Symbolic Bisimulation for Full LOTOS

Carron Shankland[1*] and Muffy Thomas[2] and Ed Brinksma[3]

[1] Department of Computing Science and Mathematics, University of Stirling
[2] Department of Computing Science, University of Glasgow
[3] Department of Computer Science, University of Twente

Abstract. A *symbolic* semantics for Full LOTOS in terms of *symbolic* transition systems is defined, following the approach taken for message passing CCS in [HL95a], altered to take account of the particular features of LOTOS (multi-way synchronisation, value negotiation, selection predicates). Symbolic bisimulation over symbolic transition systems is defined, and symbolic bisimulation on ground behaviour expressions is shown to preserve the usual concrete (strong) bisimulation on the standard semantics. Finally, a modal logic based on symbolic transition systems is defined. All are illustrated with reference to examples.

1 Introduction

Full LOTOS[4] is a message passing process algebra which combines some features of both CSP [Hoa85] and CCS [Mil89]. In order to accommodate multi-way synchronisation, i.e. associative synchronisation between two or more processes, the standard semantics of LOTOS gives meaning only to processes with *ground* data; the semantics is in terms of structured labelled transition systems. This means that query events do not correspond to a single transition, but rather a set of transitions, one for each possible ground instance of the query variable(s). For example, when $B = g?x : S; B'$ then there is a transition $B \xrightarrow{gv} B'$ for each value v of sort S (i.e. each for each equivalence class in the associated initial algebra). The implication of this semantics is that a query event offer is equivalent (with respect to strong bisimulation) to an infinite choice over all values of the data type, e.g. $in?x : Nat; P$ is equivalent to $in!0; P$ [] $in!1; P$ [] $in!2; P$ [] ...

While the advantage of this semantics is that it easily accomodates multi-way synchronisation between any number of processes (CCS only allows two-way synchronisation), it can result in infinite transition systems (both in depth and breadth) which are difficult to reason about. Moreover, by *embedding* the data values in the actions, any uniformities in the actions of the processes are lost and the semantics cannot be extended to partial specifications, i.e. open behaviour expressions; our experiences with LOTOS applications (e.g. [KT95, TO94]) confirm that this is highly desirable.

* This author was partially supported by a grant from the British Council enabling travel from Scotland to the Netherlands.

[4] Full LOTOS is Basic LOTOS plus algebraic data types. In the remainder of this paper the term LOTOS refers to Full LOTOS.

To overcome this, we define a *symbolic* semantics for LOTOS in terms of *symbolic* transition systems. This allows reasoning about data to be separated from reasoning about processes (the latter is our primary interest); we assume the existence of some oracle which will report the validity of predicates on the data. In reality this oracle will be implemented by some other proof system. Broadly, we follow the approach taken in [HL95a] for symbolic transition graphs and message passing CCS but our approach differs in several significant ways to accommodate the particular features of LOTOS (see below for details). We define symbolic bisimulation over symbolic transition systems and sketch the proof that symbolic bisimulation on ground behaviour expressions preserves the usual concrete (strong) bisimulation on the standard semantics. We also define a modal logic based on symbolic transition systems.

Throughout, the semantics, bisimulation relation and logic is illustrated by application to a telephony example.

2 Preliminaries

In this section we give some basic definitions which are used throughout the paper. Some familiarity with LOTOS [ISO88, BB89] is assumed; only a brief overview of the distinguishing features is provided.

LOTOS LOTOS has two (related) features which distinguish it from most of the standard process algebras: value negotiation, multi-way (broadcast) synchronisation and selection predicates.

Value negotiation refers to the fact that there is no simple input–output model of value passing in LOTOS; rather, an event *offers* a single value, a type of values, or a set of values drawn from a type satisfying a selection predicate. For example, a single value offer is given by $g!succ(0)$. A type of values is being offered by $g?x : Nat$, meaning, informally, that any value of the type Nat is being offered, or is acceptable as a value for x. $g?x : Nat[x > 0]$ offers only values from Nat which are greater than 0. The importance of selection predicates is that they may refer to variables which are being introduced in the current action. This differs from guards (also present in LOTOS) which may only refer to data introduced in previous actions.

! and ? offers can synchronise in any combination. For example, when u and v are ground terms, $g!u; P$ and $g!v; Q$ can synchronise iff $u \equiv v$, in the proof system associated with the data type specification.

Multi-way synchronisation means that when two actions synchronise, with possibly some data exchange taking place, the resulting action may be involved in further synchronisation. This is in contrast to, e.g. CCS, where two actions synchronise to give an unobservable τ action, which may not synchronise with any other action. So, in CCS, communication is strictly two-way and *not* associative, whereas in LOTOS synchronisation is multi-way and associative. For example,

$$(g!succ(0); P) \ |[g]| \ (g?x : Nat[odd(x)]; Q) \ |[g]| \ (g?y : Nat[y \geq 0]; R)$$

can synchronise, and is equivalent (with respect to bisimulation) to

$$g!succ(0); (P \ |[g]| \ Q[succ(0)/x] \ |[g]| \ R[succ(0)/y])$$

And this in turn can synchronise with, say,

$$g!pred(succ(succ(0))); S$$

all of which assumes an appropriate theory of Nat.

While LOTOS allows *multiple* data offers, e.g. $g!x!y?n : Nat; P$, in order to simplify the definitions concerned with transition systems and bisimulation, we will, without loss of generality, assume that only one event offer can occur at a gate/event.

Concrete Semantics and Bisimulation We refer to the standard semantics (as defined in [ISO88]) as the "concrete" semantics, and the standard strong bisimulation as "concrete" bisimulation. We write concrete bisimulation as $\sim$. (N.B. $\sim$ is defined only on closed behaviour expressions).

Variables and Substitutions We use σ to stand for substitution of data names and we write a substitution as $[z/x]$ where z is substituted for x. We assume a function **new-var** which generates fresh variable names. We call a unifier which generates new variables and uses those to unify terms a *renaming unifier*. For example, $[z/x, z/y]$ is a renaming unifier of x and y. We write the composition of two substitutions σ_1 and σ_2 as $\sigma_1\sigma_2$, where σ_2 has precedence over σ_1.

Free and Bound Variables The variables occurring in a data expression E are given by *vars(E)*. A behaviour expression may contain *free* and *bound* (data) variables; a closed behaviour expression is one with no free variables and a ground expression is one with no variables. Free variables arise in two ways: as formal process parameters, and as variables which have been introduced (and bound) earlier by a ? event. That is, ? is considered to be a binder; e.g. in $g?x; g!x; exit$, all occurrences of x are bound, but in $g!x; exit$, x is free. Following [Eer94] the free variables of a behaviour expression is defined by structural induction on terms.

Definition 1 *Let $vars(E)$ be the variables occurring in expression E. Then the free variables of behaviour expression B, fv(B), is defined by*

$$
\begin{array}{ll}
fv(stop) & = \{\} \\
fv(exit) & = \{\} \\
fv(exit(x)) & = \{x\} \\
fv(p[g]) & = \{\} \\
fv(p[g](x_0 \ldots x_n)) & = \{x_0 \ldots x_n\} \\
fv(g;\ B) & = fv(B) \\
fv(g?x{:}S[SP];\ B) & = (vars(SP) \cup fv(B)) \setminus \{x\} \\
fv(g!x[SP];\ B) & = \{x\} \cup vars(SP) \cup fv(B) \\
fv([SP] \mathbin{{-}{>}} B) & = vars(SP) \cup fv(B) \\
fv(B_1\ *\ B_2) & = fv(B_1) \cup fv(B_2), \\
 & \quad where\ * = \texttt{[]},\ \texttt{[>},\ \texttt{|[...]|} \\
fv(let\ x{=}E\ in\ B) & = vars(E) \cup (fv(B) \setminus \{x\}) \\
fv(hide\ g\ in\ B) & = fv(B) \\
fv(B_1\ \texttt{>>}\ accept\ x\ in\ B_2) & = fv(B_1) \cup (fv(B_2) \setminus \{x\})
\end{array}
$$

3 Extended Transition Systems

Following [HL95a] *Symbolic transition systems* (STS) are transition systems which separate the data from process behaviour. STS's are essentially labelled transition systems with variables, both in states and transitions, and conditions, determining the validity of a transition.

Definition 2 *Let G be set of gate/event names. A Symbolic Transition System consists of*

a (nonempty) set of states with a distinguished initial state, s_0,
a set of transitions of the form $P \xrightarrow{b \quad gx} P'$, where
- *P is the source state,*
- *b is a Boolean expression, or condition, which must hold for the transition to be valid,*
- *g is a gate, or event name, $g \in G \cup \{\mathbf{i}, \delta\}$ where* **i** *is the silent event in LOTOS, and δ is the special event produced by* **exit**,
- *x is a variable denoting data offer associated with g, $g \in G \cup \{\delta\}$,*
- *P' is the destination state.*

We give a symbolic semantics for LOTOS by associating a symbolic transition system with each LOTOS behaviour expression B, written STS(B). By an abuse of notation states are identified with their associated behaviour expression. We do not give a complete definition of the axioms and rules which define the symbolic semantics here; instead only some of the most important ones, i.e. those for action prefix, choice, guards and parallelism, are given in Figures 1 and 2. In the axioms and rules α is used to stand for gx when the particular g and x is not of interest.

Some key features (and differences from [HL95a]) of this symbolic semantics are

- both kinds of data offer, i.e. both ? and !, are represented by a transition labelled by a gate/event, a variable and a condition. This is motivated by the observation that every offer is a *set* of values constrained by a condition – an equality in the case of a ! offer and an arbitrary predicate in the case of a ? offer. As far as we are concerned, there is no distinction between ! and ? in this semantics.
- every transition introduces a new variable. This overcomes any potential variable name capture. For example, even if every ? variable in the expression is unique, it is possible that a process is invoked more than once, e.g. $P[g] \;|||\; P[g]$ where $P[g] = g?x : S; ...$. This is why transitions can be labelled gx rather than gE.
- synchronisation results in a new name being assigned to the value passed (with appropriate substitution in the subsequent processes) and conjunction of the transition conditions.
- guarding, prefix and parallelism are the only rules which alter transition conditions.
- nodes of the transition system are behaviour expressions, whereas in [HL95a] they are lists of free variables.

silent prefix axiom

$$\frac{}{i; B \xrightarrow{\mathrm{tt} \quad \mathbf{i}} B}$$

general prefix axiom

$$\frac{}{go[SP]; B \xrightarrow{SP\sigma \wedge b \quad gz} B\sigma}$$

where $z \in$ **new-var**.

$$o = \begin{cases} !E & \text{then } b = (z \equiv E) \;\; \sigma = [\,] \\ ?x : S \text{ then } b = \mathrm{tt} & \sigma = [z/x] \end{cases}$$

exit axiom

$$\frac{}{\mathrm{exit}(o) \xrightarrow{b \quad \delta z} \mathrm{stop}}$$

where $z \in$ **new-var**.

$$o = \begin{cases} E & \text{then } b = (z \equiv E) \\ \mathrm{ANY}\ \textit{sortname} & \text{then } b = \mathrm{tt} \end{cases}$$

Fig. 1. Selected Axioms for Symbolic Semantics for LOTOS

We illustrate STS's by example in the next section.

guard rule

$$\frac{B \xrightarrow{b \quad \alpha} B'}{([SP] \texttt{->} B) \xrightarrow{b \wedge SP \qquad \alpha} B'}$$

choice rules

$$\frac{B_1 \xrightarrow{b \quad \alpha} B_1'}{B_1 \; [] \; B_2 \xrightarrow{b \quad \alpha} B_1'}$$

Similarly for $B2$.

general parallelism rules

$$\frac{B_1 \xrightarrow{b_1 \quad gu} B_1' \qquad\qquad B_2 \xrightarrow{b_2 \quad gw} B_2'}{B_1 |[A]| B_2 \xrightarrow{b_1[z/u] \wedge b_2[z/w] \quad gz} B_1'[z/u] |[A]| B_2'[z/w]}$$

where $z \in$ **new-var** and $g \in A \cup \{\delta\}$

$$\frac{B_1 \xrightarrow{b \quad gx} B_1'}{B_1 |[A]| B_2 \xrightarrow{b \quad gx} B_1' |[A]| B_2}$$

$g \notin A \cup \{\delta\}$
Similarly for B_2.

instantiation rule

$$\frac{B' \xrightarrow{b \quad \alpha} B''}{p[g_1, \ldots, g_n](t_1, \ldots, t_m) \xrightarrow{b \quad \alpha} B''}$$

where $p[h_1, \ldots, h_n](x_1, \ldots, x_m) := B$ is a process definition,
$B' =$ relabel $[g_1/h_1, \ldots, g_n/h_n]$ in $B[t_1/x_1, \ldots, t_m/x_m]$

Fig. 2. Selected Inference Rules for Symbolic Semantics for LOTOS

3.1 LOTOS Examples

Consider two specifications of user behaviour in a telephone network where users are forbidden to make and receive calls to/from particular users. The two specifications are given in Figures 3 and 5, and their respective STS's in Figures 4 and 6.

Each user process is parameterised by

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

There are 5 possible events: the `con` (*connect*) and `discon` (*disconnect*) events, the `dial` (*dial*), `unobt` (*unobtainable*) and `on` (*on hook*) events. The first three events include data offers, for example, **discon!x!y** denotes the event of disconnecting the call from user x to user y. 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. Also, we do not allow that phones are engaged, or unobtainable for reasons other than being in the `out` list.

The difference between `Tel_I` and `Tel_II` is essentially the points at which choices are made, rather than the criteria involved in those choices.

```
process Tel_I[dial,con,discon,unobt,on]
       (id:userid,bar_in:idlist,bar_out:idlist) :exit :=

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

Fig. 3. LOTOS Description of Telephone I

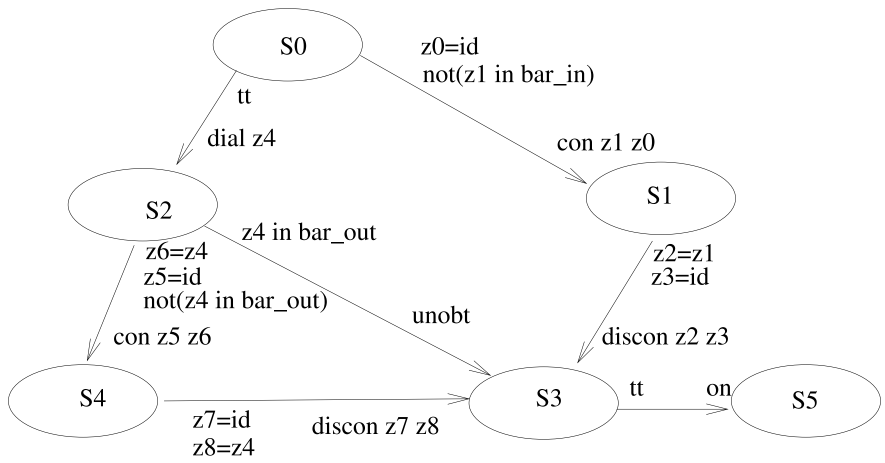

Fig. 4. STS for Telephone I

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

con?x:userid!id [not (x in bar_in)]; discon!x!id; on; exit
[]
dial?x:userid [x in bar_out]; unobt; on; exit
[]
dial?x:userid [not(x in bar_out)]; con!id!x; discon!id!x; on; exit
endproc
```

Fig. 5. LOTOS Description of Telephone II

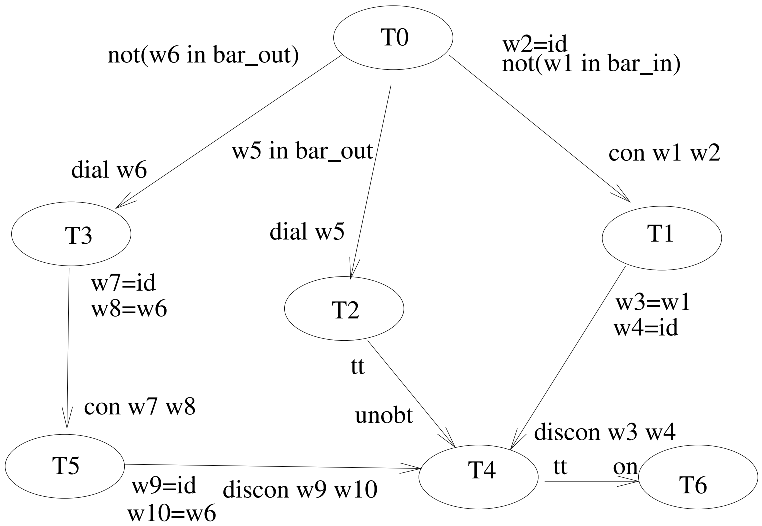

Fig. 6. STS for Telephone II

4 Bisimulation

The notion of bisimulation has proven to be useful in process algebra, to simplify specifications, and to show two specifications equivalent. We desire a similar notion for symbolic transition systems.

4.1 Symbolic Bismimulations on STS

As with symbolic transition systems, our main motivation here is to retain the separation between data and processes. When considering the equivalence of processes we must be very careful; obviously the particular value of a data variable can completely alter the behaviour of a process. Therefore, we must not discard this information.

The crux of the following definition of bisimulation is the notion that data can be partitioned according to some Boolean expressions, or predicates, e.g. $\{x < 0, x \geq 0\}$ and this may give enough information to accurately simulate a process, without assigning a particular value to the data variables. This also means that each bisimulation is a parameterised family of relations, where the parameters are the Boolean expressions. Furthermore, we only consider simulating transitions which could possibly be valid. Namely, given a particular Boolean expression, or "context" b, and a transition with condition b', we do not consider that transition at all if the context and condition are mutually inconsistent. For example, if b is $x = 0$, and b' is $x \neq 0$, then the transition can never be valid in this context, and so we do not need to consider the transition in the simulation. Hennessy and Lin do not do this; technically it is not necessary, but operationally it is desirable. The other main difference here is that we do not distinguish !, ? and neutral actions, while they must.

Definition 3 *(Symbolic Bisimulations)*

Let $\mathbf{S} = \{S^b | b \in BExp\}$ be a parameterised family of relations over terms. Then $\mathcal{SLB}(\mathbf{S})$ is the BExp-indexed family of symmetric relations defined by:

$(t_1, t_2) \in \mathcal{SLB}(\mathbf{S}^b)$ *if*

1. *whenever $t_1 \xrightarrow{b_1 \;\; gx_1} t_1'$ and $b \wedge b_1\sigma$*
   $\exists B$ *(a collection of Boolean expressions) such that $b \wedge b_1\sigma \Rightarrow \bigvee B$.*
   for each $b' \in B$ there exists a $t_2 \xrightarrow{b_2 \;\; gx_2} t_2'$ such that
   $b' \Rightarrow b_2\sigma$
   $(t_1'\sigma, t_2'\sigma) \in S^{b'}$
   where σ is a renaming unifier of x_1 and x_2.
2. *whenever $t_2 \xrightarrow{b_2 \;\; gx_2} t_2'$ and $b \wedge b_2\sigma$*
   $\exists B$ *(a collection of Boolean expressions) such that $b \wedge b_2\sigma \Rightarrow \bigvee B$.*
   for each $b' \in B$ there exists a $t_1 \xrightarrow{b_1 \;\; gx_1} t_1'$ such that
   $b' \Rightarrow b_1\sigma$
   $(t_2'\sigma, t_1'\sigma) \in S^{b'}$
   where σ is a renaming unifier of x_1 and x_2.

We use $\sim^b$ to denote the largest symbolic bisimulation, for a given b.

4.2 Example

Theorem 1 `Tel_I` *and* `Tel_II` *are symbolically bisimular under the trivial condition,* tt*; i.e.* `Tel_I` $\sim^{\mathrm{tt}}$ `Tel_II`.

Proof

There is a bisimulation consisting of the following sets (assuming the symmetric pairs in each set):

$$
\begin{array}{ll}
S^{\mathrm{tt}} & = \{\textit{(S0, T0)}\} \\
S^{u6\equiv id \wedge not(u7\ in\ bar\_in)} & = \{\textit{(S1, T1)}\} \\
S^{u6\equiv id \wedge not(u7\ in\ bar\_in)\wedge u8\equiv u6\wedge u9\equiv id} & = \{\textit{(S3, T4), (S5, T6)}\} \\
S^{u1\ in\ bar\_out} & = \{\textit{(S2, T2), (S3, T4), (S5, T6)}\} \\
S^{not(u1\ in\ bar\_out)} & = \{\textit{(S2, T3)}\} \\
S^{not(u1\ in\ bar\_out)\wedge u2\equiv id\wedge u3\equiv u1} & = \{\textit{(S4, T5)}\} \\
S^{not(u1\ in\ bar\_out)\wedge u2\equiv id\wedge u3\equiv u1\wedge u4\equiv id\wedge u5\equiv u1} & = \{\textit{(S3, T4), (S5, T6)}\}
\end{array}
$$

where $\sigma = [u1/z4, u1/w5, u1/w6, u2/z5, u2/w7, u3/z6, u3/w8, u4/z7, u4/w9,$
$u5/z8, u5/w10, u6/z0, u6/w2, u7/z1, u7/1, u8/z2, u8/w3, u9/z3, u9/w4]$

The proof relies on the partition induced by $\{$*u1 in bar_out, not (u1 in bar_out)*$\}$.

4.3 Relating Symbolic and Concrete Bisimulations

In [HL95a], concrete bisimulations are defined in terms of the symbolic transition systems, with concrete output events relating to symbolic output events, and concrete input events relating to symbolic input events. So, the relationship between the two semantics is reasonably straightforward.

Here, there is no distinction between input and output; in the concrete semantics the origins of the data offers as a ! or ? offers have been lost. Moreover, we have been given a concrete semantics over ground terms in which transitions are labelled by ground terms (strictly speaking, the equivalence classes they denote). In the symbolic semantics, we have (potentially) open terms labelling states, and transitions (possibly) offering a single variable, under some conditions; these conditions determine the set of values which may be substituted for the variable. To illustrate the standard concrete semantics and the symbolic semantics defined here, Figures 7 and 8 contain portions of the respective transition systems for the behaviour expression $g?x : Nat[x < 10]; h?y : Nat; h!x; stop$. Note that whereas the concrete system has infinite branching, the symbolic system has finite branching.

The relationship between concrete and symbolic events is given by the following. The proofs, by induction on the structure of derivations, are omitted due to lack of space.

Lemma 1 *Relating Concrete Events to Symbolic Events.*
For all ground behaviour expressions t, t' *events* g *and values* v*.*
$t \xrightarrow{gv} t' \Rightarrow \exists z, SP, u.u[v/z] = t' \wedge SP[v/z] \wedge t \xrightarrow{SP\ \ gz} u.$

Lemma 2 *Relating Symbolic Events to Concrete Events.*
For all behaviour expressions t, t' *events* g*, variable* z*, and Boolean expressions* SP*.*
$t \xrightarrow{SP\ \ gz} t' \Rightarrow \forall\ v,$ *closed substitutions* σ *s.t. domain of* σ *is* $fv(t)$*.*
$(SP\sigma[v/z]) \Rightarrow t\sigma \xrightarrow{gv} t'\sigma[v/z].$

Now these relationships can be used to show the main result: that symbolic bisimulation preserves concrete bisimulation, for closed processes. The proof is by induction on the structure of terms, and is omitted here.

Theorem 2 *For all closed processes t, u and Boolean expression b.*
$b \wedge (t \sim u) \Rightarrow t \sim^b u$.

5 A Modal Logic for LOTOS

Based on the above symbolic transition systems we now define a logic which captures the notion of symbolic bisimulation, i.e. one which is adequate with respect to symbolic bisimulation. Our ultimate aim is to give a modal *mu*-calculus in which to express properties of LOTOS; as a first step a modified version of HML [HM85] is presented.

The syntax of our logic is based on that presented in [HL95b]. Note that we allow lists of event offers in the quantifiers of the logic (rather than restricting to single offers as we have done, for simplicity, in the semantics).

Let G be a set of gate/event names. Then a logical formula is either a Φ or a Ψ, where

$$\Phi ::= B \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [a]\Phi \mid \langle a \rangle \Phi \mid [g]\Psi \mid \langle g \rangle \Psi$$

$$\Psi ::= \exists \overline{x}.\Phi \mid \forall \overline{x}.\Phi$$

where $a \in G \cup \{\mathbf{i}, \delta\}$, $g \in G \cup \{\delta\}$ and $\overline{x}$ denotes a list of variable names.

The modal operators act as usual: $[a]\Phi$ means that Φ holds after all a transitions, $\langle a \rangle \Phi$ means that Φ holds after some (at least one) a transition. The quantifiers are added to allow us to express properties of data in the logic.

We define $S \models_C \Phi$, denoting when a state S in an STS satisfies a modal formula Φ in a context C. The relation is defined inductively over the syntax of the logic by the equations of Figure 9. These equations are mostly straightforward. For formulae involving $\langle\ \rangle$ *one* transition has to be found which will lead to a state which satisfies the rest of the formula, whereas for formulae involving $[\ ]$ *all* states reached by appropriately labelled transitions must satisfy the remainder of the formula. For formulae involving $\exists$ a set of values (possibly a singleton set) to use in evaluating the rest of the expression is chosen using $P(\overline{z})$, and for formulae involving $\forall$ the variables are simply renamed (so that the same names are used in the process and in the logic).

We also have to consider the special case where the original formula has the form Ψ. This corresponds to cases where the STS contains free variables (i.e. it denotes a parameterised process or a partial specification) and the formula starts with a quantifier. In this event, we must explicitly relate the quantified variables of Ψ to the free variables of the behaviour expression (labelling the state). This essentially involves unifying the two sets of variables, and, in the case of existential quantification, we can also add an additional condition to the relation. This is defined in Figure 10.

All reasoning takes place with respect to a context C, usually initially tt. This context is the conjunction of the conditions generated by the transition system, by the logic, and possibly by the prover (usually in anticipation of some later condition in the transitions or the logic). When the end of a proof is reached we

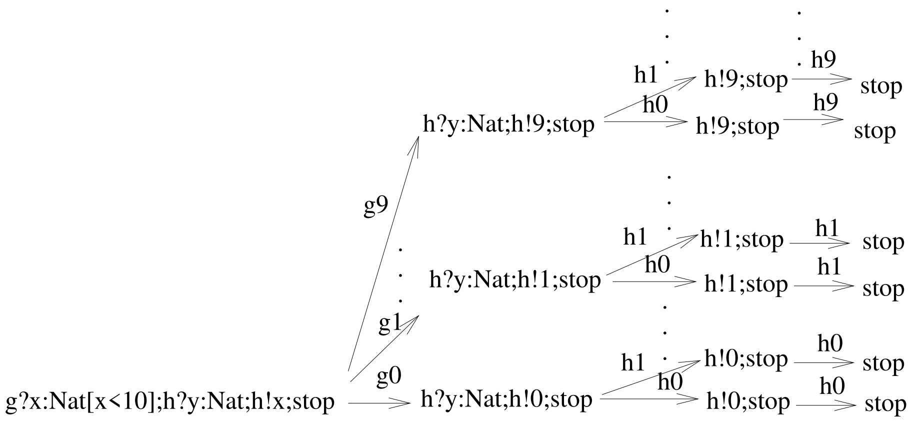

Fig. 7. Concrete Transition System

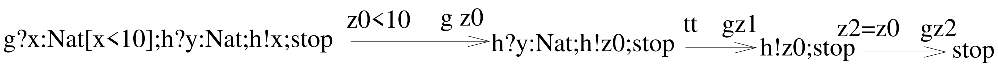

Fig. 8. Symbolic Transition System

$$
\begin{array}{lll}
S \models_C B & = & B \wedge C \\
S \models_C \Phi_1 \wedge \Phi_2 & = & S \models_C \Phi_1 \text{ and } S \models_C \Phi_2 \\
S \models_C \Phi_1 \vee \Phi_2 & = & S \models_C \Phi_1 \text{ or } S \models_C \Phi_2 \\
S \models_C \langle a \rangle \Phi & = & \exists S'.S \xrightarrow{b \quad a} S' \ \wedge \ (C \wedge b) \ \wedge \ S' \models_{C \wedge b} \Phi \\
S \models_C \langle g \rangle \exists \overline{x} \Phi & = & \exists \overline{z}.P(\overline{z}).\exists S'.S \xrightarrow{b \quad g\overline{y}} S' \ \wedge \ (C \wedge b[\overline{z}/\overline{y}] \wedge P(\overline{z})) \\
 & & \text{and } S'[\overline{z}/\overline{y}] \models_{C \wedge b[\overline{z}/\overline{y}] \wedge P(\overline{z})} \Phi[\overline{z}/\overline{x}] \\
S \models_C \langle g \rangle \forall \overline{x} \Phi & = & \forall \overline{z}.\exists S'.S \xrightarrow{b \quad g\overline{y}} S' \ \wedge \ (C \wedge b[\overline{z}/\overline{y}]) \\
 & & \text{and } S'[\overline{z}/\overline{y}] \models_{C \wedge b[\overline{z}/\overline{y}]} \Phi[\overline{z}/\overline{x}] \\
S \models_C [a] \Phi & = & \forall S'.(S \xrightarrow{b \quad a} S' \ \wedge \ (C \wedge b)) \Rightarrow S' \models_{C \wedge b} \Phi \\
S \models_C [g] \exists \overline{x} \Phi & = & \exists \overline{z}.P(\overline{z}).\forall S'.(S \xrightarrow{b \quad g\overline{y}} S' \ \wedge \ (C \wedge b[\overline{z}/\overline{y}] \wedge P(\overline{z}))) \\
 & & \Rightarrow S'[\overline{z}/\overline{y}] \models_{C \wedge b[\overline{z}/\overline{y}] \wedge P(\overline{z})} \Phi[\overline{z}/\overline{x}] \\
S \models_C [g] \forall \overline{x} \Phi & = & \forall \overline{z}.\forall S'.(S \xrightarrow{b \quad g\overline{y}} S' \ \wedge \ (C \wedge b[\overline{z}/\overline{y}] \wedge P)\overline{z}))) \\
 & & \Rightarrow S'[\overline{z}/\overline{y}] \models_{C \wedge b[\overline{z}/\overline{y}]} \Phi[\overline{z}/\overline{x}]
\end{array}
$$

Fig. 9. Relating the Modal Formulae to Symbolic Transition Systems

$$
\begin{array}{l}
\langle fv(s_0), s_0 \rangle \models_C \exists \overline{x} \Phi = \exists \overline{z}.(C[\overline{z}/fv(s_0] \wedge P(\overline{z})) \wedge s_0[\overline{z}/fv(s_0)] \models_{C[\overline{z}/fv(s_0)] \wedge P(\overline{z})} \Phi[\overline{z}/\overline{x}] \\
\langle fv(s_0), s_0 \rangle \models_C \forall \overline{x} \Phi = \forall \overline{z}.s_0[\overline{z}/fv(s_0)] \models_{C[\overline{z}/fv(s_0)]} \Phi[\overline{z}/\overline{x}]
\end{array}
$$

Fig. 10. Relating the Modal Formulae to Symbolic Transition Systems

look only for consistency in these conditions. This allows the flexibility to have a loose logical specification, or a loose process specification, and to generate the conditions under which a formula will hold during the construction of the proof.

Examples A number of properties can be defined which can be shown to hold of both the Telephony examples. We would expect this, because they are symbolically bisimilar. We omit the proofs here, due to lack of space.

$\forall id, bar\_in, bar\_out.\langle dial\rangle \exists v.not(v\ in\ bar\_out)$.

It is possible to dial a number which is not in the barred out list.

$\forall id, bar\_in, bar\_out.\langle dial\rangle \exists v.\langle unobt\rangle \mathrm{tt}$.

After dialling a number, the user might get unobtainable. Note that, in the proof, we must generate the condition under which this property holds, namely $v\ in\ bar\_out$. Alternatively, we could have specified this in the logic by $\forall id, bar\_in, bar\_out.\langle dial\rangle \exists v.(v\ in\ bar\_out) \wedge \langle unobt\rangle \mathrm{tt}$.

$\forall id, bar\_in, bar\_out.\langle dial\rangle \exists v_1.(v_1 = 999) \wedge \langle con\rangle \exists v_2, v_3.(v_3 = 999)$.

After dialling 999 the user will be connected to 999. The proof is made in the initial context that $not(999\ in\ bar\_out)$ (rather than the usual tt).

Note that this property only states that it is *possible* to be connected. A stronger statement would be $\forall id, bar\_in, bar\_out.$

$\langle dial\rangle \exists v_1.(v_1 = 999) \wedge [dial]\exists v_1.((v_1 = 999) \wedge \langle con\rangle \exists v_2, v_3.(v_3 = 999))$

$\forall id, bar\_in, bar\_out.\langle dial\rangle \forall v.not(v\ in\ bar\_out) \vee (\langle unobt\rangle \mathrm{tt} \wedge [-\mathrm{unobt}]\mathrm{ff})$.

After dialling a number in the barred list the user gets unobtainable (and no other action is possible). Here we use the shorthand $-unobt$ to mean "all actions except unobt".

Sometimes it is quite difficult to get the right formulation of these properties. For example, an alternative way to express this property is

$\forall id, bar\_in, bar\_out.\langle dial\rangle \exists v.(v\ in\ bar\_out) \wedge (\langle unobt\rangle \mathrm{tt} \wedge [-\mathrm{unobt}]\mathrm{ff}$.

In this case, the proof relies on choosing the correct partition, whereas the proof of the first version does not. Clearly they say similar, but different, things about the systems; the version using $\forall$ is stronger because it says there is no v (in *bar_out*) which doesn't do *unobt*, whereas the other version merely says that there is at least one v in *bar_out* which does *unobt*.

This alternative formulation can also be applied to the next two properties.

$\forall id, bar\_in, bar\_out.[dial]\forall v.not(v\ in\ bar\_out) \vee [con]\mathrm{ff}$.

After dialling a number in the barred list, connection is not possible. Note here that we choose to ignore the data values associated with the connection event. In this case, even if we expressed anything about them the whole boolean expression would evaluate to ff. On the other hand, it might be important to us to match data in the process with data in the logic, i.e. if the process only output one value then the above formula would still hold, but we might need to know that two values were output, even if we don't care about their values in this instance.

$\forall id, bar\_in, bar\_out.[dial]\forall v_1.(v_1\ in\ bar\_out) \vee [con]\forall v_2, v_3.(v_1 = v_3 \wedge v_2 = id)$.

After dialling a number not in the barred list, connection is possible (with the appropriate data values). Unlike the previous example, here, if we omitted

v_2 or v_3 we would merely be expressing that connection was possible (not necessarily between the desired parties).

$\forall id, bar\_in, bar\_out.[dial]\forall v.[-](\langle on\rangle \mathrm{tt} \vee [-]\langle \mathrm{on}\rangle \mathrm{tt})$.

After dialling, the second or third event will be an on hook event. Here we use $-$ as a shorthand for "all actions".

Variants of the Logic Interesting variants of the logic may also be developed. For example, we can obtain a different version of the logic by allowing that the data partition takes place after the choice of transition. This gives a more elegant version of the semantics; see Figure 11.

$$
\begin{array}{ll}
S \models_C \langle a\rangle\Phi & = \exists S'.S \xrightarrow{b \quad a} S' \ \wedge \ (C \wedge b) \ \wedge \ S' \models_{C\wedge b} \Phi \\
S \models_C \langle g\rangle\Psi & = \exists S'.S \xrightarrow{b \quad g\overline{y}} S' \ \wedge \ (C \wedge b) \wedge (\overline{y}, S') \models_{C\wedge b} \Psi \\
S \models_C [a]\Phi & = \forall S'.(S \xrightarrow{b \quad a} S' \ \wedge \ (C \wedge b)) \Rightarrow S' \models_{C\wedge b} \Phi \\
S \models_C [g]\Psi & = \forall S'.(S \xrightarrow{b \quad g\overline{y}} S' \ \wedge \ (C \wedge b)) \Rightarrow (\overline{y}, S') \models_{C\wedge b} \Psi \\
(\overline{y}, S) \models_C \exists\overline{x}\Phi & = \exists\overline{z}.P(\overline{z}) \wedge \ S[\overline{z}/\overline{y}] \models_{C[\overline{z}/\overline{y}]\wedge P(\overline{z})} \Phi[\overline{z}/\overline{x}] \\
(\overline{y}, S) \models_C \forall\overline{x}\Phi & = \forall\overline{z}.S[\overline{z}/\overline{y}] \models_{C[\overline{z}/\overline{y}]} \Phi[\overline{z}/\overline{x}]
\end{array}
$$

Fig. 11. Alternative Formulation of Logic Semantics

This means the logic is no longer adequate with respect to symbolic bisimulation, and we can therefore give properties which distinguish `Tel_I` and `Tel_II`. For example, `Tel_I` would satisfy $\forall id, bar\_in, bar\_out.[dial]\exists v.(v \ in \ bar\_out)$, while `Tel_II` would not. We observe that the change to the logic amounts to losing the difference between $[a]\exists v$ and $[a]\forall v$.

6 Conclusions and Further Work

We have defined a *symbolic* semantics for full LOTOS in terms of *symbolic* transition systems, symbolic bisimulation over those transition systems, and a symbolic modal logic. Broadly speaking, we have adopted the approach of [HL95a]; however, the features of LOTOS (especially the need to accomodate multi-way synchronisation, and the resulting model of value passing) mean that this is not a straightforward adaptation of the theory presented in [HL95a] and [HL95b].

The use of a symbolic semantics, relation, logic and proof system will allow us to reason about Full LOTOS processes, separating the data from the processes, but without losing essential information that the data supplies in terms of flow of control. Previous approaches to reasoning about Full LOTOS processes meant using considerable intuition about different representations of data, and data as processes [Got87] or using complex transformations [Bri92, T.92], only some of which preserve the data information.

There are two main streams of further work: logic-related and case studies. For the logic, we hypothesise that the logic given above is adequate with respect to symbolic bisimulation on finite processes, but have not yet proved this. We will also extend the logic with fixpoint operators, to give a variant of modal mu-calculus; this will give a more expressive language.

Finally, in order to evaluate the effectiveness of our approach we need to carry out extensive case studies in a wide range of application areas.

References

[BB89] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. In P.H.J. van Eijk, C.A. Vissers, and M. Diaz, editors, *The Formal Description Technique LOTOS*, pages 23–76. Elsevier Science Publishers B.V. (North-Holland), 1989.

[T.92] T. Bolognesi, editor. Catalogue of LOTOS Correctness Preserving Transformations. Technical Report Lo/WP1/T1.2/N0045, The LOTOSPHERE Esprit Project, 1992. Task 1.2 deliverable. LOTOSPHERE information disseminated by J. Lagemaat, email `lagemaat@cs.utwente.nl`.

[Bri92] E. Brinksma. From Data Structure to Process Structure. In K.G. Larsen and A. Skou, editors, *Proceedings of CAV 91*, LNCS 575, pages 244–254, 1992.

[Eer94] H. Eertink. *Simulation Techniques for the Validation of LOTOS Specifications.* PhD thesis, University of Twente, 1994.

[Got87] R. Gotzhein. Specifying Abstract Data Types with LOTOS. In B. Sarikaya and G.V. Bochmann, editors, *Protocol Specification, Testing, and Verification, VI*, pages 15–26. Elsevier Science Publishers B.V. (North-Holland), 1987.

[HL95a] M. Hennessy and H. Lin. Symbolic Bisimulations. *Theoretical Computer Science*, 138:353–389, 1995.

[HL95b] M. Hennessy and X. Liu. A Modal Logic for Message Passing Processes. *Acta Informatica*, 32:375–393, 1995.

[HM85] M. Hennessy and R. Milner. Algebraic Laws for Nondeterminism and Concurrency. *Journal of the Association for Computing Machinery*, 32(1):137–161, 1985.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, 1985.

[ISO88] International Organisation for Standardisation. *Information Processing Systems — Open Systems Interconnection — LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, 1988.

[KT95] 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.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall International, 1989.

[TO94] M. Thomas and T. Ormsby. On the Design of Side-Stick Controllers in Fly-by-Wire Aircraft. In *Applied Computing Review, Special Issue: Safety-Critical Software*, pages 15–20. ACM Press, Volume 2, Number 1 Spring 1994.

This article was processed using the LaTeX macro package with LLNCS style