A SYMBOLIC SEMANTICS AND BISIMULATION FOR FULL LOTOS

Muffy Calder
Department of Computing Science,
University of Glasgow, Glasgow G12 8QQ
muffy@dcs.gla.ac.uk

Carron Shankland
Department of Computing Science and Mathematics,
University of Stirling, Stirling FK9 4LA
carron@cs.stir.ac.uk

Abstract

A *symbolic* semantics for Full LOTOS in terms of *symbolic transition systems* is defined; the semantics extends the (infinitely branching) standard semantics by giving meaning to data parameterised behaviours, and provides a finitely branching representation for behaviours. Symbolic bisimulation is defined.

This extends our previous work [14], making the definitions more amenable to automated reasoning and processes with recursion.

Keywords: LOTOS, symbolic transition systems, symbolic bisimulation.

1. Introduction

LOTOS [11] is a message passing process algebra which combines two orthogonal languages: a process language, known as Basic LOTOS, with features from both CSP and CCS, and the equational abstract data type language ACT ONE. LOTOS is an ISO standard [11] whose given semantics is in terms of structured labelled transition systems. In this semantics (referred to here as the *standard semantics*), each data variable in a process is instantiated by every possible value of its corresponding type, resulting in infinite transition systems (both in breadth and in depth).

This approach has several drawbacks. First, it is impossible to use standard (finite state) model-checking techniques over infinite transition systems; the usual solution is to restrict the underlying datatypes. Indeed, any kind of automated reasoning becomes difficult, if not impossible. Second, because the data values are embedded in the transitions, any uniformities in the actions of the processes are lost. For example, the process description may make it clear that a particular action happens when the value of some variable lies between 3 and 42 (say), but that information is much harder to extract from the labelled transition system directly, especially if there are an infinite number of branches. Finally, as a consequence of this approach it is not possible to reason about partial, or data parameterised, behaviour expressions. Our experiences with LOTOS applications (e.g. [15, 16]) indicate that this is highly desirable.

The advantage of the standard approach is that it easily accommodates multi-way synchronisation, i.e. associative synchronisation between two or more processes. Multi-way synchronisation has led to a particular constraint-oriented style of specification in the LOTOS community which is particularly useful when building a system by layering behaviour on a simple building block.

The problem we address here and in [14] is how to reason over potentially infinite LOTOS processes while retaining multi-way synchronisation and without restricting the datatypes. Since the addition of data to the language is the reason for the problem, some sort of separation of the concerns of data and processes seems appropriate. There are three kinds of solution to this problem. The first is to get rid of data altogether in a brute force manner [1] which changes the behaviour of the process. The second is to construct a process representation of the data type [2, 9]. This approach only converts data operations into process operations; the data values are still present therefore the process remains infinite branching. The third solution is to adopt a symbolic approach such as the symbolic semantics for message passing CCS in [10]. We drew on this approach for our earlier work [14] in defining a symbolic semantics and bisimulation for LOTOS. That definition did not deal effectively with recursive processes, and could not be easily implemented (since it required an infinite stock of new names). These problems are addressed here and the definitions considerably revised. The symbolic approach is also taken by Eertink [8]. While Eertink's semantics achieves a separation of the concerns of data and process, without losing information, it is rather operational, concentrating on using the semantics in a simulation tool. There are no equivalences or preorder relations associated with the semantics.

Our motivation is to define an entire symbolic framework for reasoning about LOTOS specifications including a symbolic semantics, appropriate logics, relations and tools. It is important that this new framework preserves the distinguishing features of LOTOS (presented in Section 2). In this paper we present the foundation of the framework: symbolic transition systems (STSs) and the axioms and rules for generating an STS from a (possibly open) LOTOS behaviour expression (Section 3). In order to define relations over STSs we must first define the notion of substitution over STSs (Section 4). We follow this with the definition of symbolic bisimulation (Section 5). Symbolic bisimulation should not lead to processes being distinguished which are not in the standard semantics; similarly processes which are distinguished under the standard semantics are not identified in the symbolic semantics. We have proven this to hold for unparameterised processes, but the proof is the subject of a different paper. Broadly, we follow the approach taken in [10] for symbolic transition graphs and message passing CCS, but our approach differs in several significant ways to accommodate the particular features of LOTOS. Finally, we draw our conclusions and discuss future directions.

2. Distinguishing Features of LOTOS

We assume the reader is familiar with the standard LOTOS syntax and semantics [11]. An accessible tutorial to LOTOS is [12]. Here we present three related features distinguishing LOTOS from most of the standard process algebras, particularly message passing CCS: *multi-way (broadcast) synchronisation*, *value negotiation*, and *selection predicates*. These make it non-trivial to directly apply the notion of symbolic transition and bisimulation of [10].

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. As a result of multi-way synchronisation, it makes less sense in LOTOS to refer to"input" events and "output" events. In LOTOS an event *offers* a single value or a set of values drawn from a particular sort; these two cases are distinguished by the use of ! or ? respectively.

Multiway synchronisation is achieved in the underlying transition system by *encoding* data into transitions in both *!* and *?* events. This can be seen clearly by referring to the rules from the standard [11] for generating a transition system from action prefix events. The rule for *!* events is straightforward:

$$g!E;\ P \xrightarrow{gv} P$$

where E is a data expression (with no variables in the case of the standard LOTOS semantics) and $v = [E]$ (i.e. the equivalence class of E). Perhaps less obvious is the rule for ? events:

$$g?x:S;\ P \xrightarrow{gv} P[v/x]$$

where v is a ground term of sort S, and $P[v/x]$ denotes the substitution of v for x in P. This rule gives us an axiom *schema* for each $?x$ event. For example, see Fig. 1; each possible value of Nat results in a transition.

Thus, ? event offers correspond to a (possibly infinite) choice over all values of the data type. The binding of x is defined at this point, i.e. the semantics is *early*. A *late* semantics (where binding of variables to values is delayed as long as possible) has no counterpart in the (standard) concrete semantics. This is in constrast to value-passing CCS where both kinds of semantics are possible.

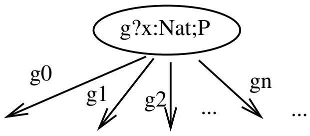

Figure 1. Standard semantics of *g?x:Nat* event offer

Encoding of values in the transitions affects the rules for synchronised parallelism. Consider a rule for CCS style two-way synchronisation. We use LOTOS syntax here for comparison:

$$\frac{g!E;\ P_1 \xrightarrow{gv} P_1 \qquad g?x:S;\ P_2 \xrightarrow{gx} P_2}{g!E;\ P_1|[g]|\,g?x:S;\ P_2 \xrightarrow{\mathbf{i}} P_1|[g]|\,P_2[v/x]}$$

There is a single transition associated with the $?$ offer (labelled with x) and a clear indication that value passing is occurring: x gets bound to the value v. However, this approach is clearly limited to two-way synchronisation because the transition label becomes the unobservable **i** action which may not synchronise with any other action.

Contrast this with the LOTOS approach to synchronisation:

$$\frac{g!E;\ P_1 \xrightarrow{gv} P_1 \qquad g?x:S;\ P_2 \xrightarrow{gv} P_2}{g!E;\ P_1|[g]|\,g?x:S;\ P_2 \xrightarrow{gv} P_1\,|[g]|\,P_2}$$

Here, one of the transitions generated by the axiom schema for $?$ offers is chosen to match the transition generated by the ! offer (where $v \equiv [E]$). That is, there may be lots of other potential transitions from the state $g?x : S;\ P_2$ labelled with g and some value, but only one which is labelled with the exact value v. In fact, this is only one case of the rule for parallelism. In particular, the processes in the premise may be a further parallel combination, yielding multi-way synchronisation. Different combinations of offers are also permitted, i.e. ! and $?$ offers can synchronise in any combination. The most unusual case is *value negotiation*, which arises when $g?x : S$ and $g?y : S$ synchronise, the result being that for every possible value, x and y are bound to the same value thereafter. Both values must have the same type.

Selection predicates may restrict data further. For example, transitions where $x \geq 42$ are denied by $g?x : S[x < 42]$. If the selection predicate evaluates to *false*, the event itself is prevented from occurring. In the case of value negotiation if both $?$ offers are qualified by selection predicates, then the value must satisfy both predicates. An important distinction from other process algebras is that selection predicates can refer to data in the current event (typically the guards found in other process algebras refer to data from previous events).

So, to preserve multi-way synchronisation and selection predicates in LOTOS, we cannot simply employ the CCS approach to ? offers. We therefore define a new semantics based on *symbolic* transition systems.

3. Symbolic Transition Systems for LOTOS

3.1. Preliminaries

We assume a countable set of *variables*, *Var*, ranged over by *x*, *y*, etc., and a (possibly infinite) set of *values*, *Val*, ranged over by v. We also assume a set of *data expressions*, *Exp*, which includes *Var* and *Val* and is ranged over by E, and a set of *Boolean expressions*, *BoolExp*, ranged over by b, including the constants $\mathtt{tt}$ (true) and $\mathtt{ff}$ (false). We also assume that we have a set of gates, G, ranged over by g. The set of events, denoted Act, ranged over by α, comprises *SimpleEv* and *StructEv*. The set of simple events, *SimpleEv*, ranged over by a, is defined as $G \cup \{\mathbf{i},\ \delta\}$. (Recall that in LOTOS **i** is the internal event and δ is the exit event.) The set of structured events, *StructEv*, contains all gate-expression combinations gE, as well as all combinations δE.

We assume the existence of the *flattening* function of the standard semantics [11]. The flattening function ensures that the given specification adheres to the LOTOS syntax, but also removes all hierarchical structure, ensures uniqueness of variable names, and that all names and types used are previously defined. The resulting object is called a *canonical LOTOS specification*.

We follow several naming conventions of the standard, including the use of a variable name to stand for a more complex structure including information not just about the name of a variable, but also its type and scope.

Variables and Substitutions. Variables and substitutions are over *data*, and *typed*, although we do not make this explicit, as noted above.

We assume a set **new-var** of fresh variable names. Strictly speaking, any reference to this set requires a context, i.e. the variable names occuring so far. For simplicity, we will assume that this context can be inferred, as required.

A *substitution* is a partial function from *Var* to *Var* ∪ *Val*, written as $[z/x]$ where z is substituted for x. A substitution is denoted σ, and the composition of two substitutions σ_1 and σ_2 is denoted $\sigma_1\sigma_2$, where σ_2 has precedence.

Structured Events. *Multiple* data offers at a gate, e.g. $g!x!y?n$: Nat; P, are allowed by LOTOS syntax. For simplicity in the following we will assume that only one event offer can occur at a gate. The obvious generalisation to lists of event offers can be easily made. The function **name**() : Act $\cup$ $\{\delta, \mathbf{i}\} \rightarrow G \cup \{\delta, \mathbf{i}\}$ extracts the gate name from a structured event and is defined in [11].

Free and Bound Variables. The variables occurring in a data expression E are given by $\mathit{vars}(E)$. A behaviour expression may contain *free* and *bound* data variables. The free variables of behaviour expressions, denoted $\mathit{fv}(P)$, are defined in Definition 5 of the Appendix.

Informally, free variables arise through usage in an expression where the variable name has been previously bound in one of several ways: as a formal process parameter, by a $?$ event, a let clause, or an enable ($>>$) with *accept* clause. For example, in $g?x;\ g!x;\ exit$, all occurrences of x are bound, but in $g!x;\ exit$, x is free.

3.2. Symbolic Transition Systems

Following [10], *symbolic transition systems* (STS) are transition systems separating data from process behaviour by making the data symbolic. We define an STS to be a labelled transition system with variables, both in states and transitions, and conditions which determine the validity of a transition.

Definition 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 $\mathit{fv}(T)$.
- *A distinguished initial state, T_0.*
- *A set of transitions $T \xrightarrow{b \quad \alpha} T'$ such that $\mathit{fv}(T') \subseteq \mathit{fv}(T) \cup \mathit{fv}(\alpha)$ and $\mathit{fv}(b) \subseteq \mathit{fv}(T) \cup \mathit{fv}(\alpha)$ and $\#(\mathit{fv}(\alpha) - \mathit{fv}(T)) \leq 1$.*

Following convention, we shall often identify an STS with its initial state. Since one possible interpretation of states is to view them as labelled by behaviour expressions, the set of free variables of an STS T, $\mathit{fv}(T)$, can be defined as $\mathit{fv}(P)$, where P is the behaviour expression labelling T.

3.3. Intuition

We give a symbolic semantics for LOTOS by associating a symbolic transition system with each LOTOS behaviour expression P, written STS(P). Before giving the axioms and rules for the symbolic semantics, we give an example illustrating the concrete (Fig. 2) and symbolic (Fig. 3) semantics for the behaviour expression

$$g?x:\mathrm{Nat}[x < 10];\ h?y:\mathrm{Nat};\ h!x;\ \mathit{stop}$$

In the standard semantics, query offers are instantiated by explicit data offers. Therefore, in Fig. 2, the $?$ offers correspond to either many or an infinite number of transitions, each labelled by a concrete offer.

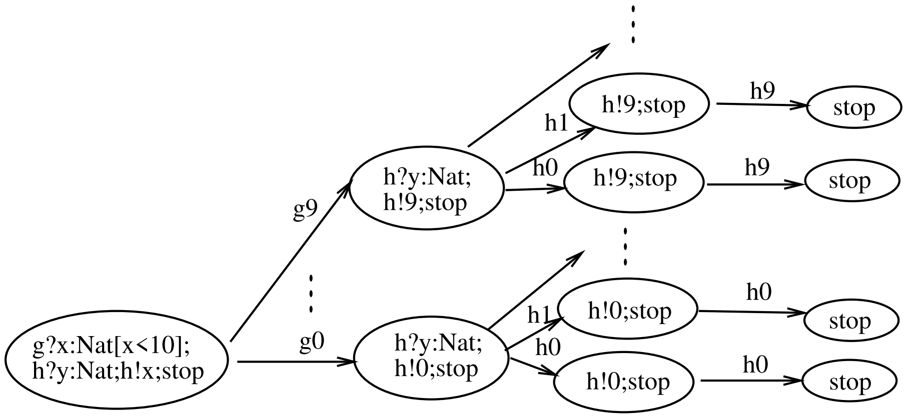

Figure 2. Standard "Concrete" Transition System

In the symbolic semantics, *open* behaviour expressions label states (e.g. $h!x;\ stop$), and transitions offer variables. The range of permissible values for the variables is determined by the Boolean conditions. Whereas the system in Fig. 2 has infinite branching, the system in Fig. 3 has only finite branching.

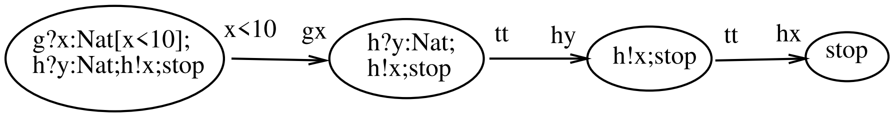

Figure 3. Symbolic Transition System

3.4. Key Features

Many of the rules given in the following section are very similar to those in the LOTOS standard; only a simple notational change to make the transition a symbolic one is required. We note here the main departures from the standard semantics, and differences from [10] and [14]).

- The syntactic distinction between the two kinds of data offer, i.e. between $?$ and $!$, has been lost. That is, both are represented by a transition labelled by a gate, an expression (possibly a simple variable) and a Boolean condition. Each offer is a *set* of values constrained in some way – expressed both by the form of the expression and by the condition of the transition. The type of offer can be determined by examining the free variables of the associated states, i.e. if $fv(E) \subseteq fv(T)$ then this is a free variable (in a ! offer, introduced previously), and if $fv(E) \not\subseteq fv(T)$ then this is a new variable (in a $?$ offer, introduced in this transition).

- Guarding, prefix and parallelism are the only rules which alter transition conditions, but unlike the LOTOS standard we do not evaluate those conditions while constructing the STS.
- Transitions associated with $?$ events may introduce new variables, in order to avoid variable name capture. For example, the query variable in $g?x : S;\ P \mid\mid\mid\ g!x;\ Q$ needs to be renamed in order avoid capturing the free variable in the right hand side. New variables may be necessary even when every $?$ variable in a specification is unique. For example, when a process is invoked more than once, e.g. $P[g]\ \mid\mid\mid\ P[g]$ where $P[g] = g?x : S;\ P'$, then one of the x variables must be assigned a unique name to avoid confusion. We assume that we may perform alpha conversion (renaming of free variables) whenever necessary.

  This is a major difference from our previous work [14], in which new variables were introduced at every data offer and synchronisation. This style of semantics gave a very clear distinction between information about a value (in the Boolean condition) and the value itself (denoted by a variable name), but would be difficult to implement since so many new names are required. Hence our revised version presented here.
- Transitions may have conditions which are not satisfiable, and may arise through unsatifiable guards in the LOTOS description, or through unsatisfiable combinations obtained through synchronisation.

3.5. Axioms and Rules of Transition

In this section we give the rules to generate STS(P), the symbolic semantics of behaviour P, from a given canonical LOTOS behaviour expression P. Following the LOTOS standard the rules are grouped according to syntactic structures (shown in the boxes). Axioms show the transition and resulting state from a given portion of LOTOS syntax (on the left of the transition). Rules show the same, but with some preconditions given above the line. Note that relabelling is not part of standard LOTOS syntax, but is introduced in order to define process instantiation (as in the LOTOS standard).

We give here a complete set of rules for deriving a symbolic semantics for a LOTOS expression; however, the most interesting rules are those for prefix, exit, guard, and parallelism.

prefix axioms

$$a;\ P \xrightarrow{\mathrm{tt}\quad a} P$$

$$g\ d;\ P \xrightarrow{\mathrm{tt}\quad gE'} P$$

$$E' = \begin{cases} E & \text{if} \quad d = !E \\ x & \text{if} \quad d = ?x : S \end{cases}$$

$$g\ d[SP];\ P \xrightarrow{SP \qquad gE'} P$$

$$E' = \begin{cases} E & \text{if} \quad d\ = !E \\ x & \text{if} \quad d\ = ?x:S \end{cases}$$

exit axioms

$$\text{exit} \xrightarrow{\mathtt{tt} \quad \delta} \text{stop}$$

$$\text{exit(ep)} \xrightarrow{\mathtt{tt} \quad \delta E'} \text{stop}$$

$$E' = \begin{cases} E & \text{if ep } = \ E \\ z & \text{if } E\ =\ \textbf{any}\ S \qquad \text{where } z \in \textbf{new-var}. \end{cases}$$

let rule

$$\frac{P[E/x] \xrightarrow{b \quad \alpha} P'}{let\ x : S = E\ in\ P \xrightarrow{b \quad \alpha} P'}$$

choice range rules

$$\frac{P[g_i/g] \xrightarrow{b \quad \alpha} P'}{choice\ g\ in\ [g_1, \ldots, g_n]\ [\,]\ P \xrightarrow{b \quad \alpha} P'}$$

for each $g_i \in \{g_1, \ldots, g_n\}$

$$\frac{P \xrightarrow{b \quad \alpha} P'}{choice\ x : S\ [\,]\ P \xrightarrow{b \quad \alpha} P'}$$

par rule

$$\frac{P[g_1/g]\ op\ \ldots\ op\ P[g_n/g] \xrightarrow{b \quad \alpha} P'}{par\ g\ in\ [g_1, \ldots, g_n]\ op\ P \xrightarrow{b \quad \alpha} P'}$$

where *op* is one of the parallel operators, $||$, $|||$, or $|[h_1, \ldots, h_m]|$, for some gate names $h_1, \ldots, h_m$.

hide rules

$$\frac{P \xrightarrow{b \quad \alpha} P'}{hide\ g_1, \ldots, g_n\ in\ P \xrightarrow{b \quad i} hide\ g_1, \ldots, g_n\ in\ P'}$$

if $\textbf{name}(\alpha) \in \{g_1, \ldots, g_n\}$

$$\frac{P \xrightarrow{b \quad \alpha} P'}{hide\ g_1, \ldots, g_n\ in\ P \xrightarrow{b \quad \alpha} hide\ g_1, \ldots, g_n\ in\ P'}$$

if $\textbf{name}(\alpha) \notin \{g_1, \ldots, g_n\}$

$\boxed{\textbf{accept rules}}$

$$\frac{P_1 \xrightarrow{b \quad \alpha} P_1'}{P_1 \gg accept\ x:S\ in\ P_2 \xrightarrow{b \quad \alpha} P_1' \gg accept\ x:S\ in\ P_2}$$

if **name**$(\alpha) \neq \delta$

$$\frac{P_1 \xrightarrow{b \quad \delta E} P_1'}{P_1 \gg accept\ x:S\ in\ P_2 \xrightarrow{b \quad i} P_2[E/x]}$$

Similarly for $\gg$ with no data.

$\boxed{\textbf{disable rules}}$

$$\frac{P_1 \xrightarrow{b \quad \alpha} P_1'}{P_1 [> P_2 \xrightarrow{b \quad \alpha} P_1' [> P_2}$$

if **name**$(\alpha) \neq \delta$

$$\frac{P_1 \xrightarrow{b \quad \alpha} P_1'}{P_1 [> P_2 \xrightarrow{b \quad \alpha} P_1'}$$

if **name**$(\alpha) = \delta$

$$\frac{P_2 \xrightarrow{b \quad \alpha} P_2'}{P_1 [> P_2 \xrightarrow{b \quad \alpha} P_2'}$$

$\boxed{\textbf{general parallelism rules (synchronising)}}$

$$\frac{P_1 \xrightarrow{b_1 \quad g} P_1' \qquad P_2 \xrightarrow{b_2 \quad g} P_2'}{P_1 |[g_1, \ldots, g_n]| P_2 \xrightarrow{b_1 \wedge b_2 \quad g} P_1' |[g_1, \ldots, g_n]| P_2'}$$

where $g \in \{g_1, \ldots, g_n, \delta\}$

$$\frac{P_1 \xrightarrow{b_1 \quad gE_1} P_1' \qquad P_2 \xrightarrow{b_2 \quad gE_2} P_2'}{P_1 |[g_1, \ldots, g_n]| P_2 \xrightarrow{b_1 \wedge b_2 \wedge E_1 = E_2 \qquad gE_1} P_1' |[g_1, \ldots, g_n]| P_2'}$$

when $vars(b_1 \cup E_1) \cap vars(b_2 \cup E_2) = \varnothing$.

$\boxed{\textbf{general parallelism rules (not synchronising)}}$

$$\frac{P_1 \xrightarrow{b \quad \alpha} P_1'}{P_1 |[g_1, \ldots, g_n]| P_2 \xrightarrow{b\sigma \quad \alpha\sigma} P_1'\sigma |[g_1, \ldots, g_n]| P_2}$$

name$(\alpha) \notin \{g_1, \ldots, g_n, \delta\}$

$$\sigma = \begin{cases} [z/x] & \text{if } \alpha = gx \text{ and } x \in vars(P_2) \\ [\,] & \text{otherwise} \end{cases} \qquad \text{where } z \in \textbf{new-var}.$$

Similarly for P_2.

choice rules

$$\frac{P_1 \xrightarrow{b \quad \alpha} P_1'}{P_1 \ [\,]\ P_2 \xrightarrow{b \quad \alpha} P_1'}$$

$$\frac{P_2 \xrightarrow{b \quad \alpha} P_2'}{P_1 \ [\,]\ P_2 \xrightarrow{b \quad \alpha} P_2'}$$

guard rule

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

stop rule

stop generates no rules.

instantiation rule

$$\frac{P[g_1/h_1, \ldots, g_n/h_n][E_1/x_1, \ldots, E_m/x_m] \xrightarrow{b \quad \alpha} P'}{p[g_1, \ldots, g_n](E_1, \ldots, E_m) \xrightarrow{b \quad \alpha} P'}$$

where $p[h_1, \ldots, h_n](x_1, \ldots, x_m) := P$ is a process definition

parenthesis rule

$$\frac{P \xrightarrow{b \quad \alpha} P'}{(P) \xrightarrow{b \quad \alpha} P'}$$

relabel rule

$$\frac{P \xrightarrow{b \quad \alpha} P'}{(P)[g_1/h_1, \ldots, g_n/h_n] \xrightarrow{b \quad \alpha'} (P')[g_1/h_1, \ldots, g_n/h_n]}$$

$$\alpha' = \begin{cases} h_i & \text{if } \alpha = g_i \text{ and } g_i \in \{g_1, \ldots, g_n\} \\ h_i E & \text{if } \alpha = g_i E \text{ and } g_i \in \{g_1, \ldots, g_n\} \\ \alpha & otherwise \end{cases}$$

4. State Equivalence and Substitution in STSs

When constructing STSs from behaviour expressions, straightforward syntactic substitution on behaviour expressions was employed. Thus, state equivalence is defined purely by syntactic equivalence of LOTOS behaviours. However, in order to define equivalence relations, preorders, or logics over STSs and to ensure that cycles (such as might arise from recursive processes) are handled correctly, we must define *substitution* on STSs. This is new from [14].

In [10], this problem is solved by introducing the concept of a “term”: a node in a symbolic transition system paired with a substitution. The same solution can be adapted for LOTOS.

Formally, a *term* consists of an STS, T, paired with a substitution, σ such that $domain(\sigma) \subseteq fv(T)$. We write this as T_σ to indicate that the substitution is not applied directly to T, and use t and u to range over terms.

Definition 2 gives the rules for transitions on terms, given the corresponding transitions on STSs. Only three cases are required: dataless transitions, transitions arising from ! offers, and transitions arising from ? offers. In all cases, $\sigma' = fv(T') \lhd \sigma$, that is, the restriction of σ to include only domain elements in the set $fv(T')$. 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 bisimulation in the next section, and the logic in [5, 6].

Definition 2 *(Transitions between terms)*

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

We note that while the vast majority of LOTOS specifications give rise to finite STSs, infinite (depth) STSs are still possible, for (data) parameterised processes. For example, *P(x) = g!x; P(x+1)* has an infinite (depth) STS. We do not consider the possibility of infinite depth a major drawback for our approach, but are beginning to investigate ways of dealing with it.

5. Symbolic Bisimulations

Standard bisimulation on transitions without data requires only the gate names to be the same. The obvious extension to transitions with data requires both gate and value to be matched exactly. In the symbolic world this is not appropriate. Consider, a symbolic transition stands for a set of concrete transitions. There are many ways to split this set into subsets; each characterisation yields a different symbolic representation. Therefore, in symbolic bisimulation, all gate names have to match exactly, but there need not be a one to one correspondence between values. A single transition in one process can be matched by one or more transitions in the other.

For example, consider the processes in Figure 4. The processes are clearly bisimilar, subject to the bisimilarity of T' with $U1$ and $U2$, but one transition in T, $x \in A \cup \bar{A}$, matches two transitions in U, $x \in A$ or $x \notin A$.

This idea of partitioning data into different sets, according to some Boolean expressions, is the crux of the following definition of symbolic bisimulation. The use of the partition means that each bisimulation is a parameterised family of relations, where the parameters are the predicates. Whereas Hennessy and Lin [10] define both an *early* and a *late* bisimulation equivalence, only an *early* bisimulation is meaningful in our context (as explained in Section 2).

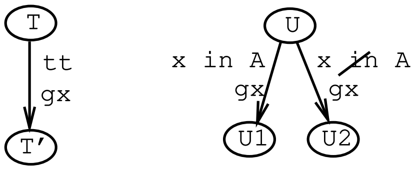

Figure 4. Bisimulation example

We give a definition of "layered" symbolic bisimulation, written $\sim_i^b$, where i is the depth of bisimulation and b is the initial context (usually simply $\mathtt{tt}$).

We assume 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 3 *(Symbolic Bisimulation on terms)*
For all b, a Boolean expression, t and u, terms:

1 $t \sim_0^b u$.

2 For all $n > 0$, $t \sim_{n+1}^b u$ iff

(a) **(dataless case)**
if t has a transition $t \xrightarrow{b_t \ \ \alpha} t'$ then there is a finite set of Boolean conditions B over fv(t) such that $(b \ \wedge \ b_t) \Rightarrow \bigvee B$ and for each $b' \in B$ there is a transition $u \xrightarrow{b_u \ \ \alpha} u'$ such that $b' \Rightarrow b_u$ and $t' \sim_n^{b'} u'$.

(b) **(data case, no new variable)**
if t has a transition $t \xrightarrow{b_t \ \ gE_t} t'$, where fv$(E_t) \subseteq$ fv(t), then there is a finite set of Boolean conditions B over fv$(t) \ \cup \ \{z\}$ such that $(b \ \wedge \ b_t \ \wedge \ z = E_t) \Rightarrow \bigvee B$, where $z =$ new(t, u), and for each $b' \in B$ either

there is a transition $u \xrightarrow{b_u \ \ gE_u} u'$, where fv$(E_u) \subseteq$ fv(u), and $b' \Rightarrow b_u$ and $b' \Rightarrow E_t = E_u$ and $t' \sim_n^{b'} u'$

or

there is a transition $u \xrightarrow{b_u \ \ gz} u'$ such that $b' \Rightarrow b_u$ and $t' \sim_n^{b'} u'$

(c) **(data case, new variable)**
if t has a transition $t \xrightarrow{b_t \ \ gz} t'$, where $z =$ new(t, u), then there is a finite set of Boolean conditions B over fv$(t) \ \cup \ \{z\}$ such that $(b \ \wedge \ b_t) \Rightarrow \bigvee B$ and for each $b' \in B$ either

there is a transition $u \xrightarrow{b_u \ \ gE_u} u'$, where fv$(E_u) \subseteq$ fv(u), and $b' \Rightarrow b_u$ and $b' \Rightarrow z = E_u$ and $t' \sim_n^{b'} u'$

or

there is a transition $u \xrightarrow{b_u \ \ gz} u'$ and $b' \Rightarrow b_u$ and $t' \sim_n^{b'} u'$

(d), (e), (f) Symmetrically, the transitions of u must be matched by t.

We may be relating processes that are parameterised. Therefore, the free variable (parameter) must be matched accordingly.

Definition 4 *($\sim^b$ for parameterised processes)*
If fv$(t) = \{x\}$, *fv*$(u) = \{y\}$, *and* $z =$ *new*(t, u) *then*

$$t \sim^b u \quad \textit{iff} \quad \forall\, z.t_{[z/x]} \sim^{b[z/x,z/y]} u_{[z/y]},$$

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

The role of the partition is to provide a step between the Boolean conditions of term t and those of term u. This can be clearly seen in case *(a)*. The intuition for cases *(b)* and *(c)* of Definition 3 is as follows. For case *(b)* we assume that the data of the t transition is a value (expression). The role of the new variable z is to provide a common language for matching transitions. In the context of b', a member of the partition, the expression E_t can either be matched by a u transition with an equivalent value E_u, or a u transition with a new variable z. The rules for transitions between terms (Definition 2) allow the new variable z to be used without explicit renaming at this point. The conditions for matching vary depending on what sort of u transition is matched. Essentially, if a new variable is matched then conditions relating to the data are captured exactly by the condition b_u. If a data expression is matched then information about data is given both by b_u and the expression E_u. Case *(c)* considers the situation in which the data of the t transition is a new variable z.

The resulting bisimulation is a Boolean condition-indexed relation. So, in most cases, when $t \sim^b u$, and t evolves to t', and u evolves to u', and $t' \sim^{b'} u'$, then b' is a different condition to b, i.e. different states in the symbolic transition systems will be related by different members of the family of relations. This is because, in a typical symbolic transition system, restrictions on data increase with depth.

We have no space here to include an example; see [7].

6. Conclusions and Further Work

We have defined a symbolic semantics for LOTOS in terms of symbolic transition systems, and symbolic bisimulation over those transition systems. Broadly speaking, we have adopted the approach of [10]; 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 [10].

Our symbolic approach eliminates infinite branching which has been a major source of difficulty in reasoning about LOTOS specifications. The inference rules for the semantics are relatively simple and intuitive, and the meaning of unparameterised processes is the same as in the standard semantics (although we did not show that here).

We have only considered strong bisimulation here, though clearly other forms of equivalence (e.g. weak bisimulation) can be defined. While we have a means of checking whether a given relation is a symbolic bisimulation we have not given here an effective method of constructing that relation. However, it is fairly easy to see that the partition has to be derived from (the cross product of) the conditions in each transition system; an algorithm has been developed and implemented in Haskell. Worst case complexity is exponential, but early recognition of zeros (in the Boolean expressions) can lessen the pain somewhat. A further interesting case is infinite (depth) transition systems. In this case we have to add a notion of state (in the imperative sense) to each (STS) state, which is beyond the scope of this paper.

Related work includes the definition of a modal logic FULL [5] which is adequate with respect to a version of the bisimulation defined here [6]. That is, it distinguishes and identifies exactly the same processes as that bisimulation. Tools to support model checking of FULL with respect to LOTOS specifications (or STSs) are also being developed based on several different implementation paradigms [3, 4, 13].

Acknowledgments

The authors would like to thank Savi Maharaj for useful input on the definition of bisimulation, and Ed Brinksma for many fruitful discussions on reasoning about LOTOS. Carron Shankland thanks the British Council, the Nuffield Foundation and the Engineering and Physical Sciences Research Council (under the project "Developing Implementation and Extending Theory: A Symbolic Approach to Reasoning about LOTOS") for their support.

Appendix: Auxiliary Definitions

Definition 5 *(Free Variables)*
Let vars(E) be the variables occurring in expression E. The set of free variables occurring in an expression is fv(E) = vars(E). The set of free variables of behaviour expression P, fv(P), is defined by

$$
\begin{array}{lcl}
fv(\mathbf{stop}) & = & \{\} \\
fv(\mathbf{exit}) & = & \{\} \\
fv(\mathbf{exit}(x)) & = & \{x\} \\
fv(P[g]) & = & \{\} \\
fv(P[g](x_1,\ldots,x_n)) & = & \{x_1,\ldots,x_n\} \\
fv(g;\ P) & = & fv(P) \\
fv(g?x:S[SP];\ P) & = & (vars(SP)\ \cup\ fv(P)) \setminus \{x\} \\
fv(g!x[SP];\ P) & = & \{x\}\ \cup\ vars(SP)\ \cup\ fv(P) \\
fv([SP]\text{->}\ P) & = & vars(SP)\ \cup\ fv(P) \\
fv(\mathbf{let}\ x = E\ \mathbf{in}\ P) & = & vars(E)\ \cup\ (fv(P) \setminus \{x\}) \\
fv(\mathbf{hide}\ g\ \mathbf{in}\ P) & = & fv(P)
\end{array}
$$

$$
\begin{array}{lcl}
fv(P_1 * P_2) & = & fv(P_1) \cup fv(P_2), \\
 & & \textit{where } * = \; [\,] \,, [> \,, \gg \,, |[g_1, \ldots, g_n]| \,, \| \,, ||| \\
fv(P_1 \gg \textbf{accept } x : S \textbf{ in } P_2) & = & fv(P_1) \cup (fv(P_2) \setminus \{x\}) \\
fv(\textbf{choice } g \textbf{ in } [g_1, \ldots, g_n] \; [\,] \; P) & = & fv(P) \\
fv(\textbf{choice } x : T \; [\,] \; P) & = & fv(P) \setminus \{x\} \\
fv(\textbf{par } g \textbf{ in } [g_1, \ldots, g_n] \; op \; P) & = & fv(P) \textit{ where } op \textit{ is one of the parallel operators}
\end{array}
$$

Similarly for the set of free variables of an action α, $fv(\alpha)$.

References

[1] T. Bolognesi, editor. Catalogue of LOTOS Correctness Preserving Transformations. Technical Report Lo/WP1/T1.2/N0045, The LOTOSPHERE Esprit Project, 1992.

[2] 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.

[3] J. Bryans and C. Shankland. Implementing a modal logic over data and processes using XTL. In this volume. Kluwer, 2001.

[4] J. Bryans, A. Verdejo, and C. Shankland. Using Rewriting Logic to implement the modal logic FULL. In D. Nowak, editor, *AVoCS'01: Workshop on Automated Verification of Critical Systems*, 2001. Oxford University Computing Laboratory technical report PRG-RR-01-07.

[5] M. Calder, S. Maharaj, and C. Shankland. A Modal Logic for Full LOTOS based on Symbolic Transition Systems. *The Computer Journal*, 2001. In press.

[6] M. Calder, S. Maharaj, and C. Shankland. An Adequate Logic for Full LOTOS. In J. Oliveira and P. Zave, editors, *Formal Methods Europe'01*, LNCS 2021, pages 384–395. Springer-Verlag, 2001.

[7] M. Calder and C. Shankland. A Symbolic Semantics and Bisimulation for Full LOTOS. Technical Report TR-2001-77, University of Glasgow, 2001. Extended version.

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

[9] 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.

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

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

[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] P. Robinson and C. Shankland. Implementing the modal logic FULL using Ergo. In D. Nowak, editor, *AVoCS'01: Workshop on Automated Verification of Critical Systems*, 2001. Oxford University Computing Laboratory technical report PRG-RR-01-07.

[14] C. Shankland and M. Thomas. Symbolic Bisimulation for Full LOTOS. In M. Johnson, editor, *Algebraic Methodology and Software Technology, Proceedings of AMAST 97*, LNCS 1349, pages 479–493. Springer-Verlag, 1997.

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

[16] M. Thomas. Modelling and Analysing User Views of Telecommunications Services. In *Feature Interactions in Telecommunications Systems*, pages 168–183. IOS Press, 1997.