineering, ICSE 14*.

Ghribi, B., & Logrippo, L. (1993). A Validation Environment for LOTOS. *Pages 93–108 of:* Danthine, A., Leduc, G., & Wolper, P. (eds), *Protocol Specification, Testing, and Verification, XIII*. Elsevier Science Publishers B.V. (North-Holland).

Gnesi, S., Madelaine, E., & Ristori, G. (1992). An Exercise in Protocol Verification. Bolognesi, T., Brinksma, E., & Vissers, C. (eds), *Third LOTOSphere workshop and seminar*. Kluwer Publishing.

Hennessy, M., & Milner, R. (1985). Algebraic Laws for Nondeterminism and Concurrency. *Journal of the Association for Computing Machinery*, **32**(1), 137–161.

ISO 8807. (1988). *Information Processing Systems — Open Systems Interconnection — LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*. International Organisation for Standardisation.

Kirkwood, C. (1993). Automating (Specification $\equiv$ Implementation) using Equational Reasoning and LOTOS. *Pages 544–558 of:* Gaudel, M.-C., & Jouannaud, J.-P. (eds), *Tapsoft '93: Theory and practice of software development*. LNCS 668.

Kirkwood, C. (1994). *Verification of LOTOS Specifications using Term Rewriting Techniques*. Ph.D. thesis, University of Glasgow. Also available as University of Glasgow technical report FM-94-07.

Kirkwood, C. (1995). *Specifying Properties of Basic LOTOS Processes Using Temporal Logic*. An extended version of this paper, to appear as a University of Glasgow Technical Report.

Kozen, D. (1983). Results on the Propositional μ-Calculus. *Theoretical Computer Science*, **27**, 333–354.

Lamport, L. (1977). Proving the Correctness of Multiprocess Programs. *IEEE Transactions on Software Engineering*, **SE-3**(2), 125–143.

Madelaine, E., & Vergamini, D. (1989). Auto: A verification tool for distributed systems using reduction of finite automata networks. *Pages 61–66 of:* Vuong, S. (ed), *Formal Description Techniques, II*. Elsevier Science Publishers B.V. (North-Holland).

Manna, Z., & Pnueli, A. (1992). *The Temporal Logic of Reactive and Concurrent Systems: Volume 1: Specification*. Springer-Verlag.

Stirling, C. (1994). An Introduction to Modal and Temporal Logics for CCS. Moller, F. (ed), *Logics for concurrency: Structure vs automata*. The VIIIth Banff Higher-Order Workshop. Banff, Alberta, Canada.

Stirling, C., & Walker, D. (1990). CCS, liveness, and local model checking in the linear time mu-calculus. *Pages 166–178 of:* Sifakis, J. (ed), *Automatic verification methods for finite state systems*. LNCS 407.

Thomas, M. (1994). The Story of the Therac-25 in LOTOS. *High Integrity Systems Journal*, **1**(1), 3–15.