Department of Computing Science and Mathematics
University of Stirling

A Symbolic Semantics and Bisimulation for Full LOTOS

Muffy Calder and Carron Shankland

Technical Report CSM-159

August 2001

Department of Computing Science and Mathematics
University of Stirling

A Symbolic Semantics and Bisimulation for Full LOTOS

Muffy Calder[1] **and Carron Shankland**

Department of Computing Science and Mathematics, University of Stirling
Stirling FK9 4LA, Scotland

Telephone +44-786-467421, Facsimile +44-786-464551
Email muffy@dcs.gla.ac.uk, carron@cs.stir.ac.uk

Technical Report CSM-159

August 2001

[1] Department of Computing Science, University of Glasgow, Glasgow G12 8QQ

Abstract

A *symbolic* semantics for Full LOTOS in terms of *symbolic* transition systems is defined; the semantics extends the standard one by giving meaning to *symbolic*, or (data) parameterised processes. Symbolic bisimulation is defined and illustrated with reference to examples.

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.

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 is an ISO standard formal description technique. The semantics of LOTOS given in the standard [ISO88] is in terms of structured labelled transition systems. In this 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. To employ model-checking as a proof technique for LOTOS systems one usually has to restrict the underlying types. 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 telephone service without recourse to the actual numbers dialled (c.f. the example in section 4.2).

The advantage of the standard approach to the semantics of LOTOS 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 is particularly useful for certain kinds of systems, and has led to a particular constraint-oriented style of specification in the LOTOS community [VSvSB91].

So, the problem we address here is how to reason over potentially infinite LOTOS processes while retaining multi-way synchronisation. 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. We cannot completely disregard data when considering the meaning of processes and equivalence between processes: the particular value of a data variable can completely alter the behaviour of a process. There are two kinds of solution to this problem. The first is to get rid of data altogether, either in a brute force manner [T.92] (which changes the behaviour of the process), or by constructing a process representation of the data type [Got87, Bri92]. The latter approach, however, still results in infinite branching. The other kind of solution is to adopt a symbolic approach. The tool SMILE [M.92] implements a symbolic semantics for Full LOTOS by Eertink [Eer94]. This semantics achieves a separation of the concerns of data and process, without losing information, but is rather operational, concentrating on using the semantics in implementing a simulation tool. There are no equivalences or preorder relations associated with the semantics. A less implementation oriented approach is taken to symbolic semantics for message passing CCS in [HL95]. The semantics is given in a modular fashion, separating the notion of a symbolic semantics from the operational aspects of making the symbolic semantics work within an implementation, for example, how variable assignments are recorded.

Our approach is to define a *symbolic* semantics for LOTOS in terms of *symbolic* transition systems. This 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. Symbolic bisimulation between symbolic transition systems is also defined. 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. The proof of this is the subject of a different paper; here we simply set up the 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 introduces LOTOS and the standard semantics, highlighting the significant features. In Section 3 symbolic transition systems are introduced, with some basic definitions and conventions that will be used throughout in Section 3.1, the main definition in Section 3.2, and the axioms and rules for generating a symbolic transition system from a (possibly open) LOTOS behaviour expression in Section 3.4. In Section 4 (strong) symbolic bisimulation (i.e. bisimulation over symbolic transition systems) is defined. Small examples are used throughout to illustrate symbolic transition systems and symbolic bisimulation; a larger example based on telephony systems is given in Section 4.2.

Finally we draw our conclusions together in Section 5 and discuss future directions.

2 LOTOS

The language Full LOTOS[1] is large and a complete presentation is outside the scope of this paper. Although we are interested in the *effect* of data we are not interested in the language *per se* therefore we do not discuss ACT ONE here. We assume only that the language has inductive data theories and a proof system. Instead we concentrate on the behavioural aspects of the language, and how that behaviour may be modified by data.

Therefore, we begin with a brief overview of the syntax and a discussion of the semantics of those features distinguishing LOTOS from other process algebras, such as CCS [Mil89] and CSP [Hoa85]. Further details of LOTOS may be found in [ISO88, BB89].

2.0.1 Syntactic Conventions

behaviour expressions	begin with P or Q.
selection predicates	(Boolean expressions) begin with SP.
gates	in their simplest form are denoted g or h. Let G be the set of gate names, then g ranges over $G \cup \{\delta\}$, and a ranges over $G \cup \{\delta, \mathbf{i}\}$, also known as *SimpleEv*. (The special gates δ and $\mathbf{i}$ are explained below.)
experiment offers	(the data associated with events) begin with d and have the form $!E$ or $?x\!:\!S$. The set of structured events (gates plus data) is denoted Act or *StructEv* and α ranges over Act $\cup$ $\{\delta, \mathbf{i}\}$. Note: $G \subseteq$ Act, therefore α ranges over G also.
(open) data expressions	begin with E.
ground data terms	begin with v.
exit parameters	begin with ep.
variables	begin with x, y, or z.
sorts	begin with S.

2.0.2 Operators of LOTOS

As in CCS and CSP, the main components of LOTOS are actions and processes. The basic processes in LOTOS are **stop**, indicating inaction (deadlock), and **exit**$(ep_1, \ldots, ep_n)$, indicating successful termination with data values. (There is also **exit** with no associated data.)

Actions occur at gates, and may or may not have data offers associated with them, e.g. $g\ d_1 \ldots d_n$ is an action at gate g with data offers $d_1 \ldots d_n$. A data offer can be a value (denoted by !, e.g. $g!4$) or a variable over a set of values (denoted by ?, e.g. $g?x\!:\!\mathrm{Nat}$). Actions may also be subject to *selection predicates*, written $g\ d_1 \ldots d_n[SP]$, where SP is a Boolean condition that may restrict the allowed data values. There are two special actions; $\mathbf{i}$ which is the (unobservable) internal event, and δ which is the successful termination event (and may have associated data).

Actions and processes are combined using the following operators:

[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.

action prefixing
written $g\ d_1 \ldots d_n;\ \ P$, meaning behave like the action $g\ d_1 \ldots d_n$ and then behave like the process P.

choice
written $P_1\ \ [\,]\ \ P_2$, meaning behave either like process P_1 or like process P_2. Nondeterministic choice results if the initial action of each branch is the same.
There are two other specialised versions of choice, one to nondeterministically choose a gate name from a given list, written **choice** g **in** $[g_1, \ldots, g_n]\ [\,]\ \ P$, and a similar one to nondeterministically choose a data value from a given range, written **choice** $x\!:\!S\ [\,]\ P$.

parallelism
In its most general form written $P_1\ |[g_1, \ldots, g_n]|\ \ P_2$, meaning perform the behaviours P_1 and P_2 in parallel, synchronising on actions in the list $g_1, \ldots, g_n$. An action in G may not proceed unless both processes are willing to perform that action. Actions not in $g_1, \ldots, g_n$ may proceed when ready.
There are three special cases: synchronising on no actions (written $P_1\ \ |||\ \ P_2$), synchronising on all actions in G (written $P_1\ \ \|\ \ P_2$), and **par** g **in** $[g_1, \ldots, g_n]\ \mathit{op}\ P$ where *op* is one of the three parallel operators described so far, meaning instantiate n copies of P in parallel, replacing g_i for g in the ith instance of P.

enable
written $P_1 \gg\ P_2$, meaning behave like process P_1 and when that terminates successfully, behave like process P_2.
Enable can also be parameterised with data $P_1 \gg$ **accept** $x_1\!:\!S_1, \ldots, x_n\!:\!S_n$ **in** P_2, where values are passed from the terminating **exit** of P_1 and bound to $x_1, \ldots, x_n$ in P_2.

disable
written $P_1\,[>\ \ P_2$, meaning behave like process P_1, but at any time P_2 may interrupt and assume control. Control never returns to P_1. If P_1 terminates successfully then P_2 can no longer interrupt.

guard
written $[SP]{-}{>}\,P$ meaning behave like process P if the Boolean condition SP is satisfied, otherwise behave like **stop**.

hide
written **hide** G **in** P, meaning behave like P, but if an action in G occurs then convert it to the internal action **i** (so it cannot be observed by the environment). This is mainly used to force the enclosed processes to communicate only with each other.

renaming
written $P[S]$, allows the gate names in P to be renamed, according to the function S.

variable declaration
written **let** $x_1 = E_1, \ldots, x_n = E_n$ **in** P, meaning bind x_i to the value E_i in P.

Recursion is also allowed and is unconstrained.

2.1 Semantics and Distinguishing Features of LOTOS

The standard semantics (i.e. according to [ISO88]) of a LOTOS process is a structured labelled transition system, as defined in Appendix A.

LOTOS has three (related) features which distinguish it from most of the standard process algebras: *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. This is in contrast to CCS and the message passing process calculus used in [HL95], where synchronisation is strictly two way: two actions synchronise yielding an unobservable action, which may not synchronise with any other action.

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 are distinguished by the use of ! or ? respectively. Either kind of offer may be subject to a selection predicate which may restrict the data further. In fact, if the selection predicate evaluates to *false*, the event itself is prevented from occurring. Note that selection predicates can refer to data in the current event (whereas the guards found in other process algebras refer to data from previous events).

! and ? offers can synchronise in any combination. For example, when v_1 and v_2 are ground terms, $g!v_1;\ P$ and $g!v_2;\ Q$ can synchronise iff $v_1 \equiv v_2$, where $\equiv$ is the equivalence induced by the data type. $g!v;\ P$ and $g?x:S;\ Q$, where v is a ground term of sort S, can synchronise with the effect of x being bound to v thereafter (i.e. in Q). Finally, $g?x:S;\ P$ and $g?y:S;\ Q$ can also synchronise, the result being that for every possible value, x and y are bound to the same value thereafter. If they are qualified by selection predicates, then the value must satisfy both predicates. This is known as value negotiation.

For example,

$$(g!succ(0);\ P_1)\ \ |[g]|\ \ (g?x:\mathrm{Nat}[odd(x)];\ P_2)\ \ |[g]|\ \ (g?y:\mathrm{Nat}[y > 0];\ P_3)$$

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

$$g!succ(0);\ (P_1\ \ |[g]|\ \ P_2[succ(0)/x]\ \ |[g]|\ \ P_3[succ(0)/y])$$

And this in turn can synchronise with, say,

$$g!pred(succ(succ(0)));\ P_4$$

all of which assumes an appropriate theory of Nat.

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 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.

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.

For example, $g!0;\ P$ offers the single transition labelled $g[0]$, while $g?x:\mathrm{Nat};\ P$ offers the transitions labelled by $g[0]$, $g[succ(0)]$, $g[succ(succ(0))]$, ... See Fig. 1. Thus, event offers of more than one value (i.e. ? offers) correspond to a (possibly infinite) choice over all values of the data type.

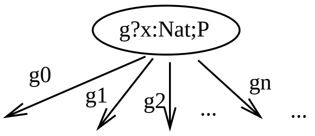


Figure 1: Standard semantics of `g?x:Nat` event offer

This encoding of values in the transitions in turn affects the rules for synchronised parallelism. Consider the rule for CCS style two-way synchronisation (we use LOTOS syntax here for comparison):

$$\frac{g!E;\ P_1 \xrightarrow{gv} P_1 \qquad g?x:S;\ P_2 \xrightarrow{gx} P_2}{g!E;\ P_1|[g]|\,g?x:S;\ P_2 \xrightarrow{\mathrm{i}} P_1|[g]|\,P_2[v/x]}\ \ g\in\ \{g_1,\ldots,g_n,\delta\}$$

Here 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 value passed disappears from the transition after synchronisation.

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

$$\frac{g!E;\ P_1 \xrightarrow{gv} P_1 \qquad g?x:S;\ P_2 \xrightarrow{gv} P_2}{g!E;\ P_1|[g_1,\ldots,g_n]|\,g?x:S;\ P_2 \xrightarrow{gv} P_1\ |[g_1,\ldots,g_n]|\,P_2}\ \ g\in\ \{g_1,\ldots,g_n,\delta\}$$

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; the matching transition may of course be matched again in another synchronisation. That is, there may be lots of other potential transitions from a state labelled with g and some value, but only one which is labelled with the value v (derived from the expression E).

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.

3 Symbolic Transition Systems for LOTOS

We must define a new semantics for LOTOS which associates a finite number of *symbolic* transitions with ? offers (like CCS), but preserves multi-way synchronisation and the selection predicates. The new semantics should remove the infinite branching introduced by ? offers, but should still be compatible with the standard semantics (i.e. it must not identify more or fewer processes).

Before giving the new *symbolic* semantics, we introduce some basic definitions and conventions.

3.1 Preliminaries

Concrete Semantics and Strong Bisimulation We refer to the standard semantics (as defined in [ISO88] and reproduced in Appendix A) as the "concrete" semantics. We refer to the corresponding strong bisimulation equivalence [Mil89] as "standard" bisimulation (equivalence), written as $\sim$. (N.B. $\sim$ is defined only on closed behaviour expressions.)

States and Boolean Formulae States in transition systems begin with T or U, and Boolean expressions (over data) begin with B (in the case of partitions in the bisimulation), and b in the case of transition conditions. We use the shorthand tt for *true* and ff for *false*.

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. 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 (data) substitution is written as $[z/x]$ where z is substituted for x; we use σ and τ to denote substitutions. Such substitutions are applied to process behaviours purely syntactically. We use a different mechanism to deal with substitution over transition systems, which deals with repeated substitions to the same variable. See Section 3.6.

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 of presentation we will assume that only one event offer can occur at a gate/event. We can easily make a suitable extension to lists of offers.

The function **name**() : 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.

Informally, free variables arise in several ways: as formal process parameters, as variables which have been introduced (and bound) earlier by a ? event, a let clause, or an >> 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.

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 B. We assume that we may perform alpha conversion (renaming of free variables) whenever necessary.

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.

3.2 Symbolic Transition Systems

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

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. (By an abuse of notation we will often confuse states and their labels.) We define the free variables of a state (behaviour expression) in Definition 4 of the appendix.

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.

We give a symbolic semantics for LOTOS by associating a symbolic transition system with each LOTOS behaviour expression P, written STS(P).

3.3 Intuition

A complete definition of the axioms and rules which define the symbolic semantics is given in the next section. Before proceeding to the formal definition, we give an example which serves to illustrate the difference between the concrete and symbolic semantics. Figs. 2 and 3 below contain portions of the respective transition systems for the behaviour expression

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

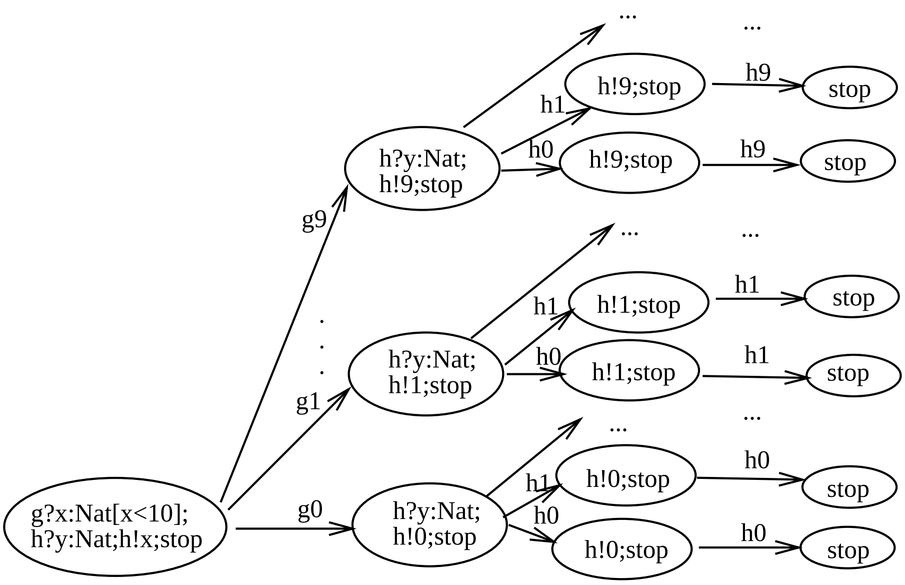


Figure 2: Concrete Transition System

In the concrete 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.

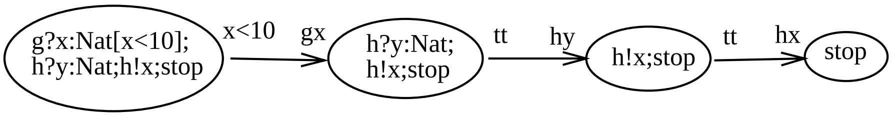


Figure 3: Symbolic Transition System

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

3.4 Symbolic Semantics: Axioms and Rules of Transition

We assume the existence of the *flattening* function of the standard semantics. The flattening function essentially ensures that the 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.

For a given canonical LOTOS specification P, the rules to generate STS(P) are as follows.

prefix axioms

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

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

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

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

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

exit axioms

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

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

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

let rule

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

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

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

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

accept rules

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

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

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

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 $(\mathit{vars}(b_1) \cup \mathit{vars}(E_1))) \cap (\mathit{vars}(b_2) \cup \mathit{vars}(E_2)) = \varnothing$.

general parallelism rules (not synchronising)

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

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

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

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

bracket rule

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

relabel rule

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

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

3.5 Key Features

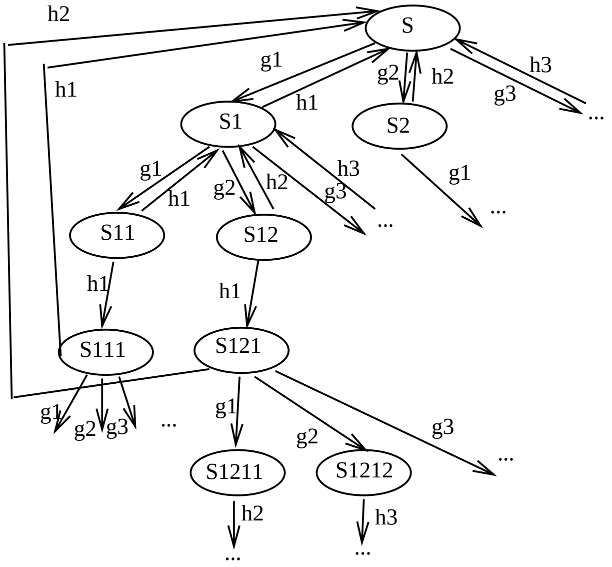
Key features (and differences from [HL95] and the concrete semantics) of the symbolic semantics defined above are:

- 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. Whether or not a variable has been bound can be determined by examining the free variables of the associated states.

- 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 to 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. $P[g]\ \ |||\ P[g]$ where $P[g] = g?x\!:\!S;\ P'$, then one of the x variables must be assigned a unique name to avoid confusion.

- Guarding, prefix and parallelism are the only rules which alter transition conditions.

- 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 previous section.

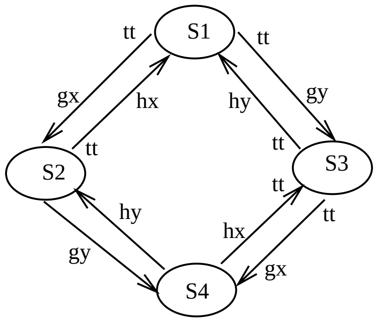


Behaviour expressions at states:

S= P ||| P
S1= h!1;P ||| P
S11 = h!1;P ||| h!1;P
S111 = P ||| h!1;P
S1211 = h!2;P ||| h!2;P

S2 = h!2;P |||P
S12 = h!1;P ||| h!2;P
S121 = P ||| h!2;P
S1212 = h!3; P ||| h!2;P

Figure 4: CTS for $P[g]\ |||\ P[g]$ *where* $P[g] = g?x;\ h!x;\ P[g]$



Behaviour expressions at states

S1 = P ||| P
S2 = h!x;P ||| P

S3 = P ||| h!y;P
S4 = h!x;P ||| h!y;P

Figure 5: STS for $P[g]\ |||\ P[g]$ *where* $P[g] = g?x;\ h!x;\ P[g]$

Figs. 4, 5, 6 and 7 contain some simple examples which illustrate some of the features of the concrete and symbolic transition systems (CTS and STS respectively).

Figs. 4 and 5 illustrate the CTS and STS, respectively, for the same behaviour expression.

Note that the processes in Figs. 6 and 7 do not have a corresponding CTS because open processes are not defined in the standard LOTOS semantics, and while the former results in a finite system, the subtle change in the recursion in the latter results in a infinite (depth) system.

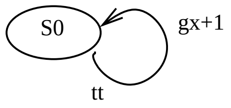


Figure 6: STS for $P[g](x) = g!x + 1;\ P[g](x)$

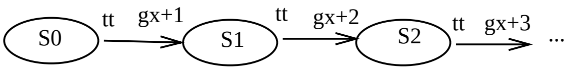


Figure 7: STS for $P[g](x) = g!x + 1;\ P[g](x + 1)$

3.6 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. The examples STSs of Figs. 5 and 6 show the applied syntactic state equivalence.

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 first define *substitution* on STSs.

Consider, for example, the simple buffer `Buff = input?x:Nat; output!x; Buff`. The STS which corresponds to `Buff` is shown in Figure 8.

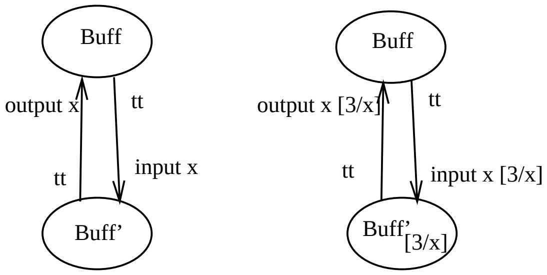


Figure 8: Failed substitution on `Buff` STS

If the first action taken by this process is to input the value `3`, then the `x` at the output gate must also be bound to that value. Since `Buff` is recursive, we expect that the next time round the loop a different value may be input, and therefore a different substitution must be applied. However, if we simply substitute `3` for `x` in the STS, as shown in Figure 8, we fail to capture this possibility.

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 *substitution* is a partial function from Var to Var ∪ Val and 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.

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

Figure 9: Rules for transitions between terms

Figure 9 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.

For example, since `Buff` is closed, it can be paired only with the empty substitution to form the term $\texttt{Buff}_{[\,]}$. The substitution is applied step by step, when necessary, as explained in the rules for transitions between terms (Figure 9). For example, below are some possible transitions starting from the term $\texttt{Buff}_{[\,]}$. The substitutions capture the fact that the variable x is discarded and then bound afresh upon each pass through the loop, making it possible to process a different value during each pass.

$\texttt{Buff}_{[\,]} \xrightarrow{\texttt{tt} \quad \texttt{input}\, z1} \texttt{Buff'}_{[z1/x]}$
$\texttt{Buff'}_{[z1/x]} \xrightarrow{\texttt{tt} \quad \texttt{output}\, z1} \texttt{Buff}_{[\,]}$
$\texttt{Buff}_{[\,]} \xrightarrow{\texttt{tt} \quad \texttt{input}\, z2} \texttt{Buff'}_{[z2/x]}$ and so on.

Terms, rather than STSs, are used as the basis for defining the bisimulation in the next section, and the logic in [CMS01a, CMS01b].

4 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. Therefore, for our purposes the gate label is too coarse a measure of equivalence; we need the extra refinement gained by also considering data, but without considering specific data values.

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. 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 2.1).

We give a definition of "layered" symbolic bisimulation over terms, written $\sim_i^b$, where i is the depth of bisimulation and b is the initial Boolean context.

We shall assume we have a function $new(t, u)$ which, given two terms t and u, returns a variable which is not among the free variables of either t or u. This is used, for example in clause (b) to introduce a new name, z, for the value associated with the "t" transition. Note that the same z is used for the "u" transition in the same clause. We know this is the same z because the rules of transitions on terms allow us to use whatever name we like, and the renaming in the rest of the term is carried out automatically.

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 κ an ordinal > 0, $t \sim_{\kappa+1}^b u$ 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 $t' \sim_\kappa^{b'} u'$.

   (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 $t' \sim_\kappa^{b'} u'$
   or
   there is a transition $u \xrightarrow{b_u \;\; gz} u'$ such that $b' \Rightarrow b_u$ and $t' \sim_\kappa^{b'} u'$

   (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 $t' \sim_\kappa^{b'} u'$
   or
   there is a transition $u \xrightarrow{b_u \;\; gz} u'$ and $b' \Rightarrow b_u$ and $t' \sim_\kappa^{b'} u'$

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

3. *For κ an ordinal and λ a limit ordinal, $t \sim_\lambda^b u$ iff $\forall \kappa < \lambda \,.\, t \sim_\kappa^b u$.*

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

$$t \sim^b u \text{ iff for } z = new(t, u),\; t_{[z/x]} \sim^{b[z/x, z/y]} u_{[z/y]}.$$

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

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). 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 (Figure 9) allow new variables to be introduced. 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. The role of the new variable z is to provide a common language for matching transitions. 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.

4.1 Examples

Some small examples illustrate symbolic bisimulation.

Example 1. Consider the following two processes, also illustrated in Fig. 10.

```
process P [c,d,e,f,g](x:Int):
                     exit :=
g!x;(   c;(  [x<0] -> f;exit
           [][x=0] -> g;exit)
     []c;(   [x=0] -> d;exit
           [][x>0] -> e;exit))
endproc
```

```
process Q [c,d,e,f,g](y:Int):
                      exit :=
g!y;(   c;(  [y<0] -> f;exit
           [][y=0] -> d;exit)
     []c;(   [y=0] -> g;exit
           [][y>0] -> e;exit))
endproc
```

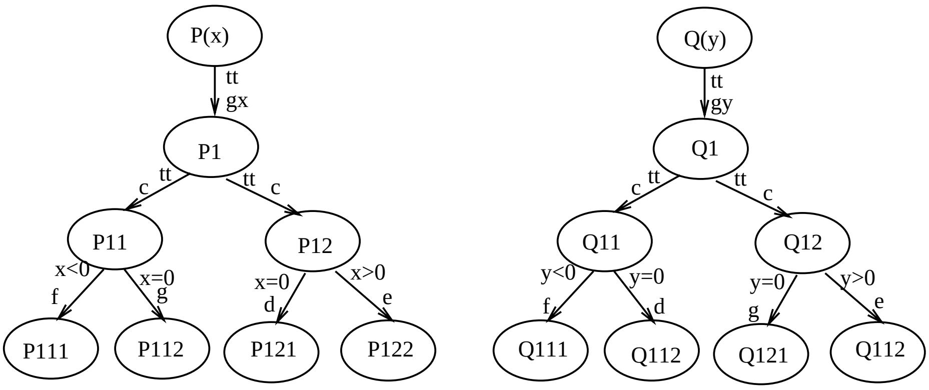


Figure 10: Example 1

These two processes are symbolically bisimilar, i.e. $P(x) \sim^{tt} Q(y)$. The crucial partitions for both P and Q are $\{z = 0, z < 0\}$, for the left c branches, and $\{z = 0, z > 0\}$, for the right c branches, where z is a unifying variable.

Specifically, we have the following relations (and their symmetric counterparts):

$$\begin{aligned}
\sim^{\mathrm{tt}} &= \{(P(z), Q(z)), (P_1, Q_1)\} \\
\sim^{z<0} &= \{(P_{11}, Q_{11}), (P_{111}, Q_{111})\} \\
\sim^{z=0} &= \{(P_{11}, Q_{12}), (P_{12}, Q_{11}), (P_{112}, Q_{121}), (P_{121}, Q_{112})\} \\
\sim^{z>0} &= \{(P_{12}, Q_{12}), (P_{122}, Q_{122})\}
\end{aligned}$$

The family of indexed relations describing the complete bisimulation is
$\{\sim^{\mathrm{tt}}, \sim^{z<0}, \sim^{z=0}, \sim^{z>0}\}$.

Example 2. Consider the following two processes, also illustrated in Fig. 11.

```
process P [c,d,e,f,g] : exit :=       process Q [c,d,e,f,g] : exit :=
g?x:Int;(   [x>0] -> c;f;exit         g?y:Int;c;(   [y>0] -> f;exit
          [][x=0] -> c;e;exit                     [][y=0] -> e;exit
          [][x<0] -> c;d;exit)                    [][y<0] -> d;exit)
endproc                               endproc
```

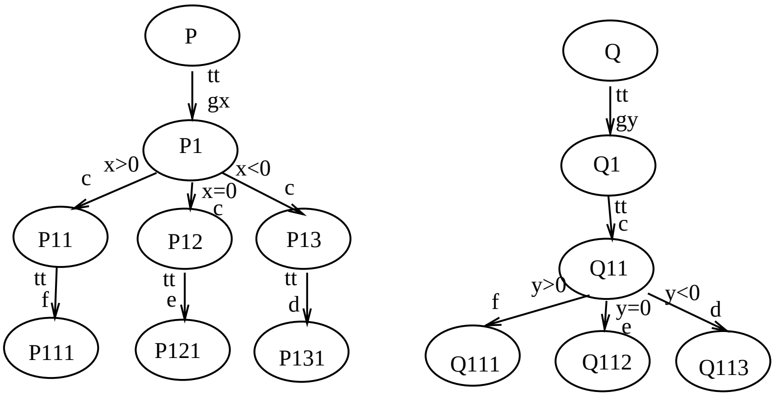


Figure 11: Example 2

These two processes are symbolically bisimilar, i.e. $P \sim^{\mathrm{tt}} Q$.

In this case the crucial partition is $\{z < 0, z = 0, z > 0\}$, where z is a unifying variable. More specifically, we have the following relations (and their symmetric counterparts):

$$\begin{aligned}
\sim^{\mathrm{tt}} &= \{(P, Q), (P_1, Q_1)\} \\
\sim^{z<0} &= \{(P_{13}, Q_{11}), (P_{131}, Q_{113})\} \\
\sim^{z=0} &= \{(P_{12}, Q_{11}), (P_{121}, Q_{112})\} \\
\sim^{z>0} &= \{(P_{11}, Q_{11}), (P_{111}, Q_{111})\}
\end{aligned}$$

That $P \sim^{\mathrm{tt}} Q$ goes against normal intuition about the effect of (nondeterministic) branching on equivalence (these processes without data would not be equivalent under $\sim$). Here as soon as the data value is bound the later actions of the process are determined, so the choice is actually deterministic.

Example 3. Consider the following two processes, also illustrated in Fig. 12.

```
process P [g,rpt,out]:exit(Nat) :=        process Q [g,rpt]:exit :=
  g?x:Nat;P1(x)                             g?y:Nat;Q1
where                                     where
process P1 [rpt,out](z:Nat):exit :=       process Q1 [rpt]:noexit :=
    [z<0] -> out;exit                       rpt; Q1
  [][z>=0] -> rpt;P1(z+1)                 endproc
endproc                                   endproc
endproc
```

These two processes are also symbolically bisimilar, i.e. $P \sim^{\mathrm{tt}} Q$, given certain assumptions about Nat.

This time the partition is not particularly interesting, but we need information from the inductive theory of Nat, namely that $\forall\, x\!:\!\mathrm{Nat}.(x >= 0 \ \wedge\ (\forall\, n\!:\!\mathrm{Nat}.x + n >= 0))$.

Note that all the right-hand transitions from P, i.e. $P_1 \longrightarrow P_{12}$, $P_{11} \longrightarrow P_{112}$, ..., are imaginary and therefore these are not considered in the bisimulation. Excluding these transitions,

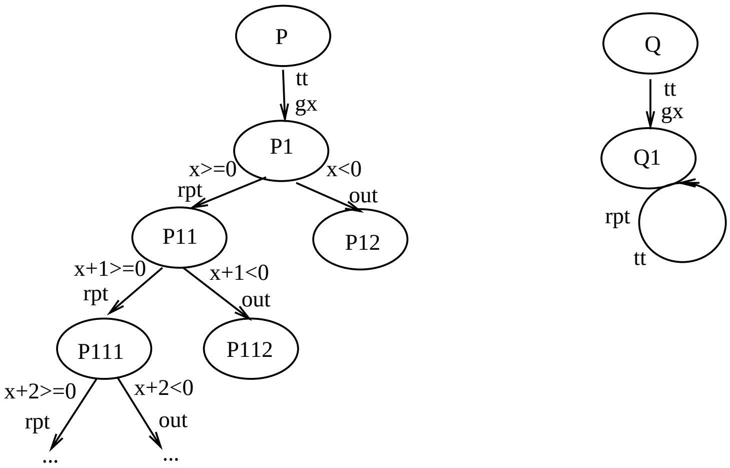


Figure 12: Example 3

we have the following infinite relation (and its symmetric counterpart):

$$\sim^{\text{tt}} \quad = \quad \{(P, Q), (P_1, Q_1), (P_{11}, Q_1), (P_{111}, Q_1), \ldots\}$$

Finally, we give an example of two processes which are *not* symbolically bisimilar.

Example 4. Consider the following two processes:

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

While we can define a suitable partition in one direction, e.g. $\{y = 3\}$, giving us $(y = 3) \Rightarrow \mathit{odd}(y)$, the symmetric case does not hold, i.e. $\mathit{odd}(y) \not\Rightarrow (y = 3)$. As we would expect, the two cannot be shown to be symbolically bisimilar.

4.2 A Larger Example

In this section we 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 Figs. 13 and 15, and their respective STS in Figs. 14 and 16. 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.

Phone Users are Bisimilar `Tel_I` and `Tel_II` are symbolically bisimilar under the trivial condition, tt; i.e. `Tel_I` $\sim^{\text{tt}}$ `Tel_II`.

```
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
```

Figure 13: LOTOS Description of Telephone I

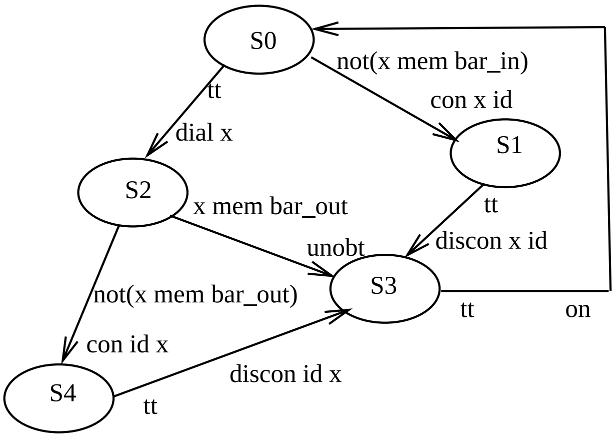


Figure 14: STS for Telephone I

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

(con?x:userid!id [not (x 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 15: LOTOS Description of Telephone II

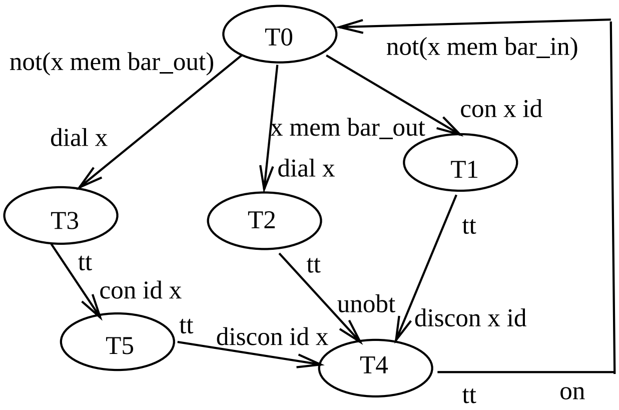


Figure 16: STS for Telephone II

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

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

The proof relies on the partition $\{x\ mem\ bar\_out, not(x\ mem\ bar\_out)\}$.

5 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 symbolic semantics, and a bisimulation relation, allows us to reason about *Full* LOTOS processes, separating the data from the processes, without losing essential information that the data supplies in terms of flow of control. The solution is simple and intuitive, unlike some previous approaches which 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 [T.92].

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 several examples 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. The interesting case is when we have infinite (depth) transition systems; these may yield an infinite number of variables, and consequently conditions, and so we must be able to recognise the relationships between them. For example, see Fig. 6.

Related work includes the definition of a corresponding modal logic [CMS01a] in which to express temporal properties of LOTOS. In a related paper [CMS01b] the logic is shown to be adequate with respect to a version of the bisimulation defined here, that is, it should distinguish and identify exactly the same processes as the symbolic bisimulation. Also planned is development of tools to support reasoning about LOTOS using symbolic transitions, namely a tool to check symbolic bisimulation, and a model checker for the logic.

Acknowledgement. The authors would like to thank Savi Maharaj for reviewing this report and for useful input on the definition of bisimulation, and Ed Brinksma for many fruitful discussions on reasoning about LOTOS. Thanks also to Jeremy Bryans and Ken Turner for reviewing a draft of this report. Carron Shankland thanks the British Council, the Nuffield Foundation and the Engineering and Physical Sciences Research Council for their support.

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.

[CMS01a] M. Calder, S. Maharaj, and C. Shankland. A Modal Logic for Full LOTOS based on Symbolic Transition Systems. *The Computer Journal*, 2001. In press.

[CMS01b] 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.

[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.

[M.92] 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`.

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

[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 Semantics of LOTOS

The semantics of processes in LOTOS are given by structured labelled transition systems. The complete syntax and semantics of full LOTOS may be found in the LOTOS standard [ISO88]; here we give only the inference rules defining the semantics of LOTOS. Notational conventions are as given in Section 2.

In order to turn a LOTOS specification into a labelled transition system, the specification is first "flattened" to give a *canonical LOTOS specification.* The inference rules of transition may then be applied to the canonical LOTOS specification to give a *class* of structured labelled transition systems, each relating to different instantiations of the formal parameters of the specification.

In the next section we give the standard definitions relating to labelled transition systems and algebras which are required for the definition of the inference rules in Section A.2.

A.1 Algebras and Transition Systems

A flattened canonical LOTOS specification, *CLS*, is given by a pair $\langle AS,\ BS\rangle$. *AS* is an algebraic specification $\langle S,\ OP,\ E\rangle$, where $\langle S,\ OP\rangle$ is a signature and *E* is a set of conditional equations. The semantics of *AS* is given by the many sorted algebra *Q(AS)* which is the quotient term algebra of *AS*. *BS* is a behaviour specification $\langle PDEFS,\ pdef_0\rangle$ where *PDEFS* is the set of all the process definition in *CLS*, and $pdef_0$ is the top level process of the specification. Each element of *PDEFS* is a pair $\langle p,\ P_p\rangle$ of a process name and the corresponding behaviour expression. All sort names and operations in *BS* are defined in *AS* (the flattening function ensures this).

The algebraic specification generates a derivation system *D*, which allows us to deduce if two terms are congruent, i.e. $D\ \vdash\ E_1\ \equiv_{AS}\ E_2$. The congruence class of a term *E*, written $[E]$, is defined as $[E] = \{E' \mid E \equiv_{AS} E'\}$.

A *labelled transition system Sys* is a 4-tuple $\langle S,\ \mathrm{Act},\ T, s_0\rangle$ $(S,\ \mathrm{Act},\ \{\xrightarrow{\alpha}\subseteq S \times S\}, s_0)$, which consists of a set S of *states*, a set Act of *transition labels*, a *transition relation* $\xrightarrow{\ \ \alpha\ \ }$, one for each $\alpha \in$ Act, with Act = $G\ \cup\ \{\mathbf{i}\}$, and a *starting state* $s_0 \in S$.

A *structured labelled transition system Struc* is a 5-tuple $\langle S, \mathrm{Act}, A, T, s\rangle$ where $A = \langle D, O\rangle$ is a many-sorted algebra such that $\langle S, Act, T, s\rangle$ is a labelled transition system, for Act $\overset{def}{=}$ $\{\mathbf{i}\}$ $\cup$ $\{gv \mid g \in G, v \in (\bigcup D)^*\}$. This is also referred to as a labelled transition system over A.

In other words, a structured labelled transition system is just like a labelled transition system, except that each label g is decorated by a string of values from D.

A.2 LOTOS Axioms and Inference Rules

$$\mathbf{i};\ P \xrightarrow{\mathrm{i}} P$$

$$g\ d_1 \ldots d_n;\ P \xrightarrow{g\ v_1\ \ldots\ v_n} [ty_1/y_1, \ldots ty_m/y_m]P$$

iff

$$\begin{array}{ll}
v_i = [E_i] & \text{if } d_i = !E_i (1 \leq i \leq n) \text{ and } E_i \text{ is a ground term,} \\
v_i \in Q(s_i) & \text{if } d_i = ?x_i (1 \leq i \leq n) \text{ with } \mathrm{sort}(x_i) = s_i, \\
ty_1, \ldots, ty_m \text{ are term instances with } v_i = [ty_i] & \text{if } d_i = ?y_j (1 \leq i \leq n, 1 \leq j \leq m) \\
 & \text{and } \{y_1, \ldots, y_m\} = \{x_i \mid d_i = ?x_i, 1 \leq i \leq n\}.
\end{array}$$

$$g\ d_1 \ldots d_n[SP];\ P \xrightarrow{g\ v_1\ \ldots\ v_n} [ty_1/y_1, \ldots ty_m/y_m]P$$

iff
v_i and ty_i defined as above, and providing $D \vdash SP'$ where SP' denotes the ground equation obtained by simultaneous replacement in SP of all x_i in SP that also occur contained in a d_i variable offer, i.e. $d_i = ?x_i(1 \leq i \leq n)$, by a term $t \in v_i$.

$$\mathbf{exit}(ep_1, \ldots, ep_n) \xrightarrow{\delta\ v_1\ \ldots\ v_n} \mathbf{stop}$$

iff

$$\begin{array}{ll}
v_i = [ep_i] & \text{if } ep_i \text{ is a ground term } (1 \leq i \leq n) \\
v_i \in Q(s_i) & \text{if } ep_i = \mathbf{any}\ s_i (1 \leq i \leq n)
\end{array}$$

$$\mathbf{exit} \xrightarrow{\delta} \mathbf{stop}$$

$$\frac{[E_1/x_1, \ldots, E_n/x_n]P \xrightarrow{\alpha} P'}{\mathbf{let}\ x_1 = E_1, \ldots, x_n = E_n\ \mathbf{in}\ P \xrightarrow{\alpha} P'}$$

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

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

$$\frac{[E/x]P \xrightarrow{\alpha} P'}{\mathbf{choice}\ x\ \ [\,]\ P \xrightarrow{\alpha} P'}$$

iff E is a ground term with $[E] \in Q(s)$, where x is a variable with $\mathrm{sort}(x) = s$.

$$\frac{(P)[g_1/g]\ \textit{par-op} \ldots \textit{par-op}\ (P)[g_n/g] \xrightarrow{\alpha} P'}{\mathbf{par}\ g\ \mathbf{in}\ [g_1, \ldots, g_n]\ \textit{par-op}\ P \xrightarrow{\alpha} P'}$$

where *par-op* is || , ||| or |[ ]| .

$$\frac{P \xrightarrow{\alpha} P'}{\mathbf{hide}\ g_1, \ldots, g_n\ \mathbf{in}\ P \xrightarrow{\alpha} \mathbf{hide}\ g_1, \ldots, g_n\ \mathbf{in}\ P'}$$
$$\text{if } \mathbf{name}(\alpha) \notin \{g_1, \ldots, g_n\}$$

$$\frac{P \xrightarrow{\alpha} P'}{\mathbf{hide}\ g_1, \ldots, g_n\ \mathbf{in}\ P \xrightarrow{\mathrm{i}} \mathbf{hide}\ g_1, \ldots, g_n\ \mathbf{in}\ P'}$$
$$\text{if } \mathbf{name}(\alpha) \in \{g_1, \ldots, g_n\}$$

$$\frac{P_1 \xrightarrow{\alpha} {P_1}'}{P_1 \gg \ \textbf{accept}\ x_1\!:\!S_1,\ldots,x_n\!:\!S_n\ \textbf{in}\ P_2 \xrightarrow{\alpha} {P_1}' \gg \ \textbf{accept}\ x_1\!:\!S_1,\ldots,x_n\!:\!S_n\ \textbf{in}\ P_2}$$
$$\textbf{name}(\alpha) \neq \delta$$

$$\frac{P_1 \xrightarrow{\delta\ v_1 \ldots v_n} {P_1}'}{P_1 \gg \ \textbf{accept}\ x_1\!:\!S_1,\ldots,x_n\!:\!S_n\ \textbf{in}\ P_2 \xrightarrow{\mathbf{i}} [E_1/x_1,\ldots,E_n/x_n]P_2}$$

where $E_1,\ldots,E_n$ are ground terms with $[E_1] = v_1,\ldots,[E_n] = v_n$.

$$\frac{P_1 \xrightarrow{\alpha} {P_1}'}{P_1\, [>\ P_2 \xrightarrow{\alpha} {P_1}'\, [>\ P_2}$$
$$\textbf{name}(\alpha) \neq \delta$$

$$\frac{P_1 \xrightarrow{\delta\ v_1 \ldots v_n} {P_1}'}{P_1\, [>\ P_2 \xrightarrow{\delta\ v_1 \ldots v_n} {P_1}'}$$

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

$$\frac{P_1 \xrightarrow{\alpha} {P_1}'}{P_1|[g_1,\ldots,g_n]|P_2 \xrightarrow{\alpha} {P_1}'\ |[g_1,\ldots,g_n]|P_2}$$
$$\textbf{name}(\alpha) \notin\ \{g_1,\ldots,g_n,\delta\}$$

$$\frac{P_2 \xrightarrow{\alpha} P_2'}{P_1|[g_1,\ldots,g_n]|P_2 \xrightarrow{\alpha} P_1\ |[g_1,\ldots,g_n]|P_2'}$$
$$\textbf{name}(\alpha) \notin\ \{g_1,\ldots,g_n,\delta\}$$

$$\frac{P_1 \xrightarrow{\alpha} P_1' \qquad P_2 \xrightarrow{\alpha} P_2'}{P_1|[g_1,\ldots,g_n]|P_2 \xrightarrow{\alpha} {P_1}'\ |[g_1,\ldots,g_n]|P_2'}$$
$$\textbf{name}(\alpha) \in\ \{g_1,\ldots,g_n,\delta\}$$

$$\frac{P_1|[\,]|P_2 \xrightarrow{\alpha} P'}{P_1|||P_2 \xrightarrow{\alpha} P'}$$

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

where $\{g_1,\ldots,g_n\} = G$ (the set of all gates).

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

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

$$\frac{P \xrightarrow{\alpha} P'}{[SP] \text{->} P \xrightarrow{\alpha} P'}$$

iff SP is a ground equation and $D \vdash SP$.

no inference rules are generated for **stop**

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

iff $\langle p, P_p\rangle \in$ BS.PDEFS
where *formal-gates(p)* $= \langle h_1,\ldots,h_n\rangle$, and *formal-vars(p)* $= \langle x_1,\ldots,x_m\rangle$.

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

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

where

$h_1,\ldots,h_n$ are *gate-names*,
$\alpha\ = gv_1\ldots v_m$,
$\alpha'\ = gv_1\ldots v_m$ if $g \notin \{h_1,\ldots,h_n\}$
$\alpha'\ = g_iv_1\ldots v_m$ if $g = h_i(1 \leq i \leq n)$

These rules and axioms completely define the structured labelled transition system of a canonical LOTOS specification.

B 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*

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