A Symbolic Semantics and Bisimulation for Full LOTOS

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

Carron Shankland
Department of Computing Science and Mathematics,
University of Stirling, Stirling FK9 4LA
email: 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 and illustrated with reference to a telecomms example.

The approach taken follows that applied to message passing CCS in [HL95], but differs in several significant aspects, taking account of the particular features of LOTOS: multi-way synchronisation, value negotiation, selection predicates.

keywords: LOTOS, symbolic transition systems, symbolic bisimulation, automated reasoning.

1 Introduction

LOTOS [ISO88] is a message passing process algebra which combines two orthogonal languages: a process language, known as Basic LOTOS, with features from both CSP [Hoa85] and CCS [Mil89], and the equational abstract data type language ACT ONE [EM85]. LOTOS[1] is an ISO standard [ISO88] 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 at that point. 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. [TO94, Tho94, Tho97, SM98]) indicate that this is highly desirable. For example, one might wish to reason about the behaviour of a telecommunications service without recourse to the actual numbers dialled (c.f. the example in section 6.1).

The advantage of the standard approach is that it easily accommodates multi-way synchronisation, i.e. associative synchronisation between two or more processes (CCS only allows two-way synchronisation). Multi-way synchronisation has led to a particular constraint-oriented style of specification in the LOTOS community [VSvSB91] which is particularly useful when building a system by layering more behaviour on a simple building block (or indeed, restricting the behaviour).

The problem we address here is how to reason over potentially infinite LOTOS processes while retaining multi-way synchronisation and without restricting the datatypes. Since the addition of

[1]Full LOTOS is Basic LOTOS (the process algebra) plus algebraic data types. In the remainder of this paper the term LOTOS refers to Full LOTOS.

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 [Bol92] which changes the behaviour of the process. The second is to construct a process representation of the data type [Got87, Bri92]. This approach, however, 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 [HL95]. We draw on this approach to define a *symbolic* semantics for LOTOS in terms of *symbolic transition systems*. Our semantics eliminates infinite branching, maintains uniformities of data, and allows the representation of partial specifications, but does not lose the advantage of multi-way synchronisation, nor the information conveyed by the data. We note that the tool SMILE [LITE] implements a symbolic semantics for Full LOTOS by Eertink [Eer94]. While this 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 LOTOS including a symbolic semantics, appropriate logics, relations and tools. Previously, we have defined a modal logic and a bisimulation relation over symbolic transition systems [CMS01]. However, the drawback of that bisimulation relation is that an infinite number of values need to be checked in order to show equivalence. In this paper we define *symbolic* bisimulation between symbolic transition systems. This allows equivalence to be checked finitely. 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, but the proof is the subject of a different paper; here we set up the semantic framework for reasoning about Full LOTOS. Broadly, we follow the approach taken in [HL95] for symbolic transition graphs and message passing CCS, but our approach differs in several significant ways to accommodate the particular features of LOTOS.

The paper is organised as follows. Section 2 highlights the distinguishing features of the standard semantics for LOTOS. In Section 3 symbolic transition systems are introduced, and in Section 4, the axioms and rules for generating a symbolic transition system from a (possibly open) LOTOS behaviour expression are given. In the following section we introduce the concept of substitution over symbolic transition systems, and explain why it is necessary to consider pairs of symbolic transition systems and substitutions. In Section 6 we define (strong) symbolic bisimulation (i.e. bisimulation over symbolic transition systems), and follow this with an example drawn from telecommunications. 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 [ISO88]. An accessible tutorial to LOTOS is [LFH92]. In this section we draw out the 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 as described in [HL95].

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 [ISO88] for generating a transition system from action prefix events. The rule for *!* events is straightforward:

$$g!v;\ P \xrightarrow{gv'} P$$

where v is a ground term and $v' = [v]$ (i.e. the equivalence class of v).

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 latter rule gives us an axiom *schema* for each $?x$ event which must be instantiated by terms/values of the appropriate sort. The binding of x is defined at this point, i.e. the semantics is *early*. A *late* symbolic semantics (meaning that the 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.

To illustrate further, $g!0$ offers the single transition labelled $g[0]$, while $g?x:Nat$ offers the transitions labelled by $g[0]$, $g[succ(0)]$, $g[succ(succ(0))]$, ... See Fig. 1. Thus, ? event offers correspond to a (possibly infinite) choice over all values of the data type.

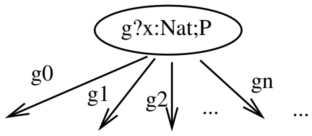

Figure 1: Standard semantics of *g?x:Nat* event offer

This encoding of values in the transitions in turn affects the rules for synchronised parallelism. Consider a form of the rule for CCS style two-way synchronisation, synchronising only on gate g for simplicity. We use LOTOS syntax here for comparison:

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

In this rule there is a single transition associated with the ? offer (labelled with x) and there is 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. Therefore the value passed effectively disappears from actions after synchronisation.

In contrast, the LOTOS rule for synchronised parallelism for the same processes makes use of the encoding of values in transitions:

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

where $v \equiv [E]$.

In the LOTOS approach, one of the transitions generated by the axiom schema for ? offers is chosen to match the transition generated by the ! offer. 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 (derived from the expression E). The resulting transition may of course be matched again in another synchronisation.

! and ? offers can synchronise in any combination. For example, when v_1 and v_2 are ground terms, $g!v_1$ and $g!v_2$ can synchronise iff $v_1 \equiv v_2$, where $\equiv$ is the equivalence induced by the data type. $g!v$ and $g?x:S$, where v is a ground term of sort S, can synchronise with the effect of x being bound to v thereafter. Finally, $g?x:S$ and $g?y:S$ can also synchronise, the result being that for every possible value, x and y are bound to the same value thereafter. Note that both values must have the same type. This last case is known as *value negotiation*.

Either kind of offer may be subject to a *selection predicate* which may restrict the data further. For example, $g?x : S[x < 42]$ denies the possibility of transitions where $x \geq 42$. In the case of value negotiation if both ? offers are qualified by selection predicates, then the value must satisfy both predicates. In fact, if the selection predicate evaluates to *false*, the event itself is prevented from occurring. 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, if we wish to preserve multi-way synchronisation and selection predicates in LOTOS (and we do), we cannot simply employ the CCS-approach to ? offers. We therefore define a new semantics based on *symbolic* transition systems.

3 Symbolic Transition Systems

Following [HL95], *symbolic transition systems* (STS) are transition systems which separate the data from process behaviour by making the data symbolic. Here, 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 $fv(T)$. The function $fv()$ is defined in Appendix A.
- *A distinguished initial state, T_0.*
- *A set of transitions written as $T \xrightarrow{b \quad \alpha} T'$ such that*
  $fv(T') \subseteq fv(T) \cup fv(E)$ *and* $fv(b) \subseteq fv(T) \cup fv(E)$ *and* $\#(fv(E) - fv(T)) \leq 1$.
  b is a Boolean expression and $\alpha \in$ Act.

We assume that we have a countable set of *variables*, *Var*, ranged over by x, y, etc., and a (possibly infinite) set of *values*, *Val*, ranged over by v. We also assume a set of *data expressions*, *Exp*, which includes *Var* and *Val* and is ranged over by E, and a set of *Boolean expressions*, *BoolExp*, ranged over by b, including the constants tt (true) and ff (false). We also assume that we have a set of gates, G, ranged over by g. The set of events, denoted Act, 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. Our rules make no distinction between these two kinds of structured events therefore we will generally treat δ as if it were a member of G in this paper. For simplicity, we do not allow structured events consisting of multiple data expressions; only *singleton* data offers are allowed. It is possible, but tedious, to extend our analysis to the case of multiple data offers.

Following convention, we shall often identify an STS with its initial state. And since one possible interpretation of states is to view states as labelled by behaviour expressions, the set of free variables of an STS T, $fv(T)$, is defined as the set of free variables of the behaviour expression labelling the initial state of T. See Defn. 4 of Appendix A. (By an abuse of notation we will often confuse state and their labels.)

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

3.1 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 which serves to illustrate the difference between the standard and symbolic semantics. Consider the behaviour expression

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

Figs. 2 and 3 contain portions of corresponding concrete and symbolic transition systems.

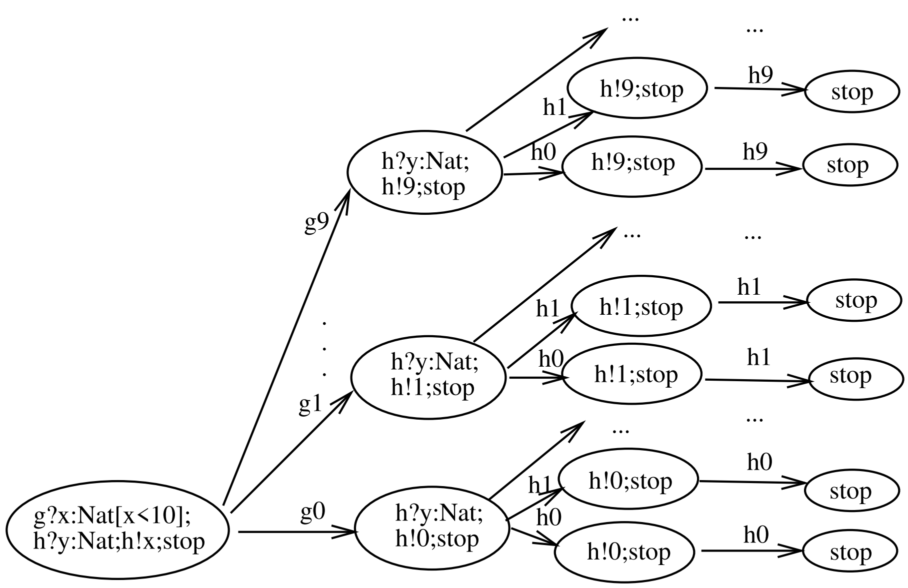

Figure 2: Concrete Transition System

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 of which is labelled by a concrete offer. Strictly speaking, the labels are the equivalence classes denoted by the ground expressions.

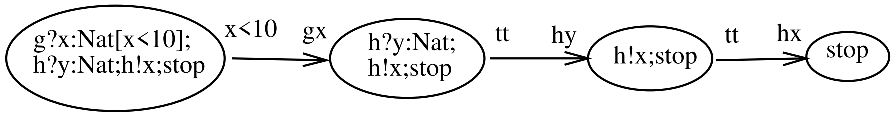

Figure 3: Symbolic Transition System

In the symbolic semantics, *open* behaviour expressions label states (e.g. $h!x;\ \mathit{stop}$), and transitions offer variables, under some conditions; these conditions determine the set of values which may be substituted for variables. Whereas the system in Fig. 2 has infinite branching, the system in Fig. 3 has only finite branching.

4 Symbolic Transition Systems for LOTOS

4.1 Preliminaries

We assume the existence of the *flattening* function of the standard semantics [ISO88]. The flattening function essentially 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 current scope.

Variables and Substitutions Variables and substitutions are over *data*. We assume all data expressions are typed, but we do not make this explicit in our definitions as noted above. Similarly, all substitutions are well typed.

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 $Var \cup Val$, written as $[z/x]$ where z is substituted for x. We use σ to denote substitutions. 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.

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

Free and Bound Variables Standard LOTOS does not distinguish between bound and free variables (because it allows no free variables), but this is an important distinction in the symbolic semantics.

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. The free variables of behaviour expressions, denoted $fv(P)$, are defined in Definition 4 of Appendix A.

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.

For a given expression E, we call a substitution σ *closing* on E when all $E\sigma$ is closed. Similarly for other objects such as states in symbolic transition systems, Boolean expressions, and terms.

4.2 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 [HL95]).

- 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/event, an expression (possibly a simple variable) and a Boolean condition. Each offer is a *set* of values constrained in some way – the constraint is usually expressed both by the form of the expression and by the condition of the transition. Whether or not a variable has been bound 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, and introduced previously), and if $fv(E) \not\subseteq fv(T)$ then this is a new variable (in a ? offer, and introduced in this transition).

- Guarding, prefix and parallelism are the only rules which alter transition conditions, but note that, unlike the LOTOS standard, we do not evaluate those conditions while constructing the STS, we simply record them for future reference (e.g. to be used in bisimulation).

- Transitions associated with ? events may introduce new variables, in order to avoid variable name capture. For example, in $g?x : S;\ P\ |||\ g!x;\ Q$, the query variable 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. $P[g] \quad ||| \quad P[g]$ 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.

- Transitions may have conditions which are not satisfiable. We will refer to these transitions as "imaginary". Imaginary transitions may, for example, be the result of unsatisfiable conditions in a LOTOS guard, e.g. $[x < 0]$ `->` P, where x is of sort Nat. Such imaginary conditions are detected when the proof system of the data type is employed to evaluate conditions generated by the axioms and rules of the next section.

4.3 Symbolic Semantics: Axioms and Rules of Transition

In this section we give the rules to generate STS(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).

prefix axioms

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

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

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

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

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

exit axioms

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

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

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

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, $|[h_1,\ldots,h_m]|$, for some gate names $h_1,\ldots,h_m$, $\|$ or $|||$.

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

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

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

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 $\mathbf{name}(\alpha) \neq \delta$

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

Similarly for $\gg$ with no data.

disable rules

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

if $\mathbf{name}(\alpha) \neq \delta$

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

if $\mathbf{name}(\alpha) = \delta$

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

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

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

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

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

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

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

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

Similarly for P_2.

$\boxed{\textbf{choice rules}}$

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

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

$\boxed{\textbf{guard rule}}$

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

$\boxed{\textbf{stop rule}}$

stop generates no rules.

$\boxed{\textbf{instantiation rule}}$

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

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

$\boxed{\textbf{parenthesis rule}}$

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

$\boxed{\textbf{relabel rule}}$

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

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

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

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

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

Figure 4: Rules for transitions between terms

5 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.

In [HL95], 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.

Figure 4 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.

6 Symbolic Bisimulations

Although our motivation is to separate reasoning about data from reasoning about processes, data cannot be disregarded when considering the equivalence of processes: the particular value of a data variable can completely alter the behaviour of a process. The standard bisimulation on transitions without data requires only the gate names to be the same. The obvious extension to transitions with data (based on the standard semantics) requires both gate and value to be matched exactly. The symbolic approach allows us to take a middle road: a single transition in one process can be matched by one or more transitions in the other. Consider, a symbolic transition stands for a set of concrete transitions. Therefore, there are many ways to characterise this set, possibly yielding more than one symbolic transition. Therefore, in symbolic bisimulation, all gate names have to match exactly, but the set of values can be reconstructed from several transitions. For example, consider the processes in Figure 5. The processes are clearly bisimilar, subject to the bisimilarity of T' with $U1$ and $U2$, but one transition in T matches two transitions in U.

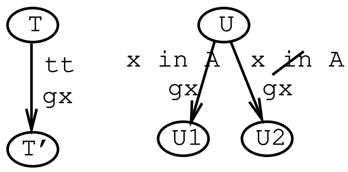

Figure 5: processes `T` and `U`

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 match transitions, without assigning a particular value to the data variables. The role of the partition is to allow transitions to be split or merged according to their data parameters.

The use of the partition means that each bisimulation is a parameterised family of relations, where the parameters are the predicates. Furthermore, we only consider simulating transitions which could possibly be valid. Namely, given a particular predicate, 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, i.e. it is "imaginary" and so we do not need to consider the transition in the simulation. We also note that whereas Hennessey and Lin [HL95] define both an *early* and a *late* bisimulation equivalence, only an *early* bisimulation is meaningful in our context (as explained in Section 2).

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 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 2 *(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 Booleans 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 Booleans 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 Booleans 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 variables must be matched accordingly.

Definition 3 *($\sim^b$ for parameterised processes)*
If $fv(t) = \{x\}$ and $fv(u) = \{y\}$, then

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

where $z = new(t, u)$.
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)*. Cases *(b)* and *(c)* of Definition 2 appear quite complex, but the intuition is as follows. For case *(b)* we assume that the data of the t transition is a value (expression). In the context of b', a member of the partition, this expression can either be matched by a u transition with the same value, or a u transition with a new variable z. The rules for transitions between terms (Definition 4) allow new variables to be introduced. The role of the new variable z is to provide a common language for matching transitions. The conditions to be matched (particularly the implication between the members of the partition and the condition of the u transition) 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)* is similar, but starting with the assumption that 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.

6.1 Examples

Consider two specifications of user behaviour in a telecomms network where users are forbidden to make and receive calls to/from particular users. The two specifications are given below with their respective STS in Figs. 6 and 7. We allow ourselves the liberty of extending the definitions to actions with multiple data offers. As noted earlier, this is a simple extension, omitted in the rest of the paper for simplicity.

Each user process is parameterised by the user id, the list of prohibited incoming callers, and the list of prohibited outgoing numbers. There are 5 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) :noexit :=
(con?x:userid!id [not (x mem bar_in)]; discon!x!id; on;
  Tel_I[dial,con,discon,unobt,on](id,bar_in,bar_out))
[]
(dial?x:userid;
   ([x mem bar_out] -> unobt; on;
     Tel_I[dial,con,discon,unobt,on](id,bar_in,bar_out)
   []
   [not(x mem bar_out)] -> con!id!x; discon!id!x; on;
     Tel_I[dial,con,discon,unobt,on](id,bar_in,bar_out)))
endproc
```

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

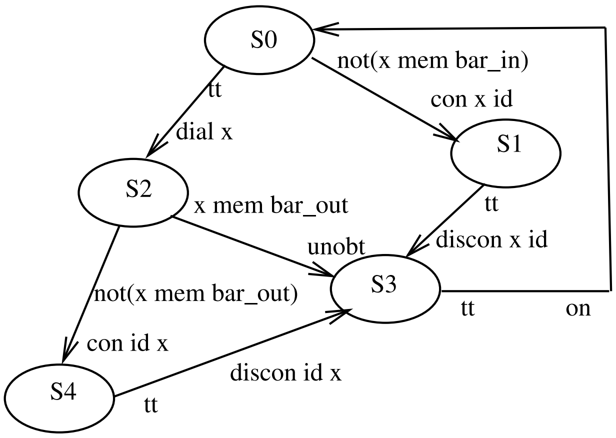

Figure 6: STS for Tel_I

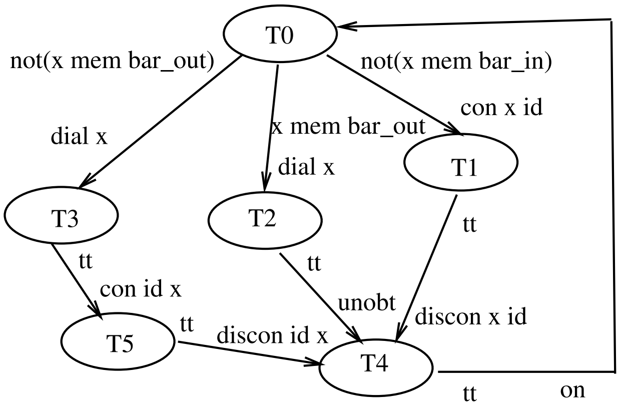

Figure 7: STS for Tel_II

The two processes are symbolically bisimilar, $Tel\_I \sim^{\mathrm{tt}} Tel\_II$, under the trivial condition tt. The (largest) symbolic bisimulation consists of the following relations (assuming the symmetric pairs in each set):

$$
\begin{array}{lcl}
S^{\mathrm{tt}} & = & \{(\mathrm{S0},\ \mathrm{T0})\} \\
S^{not(x\ mem\ bar\_in)} & = & \{(\mathrm{S1},\ \mathrm{T1}),\ (\mathrm{S3},\ \mathrm{T4})\} \\
S^{(x\ mem\ bar\_out)} & = & \{(\mathrm{S2},\ \mathrm{T2}),\ (\mathrm{S3},\ \mathrm{T4})\} \\
S^{not(x\ mem\ bar\_out)} & = & \{(\mathrm{S2},\ \mathrm{T3}),\ (\mathrm{S4},\ \mathrm{T5}),\ (\mathrm{S3},\ \mathrm{T4})\}
\end{array}
$$

The proof relies on the partition $\{x\ mem\ bar\_out, not(x\ mem\ bar\_out)\}$. We note that the *standard* semantics for both these specifications results in infinite transition systems, consequently (standard) bisimimulation could not be demonstrated using existing tools.

7 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 [HL95]; 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 [HL95].

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 they preserve the standard semantics for unparameterised processes. Other approaches to reasoning about Full LOTOS have meant using considerable intuition about different representations of data, data as processes [Got87], or using transformations which are rather complex [Bri92] or do not preserve the data information [Bol92].

We have only considered strong bisimulation here, though clearly other forms of equivalence (e.g. weak) can be defined. While we have a means of checking whether a given relation is a symbolic bisimulation, and an example illustrate this, 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; such an algorithm is currently under development. A further interesting case is infinite (depth) transition systems; with an infinite number of states, and potentially conditions, we must be able to recognise the relationships between states.

Related work includes the definition of a corresponding modal logic FULL [CMS01] which has shown to be adequate with respect to a version of the bisimulation defined here [CMS01]. That is, it should distinguish and identify exactly the same processes as the symbolic bisimulation. Further work includes the development of tools to support reasoning about LOTOS using symbolic transitions, namely a tool to check symbolic bisimulation, and model checkers for the logic based on several different implementation paradigms [BS01, BVS01, RS01].

Acknowledgement. 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.

References

[AVIS01] R. Bharadwaj, editor. *AVIS'01: First International Workshop on Automated Verification of Infinite-State Systems.* Naval Research Laboratory Technical Memorandum, 2001.

[Bol92] 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.

[BS01] J. Bryans and C. Shankland. Implementing a modal logic over data and processes using XTL. Submitted to FORTE/PSTV 2001, February 2001.

[BVS01] J. Bryans, A. Verdejo, and C. Shankland. Using Rewriting Logic to implement the modal logic FULL. In AVIS workshop [AVIS01].

[CMS01] M. Calder, S. Maharaj, and C. Shankland. An Adequate Logic for Full LOTOS. In J.N. Olivera and P. Zave, editors, *Proceedings of FME 2001*, LNCS 2021, pages 384–395, 2001.

[CMS01] M. Calder, S. Maharaj, and C. Shankland. A Modal Logic for Early Symbolic Transition Systems. *The Computer Journal*, 2001. In press.

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

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1985.

[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.

[HL95] M. Hennessy and H. Lin. Symbolic Bisimulations. *Theoretical Computer Science*, 138:353–389, 1995.

[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.

[LITE] M. Caneve and E. Salvatori, editors. LITE User Manual. Technical Report Lo/WP2/N0034/V08, The LOTOSPHERE Esprit Project, 1992. LOTOSPHERE information disseminated by J. Lagemaat, email `lagemaat@cs.utwente.nl`.

[LFH92] L. Logrippo, M. Faci and M. Haj-Hussein. An Introduction to LOTOS: Learning by Examples. *Computer Networks and ISDN Systems*, 23:325–342, 1992.

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

[RS01] P. Robinson and C. Shankland. Implementing the modal logic FULL using Ergo. In AVIS workshop [AVIS01].

[SM98] M. Sighireanu and R. Mateescu. Verification of the Link Layer Protocol of the IEEE-1394 Serial Bus (FireWire): an Experiment with E-LOTOS. *Springer International Journal on Software Tools for Technology Transfer (STTT)*, 2(1):68–88, Dec. 1998.

[Tho94] M. Thomas. The Story of the Therac-25 in LOTOS. *High Integrity Systems Journal*, 1(1):3–15, 1994.

[Tho97] M. Thomas. Modelling and Analysing User Views of Telecommunications Services. In *Feature Interactions in Telecommunications Systems*, pages 168–183. IOS Press, 1997.

[TO94] 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.

[VSvSB91] C.A. Vissers, G. Scollo, M. van Sinderen, and E. Brinksma. Specification styles in distributed systems design and verification. *Theoretical Computer Science*, 89:179–206, 1991.

A Auxiliary Definitions

Definition 4 *(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

$$
\begin{array}{lcl}
fv(\mathbf{stop}) & = & \{\} \\
fv(\mathbf{exit}) & = & \{\} \\
fv(\mathbf{exit}(x)) & = & \{x\} \\
fv(P[g]) & = & \{\} \\
fv(P[g](x_1, \ldots, x_n)) & = & \{x_1, \ldots, x_n\} \\
fv(g;\ P) & = & fv(P) \\
fv(g?x:S[SP];\ P) & = & (vars(SP)\ \cup\ fv(P)) \setminus \{x\} \\
fv(g!x[SP];\ P) & = & \{x\}\ \cup\ vars(SP)\ \cup\ fv(P) \\
fv([SP]\text{->}\,P) & = & vars(SP)\ \cup\ fv(P) \\
fv(\mathbf{let}\ x = E\ \mathbf{in}\ P) & = & vars(E)\ \cup\ (fv(P) \setminus \{x\}) \\
fv(\mathbf{hide}\ g\ \mathbf{in}\ P) & = & fv(P) \\
fv(P_1 * P_2) & = & fv(P_1)\ \cup\ fv(P_2), \\
 & & \textit{where } * = [\,],\ [>\,,\ \gg,\ |[g_1, \ldots, g_n]|\,,\ \|\,,\ ||| \\
fv(P_1 \gg \mathbf{accept}\ x : S\ \mathbf{in}\ P_2) & = & fv(P_1)\ \cup\ (fv(P_2) \setminus \{x\}) \\
fv(\mathbf{choice}\ g\ \mathbf{in}\ [g_1, \ldots, g_n]\ \ [\,]\,P) & = & fv(P) \\
fv(\mathbf{choice}\ x : T\ [\,]\ \ P) & = & fv(P) \setminus \{x\} \\
fv(\mathbf{par}\ g\ \mathbf{in}\ [g_1, \ldots, g_n]\ op\ P) & = & fv(P)\ \textit{where op is one of the parallel operators}
\end{array}
$$