An Adequate Logic for Full LOTOS

Muffy Calder[1], Savi Maharaj[2], and Carron Shankland[2]

[1] Department of Computing Science,
University of Glasgow, Glasgow G12 8QQ, UK
muffy@dcs.gla.ac.uk
[2] Department of Computing Science and Mathematics,
University of Stirling, Stirling FK9 4LA, UK
{savi,carron}@cs.stir.ac.uk

Abstract. We present a novel result for a logic for symbolic transition systems based on LOTOS processes. The logic is adequate with respect to bisimulation defined on symbolic transition systems.

1 Introduction

LOTOS [12] is a popular process description language that has been in use for well over a decade. With the aid of a number of mature verification tools, it has been successfully applied in a number of domains, including protocols and services [17], distributed systems [23, 16], and as a semantics for higher level languages such as feature descriptions [22] and use-case maps [1].

A particularly distinctive feature of LOTOS is that it 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. Specifically, there is no logic for reasoning about LOTOS processes with unconstrained data. This is a serious drawback since it has long been recognised that a more abstract, temporal logic is essential for describing and checking desired (or undesired) properties of processes [11]. Indeed, experience with case studies [21, 19, 20, 17] has shown the benefits of having data in the process description language and the need to express properties of a system in terms of data, as well as actions. Often the properties refer to data, but symbolically, rather than mentioning particular instances. For example, in the classical comparator one such property is *if process* `Comp` *inputs x and y on channel* `in`*, and x and y are equivalent, then eventually it will output true on channel* `out`.

There has been a good reason to avoid dealing with data properly: in LOTOS, 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. Therefore existing approaches have been restricted to Basic LOTOS [13], or LOTOS with only finite data types [6].

Our aim is to provide a complete approach to data. In order to do so, we base our logic on a new semantics for LOTOS which is finitely branching. This is achieved by having a *symbolic* treatment of data; the underlying state transition systems are therefore called symbolic state transition systems (STSs). Our work is heavily influenced by the symbolic transition systems and logic developed by Hennessy, Lin and Liu for CCS [9, 10]. However, it is significantly different because of the special characteristics of the STSs that result from LOTOS. These derive from the three (related) features that distinguish LOTOS from most other process algebras: *multi-way (broadcast) synchronisation, value negotiation*, and *selection predicates*. Together, these features make the definition of the similar concepts of symbolic transition, bisimulation and logic, non-trivial.

1.1 Related Work

A symbolic approach to message passing CCS is presented in [9] and a related logic in [10]. We adopt the theory of symbolic transition systems here, but the logic is not so useful for our applications. The logic of Hennessy and Liu is based on a *late* semantics, whereas we adopt an *early* semantics because the standard definition of LOTOS [12] is also early. (The late and early classification relates to binding time of variables to values.) In addition, the modal operators defined rely on the classical CCS distinction between ! and ? data offers (i.e. as corresponding to output and input events). In LOTOS the distinction between these two kinds of data offers is not so clear cut. The logic does have the advantage that it is based on symbolic transition systems, and therefore places no artificial restrictions on data values.

μCRL [8] is, like LOTOS, a process algebra with data. In [7] an extension of the modal mu-calculus [14] is presented which includes quantification over data in the modal operators. The semantics of the logic is over labelled transition systems and therefore is subject to the usual problems of state explosion. The focus of their research is on proof rules for the logic rather than adequacy with respect to some equivalence over μCRL processes.

The CADP toolkit [6] provides a number of tools to analyse Full LOTOS specifications, two of which use logic to provide an abstract description of system properties. The tool *evaluator* takes an alternation free modal mu-calculus [14] formula and assesses its truth with respect to a LOTOS expression. The modal operators are extended to allow more flexibility in dealing with actions with data, for example, precise actions or Unix regular expressions can be matched. However, it is not possible to state general predicates on data, such as *input a value which is less than 42 but more than 3*. The action formulae of this logic treat the values as syntactic entities only, whereas we provide the ability to reason about their semantics too.

Also part of the CADP toolkit is *XTL* [15]. This is an executable temporal language which describes computations over transitions. XTL allows a more general treatment of data actions than the evaluator. For example, variables over data can be declared and matched with actions, and operations over data in the LOTOS source can also be used in the logic. Various logics can be encoded in XTL; in fact, we have encoded a *restricted* form of the logic presented in this paper in XTL and carried out some limited examples.

Two important disadvantages of XTL are that the underlying semantics of labelled transition systems is concrete (i.e. fully instantiated) and that CADP must impose finiteness restrictions on the data types of the language to obtain tractability. So, any logic encoded by XTL cannot handle Full LOTOS effectively or accurately.

1.2 Structure of the Paper

The structure of the rest of this paper is as follows. In Section 2 we introduce the idea of a symbolic transition system, describe how this has had to be adapted for LOTOS, and explain the problem of defining substitution and how this is solved. In Section 3 we present the syntax and semantics of a modal logic called FULL. In Section 4 we give an alternative characterisation of the equivalence induced by the logic by showing that it coincides with bisimulation on symbolic transition systems. Finally, we discuss further work and conclude in Section 5.

2 Symbolic Transition Systems

The standard semantics of LOTOS [12] (labelled transition systems) hard codes concrete data values into the transitions. For example, `g!0; P` offers the single transition labelled `g[0]`, while `g?x:Nat; P` offers the transitions labelled by `g[0]`, `g[succ(0)]`, `g[succ(succ(0))]`, ... (Fig. 1). Thus, event offers of more than one value (i.e. `?` offers) correspond to a (possibly infinite) choice over all values of the data type. While this makes the semantics of certain language features easier to describe (particularly multiway synchronisation), it makes reasoning about specifications more difficult since transition systems are typically infinite. Existing tools such as CADP [6] deal with this problem by imposing finiteness restrictions on data types, limiting the natural numbers, for example, to a maximum of 256.

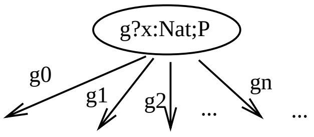

Fig. 1. Standard semantics of `g?x:Nat` event offer

An alternative solution is to restate the semantics of the language in a form which exposes the commonalities of actions and the finitary nature of the process specification. This can be done by basing the semantics on *symbolic transition systems* (STSs). These 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 [9] which gave a symbolic semantics for value passing CCS. In our research [4, 3], we have been adapting this theory for use with LOTOS. There are significant differences between LOTOS and value passing CCS which mean that this adaptation is not straightforward.

One difference is that input events in CCS are always unconstrained and there is no analogue of the *selection predicates* which can be used in LOTOS to restrict the values passed in a `?` event. For example, LOTOS allows events such as `g?x [x > 3]` meaning, *input an x which is bigger than 3.* This means that the transition conditions in the LOTOS semantics need to be able to talk about the data associated with the current transition, whereas in CCS these are concerned only with previous transitions.

Another difference is that in order to implement multi-way synchronisation LOTOS permits synchronisation between any combination of `?` and `!` events, whereas in CCS an input event (`?`) can synchronise only with an output action (`!`). This means that the distinction between `?` and `!` is much less significant in LOTOS than it is in CCS. Essentially, a `!` event is associated with an expression using constants and "known" variables while a `?` event introduces a new variable. We have found it convenient to remove the `!/?` distinction from the syntax of data expressions in STSs. We shall still need to be able to tell when a transition introduces a new variable, but this will be determined by comparing the transition's data expression with the free variables of the source of the transition.

We shall assume that we have a countable set of *variables*, Var, ranged over by x, y, etc., and a (possibly infinite) set of *values*, Val, ranged over by v. We also assume a set of *data expressions*, Exp, which includes Var and Val and is ranged over by E, and a set of *boolean expressions*, BoolExp, ranged over by b. We also assume that we have a set of gates, G, ranged over by g. The set of simple events, SimpleEv, ranged over by a, is defined as $\mathrm{G} \cup \{\mathbf{i}, \delta\}$. (Recall that in LOTOS $\mathbf{i}$ is the internal event and δ is the special event which takes place when a process is exited.) The set of structured events, StructEv contains all gate-expression combinations gE, as well as all combinations δE. Since the two kinds of structured events are handled exactly the same, we shall generally ignore δ in this paper, treating it as if it were a member of G. For simplicity, we do not allow structured events consisting of multiple data expressions; only *singleton* data offers are allowed. It is possible, but tedious, to extend our analysis to the case of multiple data offers.

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. Formally, the definition of STS is as follows:

Definition 1. *(Symbolic Transition Systems) A symbolic transition system consists of:*

- *a set of states, containing a distinguished initial state, T_0, with each state T tagged with a set of free variables, denoted $fv(T)$.*

– *a set of transitions written as* $T \xrightarrow{b \quad \alpha} T'$*,*
where $\alpha \in$ *SimpleEv* $\cup$ *StructEv and* b *is a Boolean expression and* $fv(T') \subseteq fv(T) \cup fv(\alpha)$ *and* $fv(b) \subseteq fv(T) \cup fv(\alpha)$ *and* $\#(fv(\alpha) - fv(T)) \leq 1$

Following convention, we shall often identify an STS with its initial state. For example, the set of free variables of an STS S, $fv(S)$, is defined as the set of free variables of the initial state of S.

A set of rules presented in [4] 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). This paper is concerned with STSs rather than LOTOS processes, though we shall use LOTOS syntax to describe examples.

2.1 Substitution

In the following section we present a logic on symbolic transition systems. Before we can do this, however, we must consider the question of how to define *substitution* on STSs. It is not possible to define a straightforward syntactic substitution on STSs because of the presence of cycles (such as might arise from recursive processes).

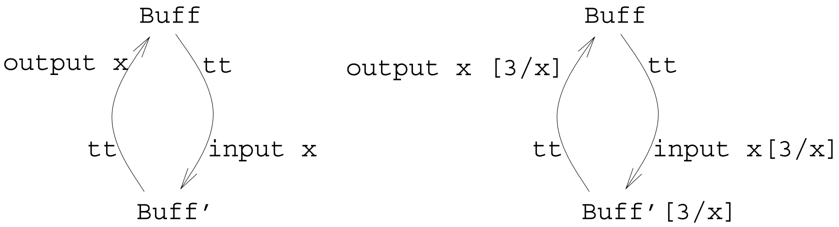

Fig. 2. Failed substitution on `Buff` STS

Consider, for example, the simple buffer `Buff = input?x:Nat; output!x; Buff`. The STS which corresponds to `Buff` is shown in Figure 2. If the first action taken by this process is to input the value `3`, then the `x` at the output gate must also be tied to that value. Since `Buff` is recursive, we expect that the next time round the loop a different value may be input, and therefore a different substitution must be applied. However, if we simply substitute `3` for `x` in the STS, as shown in Figure 2, we fail to capture this possibility.

In [9], this problem is solved by introducing the concept of a "term": a node in a symbolic transition system paired with a substitution. The same solution can be adapted for LOTOS. Formally, a *substitution* is a partial function from Var to Var ∪ Val and a *term* consists of an STS, T, paired with a substitution, σ such that $domain(\sigma) \subseteq fv(T)$. We use t and u to range over terms. For example, since `Buff` is closed, it can be paired only with the empty substitution to form the term $\texttt{Buff}_{[\,]}$. The substitution is applied step by step, when necessary, as explained in the rules for transitions between terms (Figure 2). For example, below are some possible transitions starting from the term $\texttt{Buff}_{[\,]}$. The substitutions capture the fact that the variable x is discarded and then bound afresh upon each pass through the loop, making it possible to process a different value during each pass.

$$\begin{array}{l}
\texttt{Buff}_{[\,]} \xrightarrow{\texttt{tt} \quad \texttt{input}\,z1} \texttt{Buff'}_{[z1/x]} \\
\texttt{Buff'}_{[z1/x]} \xrightarrow{\texttt{tt} \quad \texttt{output}\,z1} \texttt{Buff}_{[\,]} \\
\texttt{Buff}_{[\,]} \xrightarrow{\texttt{tt} \quad \texttt{input}\,z2} \texttt{Buff'}_{[z2/x]} \text{ and so on.}
\end{array}$$

The definition of free variables is extended to terms in the obvious way. Terms, rather than STSs, are used as the basis for defining the logic and bisimulation.

Definition 2. *Transitions on Terms*

$$\begin{array}{l}
T \xrightarrow{b \quad a} T' \; \textit{implies} \; T_\sigma \xrightarrow{b\sigma \quad a} T'_{\sigma'} \\
T \xrightarrow{b \quad gE} T' \; \textit{implies} \; T_\sigma \xrightarrow{b\sigma \qquad\quad gE\sigma} T'_{\sigma'} \\
\qquad\qquad\qquad \textit{where } fv(E) \subseteq fv(T) \\
T \xrightarrow{b \quad gx} T' \; \textit{implies} \; T_\sigma \xrightarrow{b\sigma[z/x] \quad gz} T'_{\sigma'[z/x]} \\
\qquad\qquad\qquad \textit{where } x \notin fv(T) \; \textit{and} \; z \notin fv(T_\sigma)
\end{array}$$

In all cases, $\sigma' = fv(T') \lhd \sigma$, that is, the restriction of σ to include only domain elements in the set $fv(T')$.

3 The Modal Logic FULL

In this section we present the syntax and semantics of a modal logic defined over symbolic transition systems. The logic is called Full LOTOS Logic (FULL) and is inspired by the HML presented in [18] and the data extended logic presented in [10]. The logic and the design considerations driving the choice of operators are described fully in [3]; here we simply give the syntax and semantics without discussion.

FULL is made up of two parts. The first set of formulae, ranged over by Φ, applies to closed terms. The second set, ranged over by Λ, is to be used for terms with a single free variable, as would arise from a LOTOS process with a single parameter. (The extension to multiple free variables is straightforward but tedious and is therefore omitted).

Definition 3. *(Syntax of FULL)*

$$\begin{array}{lcl}
\Phi & ::= & b \;\mid\; \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [a]\Phi \;\mid\; \langle a \rangle \Phi \\
 & & \mid\; \langle \exists x\; g \rangle \Phi \;\mid\; \langle \forall x\; g \rangle \Phi \mid [\exists x\; g]\Phi \;\mid\; [\forall x\; g]\Phi \\
\Lambda & ::= & \exists x.\Phi \;\mid\; \forall x.\Phi
\end{array}$$

Definition 4. *(Semantics of FULL) Given any closed term t, the semantics of $t \models \Phi$ is given by:*

$$
\begin{array}{lcl}
t \models b & = & b \equiv \mathrm{tt} \\
t \models \Phi_1 \wedge \Phi_2 & = & t \models \Phi_1 \textit{ and } t \models \Phi_2 \\
t \models \Phi_1 \vee \Phi_2 & = & t \models \Phi_1 \textit{ or } t \models \Phi_2 \\
t \models \langle a \rangle \Phi & = & \textit{there is a } t' \textit{ s.t. } t \xrightarrow{\mathrm{tt} \quad a} t' \textit{ and } t' \models \Phi \\
t \models [a]\Phi & = & \textit{whenever } t \xrightarrow{\mathrm{tt} \quad a} t' \textit{ then } t' \models \Phi \\
t \models \langle \exists x \; g \rangle \Phi & = & \textit{for some value } v\textit{, either} \\
 & & \quad \textit{for some } t'\textit{, } t \xrightarrow{\mathrm{tt} \quad gv} t' \textit{ and } t' \models \Phi[v/x] \\
 & & \textit{or} \\
 & & \quad \textit{for some } t'\textit{, } t \xrightarrow{b \quad gz} t' \textit{ and } b[v/z] \equiv \mathrm{tt} \\
 & & \quad \textit{and } t'_{[v/z]} \models \Phi[v/x] \\
t \models \langle \forall x \; g \rangle \Phi & = & \textit{for all values } v\textit{, either} \\
 & & \quad \textit{for some } t'\textit{, } t \xrightarrow{\mathrm{tt} \quad gv} t' \textit{ and } t' \models \Phi[v/x] \\
 & & \textit{or} \\
 & & \quad \textit{for some } t'\textit{, } t \xrightarrow{b \quad gz} t' \textit{ and } b[v/z] \equiv \mathrm{tt} \\
 & & \quad \textit{and } t'_{[v/z]} \models \Phi[v/x] \\
t \models [\exists x \; g]\Phi & = & \textit{for some value } v\textit{,} \\
 & & \textit{whenever } t \xrightarrow{\mathrm{tt} \quad gv} t' \textit{ then } t' \models \Phi[v/x] \textit{ and} \\
 & & \textit{whenever } t \xrightarrow{b \quad gz} t' \textit{ and } b[v/z] \equiv \mathrm{tt} \textit{ then } t'_{[v/z]} \models \Phi[v/x] \\
t \models [\forall x \; g]\Phi & = & \textit{for all values } v\textit{,} \\
 & & \textit{whenever } t \xrightarrow{\mathrm{tt} \quad gv} t' \textit{ then } t' \models \Phi[v/x] \textit{ and} \\
 & & \textit{whenever } t \xrightarrow{b \quad gz} t' \textit{ and } b[v/z] \equiv \mathrm{tt} \textit{ then } t'_{[v/z]} \models \Phi[v/x]
\end{array}
$$

Given any term t with one free variable z the semantics of $t \models \Lambda$ is given by:

$$
\begin{array}{lcl}
t \models \exists x.\Phi & = & \textit{there is some value } v \textit{ such that } t_{[v/z]} \models \Phi[v/x] \\
t \models \forall x.\Phi & = & \textit{for all values } v\textit{, } t_{[v/z]} \models \Phi[v/x]
\end{array}
$$

A property of FULL is that for every formula it is possible to construct the negation, *neg*, of that formula. (We assume that negation is available in the underlying language of boolean expressions.) For example, $neg([\forall x \; g]\Phi)$ is $\langle \exists x \; g \rangle neg(\Phi)$.

To each formula in FULL is associated a *depth*, n, which is defined in the obvious inductive way.

4 Bisimulation and Adequacy of FULL

In developing the logic FULL we were motivated by two goals. The first was to develop a logic which allowed properties concerning data to be expressed in a natural way. The second was to ensure that the logic was *adequate* with respect to other notions of equivalence between processes, in the sense that equivalent processes should satisfy the same set of logical formulae. One important relationship between processes is that of *bisimulation.* In this section we show how bisimulation is defined upon terms and prove that FULL is adequate with respect to bisimulation.

We shall assume we have a function $new(t, u)$ which, given two terms t and u, returns a variable which is not among the free variables of either t or u.

Definition 5. *Bisimulation on terms*
Given two closed terms t and u,

1. $t \sim_0 u$
2. *for all $n > 0$, $t \sim_n u$ provided that:*
   (a) **(simple event)**
   whenever $t \xrightarrow{\mathrm{tt}\ \ a} t'$, then for some u', $u \xrightarrow{\mathrm{tt}\ \ a} u'$ and $t' \sim_{n-1} u'$
   (b) **(structured event, no new variable)**
   whenever $t \xrightarrow{\mathrm{tt}\ \ gv} t'$, then either
   for some u', $u \xrightarrow{\mathrm{tt}\ \ gv} u'$ and $t' \sim_{n-1} u'$
   or
   for some u', $u \xrightarrow{b_u\ \ gz} u'$ and $b_u[v/z] \equiv \mathrm{tt}$ and $t' \sim_{n-1} u'_{[v/z]}$, where $z = new(t, u)$.
   (c) **(structured event, new variable)**
   whenever $t \xrightarrow{b_t\ \ gz} t'$, where $z = new(t, u)$, then, for all v s.t. $b_t[v/z] \equiv$ tt, either
   for some u', $u \xrightarrow{\mathrm{tt}\ \ gv} u'$ and $t'_{[v/z]} \sim_{n-1} u'$
   or
   for some u', $u \xrightarrow{b_u\ \ gz} u'$ and $b_u[v/z] \equiv \mathrm{tt}$ and
   $t'_{[v/z]} \sim_{n-1} u'_{[v/z]}$.
   (d), (e), (f) *Symmetrically, the transitions of u must be matched by t.*

Given two terms t and u with free variables $\{x\}$ and $\{y\}$, respectively, $t \sim_n u$ provided that for all values v, $t_{[v/x]} \sim_n u_{[v/y]}$.

The four theorems which follow show that FULL is adequate with respect to bisimulation. Theorems 1 and 2 give the result for closed terms, and are then used to prove the result for terms of one free variable (Theorems 3 and 4).

Theorem 1. *(FULL distinguishes non-bisimilar closed terms) For all n, for all closed terms t and u, if $t \not\sim_n u$ then there is a formula Φ such that $t \models \Phi$ and $u \not\models \Phi$*

Proof *The proof is by induction on n. If $n = 0$ then the result is vacuously true. In the case where $n > 0$, we examine all the ways in which bisimulation can fail and, in each case, construct a formula which is satisfied by t but not by u. We shall illustrate the construction by showing the case where rule (c) of Definition 5 fails. The other cases are simpler and are omitted.*

If rule (c) fails, then there is a transition $t \xrightarrow{b_t\ \ gz} t'$, where $z = new(t, u)$, but there is some value v such that $b_t[v/z] \equiv \mathrm{tt}$ and for all transitions of the form $u \xrightarrow{\mathrm{tt}\ \ gv} u'$, $t'_{[v/z]} \not\sim_{n-1} u'$, and for all transitions of the form $u \xrightarrow{b_u\ \ gz} u'$ where

$b_u[v/z] \equiv \mathrm{tt}$, $t'_{[v/z]} \not\sim_{n-1} u'_{[v/z]}$. Suppose that that there are k of the first kind of transition and m of the second kind, where k and m are natural numbers. Then, by the induction hypothesis, each of the u_i's of the first kind can be distinguished from $t'_{[v/z]}$ by some formula Φ_i, and for each of the u_i's of the second kind, there is a formula Ψ_i which distinguishes $t'_{[v/z]}$ from $u'_{i[v/z]}$. Then, t and u can be distinguished by the formula $[\exists g\ x](x = v) \wedge \bigwedge\{\Phi_1, ...\Phi_k\} \wedge \bigwedge\{\Psi_1, ...\Psi_m\}$.

Theorem 2. *(Bisimilar closed terms satisfy the same formulae) For all n, for all closed terms t and u, if $t \sim_n u$ then, for all formulae Φ such that $depth(\Phi) \leq n$, $t \models \Phi$ if and only if $u \models \Phi$.*

Proof *The proof is by induction on n. If $n = 0$, then the formula Φ must be of depth 0, and must therefore be a simple boolean b. By the semantics of FULL, it is clear that for any t and u, $t \models b$ iff $u \models b$.*

In the case where $n > 0$, we take any t and u and assume that $t \sim_n u$. We must show that for all formulae Φ such that $depth(\Phi) \leq n$, $t \models \Phi$ if and only if $u \models \Phi$. This is done by induction on the structure of Φ. There are 9 cases to consider. We illustrate the arguments used by showing one of the most complex cases:

Consider the case where Φ is of the form $[\forall x\ g]\Phi'$. Suppose that $t \models \Phi$. Then, by the semantics of FULL, for all values v, whenever there is a t' such that $t \xrightarrow{\mathrm{tt}\quad gv} t'$ then $t' \models \Phi'[v/x]$, and whenever there is a t' such that $t \xrightarrow{b_t\quad gz} t'$ (for some new variable z) and $b_t[v/z] \equiv \mathrm{tt}$ then $t'_{[v/z]} \models \Phi'[v/x]$. We must show that $u \models \Phi$. Take any value v. We must consider all u transitions on v. These can be of two kinds:

Case (1) *Suppose there is a transition of the form $u \xrightarrow{\mathrm{tt}\quad gv} u'$. By bisimilarity, this is matched by a t transition. There are two possibilities.*

The matching transition may be of the form $t \xrightarrow{\mathrm{tt}\quad gv} t'$, where $t' \sim_{n-1} u'$. Then, we know that $t' \models \Phi'[v/x]$ and, by the main induction hypothesis, we get that $u' \models \Phi'[v/x]$.

The matching transition may be of the form $t \xrightarrow{b_t\quad gz} t'$, where $z = new(t, u)$ and $b_t[v/z] \equiv \mathrm{tt}$ and $t'_{[v/z]} \sim_{n-1} u'$. Then, we know that $t'_{[v/z]} \models \Phi'[v/x]$ and, by the main induction hypothesis, we get that $u' \models \Phi'[v/x]$.

Case (2) *Suppose there is a transition of the form $u \xrightarrow{b_u\quad gz} u'$, (for some fresh z) and $b_u[v/z] \equiv \mathrm{tt}$. We wish to show that $u'_{[v/z]} \models \Phi'[v/x]$. Now, since z is fresh, we can replace z by z' where $z' = new(t, u)$. In other words, we are looking instead at the transition $u \xrightarrow{b_u[z'/z]\quad gz'} u'_{[z'/z]}$. For this transition, we get that $b_u[v/z'] \equiv \mathrm{tt}$. And, we need to show that $u'_{[v/z']} \models \Phi'[v/x]$.*

By bisimilarity, this transition is matched by a t transition. There are two possibilities.

The matching transition may be of the form $t \xrightarrow{\mathrm{tt}\quad gv} t'$, where $t' \sim_{n-1} u'_{[v/z']}$. Then, we know that $t' \models \Phi'[v/x]$ and, by the main induction hypothesis, we get that $u'_{[v/z']} \models \Phi'[v/x]$.

The matching transition may be of the form $t \xrightarrow{b_t \quad gz'} t'$*, where* $b_t[v/z'] \equiv \mathrm{tt}$ *and* $t'_{[v/z']} \sim_{n-1} u'_{[v/z']}$*. Then, we know that* $t'_{[v/z']} \models \Phi'[v/x]$ *and, by the main induction hypothesis, we get that* $u'_{[v/z']} \models \Phi'[v/x]$*.*

Theorem 3. *(FULL distinguishes non-bisimilar open terms) For all* n*, for all terms* t *and* u *with one free variable, if* $t \not\sim_n u$ *then there is a formula* Λ *such that* $t \models \Lambda$ *and* $u \not\models \Lambda$*.*

Proof *Suppose that the free variables of* t *and* u *are* z_1 *and* z_2*, respectively. Since* $t \not\sim_n u$*, then there is some value* v *such that* $t_{[v/z_1]} \not\sim_n U_{[v/z_2]}$*. By Theorem 1 there is then a formula* Φ *such that* $t_{[v/z_1]} \models \Phi$ *but* $u_{[v/z_2]} \not\models \Phi$*. We construct the formula* $\Lambda = \forall\, x.(x \neq v) \vee \Phi$*. Then,* $t \models \Lambda$ *but* $u \not\models \Lambda$*.*

Theorem 4. *(Bisimilar open terms satisfy the same formulae) For all* n*, for all terms* t *and* u *with one free variable, if* $t \sim_n u$ *then, for all* Λ *such that* $depth(\Lambda) \leq n$*,* $t \models \Lambda$ *if and only if* $u \models \Lambda$*.*

Proof *This is a straightforward consequence of Theorem 2.*

5 Further Work

The results presented in this paper provide a foundation upon which to build a system for verifying properties of specifications in Full LOTOS. In this section we discuss the further work, both theoretical and practical, which needs to be done to realise this goal.

Extensions of the Logic The logic we have developed is relatively sparse, and there are several useful ways in which it could be extended and made more expressive. However, care must be taken to ensure that this is not done at the expense of adequacy. Two important features which we intend to focus upon are ways of handling multi-sorted data, and fixpoint operators to handle recursion.

User-defined algebraic datatypes are an important and heavily used feature of LOTOS so it is essential to extend FULL to deal in some way with multiple data types. One obvious way of doing this is to encode types as predicates over values. The details of this need to be worked out and alternative solutions explored.

Recursion is another heavily-used feature of LOTOS, and the usefulness of FULL would be significantly enhanced by the addition of fixpoint operators for reasoning about recursive or infinitary behaviour. This is a topic which has been much studied in the theory of concurrency and we hope to be able to adapt existing solutions to the needs of LOTOS.

Further Theoretical Analysis Some areas of the theory underlying symbolic transition systems for LOTOS are as yet incomplete. For example, the relationship between our symbolic semantics and the standard semantics of LOTOS has not yet been fully analyzed. We conjecture that the two semantics coincide for closed terms, in the sense that bisimilar terms in the symbolic semantics correspond to bisimilar processes in the standard semantics. The details of this remain to be checked.

Another interesting area of study is *symbolic bisimulation.* The bisimulation presented in this paper is of limited practical use because it requires a possibly infinite number of values to be examined (cf rules *2(c)* and *2(f)* of Definition 5). This problem can be solved by turning to symbolic bisimulation, as introduced in [9]. Symbolic bisimulation solves the problem of infinite values by dividing the value space that must be examined into a finite number of partitions described by boolean expressions. We have defined symbolic bisimulation for LOTOS [4] and are working on its theoretical underpinnings and the development of a bisimulation-checking tool to support it.

Algorithms and Tools The eventual goal of this research is the development of tools to support reasoning about specifications in Full LOTOS. Work is in progress on the development of algorithms for reasoning within FULL. In tandem with this, there is also work on the implementation of tools to support reasoning in FULL. At the present time, a restricted version of the logic has been implemented in CADP. The logic is also being implemented in the Ergo theorem prover [2] and in the Maude system [5].

Acknowledgement. The authors would like to thank the Engineering and Physical Sciences Research Council and the Nuffield Foundation Newly Appointed Lecturer scheme for supporting this research.

References

1. D. Amyot, L. Charfi *et al.* Feature Description and Feature Interaction Analysis with Use Case Maps and LOTOS. In M. Calder and E. Magill, editors, *Feature Interactions in Telecommunications and Software Systems VI.* IOS Press, May 2000.
2. H. Becht, A. Bloesch *et al.* Ergo 4.1 Reference Manual. Technical Report 96-31, Software Verification Research Centre, University of Queensland, Australia, November 1996
3. M. Calder, S. Maharaj, and C. Shankland. A Modal Logic for Early Symbolic Transition Systems. *The Computer Journal*, 2001. To appear.
4. M. Calder and C. Shankland. A Symbolic Semantics and Bisimulation for Full LOTOS. To appear as a University of Stirling Technical Report, 2000.
5. M. Clavel, F. Duran *et al.* Maude: Specification and Programming in Rewriting Logic. Maude System documentation. Computer Science Laboratory, SRI, Menlo Park, California, March 1999.
6. J-C. Fernandez, H. Garavel *et al.* CADP (CAESAR/ALDEBARAN Development Package): A Protocol Validation and Verification Toolbox. In R. Alur and T.A. Henzinger, editors, *Proceedings of CAV'96*, number 1102 in Lecture Notes in Computer Science, pages 437–440. Springer-Verlag, 1996.
7. J.F. Groote and R. Mateescu. Verification of Temporal Properties of Processes in a Setting with Data. In *Proceedings of the 7th International Conference on Algebraic*

Methodology and Software Technology AMAST'98, Amazonia, Brazil, volume 1548 of *Lecture Notes in Computer Science*, pages 74–90, 1999.

8. J.F. Groote and A. Ponse. The Syntax and Semantics of μ-CRL. In *Proceedings of Algebra of Communicating Processes, Utrecht 1994*, Workshops in Computing. Springer-Verlag, 1995.
9. M. Hennessy and H. Lin. Symbolic Bisimulations. *Theoretical Computer Science*, 138:353–389, 1995.
10. M. Hennessy and X. Liu. A Modal Logic for Message Passing Processes. *Acta Informatica*, 32:375–393, 1995.
11. M. Hennessy and R. Milner. Algebraic Laws for Nondeterminism and Concurrency. *Journal of the Association for Computing Machinery*, 32(1):137–161, 1985.
12. International Organisation for Standardisation. *Information Processing Systems — Open Systems Interconnection — LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, 1988.
13. C. Kirkwood. Specifying Properties of Basic LOTOS Processes Using Temporal Logic. In G. v Bochmann, R. Dssouli, and O. Rafiq, editors, *Formal Description Techniques, VIII*, IFIP. Chapman Hall, April 1996.
14. D. Kozen. Results on the Propositional μ-Calculus. *Theoretical Computer Science*, 27:333–354, 1983.
15. R. Mateescu and H. Garavel. XTL: A Meta-Language and Tool for Temporal Logic Model-Checking. In *Proceedings of the International Workshop on Software Tools for Technology Transfer STTT'98 (Aalborg, Denmark)*, 1998.
16. C. Pecheur. Using LOTOS for specifying the CHORUS distributed operating system kernel. *Computer Communications*, 15(2):93–102, March 1992.
17. 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.
18. 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.
19. M. Thomas. The Story of the Therac-25 in LOTOS. *High Integrity Systems Journal*, 1(1):3–15, 1994.
20. M. Thomas. Modelling and Analysing User Views of Telecommunications Services. In *Feature Interactions in Telecommunications Systems*, pages 168–183. IOS Press, 1997.
21. M. Thomas and B. Ormsby. On the Design of Side-Stick Controllers in Fly-by-Wire Aircraft. *A.C.M. Applied Computing Review*, 2(1):15–20, Spring 1994.
22. Kenneth J. Turner. An architectural description of intelligent network features and their interactions. *Computer Networks*, 30(15):1389–1419, September 1998.
23. A. Vogel. On ODP's architectural semantics using LOTOS. In J. de Meer, B. Mahr, and O. Spaniol, editors, *Proc. Int. Conf. on Open Distributed Processing*, pages 340–345, September 1993.