Towards a Symbolic Modal Logic for LOTOS

Carron Kirkwood*
Department of Computer Studies, Glasgow Caledonian University
Glasgow, Scotland

Muffy Thomas
Department of Computing Science, University of Glasgow
Glasgow, Scotland

Abstract

We present a version of HML [7], adapted to allow description of properties of Full LOTOS [8] specifications using symbolic values and conditions in modal formulae as well as concrete data values. In modifying HML we especially consider the features of LOTOS which distinguish it from other process algebras and discuss how these alter the requirements of the logic, and how this in turn is reflected in the semantics of the logic.

Our logic is given in relation to a late symbolic semantics for Full LOTOS, clearly separating the process reasoning from the data reasoning.

This work is motivated by our experiences of using LOTOS in several different applications [10, 15], in which we have identified a clear need for temporal reasoning about symbolic, and possibly partial, LOTOS specifications.

1 Introduction

The use of modal and/or temporal logics to describe abstract properties of systems is well established, as is their use in conjuction with more concrete process algebra descriptions of the same system. See, for example, [7, 13, 9]. In these cases, either the process algebra concerned does not involve data values, or concrete data values are used, either implicitly (e.g. in CCS) or explicitly (e.g. in LOTOS).

Full LOTOS [8] is a process algebra with algebraic data types and multi-way synchronisation[1]. Although LOTOS[2] allows data type process parameters, and data type expressions as process guards and structured event offers, the standard semantics of LOTOS does not give meaning to *open* specifications, that is, processes with uninstantiated data. (This is similar to the way CCS handles data values.) Thus, we cannot reason about the temporal behaviour of open specifications.

Open processes arise in two ways: because data variables may be formal parameters to processes, or because they are essentially "input" variables in structured events. For example, in the LOTOS process

```
P(x)[g] := g?y:nat[y>2];g!x+y; ...
```

`x` is a formal parameter; `y` is an input variable; `y>2` is a condition on the input, and `x+y` is a single (symbolic) output value.

But, often it is desirable to be able reason about such specifications, since they pervade any realistic system description. For example, in a telephone network, it is useful to be able to reason about (temporal) properties of the system which are independent of the data, as well as being able to reason about concrete properties and properties which *do* depend on the data values. Examples of data independent properties include *after a user dials a number, it is possible that the user will become connected to that number* and *after a user dials a number, the user will always become connected to that number*.

*Carron Kirkwood is now Carron Shankland and can be contacted at: Department of Computing Science and Mathematics, University of Stirling, Scotland

[1] See [12] for an introduction to the language.

[2] In the remainder of this paper "LOTOS" is used to refer to Full LOTOS. Basic LOTOS refers to the process part only.

The traditional handling of temporal properties of LOTOS processes is best represented in the toolset CADP [3] in which LOTOS specifications (with some syntactic restrictions) are translated into labelled transition systems, instantiating free variables by all possible data values; the validity of alternation-free modal μ-calculus [11] formulae is checked with respect to that labelled transition system. No symbolic values are permitted in the logical formulae.

Clearly, although a desirable and powerful logic is used here, the specifications do not retain their symbolic nature, nor can the logic express it. For example, we cannot express the ability of a process to perform the action `g!x+y` without instantiating `x` and `y`, nor can we express a property concerning `g?y` without instantiating `y`. These problems are overcome, to some extent, in Eludo [4]. Here, the validity of CTL [2] formulae are checked with respect to LOTOS specifications. Variables in the LOTOS remain symbolic, including unmatched query variables (i.e. they have not been matched with `!` offers), predicates about the variables are built up as the model is explored, and narrowing is used to obtain concrete values where possible. However, CTL cannot describe the cyclical properties we ultimately desire, and the (non-standard) semantics of LOTOS used here has not yet been formalised (other than by the Prolog rules of the implementation) nor compared with the standard semantics.

A third approach is described in [6], in which symbolic data values are allowed in the modal logic and the semantics of the logic is based on symbolic transition systems. The message-passing process algebra used is similar to CCS. This work is not applicable to LOTOS because it relies on features of CCS which are quite different from those in LOTOS: in particular conditions on input variables and multi-way synchronisation (N.B. synchronisation is two-way in CCS and results in an unobservable event. In LOTOS, multi-way synchronisation requires the result of two-way synchronisation to be observable and available for synchronisation with another process. This fundamentally affects the treatment of symbolic values, since in LOTOS we cannot assume that each `?` offer matches with exactly one `!` offer.). We diverge significantly from this approach.

Two main technical developments are described in this paper: a symbolic modal logic for LOTOS and a corresponding proof system. Although every logic is symbolic, we call our logic symbolic to highlight the fact that the data is symbolic. Note that the logic used is an extended form of HML [7]. Although this logic is weaker than CTL, our intention is to build on the work presented here by adding fixed points to the logic, yielding something more closely related to the modal μ-calculus, which will eventually give us the expressive power we desire.

Both the logic and the proof system are defined with reference to a late symbolic semantics for LOTOS which, unlike the standard semantics of LOTOS but similarly to the semantics of [5], clearly separates the process and the data. Conditions concerning the data control the exploration of the transition system; in the proof system these (data) proof obligations are discharged using a separate data reasoning system.

The paper is organised as follows. We begin with motivation in the form of an example which illustrates some of the requirements of the logic. The syntax and semantics of the logic itself are presented in Section 3 together with related topics: a late symbolic semantics of LOTOS in Section 3.1 and a discussion of important semantic issues in Section 3.4.

The proof system is presented in Section 3.5, and used in Section 4 to prove the validity of the properties described informally in Section 2. Finally, we conclude and discuss further work.

2 An Example

As motivation, consider a simple example of a LOTOS specification and some typical temporal properties. While the example is clearly contrived (LOTOS is typically used to describe communications protocols and system architectures c.f. [10, 15]) it does provide a simple, intuitive example of the issues to be addressed.

The example is given in Figure 1 and concerns user behaviour in a telephone network where users are forbidden to make and receive calls from particular users. Each user process is parameterised by

- the user id
- the list of prohibited incoming callers
- the list of prohibited outgoing numbers.

Telephone processes will synchronise with each other on the `con` (*connect*) and `discon` (*disconnect*) events, that is the entire network of users will be the (synchronised) parallel composition of several instantiations of the parameterised process `Tel`. The synchronising (structured) events are of the form `gate!x!y`, where `x` and `y` are the originating and terminating users, respectively. For example, `discon!x!y` denotes the event of disconnecting the

```
process Tel[dial,con,discon,unobt,busy,on]
        (id:userid,bar_in:idlist,bar_out:idlist) :noexit :=

(con?x:userid!id [not (x mem bar_in)]; discon!x!id; on; Tel(id,bar_in,bar_out))
[]
(dial?x:userid;
    ([x mem bar_out] -> unobt; on; Tel(id,bar_in,bar_out)
    []
    [not(x mem bar_out)] -> busy; on; Tel(id,bar_in,bar_out)
    []
    [not(x mem bar_out)] -> con!id!x; discon!id!x; on; Tel(id,bar_in,bar_out)
    []
    unobt; on; Tel(id,bar_in,bar_out)))
endproc
```

Figure 1: LOTOS Description of Telephone

call from user x to user y[3]. Note that in `Tel`, conditions are used both to guard processes (within a choice) and to qualify structured input events. For brevity, details of the datatype `userid` and `idlist` have been omitted.

We might desire the following, informally stated, properties to hold of `Tel`.

1. it is possible to dial a number which is not in the `bar_out` list,
2. after dialling a number, it is possible that the user hears an `unobt` tone,
3. after dialling `999`, it is possible that the user is connected to `999`,
4. after dialling a number which is in the `bar_out` list, the user can only hear the `unobt` tone.
5. after dialling a number which is in the `bar_out` list, it is not possible for the user to connect to the number dialled,
6. after dialling a number which is not in the `bar_out` list, it is possible that the user is connected to the number dialled,
7. after dialling a number, it is possible that after either 1 or 2 events, the user puts the handset `on`,

Note: it is assumed that `999` is not a member of the list `bar_out`.

These properties refer to concrete values, e.g. `999`, and also to symbolic values, e.g. *a number*. Thus, the modalities of the logic should include symbolic data values in addition to concrete ones. Moreover, events are qualified by predicates over the data, e.g. *a number which is not in the* `bar_out` *list*, thus the logic should also include similar qualification of events.

Having a concrete idea of the sorts of properties to be described will help us develop the logic and its semantics.

3 A Symbolic Modal Logic

In this section we present the syntax (Section 3.2) and semantics (Section 3.3) of the logic. The semantics of the logic is based on a symbolic semantics for LOTOS, an overview of which is given in Section 3.1. It is beyond the scope of this paper to give the full details of that semantics, see [1] for further information. In Section 3.4 we give a detailed insight into the important semantic issues considered during the development of the logic. Finally, we present the proof system.

We begin with the symbolic semantics of LOTOS.

[3] We note that strictly speaking, the synchronisation of structured events in this way is not possible in LOTOS; this is one of the features of ELOTOS, the new version of LOTOS which is currently being standardised. This is not a significant drawback in this example because only the properties of the process `Tel` are investigated, rather than those of a particular network. That is, we want to ensure that we have captured the right behaviour of a single user process, before considering the behaviour of a network of users.

3.1 Symbolic LOTOS

The standard LOTOS semantics [8] is given by structured labelled transition systems. This symbolic semantics (presented in [1]) enhances those transition systems in two ways. First, we allow variables in both states and labels. Second, we add an extra component to transitions.

In the symbolic semantics, transitions take the form $S \xrightarrow{D \quad gE} S'$ where S and S' are states, D is a boolean condition based on the data type language of the LOTOS description, and gE is a transition label consisting of g, a gate name, and a list of event offers $E = d_1 \ldots d_n$, where $d_i = ?e_i$ or $!e_i$. Informally, D describes the "leaving" conditions for a state and is a condition on data only. When D is unsatisfiable, the associated transition and destination state are considered to be unreachable, or "imaginary".

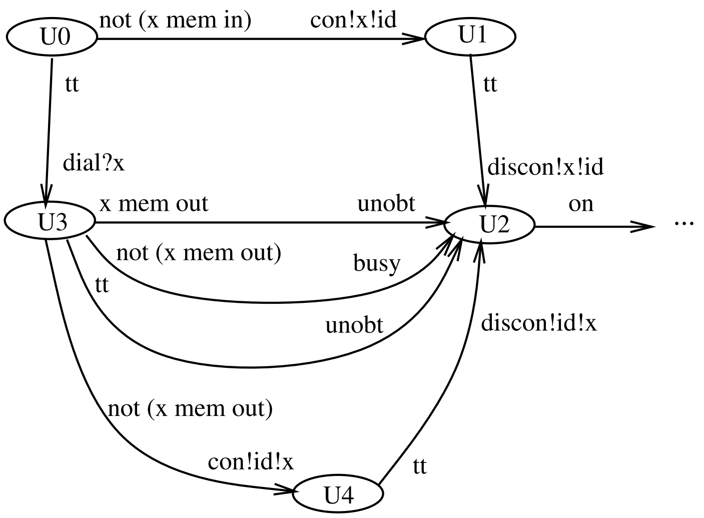

Figure 2: Telephone

To illustrate the semantics a pictorial representation of the semantics of the telephone process of Figure 1 is given in Figure 2. Note that while states in the diagram have been identified for ease of interpretation (e.g. state U2), state equivalence is not formally defined. State equivalence becomes more important when fixed point operators are to be introduced to the logic (a topic not considered here).

We now present the syntax of the logic, followed by its semantics (which is based on this symbolic semantics of LOTOS).

3.2 Syntax

The syntax of our logic is based on that of HML as presented in [13]. Here, the sets permitted in the modalities are enriched by allowing structured events and conditions, both of which may involve variables.

$$::= \mathrm{tt} \mid \mathrm{ff} \mid {}_1 \wedge {}_2 \mid {}_1 \vee {}_2 \mid [\kappa] \mid \langle\kappa\rangle$$

$$\kappa ::= K \mid \{gE \mid R\}$$

where $K \subseteq G \cup \{i\}$, G is the set of gate names, $g \in G$, $a \in G \cup \{i\}$, E is a list of *event offers*, i.e. data type expressions prefixed by `!` or `?`, and R is a *condition* over some variables (typically those in E). The logic must share the appropriate LOTOS abstract data types with the process definition (otherwise formulation of R is impossible).

Braces in formulae are typically omitted. Other standard useful abbreviations are also adopted. For example, $\langle - \rangle$ to denote $\langle G \cup \{i\}\rangle$ and $\langle -K\rangle$ to denote $\langle (G \cup \{i\}) - K\rangle$; similarly for $[-]$ and $[-K]$.

To give meaning to the above syntax it must be related to the (symbolic) semantics of processes (as given in Section 3.1.

3.3 Semantics of Symbolic Modal Logic

We define $\|\ \|_C$, the set of processes satisfying a modal formula in context C, where a context is a collection of conditions, inductively over the syntax of the logic by the following equations:

$$
\begin{array}{lcl}
\| \mathrm{tt} \|_C & = & \mathcal{P} \\
\| \mathrm{ff} \|_C & = & \emptyset \\
\| {}_1 \vee {}_2 \|_C & = & \| {}_1 \|_C \cup \| {}_2 \|_C \\
\| {}_1 \wedge {}_2 \|_C & = & \| {}_1 \|_C \cap \| {}_2 \|_C \\
\| \langle a \rangle \|_C & = & \{P \in \mathcal{P} :\ \exists P' \in \|\ \|_C.\ P \xrightarrow{\mathrm{tt} \quad a} P'\} \\
\| [a] \|_C & = & \{P \in \mathcal{P} :\ \forall P'. P \xrightarrow{\mathrm{tt} \quad a} P' \ \Rightarrow\ P' \in \|\ \|_C\} \\
\| \langle gE \mid R \rangle \|_C & = & \{P \in \mathcal{P} : \exists P'.\exists \sigma. P \xrightarrow{D \quad gE'} P' \wedge\ E\sigma \equiv E' \wedge \\
 & & \quad (C \wedge\ D \Rightarrow\ R\sigma)\ \wedge (C \wedge\ D) \wedge P' \in \|\ \sigma \|_{C \wedge D}\} \\
\| [gE \mid R] \|_C & = & \{P \in \mathcal{P} : \forall P'.\exists \sigma.((C \wedge\ D)\ \wedge\ (C \wedge\ D \Rightarrow R\sigma) \wedge \\
 & & \quad ((P \xrightarrow{D \quad gE'} P' \wedge E\sigma \equiv E') \Rightarrow P' \in \|\ \sigma \|_{C \wedge D}))
\end{array}
$$

Note that $\mathcal{P}$ is taken to be the set of all LOTOS processes.

The last two equations require some explanation. In each case, we match the event offers in E with those in E' by a substitution σ, allowing the `!`'s and `?`'s to match in any combination. Furthermore, the matching is done with respect to the equivalence of the underlying data type. These, and other, features are more fully explained in Section 3.4.

In order to express that a particular process P satisfies a modal formula we show

$$P \in \|\ \|^{\mathcal{P}}_{\mathrm{tt}}$$

which we also write more conveniently as

$$P \models$$

The set $\|\ \|^{\mathcal{P}}_{\mathrm{tt}}$ can always be constructed if the underlying data type theory is complete; however, in general we cannot necessarily determine the validity of the data conditions.

3.4 Semantic Issues

A number of semantic issues arise from the introduction of symbolic values and qualifying conditions on events:

- the difference between `!` and `?` event offers,
- qualification of event offers by conditions,
- "symbolic event matching", i.e. under what conditions does a (symbolic) label in the transition system satisfy a (symbolic) event in the logic?
- interpretation of the modal operators $\langle gE \mid R\rangle$ and $[gE \mid R]$.

These issues are significant because the decisions made affect the expressiveness of the logic.

When considering these issues our guiding principle is that the semantics satisfies

$$(P \models\ ) \Leftrightarrow (\forall \sigma . P\sigma \models\ )$$

where σ provides a concrete instantiation of P. In other words, if the symbolic P satisfies then there should be no instantiation of P which does not satisfy , and vice versa.

3.4.1 Structured and Qualified Events

Here we are concerned with the difference between `!` and `?` event offers; in particular, what are the consequences of allowing uninstantiated variables in offers? The standard semantics [8] of LOTOS gives the following interpretation to structured events. A `g!e` event corresponds to one transition labelled by `g v` where `v` is the valuation of `e` (where `e` is not an open term). A `g?x:S` event corresponds to many (possibly infinite) transitions, one for each value in `S`. In addition, the `?` is a binding occurrence of an identifier, while the `!` is a using occurrence.

In the symbolic semantics, there is one (symbolic) transition corresponding to `g?x:S` labelled by `g x`, which corresponds to a set of transitions in the standard transition system. Under this interpretation, conditions on input event offers become simply restrictions on the sets of the underlying events, and therefore transitions. Similarily, there is one (symbolic) transition corresponding to `g!e`, where `e` *may* be an open term.

3.4.2 Symbolic Event Matching

How do we match events in the logic with labels in the transition system?

In Basic LOTOS simple event names (gates) appear as transition labels and in the modalities; therefore event matching is trivial – they must by syntactically equal. In Full LOTOS we must not only match gate names, but also event offers (i.e. data expressions). Moreover, we must also take into account the equivalence induced by the data theory *and* any conditions imposed on the logical events *and* any conditions imposed on the process input variables. For example, when does `g!x` match `g!y`? Can `g!x` match `g?z`? Can `g?x [x<10]` match `g?y [y>10]`?

Our initial observation is that when matching symbolic event offers it does not matter which kind of event offer we have, i.e. `!` or `?`, since they both relate to sets of (concrete) events. Thus we cannot distinguish between, say, `!x` and `?x`, since both are offering a set of values denoted by `x`. We can therefore ignore the `!` and `?` when matching; the data expressions and the conditions alone determine matching.

Let us consider some simple concrete examples in which neither the transition label nor the logical event involves conditions. These are given in the top left portion of the table in Figure 3. Possible transition labels (drawn mainly from `Tel`) appear in the left hand column, and possible logical events from a logical formula appear across the top. In the example, we assume the usual theory of numbers (e.g. $2 \not\equiv 3$).

In these simple cases, the transition label satisfies the logical event when their respective offers, i.e. data expressions, can be matched (modulo the underlying data theory). For example `dial?x` is matched by $dial?y$ because $\mathtt{x} \equiv y\sigma$, where σ renames y by `x`. In other words, both of these are event offers which could be understood to stand for the same set of transitions (assuming `x` and y are of the same type and therefore range over the same possible data values). On the other hand, `dial?x` cannot be matched by $dial!2$ because there is no substitution which allows us to replace 2 by `x`. We cannot allow matching in the opposite "direction" (i.e. match transition label to logical event) because then there could be instantiations of `x` which might not satisfy the whole property, thus violating our guiding principle.

Now consider the more complex case where we also have conditions on the events (in either the transition system or the logic). The rest of the table in Figure 3 illustrates such situations. For convenience we use $\in$ for list membership rather than the more cumbersome `mem` of the LOTOS description. Again, we assume the usual theory of numbers, but make no assumptions about the sets `bar_out` and `bar_in`. Thus, we are left with some unreduced formulae (e.g. (x $\notin$ bar_out) $\Rightarrow$ (x $\in$ bar_out)).

The relation between conditions over the logical events and conditions over the process events can perhaps be best understood in terms of the underlying correspondence with sets of concrete events. If the logical event relates to a set A of concrete events and the process event to a set B of events, should we, when matching, insist that $A = B$ or $A \subseteq B$ or $B \subseteq A$? Alternatively, this can be expressed in terms of the conditions by $C \Leftrightarrow R$, $C \Rightarrow R$ or $R \Rightarrow C$.

Again, we use our guiding principle to help us decide, i.e. it doesn't matter what concrete values we give P in place of the symbolic values, the resulting process still satisfies the formula. This means we must require that the logical property is a looser specification of events than the process, i.e. we require $C \Rightarrow R$. (We note that could have adopted other views, which would result in a different semantics and proof system.)

	$dial?y$	$dial!2$	$dial?y \| y \in bar\_out$
`dial?x`	tt	ff	(x ∈ bar_out)
`dial!x`	tt	ff	(x ∈ bar_out)
`dial!2`	tt	tt	(2 ∈ bar_out)
`dial!3`	tt	ff	(3 ∈ bar_out)
`dial?x` [x ∈ bar_out]	(x ∈ bar_out) ⇒ tt	(2 ∈ bar_out) ⇒ tt	(x ∈ bar_out) ⇒ (x ∈ bar_out)
`dial?x` [x ∉ bar_out]	(x ∉ bar_out) ⇒ tt	(2 ∉ bar_out) ⇒ tt	(x ∉ bar_out ⇒ x ∈ bar_out)
`dial?x` [x ∈ bar_in]	(x ∈ bar_in) ⇒ tt	(2 ∈ bar_in) ⇒ tt	(x ∈ bar_in ⇒ x ∈ bar_out)

Figure 3: Matching Logical Events with Transition Labels

For example, intuitively we expect `dial!2; exit` $\models \langle dial?x\rangle$tt, because the logic allows many more events than the process. Similarly, we expect `dial?x[nastychatline(x)]; unobt; exit` $\models \langle dial?u|u \in bar_out\rangle$tt, assuming every number to a nasty chat line is in the bar_out list.

On the other hand, since `dial?x; exit` allows more events than $\langle dial!y|y \notin bar_out\rangle$tt, we do not have `dial?x; exit` $\models \langle dial!y|y \notin bar_out\rangle$tt. For example, we could instantiate x by a value v which is in bar_out, and then `dial?v; exit` $\not\models \langle dial!y|y \notin bar_out\rangle$tt.

The result of this discussion is that symbolic event matching depends not only on syntactic matching of event offers, modulo the underlying data theory, but also on the implication $C \Rightarrow R$, i.e. the process condition is stronger than the logical condition.

3.4.3 Modal Operators

The final issue to be considered relates to the interpretation of the modal operators in the new setting.

In the semantics of HML [7] the $\langle g\rangle$ operator is interpreted as being satisfied by just one branch from the current state with label g. In our new logic this interpretation does not change. We require that the event labelling the transition matches the event in the logic, and that the transition condition implies the logical condition (if there is one). If such a condition exists then we will eventually choose it.

On the other hand, the box operator permits more scope for alternative interpretations. Consider the following process:

```
P :=   g!5; g!6; stop [] g!6; g!7; stop
     []
       g!1; g!1; stop [] g!0; g!1; stop
     []
       h!8; g!4; stop
```

Should $P \models [g!x+1 \mid x > 2]\langle g!x+2\rangle$tt hold?

In the semantics of HML [7] the $[g]$ operator is interpreted as being satisfied by all branches from the current state with label g. Given the introduction of conditions, there are more interpretations of the set of transitions which satisfy $[gE|R]$. First we must select the set of transitions of interest; we can have either

- all transitions with labels matched by gE, or
- all transitions whose labels match gE and whose conditions satisfy R, i.e. $C \Rightarrow R$.

Having chosen our set we must then check that the conditions on each transition in the set satisfies R. (For the latter choice of set obviously this is trivially true.)

The different interpretations can be illuminated by considering the above example: Under the first interpretation the set of interest contains the transitions labelled `g!5`, `g!6` and `g!1`. The `h` branch is excluded because the gate name does not match and the `g!0` branch is excluded because there is no substitution σ such that $0 = (x+1)\sigma$ (assuming we are dealing with natural numbers). Given this set, the property $[g!x+1 \mid x > 2][g!x+2]$tt fails to hold because, although the `g!1` branch matches $g!x+1$ with $[x/0]$, the condition is not satisfied, i.e. $0 \not> 2$). We conclude that the first interpretation is too weak; too many undesirable branches are included.

Under the second interpretation the troublesome branch would be excluded from the set of interest, therefore the condition would succeed (with branches `g!5` and `g!6`).

A further variation in interpretation may be given by asking: should the same substitution be used to match transition labels in each branch with the logical event, or can we allow the substitutions for each branch to be different? In our example it seems clear that it is desirable to allow different matching for each branch; in the first branch x is instantiated by 5 and in the second it is instantiated by 6. If we did not allow this then the property would fail. Insisting on the same matching for each is too strong.

Each of the possible interpretations described above could be valid; we have chosen one which appears neither too strong nor too weak (i.e. choose all branches whose labels match a, allowing different matching for each branch, and whose conditions imply the logical conditions). The others give a slightly different flavour to the logic (a topic for future research might be to investigate the difference in expressive power between each of the possible interpretations of the box operator).

3.5 Proof System

The semantics given in Section 3.3 is not very useful to us when trying to show a particular process P satisfies a property ; it is too expensive to build the set of all processes which satisfy and then check if $P \in \|\ \|_{\mathrm{tt}}$.

Instead, we would like a proof system which allows us *derive* $P \vdash$ with reference only to the particular P in which we are interested. Such a system (for HML) is presented in Stirling and Walker [14]. The proof system presented here is derived from that of [14], but has two important differences:

Firstly, we assume the existence of a separate proof system for the data which we can use as an "oracle" during the proof process. In reality, the oracle will be implemented by an auxiliary proof system.

Secondly, our proof system fulfills two roles. The first is to determine the satisfaction relation between a process, under some initial conditions, and a formula. We refer to this as (unconditional) satisfaction. This is described in more detail in Section 3.5.1, and the rules are given in Figure 4. The second is the synthesis of additional conditions which, if imposed on the process, would make the satisfaction relation hold, should it not hold for the given conditions. We call this (conditional) satisfaction. This is described in more detail in Section 3.5.2, and the rules are given by adding the rules in Figure 5 to those of Figure 4. We have combined both roles in one system and therefore proofs are carried out with respect to a process (state) P, an initial condition C and a set of additional conditions A.

Conditions which are imposed by the process are collected in C as the proof progesses. Additional conditions, A, come from the logical formula and may only be introduced in the (conditional) rules for $[gE|R]$ and $\langle gE|R\rangle$. Additional conditions take the form of R.

Sequents of the form $(P, C, A) \vdash$ appear in the rules of Figure 4 and Figure 5. We may read this as "P satisfies given that the context $C \wedge A$ is valid". By an abuse of notation we allow A in the conjunction to mean $\bigwedge\{A\}$. We also take $\bigwedge\{\}$ to be equivalent to tt.

To prove $(P, C, A) \vdash$ holds a successful tableau which has $(P, \mathrm{tt}, \mathrm{tt}) \vdash$ as the root goal must be constructed.

Of course, we want our proof system to be compatible with the semantics given in Section 3.3, therefore we require

$$(P, \mathrm{tt}, \mathrm{tt}) \vdash \quad \Leftrightarrow \mathrm{P} \models$$

This holds (the proof is trivial) for (unconditional) satisfaction.

3.5.1 (Unconditional) Satisfaction

(Unconditional) satisfaction means that we can build a successful tableau without having to synthesise extra conditions in A. Formally, $P \in \|\ \|_C$, i.e. $(P, C) \models$, then we prove $(P, C, \mathrm{tt}) \vdash$ as follows.

We must construct an (unconditionally) successful tableau which has this sequent as the root goal. An (unconditionally) successful tableau is a finite proof tree in which all leaves are (unconditionally) successful. A leaf is (unconditionally) successful if

1. it has the form $(P', C', \mathrm{tt}) \vdash$ [K] and none of the (unconditional) rules of figure 4 can be applied.
2. it has the form $(P', C', \mathrm{tt}) \vdash$ tt.

where C is the data condition generated by the proof system (from the process conditions).

All other leaves are unsuccessful.

Note that for (unconditional) satisfaction, we must always apply the uncondional form of the box and diamond rules. Application of the rules, even by hand, is fairly mechanical; only the $\vee$ and $\langle K\rangle$ rules give a choice as to the next level of the tableau, making tableau construction nondeterministic.

If a proof tree has any unsuccessful leaves then the tableau is unsuccessful and the original sequent *may* not hold. We can backtrack from an unsuccessful leaf to a choice point and choose a different path (which we hope may lead to success). In proving the validity of a formula the important point is the existence of a successful tableau, not the non-existence of an unsuccessful tableau.

When the reason for failure is that there are leaves with additional conditions in A, then we may try to prove (conditional) satisfaction.

3.5.2 (Conditional) Satisfaction

When $P \notin \| \ \|_C$, i.e. $(\mathtt{P}, \mathtt{C}, \mathrm{tt}) \not\vdash$, we may try to generate the additional conditions A such that $P \in \| \ \|_{C \wedge A}$, or, put another way, discover how to modify P into P' such that $P' \in \| \ \|_C$.

In either case, we try to synthesise sets of conditions during the proof process which if we choose to enforce them, either by modifying the process or the initial conditions, the satisfaction relation will hold. For example, if the formula has the form $\langle dial?u | u \in bar\_out \rangle$ and there is a transition $P \xrightarrow{tt \quad dial?x} P'$ then we want to record that the formula holds, only if the process imposes the stronger condition $x \in bar\_out$.

$(P, C, \mathrm{tt}) \vdash$ *conditionally* when a (conditionally) successful tableau can be constructed which has this sequent as the root goal. A (conditionally) successful tableau is a finite proof tree in which all leaves are (conditionally or unconditionally) successful and there is at least one leaf which is (conditionally) successful. A leaf is (conditionally) successful if

1. it has the form $(P', C', A) \vdash [K]$, and none of the rules of Figure 4 or Figure 5 can be applied.
2. it has the form $(P', C', A) \vdash \mathrm{tt}$.

Similar clauses in terms of $\langle K \rangle$ and ff identify unsuccessful leaves.

A (conditional) proof tableau can be used in two ways: we can combine the additional conditions at the leaves to strengthen the initial condition, or we can impose the additional conditions in the process at the appropriate places. The latter approach must be adopted when the additional conditions at the leaves are inconsistent, e.g. we may have the condition $x \in bar\_out$ at one leaf and $x \notin bar\_out$ at another leaf.

$$\wedge\ \frac{(P,C,A) \vdash {}_1 \wedge {}_2}{(P,C,A) \vdash {}_1 \quad (P,C,A) \vdash {}_2}$$

$$\vee\ \frac{(P,C,A) \vdash {}_1 \vee {}_2}{(P,C,A) \vdash {}_1} \qquad\qquad \vee\ \frac{(P,C,A) \vdash {}_1 \vee {}_2}{(P,C,A) \vdash {}_2}$$

$$[A]\ \frac{(P,C,A) \vdash [K]}{(P_1,C,A) \vdash \ \ldots (P_n,C,A) \vdash}$$

$$\{P_1,\ldots,P_n\} = \{P' \mid \forall P'.P \xrightarrow{\mathrm{tt} \quad a} P' \text{ and } a \in K\}$$

$$\langle A\rangle\ \frac{(P,C,A) \vdash \langle K\rangle}{(P',C,A) \vdash}\ \exists P'.P \xrightarrow{\mathrm{tt} \quad a} P' \text{ and } a \in K$$

$$[gE|R]\ \frac{(P,C,A) \vdash [gE|R]}{(P_1, C \wedge D_1, A) \vdash \sigma_1 \ \ldots (P_n, C \wedge D_n, A) \vdash \sigma_n}$$

where $\{P_1,\ldots,P_n\} = \{P' \mid \forall P'.P \xrightarrow{D_i \quad gE_i} P'\}$
and $\exists \sigma_1 \ldots \sigma_n.\ E\sigma_1 \equiv E_1 \ldots E\sigma_n \equiv E_n$
and $\forall i \in \{1 \ldots n\}\ ((C \wedge D_i) \Rightarrow R\sigma_i \wedge A\sigma_i)\ \wedge\ (C \wedge D_i)$.

$$\langle gE|R\rangle\ \frac{(P,C,A) \vdash \langle gE|R\rangle}{(P', C \wedge D, A) \vdash \sigma}$$

$\exists P'.P \xrightarrow{D \quad gE'} P'$ and $\exists \sigma.\ E\sigma \equiv E'$
and $(C \wedge D \Rightarrow R\sigma \wedge A\sigma)\ \wedge\ (C \wedge D)$.

Figure 4: Inference Rules for the (Unconditional) Satisfaction

$$\langle gE|R\rangle cond\ \frac{(P,C,A) \vdash \langle gE|R\rangle}{(P'\sigma, C\sigma \wedge D\sigma, A\sigma \cup R\sigma) \vdash \sigma}$$

$\exists P'.P \xrightarrow{D \quad gE'} P'$ and $\exists \sigma.E\sigma \equiv E'\sigma$ and $(C\sigma \wedge D\sigma \wedge A\sigma \wedge R\sigma)$.

.

$$[gE|R]cond\ \frac{(P,C,A) \vdash [gE|R]}{(P_1\sigma_1, C\sigma_1 \wedge D_1\sigma_1, A\sigma_1 \cup R\sigma_1) \vdash \sigma_1 \ \ldots (P_n\sigma_n, C\sigma_n \wedge D_n\sigma_n, A\sigma_n \cup R\sigma_n) \vdash \sigma_n}$$

where $\{P_1,\ldots,P_n\} = \{P' \mid \forall P'.P \xrightarrow{D_i \quad gE_i} P'\}$
and $\exists \sigma_1 \ldots \sigma_n.\ E\sigma_1 \equiv E_1\sigma_1 \ldots E\sigma_n \equiv E_n\sigma_n$
and $\forall i \in \{1 \ldots n\}\ (C\sigma_i \wedge D_i\sigma_i \wedge A\sigma_i \wedge R\sigma_i)$.

Figure 5: Inference Rules for (Conditional) Satisfaction

4 The Proof System in Action

In this section we introduce a second specification of the telephone user, we formalise the example properties from Section 2, and then we consider the satisfaction of those properties by the two specifications.

The second user process specification is given in Figure 6; its underlying (symbolic) lts is given in Figure 7. The difference between `Tel` and `Tel_II` is essentially the points at which choices are made, rather than the criteria involved in those choices.

```
process Tel_II[dial,con,discon,unobt,busy]
        (id:userid,bar_in:idlist,bar_out:idlist) :noexit :=

con?x:userid!id [not (x mem bar_in)]; discon!x!id; on;
        Tel_II(id, bar_in, bar_out)
[]
dial?x:userid [x mem bar_out]; unobt; on; Tel_II(id, bar_in, bar_out)
[]
dial?x:userid [not(x mem bar_out)]; on; busy; Tel_II(id, bar_in, bar_out)
[]
dial?x:userid [not(x mem bar_out)]; con!id!x; discon!id!x; on;
        Tel_II(id, bar_in, bar_out)
[]
dial?x:userid; unobt; on; Tel_II(id, bar_in, bar_out)
endproc
```

Figure 6: LOTOS Description of Telephone II

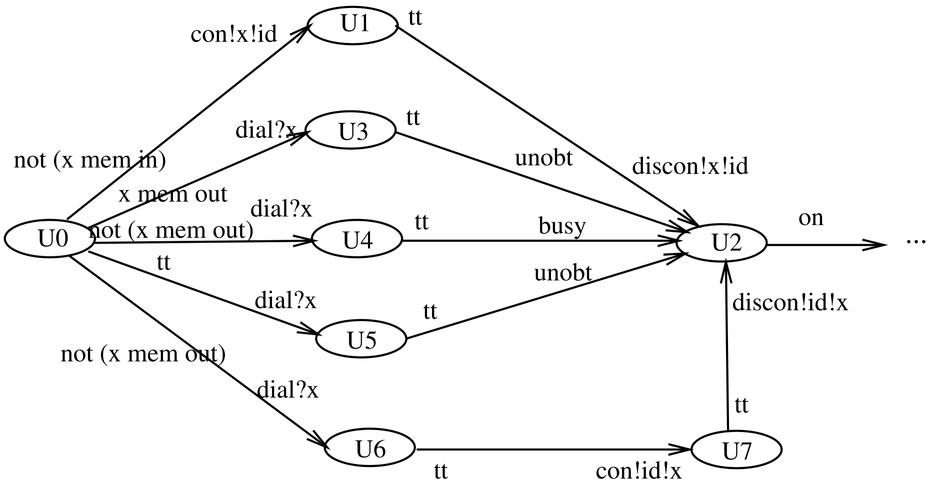

Figure 7: Telephone II

The properties informally described in Section 2 are formalised below.

1. $\langle dial?u|u \notin bar_out\rangle$tt
2. $\langle dial?u|$tt$\rangle\langle$unobt$\rangle$tt
3. $[dial!999|$tt$]\langle$con!id!999|tt$\rangle$tt
4. $\langle dial?u|u \in bar_out\rangle(\langle unobt\rangle$tt $\wedge$ $[-$unobt$]$ff$)$
5. $\langle dial?u|u \in bar_out\rangle[con!id!u|tt]$ff
6. $\langle dial?u|u \notin bar_out\rangle\langle con!id!u|tt\rangle$tt
7. $[dial?u|$tt$][-](\langle$on$\rangle$tt $\vee$ $[-]\langle$on$\rangle$tt$)$

Note that in the logic we have to be more precise about the scope of the modal operators than in natural language. In the informal descriptions, only one quantifier (e.g. "it is possible" or "always") appeared in each of the formula, the scope of which is not completely clear. Since the purpose of this paper is not to explore rigorously the formalisation of informal properties, we have simply chosen a reasonable formalisation, in each case.

For each formula above we try to derive $(\texttt{Tel}, \text{tt}, \text{tt}) \vdash$ and $(\texttt{Tel\_II}, \text{tt}, \text{tt}) \vdash$ first using the unconditional notion of satisfaction, and, if that fails, using the conditional notion of satisfaction.
The results are as follows:

1. `Tel` fails to satisfy the formula unconditionally because the side conditions require that tt $\Rightarrow$ x $\notin$ bar_out (which does not hold). At first glance this is a rather surprising result; however, this result is consistent with our guiding principle. While the transition system does allow us to dial some numbers which do not belong to `bar_out`, it does not prevent us from dialling numbers which are in `bar_out`, e.g. `nastychatline(x)`. There are therefore other instantiations of `x` which will not satisfy the formula.

   We can show conditional satisfaction, generating the additional condition $x \notin bar\_out$.

   `Tel_II` satisfies the formula unconditionally (because there are several `dial` actions, one of which has the condition $x \notin bar\_out$ as required).

2. Both `Tel` and `Tel_II` satisfy the formula unconditionally, generating no conditions.

3. For `Tel` no successful tableau can be found for unconditional satisfaction. This is because there is no substitution σ to allow us to match $x = 999\sigma$. Again, the failure to prove unconditional satisfaction arises because there may be other instantiations of `Tel` which do not satisfy the formula. `Tel` satisfies the formula conditionally, given the substitution $x = 999$ (note that conditional satisfaction allows unification).

   `Tel_II` fails to unconditionally satisfy this property for the same reasons as above; however, `Tel_II` also fails to conditionally satisfy the property. This is because the semantics of the box operator requires that all possible transitions `dial!999` are examined. There are four such transitions in `Tel_II`, and only one of those leads to a state in which the `con` action is possible.

4. `Tel` only satisfies the formula conditionally (the failure of the unconditional proof is for the same reasons as in property 1). The leaves of the proof are: $(U2, \text{tt}, \text{x} \in \text{bar\_out}) \vdash$ tt and $(U3, \text{tt}, \text{x} \in \text{bar\_out}) \vdash [-\text{unobt}]\text{ff}$.

   `Tel_II` satisfies the property unconditionally.

5. This property does not hold of `Tel` under unconditional satifaction (as above) but does hold conditionally, with leaf $(U3, \text{tt}, \text{x} \in \text{bar\_out}) \vdash [\text{con!id!x}|\text{tt}]\text{ff}$ (no transitions are possible because the condition $x \notin bar\_out \Rightarrow x \in bar\_out$ does not hold).

   `Tel_II` satisfies the property unconditionally (again, because we finish with the leaf $(U3, x \in bar\_out, \text{tt}) \vdash [\text{con!id!x}|\text{tt}]\text{ff}$ and no transitions are possible).

6. `Tel` satisfies the formula conditionally with leaf $(U4, x \notin bar\_out, x \notin bar\_out) \vdash$ tt and `Tel_II` satisfies the formula unconditionally with leaf $(U7, x \notin bar\_out, \text{tt}) \vdash$ tt.

7. Both `Tel` and `Tel_II` satisfy the formula unconditionally.

In two of the above cases a formula which we might intuitively expect to have been valid was not. This is because of certain decisions made in developing the semantics of the logic. Another interpretation would make these properties hold, but would probably also make other, currently valid properties fail. There is no middle line. What we would like, however, is for such failed proofs to give us more information. For example, the first property fails because we insist that the value of x belongs to bar_out. We could use this information to refine either the process or the formula. This information about conditions is gained through the conditional proof.

5 Conclusions and Further Work

In this paper we have presented a symbolic modal logic which can be used to describe properties of a LOTOS specification. There was some scope for alternative formulations of the logic: in particular, the satisfaction of structured events and conditions, and the interpretation of the operator. We tried to make clear the motivation behind our design decisions, where possible. The semantics of the logic is tied to a late symbolic semantics of LOTOS (developed in [1]). We also presented a proof system in which LOTOS processes can be shown to satisfy logical formulae. By use of example properties we demonstrated the logic and the proof system.

Our approach may be viewed as a hybrid of two of the approaches described in Section 1, providing the flexibility of symbolic data values of Eludo [4] in a formal framework, as in [6]. It may also be the case that the semantics outlined in Section 3.1 provides a formal basis for the Eludo work (though we have not fully explored this).

It is interesting to consider the role of the initial data condition in the tableaux (and in the semantics). As described here, the initial data context is always simply tt; however, when considering partial specifications we may want to reason from a more complex context. The proof system allows this, although we did not demonstrate that feature here. Moreover, in the examples, we also identified the possibility of synthesising additional data conditions which might be used to refine the process under consideration, or to constrain the input values, in the case that the formula does not hold.

Further developments of this work are to show that the semantics and logic are downwards compatible, that is they preserve the standard semantics and modal logics of, say, [9]. Also, as stated in the introduction, we wish to increase the expressive power of the logic to the level of the modal μ-calculus by adding fixed point operators. In order to do this we must also consider the way we deal with recursion in our symbolic semantics, and define state equivalence for (symbolic) LOTOS states in the semantics.

Acknowledgements

Ed Brinksma helped us enormously in the development of the symbolic semantics of LOTOS and in shaping the form of the modal logic presented here.

References

[1] E. Brinksma, C. Kirkwood, and M. Thomas. A Symbolic Semantics for Full LOTOS Processes. Draft Manuscript. Contact the authors (ces@cs.stir.ac.uk or muffy@dcs.gla.ac.uk) to obtain a copy, 1996.

[2] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications. In *ACM TOPLAS*, volume 8, 1986.

[3] J-C. Fernandez, H. Garavel, L. Mounier, A. Rasse, C. Rodriguez, and J. Sifakis. A Toolbox for the Verification of LOTOS Programs. In L.A. Clarke, editor, *Proceedings of the 14th International Conference on Software Engineering, ICSE 14*, 1992.

[4] B. Ghribi and L. Logrippo. A Validation Environment for LOTOS. In A. Danthine, G. Leduc, and P. Wolper, editors, *Protocol Specification, Testing, and Verification, XIII*, pages 93–108. Elsevier Science Publishers B.V. (North-Holland), 1993.

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

[6] M. Hennessy and X. Liu. A Modal Logic for Message Passing Processes. In C. Courcoubetis, editor, *Proceedings of CAV'93*, LNCS 697, pages 359–370, 1993. Also available as University of Sussex Computer Science Technical report, number 93:03.

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

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

[9] C. Kirkwood. Specifying Properties of Basic LOTOS Processes Using Temporal Logic. In *Formal Description Techniques, VIII*. Chapman Hall, 1995.

[10] C. Kirkwood and M. Thomas. Experiences with LOTOS Verification: A Report on Two Case Studies. In *Workshop on Industrial-Strength Formal Specification Techniques*, pages 159–171. IEEE Computer Society Press, 1995.

[11] D. Kozen. Results on the Propositional μ-Calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[12] L. Logrippo, M. Faci, and M. Haj-Hussein. An Introduction to LOTOS: Learning by Examples. *Computer Networks and ISDN Systems*, 23:325–342, 1992.

[13] C. Stirling. Modal and Temporal Logics for Processes. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure vs Automata*, LNCS 1043, pages 149–237, 1996. The VIIIth Banff Higher-Order Workshop. Banff, Alberta, Canada.

[14] C. Stirling and D. Walker. CCS, liveness, and local model checking in the linear time mu-calculus. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, LNCS 407, pages 166–178, 1990.

[15] 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.