A Modal Logic for Full LOTOS based on Symbolic Transition Systems

M. Calder[1], S. Maharaj[2] and C. Shankland[2]

[1]*Department of Computing Science, University of Glasgow, G12 8QQ*
[2]*Department of Computing Science and Mathematics, University of Stirling, FK9 4LA*
Email: ces@cs.stir.ac.uk

Symbolic transition systems separate data from process behaviour by allowing the data to be *uninstantiated*. Designing a HML-like modal logic for these transition systems is interesting because of the subtle interplay between the quantifiers for the data and the modal operators (quantifiers on transitions). This paper presents the syntax and semantics of such a logic and discusses the design issues involved in its construction. The logic has been shown to be adequate with respect to strong early bisimulation over symbolic transition systems derived from Full LOTOS. We define what is meant by adequacy and discuss how we can reason about it with the aid of a mechanised theorem prover.

Keywords: Modal logic, symbolic transition systems, LOTOS, bisimulation, adequacy

Received March 31, 2000; revised December 20, 2000; accepted August 23, 2000

1. INTRODUCTION

The ISO formal description technique LOTOS [1] has been used over the last twenty years for a number of applications, including OSI protocols, telecommunications systems and even children's games. A particularly useful feature of the language is that it allows description both of process flow of control and data passed between processes. Unfortunately, theory allowing analysis of such descriptions has been slower to develop; most work has concentrated on a restricted version of LOTOS without data. Our ultimate goal is to design a framework for reasoning about Full LOTOS (that is, processes plus data). In this paper we present a modal logic for Full LOTOS, similar in spirit to that defined by Hennessy and Liu [2] for value passing CCS.

The main problem to be overcome when reasoning about Full LOTOS specifications is that the standard semantics [1] instantiates all data, introducing the possibility of infinite branching in the transition systems. Infinite systems are difficult to reason about and present problems in the development of tools such as model-checkers which work by exploring all possible states. One method of dealing with infinite branching is to impose strict limitations on all data types; in effect, requiring them to be finite. This is the approach adopted in the toolkit CADP [4], which provides a modal μ-calculus for reasoning about Full LOTOS specifications. This logic is powerful and expressive but does not truly address the issue of infinite branching, since types are limited by underlying semantics. For example, the type of natural numbers in CADP has only 256 values.

Hennessy and Liu [2] have defined a modal logic for value-passing CCS that avoids the problem of infinite branching by using the *late* semantics of value-passing CCS. The late/early distinction relates to the binding time of variables to values: in the early semantics an input action $g?x$ results in x being bound to a specific value immediately, whereas in the late semantics x is bound to an *abstraction* which can be later instantiated to some specific value. Therefore in the early semantics, the action $g?x$ gives rise to a multitude of transitions (one for each possible value of x) whereas in the late semantics there is only a single transition, and therefore the transition system is finitely branching. Unfortunately, this approach is not suitable for LOTOS because operators such as multi-way synchronisation naturally give rise to an early semantics. To retain early semantics but recover finite branching we therefore turn to Symbolic Transition Systems.

Symbolic Transition Systems (STSs) [5] are transition systems that separate data from process behaviour by allowing the data to be *symbolic*, that is, uninstantiated. STSs are finitely branching because an infinite set of concrete transitions on specific values is replaced by a finite set of symbolic transitions on symbolic data items. In an STS, each transition which involves data is labelled by a gate, a symbolic data item, and a condition expressing a constraint over the data. For example, consider the process given by $g?x : S[x > 3]; P$. This process inputs a value of type S on gate g, provided that the value is greater than 3. The corresponding STS has a transition labelled with the gate g, the variable, or data parameter, x, (symbolically representing the input) and the condition $x > 3$.

Previous work [6] presents a set of rules for generating STSs from Full LOTOS processes. In this paper, we build upon our framework for reasoning about Full LOTOS specifications by developing a logic for describing abstract properties of these STSs.

The paper is organised as follows. In Section 2 we remind the reader of the syntax of LOTOS and present the formal definition of symbolic transition systems. Section 3 presents the logic. We begin with a discussion of some of the issues and choices to be made when designing a modal logic for such an STS, illustrated by various examples in Section 3.1. The denotational semantics for the logic, in terms of satisfaction by STSs generated from Full LOTOS, is given in Section 3.2. A desirable property is that the logic should neither identify (distinguish) more processes than those identified (distinguished) by strong early bisimulation on STSs, that is, it should be adequate with respect to this bisimulation [7]. This is formally stated in Section 4, although the importance of adequacy is stressed throughout. In Section 5 we discuss how we have been using the theorem prover PVS [8] to help develop this work. Finally, we summarize and mention further work in Section 6.

2. PRELIMINARIES

2.1. LOTOS

The reader is assumed to have some familiarity with process algebras, therefore we give only a brief overview here. Many authors have produced tutorials for LOTOS, for example, Logrippo *et al* [9].

LOTOS is a verbose language, with many operators. In this paper we use action prefix (denoted `a; P`) and choice (denoted `a; P [] b; Q`). Actions may be simple events (denoted *SimpleEv*) or structured events (denoted *StructEv*). Simple events come from some set G of actions, plus the distinguished events `i` (like τ in CCS) and δ. Structured events are of the form gE where $g \in G \cup \{\delta\}$ and E is an expression denoting a data offer. For example, `send!4` denotes the offer of the value `4` at gate `send`, while `rec?x:Nat` denotes the offer of all values of `Nat` at gate `rec`. Variables are bound by `?`.

Data offers can be thought of as input (`?`) or output (`!`) events, but it is important to realise that since LOTOS has multiway synchronisation data offers can synchronise in any combination (not just as input/output pairs as in CCS). A better model is to think of a data offer as offering a set of values. For `!` actions this set is always a singleton, whereas for `?` actions the set may range from empty to infinite. Then, multiway synchonisation may be seen as the intersection of the sets of values offered by all the events involved in the synchronisation.

LOTOS also has guarded events. Guards can precede actions (`[x > 0] -> a; P`, where `x` is free) or be incorporated in actions as selection predicates (`rec?x:Nat[x>5]`, where `x` is not free). These also restrict the set of values offered by an event.

2.2. Symbolic Transition Systems

STSs are essentially labelled transition systems in which states and transitions may be open, and transitions are labelled with a Boolean condition in addition to the usual gate name and data value. Although LOTOS has multiple data offers for each action, we assume for simplicity only one data offer.

The restrictions on free variables in our STSs are different from those of Hennessy and Lin [5]. This is a consequence of the way in which our STSs are derived from the syntax of LOTOS. In particular, in the STSs of Hennessy and Lin [5] the Boolean condition of a transition can use only the free variables of the source of that transition, whereas we allow this Boolean to also use the variable (if any) bound by the transition. This reflects selection predicates directly, and ties us to an *early* semantics of LOTOS. This is acceptable since multiway synchronisation implies an early semantics.

DEFINITION 2.1. Symbolic Transition Systems
A symbolic transition system consists of:

- A (nonempty) set of states. Each state T is associated with a set of free variables, denoted $fv(T)$.
- A distinguished initial state, T_0.
- A set of transitions written as $T \xrightarrow{b \quad \alpha} T'$ such that $fv(T') \subseteq fv(T) \cup fv(\alpha)$ and $fv(b) \subseteq fv(T) \cup fv(\alpha)$ and $\#(fv(\alpha) - fv(T)) \leq 1$. b is a Boolean expression and $\alpha \in$ SimpleEv $\cup$ StructEv.

We say that a state is *closed* if its set of free variables is empty. An STS is *closed* if its initial state is closed.

Another consequence of multiway synchronisation is that the distinction between ! and ? events is less important than in the STSs defined by Hennessy and Lin [5]. Hence the transitions of our STSs do not have special notation for each kind of action (they are both just data offers). However, it is always possible to tell if a variable is being bound by examining the free variables of the associated states. So, for example, whenever α introduces a new variable, we know that it is a new binding because $fv(\alpha) \not\subseteq fv(T)$.

These differences between LOTOS and value-passing CCS are significant, and make it non-trivial to adapt the work of Hennessy and Lin [5] to the LOTOS setting.

2.3. Operational Semantics

Before we can present the logic over symbolic transition systems 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 arise from recursive processes).

This problem is solved by introducing the concept of a "term": a node in a symbolic transition system paired with a substitution. Formally, a *substitution* is a partial function from Var to Var ∪ Val and a *term*, T_σ, consists of a state in an STS, T, paired with a substitution, σ, such that $domain(\sigma) \subseteq fv(T)$. The substitution is applied step by step, when necessary, as explained in the rules for transitions between terms (Definition 2.2). Note that the substitution is allowed either to rename variables or to provide an evaluation.

We use t and u to range over terms. 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. The notation $t_{[e/x]}$ is used to mean the term t with the mapping $x \mapsto e$ added to its substitution.

DEFINITION 2.2. Transitions on Terms

$$T \xrightarrow{b\quad a} T' \quad \text{implies} \quad T_\sigma \xrightarrow{b\sigma\quad a} T'_{\sigma'}$$

$$T \xrightarrow{b\quad gE} T' \quad \text{implies} \quad T_\sigma \xrightarrow{b\sigma\qquad gE\sigma} T'_{\sigma'} \quad \text{where } fv(E) \subseteq fv(T)$$

$$T \xrightarrow{b\quad gx} T' \quad \text{implies} \quad T_\sigma \xrightarrow{b\sigma[z/x]\quad gz} T'_{\sigma'[z/x]} \quad \text{where } x \notin fv(T) \text{ and } z \notin fv(T_\sigma)$$

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

To improve readability, we shall use a somewhat informal notation to express quantification over transitions, sometimes omitting some of the items over which we are quantifying. For example, we write "for some t', $t \xrightarrow{b\quad gE} t'$" to mean "for some t', b, E, $t \xrightarrow{b\quad gE} t'$" and "whenever $t \xrightarrow{b\quad gE} t'$" to mean "for all t', b, E such that $t \xrightarrow{b\quad gE} t'$".

3. A MODAL LOGIC FOR LOTOS

Our aim is to design a logic which is expressive enough to describe desirable (and undesirable) properties of a system, as well as to capture the notion of strong early bisimulation over STSs [6]. We start with the basic concepts of HML [3] and consider how to add data.

In addition to the usual constants tt, ff and binary operators ∧ and ∨, HML has two modal operators: the diamond $\langle g \rangle$, corresponding to existential quantification over transitions, and the box $[g]$, corresponding to universal quantification over transitions. We add existential and universal quantification over data values to these, so that modal operators can express quantifications over both transitions and data.

Informally, our understanding of these operators is as follows, using variable y to stand for data and g for a gate name.

$\langle \exists y\ g \rangle$	One value, one g transition.
$\langle \forall y\ g \rangle$	Enough g transitions to cover all values.
$[\exists y\ g]$	All g transitions for a particular value.
$[\forall y\ g]$	All values, all g transitions.

As noted, our STSs do not make a syntactic distinction between ! and ? events; therefore neither does our logic. This contrasts with, for example, the logic of Hennessy and Liu [2] in which there is a one to one correspondence between a quantifier in the logic and a matching transition. Here, a quantifier may need to be matched by several transitions. Recall that, due to multiway synchronisation and selection predicates, each data offer can be seen as a set of values. In particular, individual transitions may be associated with a strict subset of values for the type. Therefore when matching a universal quantifier more than one transition may be required in order to provide the complete set of values for the type, and these transitions may have been generated from either ! or ? events.

To illustrate this point, and to show informally the semantics of each modal operator given by the combinations of [ ], ⟨ ⟩, ∃ and ∀, we give several examples in the next section.

3.1. Examples

Consider the process P (the STS is given in Figure 1). We assume Num ranges from 1 to 10.

```
process P [g,h,k]: exit :=
   g?x:Num [x < 5]; h; exit
 □ g!4; k; exit
 □ g?x:Num [x = 5]; (h; exit □ k; exit)
 □ g!5; h; exit
 □ g?x:Num [x > 5]; h; exit
 □ g!10; k; exit
endproc
```

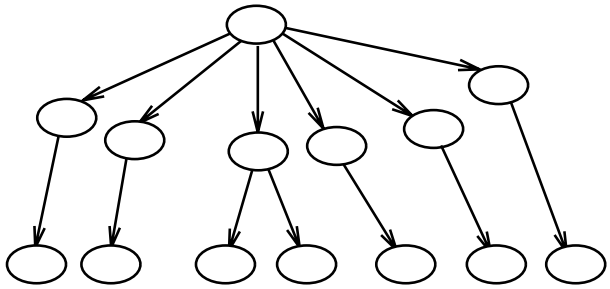

FIGURE 1. process P

To illustrate the full capabilities of the logic, we have chosen a process which has several overlapping conditions, that is, there are non-deterministic choices. In the diagrams below the highlighted branches are the ones used in the evaluation of the modal formulae.

To start with a simple example, consider the property that a process can possibly do an action, with data, and it might depend on a particular Boolean condition being satisfied. For example, *P can perform a g action with some data y which is equal to 4*. This is phrased as $\mathtt{P} \models \langle \exists y\ g \rangle (y = 4)$ and a transition showing that P satisifes the property is given in Figure 2. Only a single path in P is required to satisfy the property.

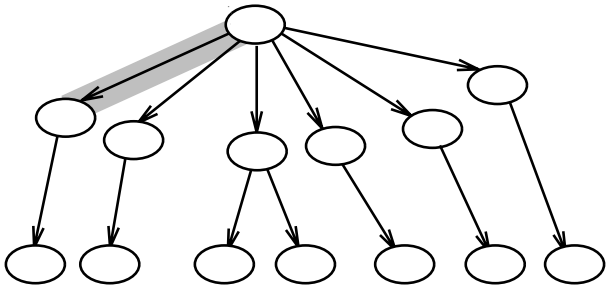

FIGURE 2.

Now consider composing together operators, expressing a chain of actions, and also the combination of $\forall$ and $\langle\,\rangle$. This combination is slightly counter-intuitive, since the usual understanding of $\langle\,\rangle$ is that only one g transition is required to satisfy it, while for $\forall$ we require all values. For example, *for all values y, P can do a g action, and then an h action.* This is phrased as $\mathtt{P} \models \langle \forall y\ g\rangle\langle h\rangle \mathrm{tt}$ and the transitions showing that P satisfies this property are given in Figure 3. Note that all values of the type of y must be considered when evaluating this property, but that this is not the same as all transitions labelled by a g action. If there was one transition $P \xrightarrow{tt \quad gx} P'$, where x has type Num, then this would be sufficient to satisfy the first part of the property. There is no such transition, therefore the set Num must be partitioned and one transition found for each member of the partition. Thus, several paths in P are required to satisfy the property, but only enough to provide all elements of the set Num.

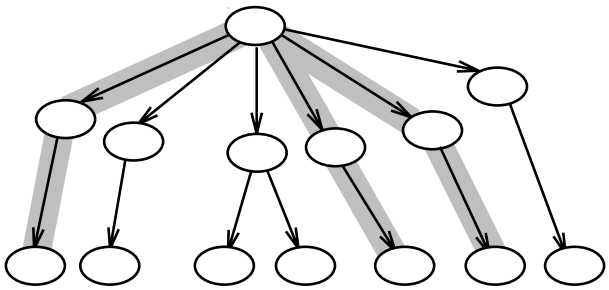

FIGURE 3.

A similar property $\langle \forall y\ g\rangle\langle k\rangle \mathrm{tt}$ fails to hold for P. Figure 4 shows the transitions where a g action followed by a k action is possible, but these only yield the set $\{4, 5, 10\}$, which does not partition Num. Therefore, P does not have this property. On the other hand, Figure 4 demonstrates that there are some paths where it is possible to do a k action. Any of these paths shows $\mathtt{P} \models \langle \exists y\ g\rangle\langle k\rangle \mathrm{tt}$ holds.

The combination of $[\,]$ and $\exists$ is also useful, since we choose a single value, but pursue all paths with that value and the given gate name. For example, the property $\mathtt{P} \models [\exists y\ g]\langle h\rangle \mathrm{tt}$, that is, *for some value y, no matter which g action is chosen it is possible to do an h action subsequently.* The example in Figure 5 shows

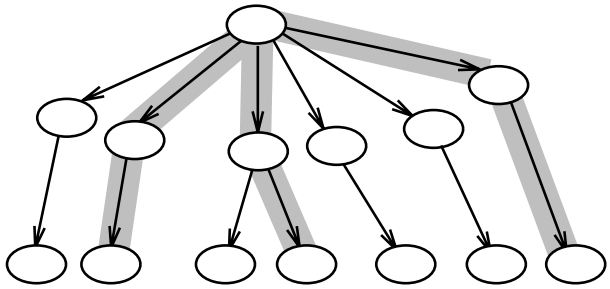

FIGURE 4.

this to be the case for the value 5. This property would also be true for any value other than 4 and 10. As with $\langle \forall y\ g\rangle$ several paths are selected, but this time all paths for a single data value are required, rather than a combination of paths covering all values.

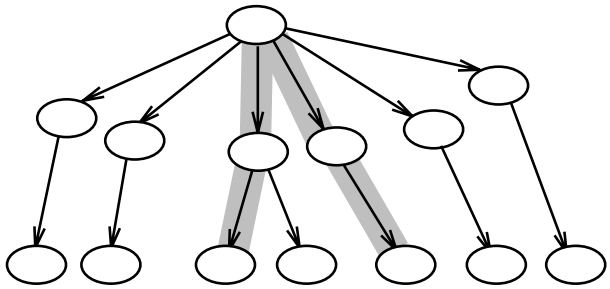

FIGURE 5.

In contrast, if the $\exists$ is replaced by a $\forall$, giving $[\forall y\ g]\langle h\rangle \mathrm{tt}$, then the property no longer holds for P, as illustrated in Figure 6. In this case all g transitions, for all values, must be considered, but it is not always possible to do an h action after a g action (specifically, after the actions $g4$ and $g10$). If the formula is extended to $[\forall y\ g](\langle h\rangle \mathrm{tt} \vee \langle k\rangle \mathrm{tt})$ then it does hold for process P.

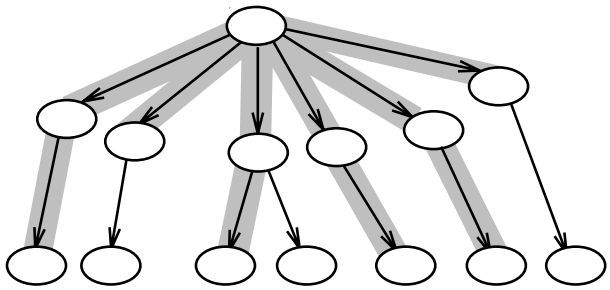

FIGURE 6.

As mentioned above, a motivating design requirement was adequacy of the logic with respect to bisimulation on STSs. Consider the process Q as follows (the corresponding STS is given in Figure 7).

```
process Q [g,h,k]: exit :=
   g?x:Num; h; exit
 [] g?5:Num; (h; exit [] k; exit)
 [] g?x:Num [x=4 or x=10]; k; exit
```

```
endproc
```

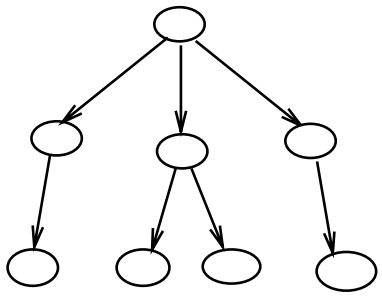

FIGURE 7. process `Q`

Informally, we can see that processes `P` and `Q` exhibit the same behaviour, that is, we expect `P` to be bisimilar to `Q`, and that they satisfy (or fail to satisfy) the same properties. Certainly this is true for the properties described above.

Finally, it is illuminating to consider an example of two processes that are *not* bisimilar (the STSs are given in Figure 8).

```
process R[g]:exit := g!3; exit endproc
process S[g]:exit := g?y:Nat[odd(y)]; exit endproc
```

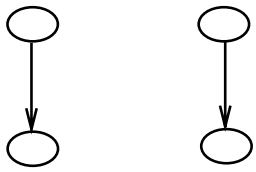

FIGURE 8. processes `R` and `S`

Process `S` clearly has actions which are not available to process `R`. A formula distinguishing these two processes is $\langle \exists x\ g \rangle (x \neq 3)$ (which `S` satisfies but `R` does not).

The examples above demonstrate that every combination of operator is potentially useful since each corresponds to some informal idea about exploring paths through the STS. Moreover, the operators seem to capture an established notion of equivalence between processes. Therefore all the combinations are included in the logic FULL (FUll LOTOS Logic). The formal syntax and semantics of this logic are presented next.

3.2. Syntax and Semantics

The syntax of FULL is based on a variant of HML, as presented by Stirling [10], with data and quantifiers added. There are two classes of formulae. The first class, ranged over by Φ, applies to closed terms. The second class, ranged over by Λ, applies to 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 3.1. Syntax of FULL

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

Here b is a Boolean expression, $a \in G \cup \{\mathrm{i}, \delta\}$, $g \in G \cup \{\delta\}$ and x denotes a variable name. We have deliberately left b unspecified, as it depends on the language of data as described in the LOTOS specification from which the STS is generated. We assume that it at least includes the usual Boolean constants.

We now give the formal semantics of the logic. First we define $t \models \Phi$, denoting that a closed term t satisfies a closed modal formula Φ (Definition 3.2). Note that although some transitions may introduce new variables, the states and formulae remain closed because of the substitutions applied. This means that when we consider which transitions to match there are only two cases: either the expression has the closing substitution applied, yielding a value, or a new variable is bound.

DEFINITION 3.2. Semantics of FULL: Closed Terms

Given any closed term t, the semantics of $t \models \Phi$ is given by:

$t \models b$ = $b \equiv \mathrm{tt}$

$t \models \Phi_1 \wedge \Phi_2$ = $t \models \Phi_1$ and $t \models \Phi_2$

$t \models \Phi_1 \vee \Phi_2$ = $t \models \Phi_1$ or $t \models \Phi_2$

$t \models \langle a \rangle \Phi$ = for some t', $t \xrightarrow{\mathrm{tt}\ \ a} t'$ and $t' \models \Phi$

$t \models [a]\Phi$ = whenever $t \xrightarrow{\mathrm{tt}\ \ a} t'$ then $t' \models \Phi$

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

$t \models \langle \forall x\ g \rangle \Phi$ = for all values v, either
for some t', $t \xrightarrow{\mathrm{tt}\ \ gv} t'$ and $t' \models \Phi[v/x]$
or
for some t', $t \xrightarrow{b\ \ gz} t'$ and $b[v/z] \equiv \mathrm{tt}$ and $t'_{[v/z]} \models \Phi[v/x]$

$t \models [\exists x\ g]\Phi$ = for some value v,
whenever $t \xrightarrow{\mathrm{tt}\ \ gv} t'$ then $t' \models \Phi[v/x]$ and
whenever $t \xrightarrow{b\ \ gz} t'$ and $b[v/z] \equiv \mathrm{tt}$ then $t'_{[v/z]} \models \Phi[v/x]$

$t \models [\forall x\ g]\Phi$ = for all values v,
whenever $t \xrightarrow{\mathrm{tt}\ \ gv} t'$ then $t' \models \Phi[v/x]$ and
whenever $t \xrightarrow{b\ \ gz} t'$ and $b[v/z] \equiv \mathrm{tt}$ then $t'_{[v/z]} \models \Phi[v/x]$

The first five rules are standard. The semantics of the new modal operators is essentially driven by the desire for adequacy. For structured transitions, the logic formula must contain a modality/quantifier pair. One

approach is to treat the different kinds of quantifiers (that is, over transitions or over data) separately in the semantics. For example, in the logic of Hennessy and Liu [2] the transition is chosen first, and the inductive step involves an abstraction (a variable, STS pair) and an open modal formula. This is possible because transition conditions only involve variables already bound, and a late semantics is used.

However, this order of evaluation is not appropriate for an early semantics. Consider the processes given in Figure 9. Assume the set A is neither empty nor universal. T and U are clearly bisimilar. If we defined a semantics for the logic in which the transition is chosen first, independently of the data, then a formula distinguishing these two processes can be constructed (so the logic is clearly not adequate). Specifically, $T \models [\forall x\ g]\mathrm{tt}$ while $U \not\models [\forall x\ g]\mathrm{tt}$. The latter fails because we are forced to choose a single g transition (and thus have to satisfy either $\forall\, x.x$ in A or $\forall\, x.x \not\in A$, which cannot be true if A is not empty or universal). Under an early interpretation quantifiers must be treated in the reverse order: data quantifier first, then transition quantifier. The structure of the syntax (putting the data quantifier inside the modal quantifier) means we have to treat the quantifier pairs in a single step in the semantics of Definition 3.2.

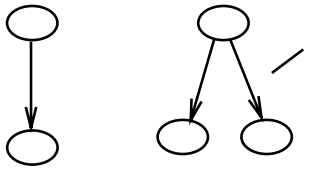

FIGURE 9. processes T and U

The rules for $t \models \Lambda$ (Definition 3.3), where T is a term with one free variable, relate this free variable to the (single) variable quantified over by a formula Λ.

DEFINITION 3.3. Semantics of FULL: Open Terms
Given any term t with one free variable, z, the semantics of an open formula, $t \models \Lambda$, is given by:

$$t \models \exists x.\Phi \quad = \quad \text{for some value } v,\ t_{[v/z]} \models \Phi[v/x]$$
$$t \models \forall x.\Phi \quad = \quad \text{for all values } v,\ t_{[v/z]} \models \Phi[v/x]$$

4. BISIMULATION OVER TERMS

An important relationship between processes is that of *bisimulation*. In this section we define (strong, early) bisimulation over terms and state the theorem that FULL is adequate with respect to this 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 4.1. 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 $t \xrightarrow{\mathrm{tt}\quad a} t'$, then for some u', $u \xrightarrow{\mathrm{tt}\quad a} u'$ and $t' \sim_{n-1} u'$
   (b) **(structured event, no new variable)**
   whenever $t \xrightarrow{\mathrm{tt}\quad gv} t'$, then either
   for some u', $u \xrightarrow{\mathrm{tt}\quad gv} u'$ and $t' \sim_{n-1} u'$
   or
   for some u', $u \xrightarrow{b_u\quad gz} u'$ 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 $t \xrightarrow{b_t\quad gz} t'$, where $z = new(t, u)$, then, for all v s.t. $b_t[v/z] \equiv \mathrm{tt}$, either
   for some u', $u \xrightarrow{\mathrm{tt}\quad gv} u'$ and $t'_{[v/z]} \sim_{n-1} u'$
   or
   for some u', $u \xrightarrow{b_u\quad gz} u'$ 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, $t_{[v/x]} \sim_n u_{[v/y]}$.

4.1. Adequacy of the Logic

As discussed previously, a desirable property of the logic is that it is *adequate* with respect to bisimulation. Some supporting examples for adequacy were given in Section 3.1; we now state this conjecture formally. The statement of the theorem relies on associating with each formula of FULL a depth, n, which is defined inductively on the structure of the formula.

THEOREM 4.1. For all n, for all terms t and u, $t \sim_n u$ if and only if t and u satisfy the same formulae, for all formulae of depth n.

The proof is omitted for lack of space but is presented elsewhere [7].

5. USING PVS TO PROVE ADEQUACY

The automated theorem prover PVS was used extensively in developing the definitions presented in this paper, and to some extent in proving adequacy. Here we discuss the reasons for choosing to use such a tool, and the resulting benefits and drawbacks.

The main reason for choosing to use a tool was to facilitate experimentation with different definitions of the logic. Many variations of the logic were considered before the definition in this paper was selected. As there are various constructors for formulae and many cases in the semantics of each constructor, much work needed to be done to analyze the consequences of each change (for example, checking that key lemmas continued to hold true.) On paper this was a tedious and error-prone task. With PVS, however, a proof can be edited and re-run,

and the user can be confident that every part of the proof has been thoroughly re-checked.

Another consequence of using PVS was that we were forced to make all definitions and proofs fully formal. This was both a benefit and a disadvantage.

On the positive side, the exercise of expressing all our definitions formally in PVS improved our understanding of many issues. For example, on paper, we had been able to be informal about issues such as how to define substitution on STSs. PVS forced us to scrutinise the details of this definition, and in so doing, brought us to a full appreciation of the reasons why substitution on STSs could not be defined satisfactorily, and why the concept of a "term" was required. This was a crucial step in arriving at a correct set of definitions and in proving the adequacy theorem.

On the other hand, once the right definitions had been found, the need to be fully formal became more of a hindrance. There were many simple subgoals in the proofs which were obvious to the human user, but were required to be proved in PVS. We judged that the benefits of full formal proof were not worth the extra time and effort it would require either to prove these subgoals in PVS or to configure the tool to prove them automatically (for example, by adding lemmas to be used for automatic proof). PVS was therefore used only in the initial stages of the proof; once we were confident that we had the right definitions and proof technique, the proof was completed on paper.

To give some idea of the amount of work done in PVS, the formalisation of the logic is about 150 lines long and the formalisation of STSs and related concepts is about 200 lines long. These numbers do not include definitions generated automatically by PVS.

6. SUMMARY AND FURTHER WORK

The standard semantics of Full LOTOS is an early semantics that instantiates all data, introducing the possibility of infinite branching in the underlying transition systems. This poses serious problems for any associated reasoning, particularly when demonstrating the expressive power of a logic or developing practical reasoning techniques such as model-checking. Consequently, we have developed an (early) *symbolic* semantics, based on symbolic transition systems (STSs), which eliminates infinite branching. The semantics, and (early) strong bisimulation for Full LOTOS, are presented in detail elsewhere [6]. Here, we have concentrated on the form of an associated logic, called FULL.

While our logic bears some similarity to that of Hennesey and Liu [2] for value-passing CCS, there are significant distinctions, arising primarily from the treatment of multiway synchronisation and selection predicates in Full LOTOS. These affect both the form of the STSs and any associated logics. Our main consideration is the possible combinations of data and event quantifiers to form new modal operators, and matching those operators with symbolic transitions in a way that corresponds with our intuitions, and with bisimulation.

We have illustrated the possible choices, through a set of examples. When making design choices, an overriding motivation is that the logic should be adequate with respect to our chosen bisimulation. In consideration of this, a formal syntax and semantics for FULL are given and adequacy has been proved. An important aspect of developing the proof, in the initial stages, was the use of an automated theorem prover.

In future work, we aim to build upon the logic FULL by adding useful extensions such as fixpoint operators. Work is also in progress on the development of practical tools to support reasoning in FULL and case studies to demonstrate its use.

ACKNOWLEDGEMENTS

The authors would like to thank the referees for their helpful comments, and the British Council, the Engineering and Physical Sciences Research Council, and the Nuffield Foundation for financial support.

REFERENCES

[1] ISO 8807 (1989). *Information Processing Systems — Open Systems Interconnection — LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*. International Organisation for Standardisation.

[2] Hennessy, M. and Liu, X. (1995). A Modal Logic for Message Passing Processes. *Acta Informatica*, Vol 32.

[3] Hennessy, M. and Milner, R. (1985). Algebraic Laws for Nondeterminism and Concurrency. *Journal of the Association for Computing Machinery*, Vol 32(1).

[4] Fernandez, J.-C., Garavel, H. *et al* (1996). CADP (CAESAR/ALDEBARAN Development Package): A Protocol Validation and Verification Toolbox. In *Proceedings of CAV'96*, LNCS 1102, Springer-Verlag.

[5] Hennessy, M. and Lin, H. (1995). Symbolic Bisimulations. *Theoretical Computer Science*, Vol 138.

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

[7] Calder, M., Maharaj, S., and Shankland, C. (2000). An Adequate Logic for Full LOTOS. To appear in the proceedings of Formal Methods Europe 2001.

[8] Shankar, N., Owre, S., and Rushby, J. M. (1993). PVS Tutorial. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA.

[9] Logrippo, L., Faci, M., and Haj-Hussein, M. (1992). An Introduction to LOTOS: Learning by Examples. *Computer Networks and ISDN Systems*, Vol 23.

[10] Stirling, C. (1989). Temporal Logics for CCS. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, LNCS 354, Springer-Verlag.