Goal Oriented Execution for LOTOS

Mazen Haj-Hussein
Luigi Logrippo
Jacques Sincennes
University of Ottawa, Computer Science Department
Protocols and Software Engineering Research Group
Ottawa, Ontario, Canada, K1N 6N5

Abstract

The dynamic semantics of LOTOS are defined in terms of axioms and inference rules which generate, from a given behaviour expression, the next possible actions and their resulting behaviour expressions. In this paper, we present a new type of inference rules, which are capable of generating traces of actions leading to a pre-selected action in the specification. These inference rules are guided by the *static derivation path* of the pre-selected action, which locates the action in the abstract syntactic tree of the current behaviour statically. This allows a considerable reduction of the search space. Such a technique often permits the analysis of divergent specifications that are generally beyond the capabilities of verification tools based on traces.

Introduction

LOTOS interpreters in existence today can function usually according to two main execution modes:

Step-by-step execution [LOBF88, vE88, vE89, Tre89, GHHL88, HH89]: the specification is executed action by action where the set of possible next actions is determined after the execution of an action. The user plays the role of the environment and resolves non-determinism, by deciding what next action should be selected, and by providing the required value expressions. Obviously, this execution mode is tedious if one wishes to execute the specification for more than a few dozen steps.

Eager execution: in this execution mode, the system attempts to go as far as possible, without user intervention, in the calculation of the behaviour tree of the specification. Values to be provided by the environment are replaced by symbolic values, or are provided by a "narrowing" algorithm [RKKL85]. An option in this execution mode is recognizing the fact that a previously encountered behaviour expression is encountered again, which means that a loop has been found [GL89, QFP88]. The output of eager evaluation can be either a symbolic behaviour tree [GL89, HH89, Ash92], or an expanded version of the specification [QFP88, Ash92]. The main problem with this execution mode is that symbolic behaviour trees grow often too quickly, although remedies have been considered, such as cutting off infeasible paths by narrowing, or reducing the size of the tree by compacting it [QFP88].

There are, of course, intermediate solutions. *Smile* [vEE91] enables the user to explore interactively symbolic behaviour trees. In any system, it is possible to restrict exploration to certain sub-trees by adding constraints, such as testing processes [GHHL88].

In this paper, we consider another type of execution mode, the *goal-oriented execution*. In this type of execution, the user specifies an action to be reached, usually an action that is not immediately derivable. The system then proceeds in a sort of selective eager execution, being able to select traces likely to reach the action. These traces can be found with the help of a static analysis of the behaviour expression. For example, if the behaviour expression is $(a;\ b;\ \mathrm{stop}\ |[b]|\ b;\ c;\ \mathrm{stop})[]c;\ d;\ f;\ \mathrm{stop}$ and the user wants to be given an (or all) execution trace(s) reaching *f* , then the left-hand side of the behaviour expression does not need to be expanded at all. The considerable savings in computing time and space are obvious from the example. When process instantiation is present, the problem of finding statically the sub-expressions that contain the desired action is slightly more complicated, but still quite manageable as we shall see. The technique described here is more powerful than the "constraint" technique described above, because the latter cannot be guided statically.

Our method allows one to look for execution traces according to several properties. Among others, two basic trace characteristics can be expressed:

- $(a, B)/A$ describes an execution trace leading from behaviour *B* to action *a* without passing through actions in $A \cup \{a\}$
- $(\langle a, b, c\rangle, B)/A$ describes an execution trace including actions *a* and *b* in that order, and terminating with *c* without passing by any action in *A*, or other occurrences of actions *a*,*b*, or *c*

In the first case, it can be specified that the targeted action can be *any* action. This is expressed by a $'-'$ in lieu of an action name.

We start by formalizing the process of statically finding a path leading to the targeted action. Viewed as a tree composed of LOTOS operators as *nodes* and gates as *edges*, the behaviour expression is traversed in order to detect the paths leading to the targeted action. These are the static derivation paths (SDPs). Further, traces leading to the targeted action are found by using goal oriented inference rules, defined on traces and guided by SDPs.

A different technique to achieve a similar goal, based on Petri Nets, is discussed in [CS92].

For the sake of simplicity, only Basic LOTOS is considered

Definitions

Conventions

The following conventions are used in this paper:

- *B, B',*B_i (i≥1) stand for LOTOS behaviour expressions
- Lower case letters, except **i**, stand for observable or unobservable actions, unless otherwise specified
- **i** stands for the internal action
- δ stands for the action performed by an *exit* construct
- $\alpha(B)$ is the set of all observable actions that appear in behaviour B

- $(B)[g_1/h_1, ..., g_n/h_n]$ stands for a relabeled behaviour expression where each action h_i that B can perform is relabeled as g_i

Traces and Operations on Traces

Following are some definitions, mostly taken from [GLO91, Gal89].

A trace is a finite-length sequence of symbols. A symbol is an identifier that represents, in our case, an observable action. Each trace can then be interpreted as a sequence of actions that may take place between a process and its environment. A trace is denoted as follows:

$\langle\rangle$	An empty trace.
$\langle a\rangle$	A trace containing only one action a.
$\langle a_1, a_2\rangle$	A trace containing two actions, a_1 followed by a_2.
$a.t$	A trace containing the action a followed by the trace t.

Here are some trace operators:

- **Projection**: The projection of a trace t on an alphabet A, denoted by $t\lfloor A$, is the trace t excluding all actions *not included* in A.
- **Inverse Projection**: The inverse projection of a trace t on an alphabet A, denoted by $t\lceil A$, is the trace t excluding all actions *included* in A.
- **Concatenation**: The concatenation of two traces t_1 and t_2, denoted by $t_1 \frown t_2$, is the trace containing the action sequence of t_1 followed by the action sequence of t_2.
- **Containment**: Denoted by $a\ in\ t$, is used to express the fact that the trace t contains the action a.
- **Alphabet**: Denoted by $\alpha(t)$, is the set of all actions in t.
- **Relabeling**: Denoted by $t[a_1/b_1, ..., a_n/b_n]$, where each b_i with $b_i\ in\ t$, for (1≤i≤n), is replaced by a_i.
- **Last**: denoted by $t^{\wedge}$, is the last action in a nonempty trace t.
- **Merging**: Denoted by $t_1|\{A\}|t_2$. This describes the set of traces resulting from composing two LOTOS processes, say P and Q, by means of the parallel composition operator $|[A]|$, where t_1 and t_2 are traces generated by processes P and Q respectively.

Transition Derivation System

The operational semantics of LOTOS behaviour expressions are defined by means of axioms and inference rules [ISO89], which permit the derivation of the observable and unobservable actions that a behaviour expression can perform. This defines the labelled transition relation $B \text{—}a\rightarrow B'$, which means that the behaviour expression B can perform the action a then behaves as B'.

The relation $B\text{-}a_1,...,a_n\rightarrow B'$ holds iff there exists $B_1,...,B_{n+1}$ such that $B_1\text{–}a_1\rightarrow B_2,...,$ $B_n\text{-}a_n\rightarrow B_{n+1}$, with $B=B_1$ and $B'=B_{n+1}$.

Let t be a trace. The relation $B=t=>B'$ is defined as:

1. $B = \langle\rangle \Rightarrow B'$ iff either $B = B'$ or there exists a natural number n with $B\text{—}\mathbf{i}^n\rightarrow B'$

2. $B = \langle a_1, ..., a_n \rangle \Rightarrow B'$ iff there exist B_1, B_2 such that $B = \langle\rangle \Rightarrow B_1$, $B_1 - a_1 \rightarrow B_2$ and $B_2 = \langle a_2, ..., a_n \rangle \Rightarrow B'$

Static Derivation Paths

A *static derivation path* (SDP) of an action *a* in a given behaviour *B* is a sequence identifying a path in the abstract syntactic tree of *B* in which *a* occurs. This path reflects the *directed* traversal of the operators composing the behaviour (e.g. ..., *right part of enable* then *nested* then *left part of choice* then ...). Process instantiations are done by using static relabeling. Although relabeling must be done dynamically from the operational semantics point of view, at this point, the only concern is *where* in a behaviour expression a given action may be found, and not *how* it is derived.

An SDP has the following form:

[] An empty path.

[*e***]** A path containing only one element *e*.

[*e*$_1$**,***e*$_2$**]** A path containing two elements, e_1 followed by e_2.

e.s A path containing the element *e* followed by the path *s*.

An element of an SDP is a symbol identifying the type of the current behaviour construct in the abstract syntactic tree. The symbols are names chosen after the LOTOS operators they represent (e.g. *choice*, *nested*). If the behaviour is involved in a binary operator, i.e. $|||$, $|[G]|$, $||$, $[]$, $[>$ or $>>$, then branches for the left and right behaviour of the operator are identified by the symbol *left* and *right* respectively, preceded by the symbol ^ .

```
specification producer_consumer[g1,g2]:noexit:=
behaviour
    hide g11,g22 in
       (producer[g1,g11] ||| consumer[g22,g2])
       |[g11,g22]|
       channel[g11,g22]
    where
process producer[g,p] :noexit
   g;p;producer[g,p]
endproc
process consumer[c,a] :noexit
   c;a;consumer[c,a]
endproc
process channel[r,s] :noexit
   r;s;channel[r,s]
endproc
endspec
```

Figure 1 A LOTOS Specification

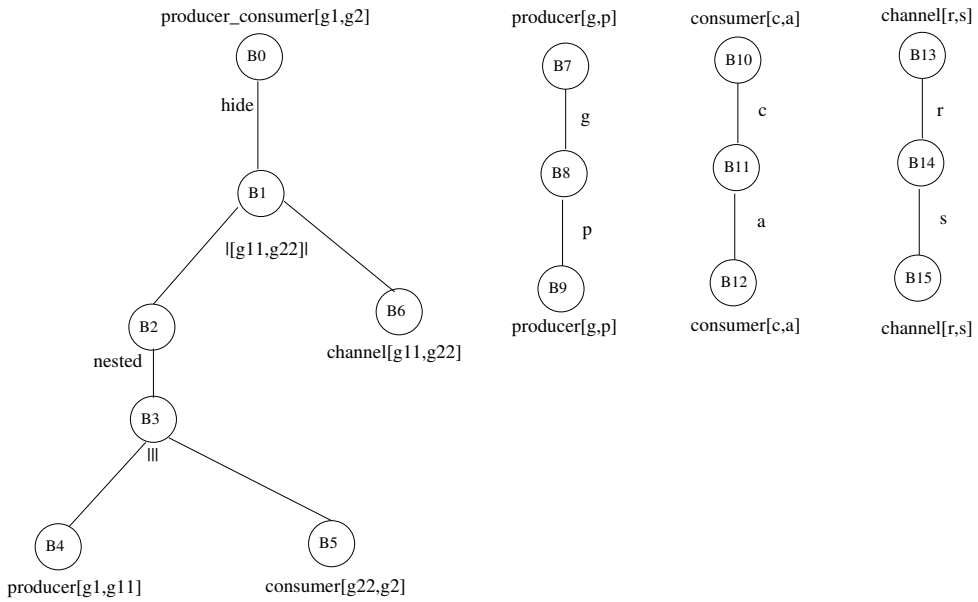

Figure 2 Abstract Syntactic Tree of Figure 1

Consider the LOTOS specification and its abstract syntactic tree given in figure 1 and figure 2 respectively. A static derivation path for the action *output* in the behaviour producer_consumer[*input*, *output*] would be [*instance, hide, parallel^left, nested, parallel^right, instance, prefix, prefix*]. It identifies the action *output* to be the action *a* in the process consumer[*c*,*a*] following the above path.

The set of all possible static derivation paths of an action *a* in a given behaviour *B* is denoted by *sdpset(a, B)*.

A restricted static derivation path of an action *a* in a given behaviour *B* with respect to a set of excluded actions *A*, is a static derivation path not having any prefixed actions in *A*. We also offer the possibility of looking for static derivation paths reaching *any action* not in *A*.

The restricted set of all static derivation paths of an action *a* in a given behaviour *B* is denoted by *sdpset(a,B)/A*. For example, considering the example given in figure 3:

$$sdpset(g_3, testing[g_1, g_2, g_3, g_4, g_5])/\{g_2\} =$$
$$\{[instance, parallel\hat{}left, nested, disable\hat{}right, choice\hat{}right, prefix, prefix],$$
$$[instance, parallel\hat{}right, choice\hat{}right, prefix, prefix, prefix]\}$$

```
specification testing[a,b,c,d,e]:noexit:=
behaviour
   ( a;b;stop
     []
     b;c;stop
       [>
          a;b;stop
          []
          d;c;stop )
|[a,c]|
   b;c;stop
   []
   a;e;c;stop
endspec
```

Figure 3 A LOTOS Specification

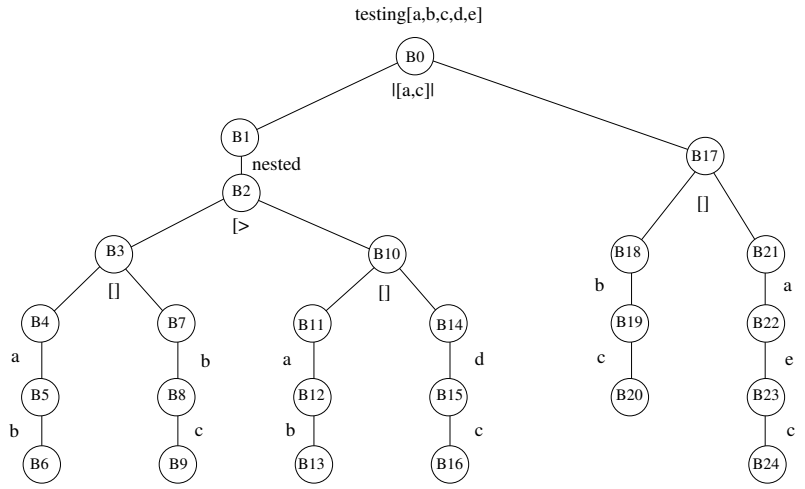

Figure 4 Abstract Syntactic Tree of figure 3

The restrictions on the use of the function *sdpset(a,B)/A* are:

1. $A \subseteq \alpha(B) \cup \{\delta\}$
2. $a \in (\alpha(B) \cup \{\delta, \text{`}-\text{`}\} - A)$, where $a = \text{`}-\text{`}$ stands for *any action*. For example, $sdpset(-, B)/A$, is the set of all static derivation paths of every action $a \notin A$ not containing actions in A

The complete definition of restricted *sdpset(a,B)/A* is given below.

Successful Termination of sdpset()

The following describes the successful ending of the traversal of the static behaviour, namely when the targeted action is found:

$$sdpset(a_1,\ a_2; B)/A\ = \{[prefix]\},\ if\ (a_1 = a_2)\ or\ (a_1 = \text{`}-\text{`}\ and\ a_2 \notin A)$$
$$sdpset(a,\ \text{exit})/A = \{[exit]\},\ if\ (a = \delta)\ or\ (a = \text{`}-\text{`}\ and\ \delta \notin A)$$

Unsuccessful Termination of sdpset()

Unsuccessful termination may result from:

1. reaching a *stop*
2. reaching an *exit*
3. the action encountered is one to avoid (i.e. belongs to *A*)
4. the instantiation of a process whose gate list does not contain the targeted action
5. the targeted action is contained in the list of hidden gates

More formally:

$$sdpset(a, stop)/A = \emptyset$$
$$sdpset(a_1,\ a_2; B)/A\ = \emptyset,\ if\ \ a_2 \in\ A$$
$$sdpset(a,\ \text{exit})/A = \emptyset,\ if\ \delta \in\ A\ or\ \left(a \neq\ '-'\ \ and\ a \neq \delta\right)$$
$$sdpset(a, p[g_1, ..., g_n])/A = \emptyset,\ if\ a \notin \{g_1, ..., g_n\}$$
$$sdpset(a, hide\ GL\ in\ B)/A\ =\ \emptyset,\ if\ a \in GL$$

Recursion

In all other situations, the behaviour has to be analyzed further. This is done by carrying the evaluation of sdpset() to the sub-behaviour(s), according to the specific rules for each type of construct, as described below. Informally, the recursive generation of the set of SDP of the current behaviour has one of the 2 forms:

1. *unary operators*: sdpset(a,*op B*) is a composition of the elements of sdpset(a,*B*), prefixing each element with the symbol representing *op*
2. *binary operators*: sdpset(a,B_1 *op* B_2) is a composition of the elements of sdpset(a,B_1) and sdpset(a,B_2), prefixing each element with the symbol representing *op* followed by *^direction*, with *direction* being *left* for elements from sdpset(a,B_1) and *right* for elements from sdpset(a,B_2)

- **Prefix**

$$sdpset(a_1,\ a_2; B)/A\ = \{[prefix.s]\ |\ \ s \in sdpset(a_1,\ B)/A\},$$
$$if\ a_1 \neq a_2\ and\ a_2 \notin\ A$$

- **Choice**

$$sdpset(a,\ B_1[]B_2)/A = \{[choice\hat{}left.s] \mid s \in sdpset(a,\ B_1)/A\} \cup \{[choice\hat{}right.s] \mid s \in sdpset(a,\ B_2)/A\}$$

- **Nested**

$$sdpset(a,\ (B))/A\ =\ \{[nested.s] \mid s \in sdpset(a,\ B)/A\}$$

- **Hiding**

$$sdpset(a,\ hide\ GL\ in\ B)/A\ =\ \{[hide.s] \mid s \in sdpset(a,\ B)/A\} \quad if\ a \notin GL$$

- **Enabling**

$$sdpset(a,\ B_1 >>\ B_2)/A = \{[enable\hat{}left.s] \mid s \in sdpset(a,\ B_1)/A\} \cup \{[enable\hat{}right.s] \mid s \in sdpset(a,\ B_2)/A\},\ if\ a\ \neq \delta$$

$$sdpset(a,\ B_1 >>\ B_2)/A = \{[enable\hat{}right.s] \mid s \in sdpset(a,\ B_2)/A\} \quad if\ a\ = \delta$$

In this case, if δ action exists in some trace, it will be found at the end of the execution of B_2. All other δ actions are transformed into internal actions by the enable operator.

- **Disabling**

$$sdpset(a,\ B_1[>\ B_2)/A = \{[disable\hat{}left.s] \mid s \in sdpset(a,\ B_1)/A\} \cup \{[disable\hat{}right.s] \mid s \in sdpset(a,\ B_2)/A\},$$

- **Parallelism**

$$sdpset(a,\ B_1|||B_2)/A = \{[parallel\hat{}left.s] \mid s \in sdpset(a,\ B_1)/A\} \cup \{[parallel\hat{}right.s] \mid s \in sdpset(a,\ B_2)/A\}$$

$$sdpset(a,\ B_1|[GL]|B_2)/A = \{[parallel\hat{}left.s] \mid s \in sdpset(a,\ B_1)/A\} \cup \{[parallel\hat{}right.s] \mid s \in sdpset(a,\ B_2)/A\}$$

$$sdpset(a,\ B_1||B_2)/A = \{[parallel\hat{}left.s] \mid s \in sdpset(a,\ B_1)/A\} \cup \{[parallel\hat{}right.s] \mid s \in sdpset(a,\ B_2)/A\}$$

- **Parallelism**

$$dpset(a,\ B_1\ op\ B_2)/A = \{[parallel\hat{}left.s] \mid s \in sdpset(a,\ B_1)/A\} \cup \{[parallel\hat{}right.s] \mid s \in sdpset(a,\ B_2)/A\} \quad with\ op \in \{|||, |[GL]|, ||\}$$

- **Process Instantiation**

$$sdpset(a,\ p[g_1, ..., g_n])/A = \left\{[instance.s] \mid s \in sdpset\left(a', B\right)/A'\right\}, \quad if\ \exists p[h_1, ..., h_n] := B, where: \quad a/a' \in \{g_1/h_1, ..., g_n/h_n\}, and \quad A' = \left\{b' \mid\ b \epsilon\ A,\ b/b' \epsilon\ \{g_1/h_1, ..., g_n/h_n\}\right\}$$

- **Relabeling**

$$sdpset(a,\ (B)[g_1/h_1, ..., g_n/h_n])/A\ =\ \left\{[relabel.s] \mid s \in sdpset\left(a',\ B\right)/A'\right\}, \quad where\ \ a/a' \in \{g_1/h_1, ..., g_n/h_n\}, and \quad A' = \left\{b' \mid b \in A, b/b' \in \{g_1/h_1, ..., g_n/h_n\}\right\}$$

Goal Oriented Inference Rules

What has been described so far were the operations necessary to analyze the behaviour *B*; extracting from it a set of SDP, i.e. the set of traces of *B* likely to reach the targeted action. Some of those traces may not be executable. For instance, although *sdpset(a, b;exit || a;exit)* finds [parallel^right.prefix] as the set of SDP, *[parallel^right.prefix]* could not be executed due to the lack of synchronization.

This leads us to the next step in goal oriented execution, namely the use of the SDPs in the application of inference rules. We shall now introduce the Goal Oriented Inference Rules. They are consistent with the usual inference rules, but differ in the following respects:

1. they define the derivation of a behaviour *B* not only on a single action, but also on traces
2. they do not describe all possible derivations of a behaviour *B* in general, but only those that comply with the SDP imposed on them

With the same restrictions as those imposed to the use of the function *sdpset*, the relations $\Rightarrow$ and $\underset{2}{\Rightarrow}$ are defined by:

1. $(a, B)/A = \langle a_1, ..., a_n \rangle \Rightarrow B'$, with $a = {}'-'$ or $a = a_n$, iff $B = \langle a_1, ..., a_n \rangle \Rightarrow B'$ such that $\forall a_i,\ 1 \leq i \leq n,\ a_i \notin A$
2. $(\langle a_1, ..., a_n \rangle, B)/A = t \underset{2}{\Rightarrow} B_n$ iff $B = \langle b_1, ..., b_m \rangle \Rightarrow B_n$, with $a_n = b_m$, and $m \geq n$, such that $\langle b_1, ..., b_m \rangle \lfloor \alpha(\langle a_1, ..., a_n \rangle) = \langle a_1, ..., a_n \rangle$, and $\forall d \in \langle b_1, ..., b_m \rangle \lceil \alpha(\langle a_1, ..., a_n \rangle), d \notin A$. An alternative definition is $(\langle a_1, ..., a_n \rangle, B)/A = t \underset{2}{\Rightarrow} B_n$ iff $(a_i, B_i)/(A - \{a_i\}) = t_i \Rightarrow B_{i+1}$ for $(1 \leq i \leq n)$, and $t = t_1 \frown ... \frown t_n$
3. $(\langle\rangle, B)/A = t \underset{2}{\Rightarrow} B_n$ iff $B = \langle\rangle \Rightarrow B_n$ or $B = \langle b_1, ..., b_n \rangle \Rightarrow B_n$, such that $\forall b_i,\ 1 \leq i \leq n,\ b_i \notin A$

Informally, the relation $\Rightarrow$ defines the derivation of behaviour *B* on a sequence of actions $\langle a_1, ..., a_n \rangle$, $i \geq 1$ with a_n being the targeted action (or any action not in *A* if *a* = '-'); while $\underset{2}{\Rightarrow}$ defines the derivation of behaviour *B* on a sequence of actions *t*, such that *t* contains a predetermined series of actions $\langle a_1, ..., a_n \rangle$, $i \geq 1$.

Note that relation 1 (on which the other two relations are defined) requires that all execution traces be generated, and only those satisfying the restriction be kept. Computationally, this is usually infeasible.

We provide an efficiently computable definition by means of *goal-oriented inference rules*, where the trace generation is guided by the static derivation paths. Thus, we redefine the relation $(a, B)/A = \langle a_1, ..., a_n \rangle \Rightarrow B'$ as $(sdp, B)/A = \langle a_1, ..., a_n \rangle \Rightarrow B'$ for $sdp \in sdpset(a, B)/A$. For the purpose of readability, we overload the notation by allowing to use either action symbols or SDPs equivalently. The formal definition of the latter relation is given below. The behaviours *Bi* used in our examples refer to the behaviour tree in figure 4. For example *B3* identifies the behaviour *a;b;*stop [] *b;c;*stop.

Base Case

$$([], B)/A = \langle\rangle \Rightarrow B \tag{1}$$

When the end of the SDP is reached, an empty trace and the current behaviour expression are returned.

Successful Termination

$$([exit], \text{exit})/A = \langle\delta\rangle \Rightarrow \text{stop} \tag{2}$$

The action δ can only appear at the end of a trace.

Action Prefix

$$\frac{(s, B)/A = t \Rightarrow B'}{([prefix.s], a; B)/A = \langle a.t\rangle \Rightarrow B'} \tag{3}$$

This rule accumulates the actions in the derived trace. For example, taking behaviour *B7* = (*b; c;* stop) in figure 4, the relation $([prefix, prefix], B7)/\{\} = t \Rightarrow B'$ will be satisfied with $t = \langle b, c\rangle$ and $B' = B9 = \text{stop}$.

Nested

$$\frac{(s, B)/A = t \Rightarrow B'}{([nested.s], (B))/A = t \Rightarrow B'} \tag{4}$$

Nesting has no effect on the derived trace.

Hiding

$$\frac{(s, B)/A = t \Rightarrow B'}{([hide.s], \text{hide } GL \text{ in } B)/A = t\lceil\{GL\} \Rightarrow \text{hide } GL \text{ in } B'} \tag{5}$$

The trace generated by the hide operator is filtered against the list of hidden gates ($t\lceil\{GL\}$). For example, let $B = \text{hide } a, b \text{ in } c; a; d; \text{stop}$, then $([hide, prefix, prefix], B)/\{\} = t \Rightarrow B'$, will be satisfied with $t = \langle c, a\rangle\lceil\{a, b\} = \langle c\rangle$ and $B' = \text{hide } a, b \text{ in } d; \text{stop}$

Enabling

$$\frac{(s, B_1)/A = t_1 \Rightarrow B_{11}}{([enable\hat{}left.s],\ B_1 >> B_2)/A = t_1 \Rightarrow B_{11} >> B_2} \tag{6}$$

$$\frac{\begin{array}{c}(s,\ B_2)/A = t_2 \Rightarrow B_{21}, \\ (r, B_1)/(A \cup (\{t_2^{\wedge}\} - \{\delta\})) = t_1 \Rightarrow B_{11}\end{array}}{\begin{array}{c}([enable\hat{}right.s],\ B_1 >> B_2)/A = t_1 \lceil \{\delta\} \frown t_2 \Rightarrow B_{21} \\ with\ r\ \in\ sdpset(\delta, B_1)\end{array}} \tag{7}$$

The first rule states that if the goal action is in B_1 then the resulting trace will be a trace t_1 generated from B_1 guided by the remainder SDP s, and the resulting behaviour will be B_{11}>>B_2 where B_{11} is the behaviour B_1 after trace t_1. The second rule states that if the goal action is in B_2 then the resulting trace will be the concatenation of trace t_1 excluding the δ action at the end, and trace t_2, where t_1 is a trace from B_1 leading to δ and not including the target element (i.e. $t_2^{\wedge}$) nor any element in A, and t_2 is a trace from B_2 guided by the remainder SDP s. The resulting behaviour will simply be behaviour B_2 after trace t_2.

For example, let behaviour $B = a; b; \text{exit} >> c; \text{exit}$, then the set of static derivation paths $sdpset(\delta, B)/\{\} = \{[enable\hat{}right, prefix, exit]\}$. Therefore the relation $(\delta,\ B)/\{\} = t \Rightarrow B'$ is defined as $([enable\hat{}right, prefix, exit],\ B)/\{\} = t \Rightarrow B'$ which will match rule (7) where the following relations must hold:

1. $([prefix, exit],\ c; \text{exit})/\{\} = t_2 \Rightarrow Br_2$
2. $(\delta,\ a; b; \text{exit})/(\{t_2^{\wedge}\} - \{\delta\}) = t_1 \Rightarrow Br_1$

The first relation will be satisfied with $t_2 = \langle c, \delta \rangle$ and $Br_2 = \text{stop}$. The second relation then becomes $(\delta,\ a; b; \text{exit})/\{\} = t_1 \Rightarrow Br_1$, which will be defined as $([prefix, prefix, exit],\ a; b; \text{exit})/\{\} = t_1 \Rightarrow Br_1$, where $[prefix, prefix, exit] \in sdpset(\delta, a; b; \text{exit})/\{\}$, and will be satisfied with $t_1 = \langle a, b, \delta \rangle$ and $Br_1 = \text{stop}$. Therefore the original relation $([enable\hat{}right, prefix, exit],\ B)/\{\} = t \Rightarrow B'$, will be satisfied with $t = t_1 \lceil \{\delta\} \frown t_2 = \langle a, b, \delta \rangle \lceil \{\delta\} \frown \langle c, \delta \rangle = \langle a, b, c, \delta \rangle$ and $B' = Br_2 = \text{stop}$

Disabling

$$\frac{(s, B_1)/A = t_1 \Rightarrow B_{11},\ t_1^{\wedge} = \delta}{([disable\hat{}left.s],\ B_1 [> B_2)/A = t_1 \Rightarrow B_{11}} \tag{8}$$

$$\frac{(s, B_1)/A = t_1 \Rightarrow B_{11},\ t_1^{\wedge} \neq \delta}{([disable\hat{}left.s],\ B_1 [> B_2)/A = t_1 \Rightarrow B_{11} [> B_2} \tag{9}$$

$$\frac{\begin{array}{c}(s, B_2)/A = t_2 \Rightarrow B_{21}, \\ (\langle\rangle, B_1)/(A \cup \{t_2^{\wedge}\} \cup \{\delta\}) = t_1 \underset{2}{\Rightarrow} B_{11}\end{array}}{([disable\hat{}right.s],\ B_1 [> B_2)/A = t_1 \frown t_2 \Rightarrow B_{21}} \tag{10}$$

The first and second rule handle the case where the goal action is in B_1. Let B_{11} be the behaviour B_1 after trace t_1 guided by the remainder SDP s. The resulting behaviour expression

of the above rule will be B_{11}, namely **stop**, if trace t_1 ends with δ; otherwise the resulting behaviour will be $B_{11}[>B_2$. The third rule states that if the search is guided to the right behaviour B_2, then the resulting trace will be a trace t_1 concatenated to trace t_2, where t_1 is any trace derived from B_1 with length ≥ 0 and not including actions in $A \cup \{t_2^\wedge\}$, and t_2 is generated from B_2 guided by the remainder SDP s. The resulting behaviour will be B_2 after trace t_2, namely B_{21}.

For example, considering behaviour $B2$ in figure 4 where $sdpset(c,\ B2)/\{b\} = \{[disable\hat{}right, choice\hat{}right, prefix, prefix]\}$ then the relation $(c,\ B2)/\{b\} = t \Rightarrow B'$, is defined as $([disable\hat{}right, choice\hat{}right, prefix, prefix],\ B2)/\{b\} = t \Rightarrow B'$. This matches rule (10) where the following relations must be satisfied:

1. $([choice\hat{}right, prefix, prefix], B10)/\{b\} = t_2 \Rightarrow Br_2$
2. $(\langle\rangle,\ B3)/\{b, t_2^\wedge, \delta\} = t_1 \underset{2}{\Rightarrow} Br_1$

The first relation will be satisfied with $t_2 = \langle d, c\rangle$ and $Br_2 = \text{stop}$; the second relation then becomes $(\langle\rangle,\ B3)/\{b, c, \delta\} = t_1 \Rightarrow Br_1$, and will be satisfied with two results:

1. $t_1 = \langle\rangle,\ Br_1 = B3 = a; b; \text{stop}[]b; c; \text{stop}$
2. $t_1 = \langle a\rangle,\ Br_1 = B5 = b; \text{stop}.$

Therefore the original relation $([disable\hat{}right, choice\hat{}right, prefix, prefix],\ B2)/\{b\} = t \Rightarrow B'$ will be satisfied with $t = t_1 \frown t_2, \text{where}\, t_1 \in \{\langle\rangle, \langle a\rangle\}, t_2 = \langle d, c\rangle$ and $B' = Br_2 = \text{stop}$.

Selected Synchronization

$$\frac{\begin{array}{c}(s, B_1)/A = t_1 \Rightarrow B_{11},\\ (t_1 \lfloor (\{S\} \cup \{\delta\}),\ B_2)/(A \cup \{S\} \cup \{\delta\}) = t_2 \underset{2}{\Rightarrow} B_{21}\end{array}}{([parallel\hat{}left.s],\ B_1|[S]|B_2)/A = t_1|\{S\}|t_2 \Rightarrow B_{11}|[S]|B_{21}} \tag{11}$$

$$\frac{\begin{array}{c}(s, B_2)/A = t_2 \Rightarrow B_{21},\\ (t_2 \lfloor (\{S\} \cup \{\delta\}),\ B_1)/(A \cup \{S\} \cup \{\delta\}) = t_1 \underset{2}{\Rightarrow} B_{11}\end{array}}{([parallel\hat{}right.s],\ B_1|[S]|B_2)/A = t_1|\{S\}|t_2 \Rightarrow B_{11}|[S]|B_{21}} \tag{12}$$

The inference rule with $parallel\hat{}left$ means that the desired action is in B_1. Therefore the remainder of the SDP, namely s, is the static derivation path that will guide the inference rules to generate a trace t_1 such that $(s, B_1) = t_1 \Rightarrow B_{11}$. In this case the resulting trace $t = t_1|\{S\}|t_2$ will be valid if we can generate a trace t_2 from B_2 such that $t_2 \lfloor (\{S\} \cup \{\delta\}) = t_1 \lfloor (\{S\} \cup \{\delta\})$ and t_2 does not contain any elements in the restricted set A. This can be done efficiently using the relation:

$$(t_1 \lfloor (\{S\} \cup \{\delta\}), B_2)/(A \cup \{S\} \cup \{\delta\}) = t_2 \underset{2}{\Rightarrow} B_{21}$$

For example, consider behaviour $B0$ in figure 4. We have

$$\begin{array}{c}sdpset(c, B0)/\{b\} = \{[parallel\hat{}left, nested, disable\hat{}right, choice\hat{}right, prefix, prefix],\\ [parallel\hat{}right, choice\hat{}right, prefix, prefix, prefix]\}\end{array}$$

Then the relation $(c,\ B0)/\{b\} = t \Rightarrow B'$ is defined as ([*parallel^left, disable^right, choice^right, prefix, prefix*],*B*0)/{*b*} = *t* ⇒*B'*. This matches rule (11) where the following relations must hold:

1. $([nested, disable\hat{}right, choice\hat{}right, prefix, prefix],\ B1)/\{b\} = t_1 \Rightarrow Br_1$
2. $(t_1 \lfloor (\{a, c, \delta\}),\ B17)/\{a, b, c, \delta\} = t_2 \underset{2}{\Rightarrow} Br_2$

The first relation will be satisfied with two results from the previous example:

1. $t_1 = \langle d, c\rangle, Br_1 = \text{stop}$
2. $t_1 = \langle a, d, c\rangle, Br_1 = \text{stop}$

The second relation then becomes

$(\langle d, c\rangle \lfloor (\{a, c, \delta\}),\ B17)/\{a, b, c, \delta\} = t_2 \underset{2}{\Rightarrow} Br_2$
or $(\langle a, d, c\rangle \lfloor (\{a, c, \delta\}),\ B17)/\{a, b, c, \delta\} = t_2 \underset{2}{\Rightarrow} Br_2$

The first relation will not hold since $sdpset(c,\ B17)/\{a, b, \delta\} = \emptyset$, and the second rule will succeeds with $t_2 = \langle a, e, c\rangle, Br_2 = \text{stop}$. And as a conclusion, the original relation

$$([parallel\hat{}left, disable\hat{}right, choice\hat{}right, prefix, prefix], B0)/\{b\} = t \Rightarrow B'$$

will hold with: $t = t_1|[a, c]|t_2 = \langle a, d, c\rangle|[a, c]|\langle a, e, c\rangle$, $B' = Br_1|[a, c]|Br_2 = \text{stop}|[a, c]|\text{stop}$. Therefore $t \in \{\langle a, d, e, c\rangle, \langle \text{a, e, d, c}\rangle\}$.

Interleave Parallelism

$$\frac{(s,\ B_1|[]|B_2)/A = t \Rightarrow B'}{(s,\ B_1|||B_2)/A = t \Rightarrow B'} \tag{13}$$

The interleave operator is treated as the selected synchronization operator with an empty list of synchronization gates.

Full Synchronization

$$\frac{(s,\ B_1|[\alpha(B_1) \cup \alpha(B_2)]|B_2)/A = t \Rightarrow B'}{(s,\ B_1||B_2)/A = t \Rightarrow B'} \tag{14}$$

The full synchronization operator is treated as the selected synchronization operator with the list of synchronization gates composed with the alphabet of behaviours B_1 and B_2.

Process Instantiation

$$\frac{\exists p[h_1, ..., h_n] := B, ([relabel.s], (B)[RL])/A' = t \Rightarrow B'}{\begin{array}{c}([instance.s],\ p[g_1, ..., g_n])/A = t \Rightarrow B' \\ where\ RL = g_1/h_1, ..., g_n/h_n, \\ A' = \{a' | a \in A, a/a' \in RL\}\end{array}} \tag{16}$$

In this rule the element *instance* in replaced by *relabel*, since a process instantiation relabels the behaviour of the process. Given the specification in figure 3, suppose we want to reach action g_3 without passing by g_2 from the behaviour $testing[g_1,\ g_2,\ g_3,\ g_4,\ g_5]$. This can be specified by $(g_3, testing[g_1, g_2, g_3, g_4, g_5])/\{g_2\} = t \Rightarrow B'$ which is defined as $(sdp,\ testing[g_1,\ g_2,\ g_3,\ g_4,\ g_5])/\{g_2\} = t \Rightarrow B',\ \ sdp \in sdpset(g_3, testing[g_1,\ g_2,\ g_3,\ g_4,\ g_5])/\{g_2\}$

For *sdp* = [*instance, relabel, parallel^left, nested, disable^right, choice^right, prefix, prefix*] we have the relation $([instance.s],\ testing[g_1,\ g_2,\ g_3,\ g_4,\ g_5])/\{g_2\} = t \Rightarrow B'$ that matches rule (16) and yields to $(s,\ (B0)[g_1/a, g_2/b, g_3/c, g_4/d, g_5/e])/\{b\} = t \Rightarrow B'$ where *s* = [*relabel, parallel^left, nested, disable^right, choice^right, prefix, prefix*]. This is the same relation we had in the previous section that resulted in:

$t = t_1[g_1/a, g_2/b, g_3/c, g_4/d, g_5/e]$

$B' = (\mathrm{stop}|[a,c]|\mathrm{stop})[g_1/a, g_2/b, g_3/c, g_4/d, g_5/e]$

Which implies $t \in \{\langle g_1, g_4, g_5, g_3\rangle, \langle g_1, g_5, g_4, g_3\rangle\}$

Relabeling

$$\frac{(s, B)/A = t \Rightarrow B'}{([relabel.s],\ (B)[RL])/A = t[RL] \Rightarrow (B')[RL]} \quad (15)$$
$$where\ RL = g_1/h_1, ..., g_n/h_n$$

For example, with behaviour $B0$ of figure 4, letting

$$s = [parallel\hat{}left, disable\hat{}right, choice\hat{}right, prefix, prefix]$$

the relation $([relabel.s],\ (B0)[g_1/a, g_2/b, g_3/c, g_4/d, g_5/e])/\{b\} = t \Rightarrow B'$ matches rule (15) where the relation $(s,\ B0)/\{b\} = t_1 \Rightarrow Br_1$ must hold, as in the previous example, with:

1. $t_1 = \langle a, d, e, c\rangle, B' = \mathrm{stop}|[a,c]|\mathrm{stop}$
2. $t_1 = \langle a, e, d, c\rangle, B' = \mathrm{stop}|[a,c]|\mathrm{stop}$

Therefore the original relation $([relabel.s],\ (B0)[g_1/a, g_2/b, g_3/c, g_4/d, g_5/e])/\{b\} = t \Rightarrow B'$ will be satisfied with

$t = t_1[g_1/a, g_2/b, g_3/c, g_4/d, g_5/e]$

$B' = (\mathrm{stop}|[a,c]|\mathrm{stop})[g_1/a, g_2/b, g_3/c, g_4/d, g_5/e]$

This implies $t \in \{\langle g_1, g_4, g_5, g_3\rangle, \langle g_1, g_5, g_4, g_3\rangle\}$

Limitations and Future Work

We foresee several applications for the concept of goal-oriented execution:

- It offers a very attractive alternative to the two execution modes mentioned in the introduction:
  - — instead of *manually* attempting to reach an action *a* not immediately derivable, one could rather ask for a list of paths leading to it

— in a graphic environment, one could also think of successively pointing at a sequence of LOTOS actions on the screen, thus expressing the wish of seeing a path reaching those actions in the selected order

- It enables verification of some types of temporal logic properties, for example that a certain action can be followed by another action.
- Because execution paths in a specification are also test cases, our method can be used to obtain test cases corresponding to certain test intents. Similarly, the method can be used to find preambles and postambles for test sequences.
- [vdS92] shows how goal-oriented inference rules can be used to find test cases corresponding to selection criteria based on dataflow analysis..

Of course, we do not claim that the techniques proposed in this paper completely answer these needs, however they are a step in that direction. Everything we have presented in this paper has been implemented in Prolog. Work is currently being carried out in extending the method to full LOTOS. This basically requires the use of narrowing techniques [RKKL85] in order to find values for value expressions and prune infeasible paths.

Concerning the limitations, it shall be mentioned that the technique may cause the generation of infinite traces due to the *depth-first* searching strategy. The *breadth-first* searching strategy may resolve this problem to some extent, but is memory consuming. The current technique still handles many cases where the existing interpreters may fail. For example, it is able to reach action *a* in the following recursive processes:

$$P[a] := a; \mathrm{stop}[]P[a]$$
$$P[a] := a; \mathrm{stop}||P[a]$$
$$P[a,b,c] := (b; \mathrm{stop}[](c; b; a; \mathrm{stop}|[b]|P[a,b,c]))$$

as well as:

$$P[a,b] := b; P[b,a]$$

which involves multiple instantiations, but it could not in:

$$P[a] := a; stop||P[a]$$

where *P[a]* is instantiated an infinite number of times.

Our implementation of static derivation paths excludes all paths having more than one appearance of an instantiation with a given gate list (this is not specified in our formal definition of SDPs). Therefore, our tool is able to reach action *a* in process *P* given below:

$$P[a,c] := c; \mathrm{Q[c]} >> R[a]$$
$$Q[c] := \mathrm{hide}\ a\ \mathrm{in}\ (a; c; Q[c]\ []\ i; \mathrm{exit})$$
$$R[d] := d; stop$$

a is found in process *R*, forcing the inference rule of *enable* to reach δ in $c; Q[c]$, whose SDP is $[prefix, instance, hide, choice\hat{}right, prefix, prefix]$.

Acknowledgment

We acknowledge the support of the Telecommunications Research Institute of Ontario; the Natural Sciences and Engineering Research Council of Canada; the Department of Communications; and Bell-Northern Research. We should also acknowledge many discussions with the colleagues of the LOTOS group of the University of Ottawa, and especially Hans van der Schoot.

Bibliography

[Ash92] P. Ashkar. Symbolic Execution of LOTOS Specifications. Master's thesis, University of Ottawa, 1992.

[CS92] T.Y. Cheung and S.Ren. Operational coverage and selective test sequence generation for LOTOS specification, January 1992. TR-92-07.

[Gal89] S. Gallouzi. Trace Analysis of LOTOS Behaviours. Master's thesis, University of Ottawa, 1989.

[GHHL88] R. Guillemot, M. Haj-Hussein, and L. Logrippo. Executing Large LOTOS Specifications. In S. Aggarwal and K. Sabnani, editors, *Protocol Specification, Testing, and Verification VIII*, pages 399–410. North-Holland, 1988.

[GL89] R. Guillemot and L. Logrippo. Derivation of Useful Execution Traces from LOTOS Specifications by using an Interpreter. In K. J. Turner, editor, *Formal Description Techniques*, pages 311–325. North-Holland, Amsterdam, 1989.

[GLO91] S. Gallouzi, L. Logrippo, and A. Obaid. An Expressive Trace Theory for LOTOS. In B. Pehrson B. Johnsson, J. Parrow, editor, *Protocol Specification, Testing, and Verification, XI*, pages 159–175. North-Holland, Amsterdam, 1991.

[HH89] M. Haj-Hussein. ISLA: An interactive system for lotos applications. Master's thesis, University of Ottawa, 1989.

[ISO89] ISO, IS 8807. *Information processing systems - Open systems interconnection - LOTOS - A formal description technique based on the temporal ordering of observational behaviour*, May 1989.

[LOBF88] L. Logrippo, A. Obaid, J.P. Briand, and M.C. Fehri. An interpreter for LOTOS, a specification language for distributed systems. *Software-Practice and Experience*, 18:365–385, 1988.

[QFP88] J. Quemada, A. Fernandez, and S. Pavon. Transformating LOTOS Specifications with LOLA. the parametrized expansion. In K. J. Turner, editor, *Proceedings of the first National Conference on Formal Description Techniques (FORTE)*, pages 45–54. North-Holland, Amsterdam, 1988.

[RKKL85] P. Rety, C. Kirchner, H. Kirchner, and P. Lescanne. NARROWER: a new algorithm for unification and its application to logic programming. In Dijon, editor, *Proceedings of the First International Conference on Rewriting Techniques and Applications*, pages 141–157, 1985.

[Tre89] J. Tretmans. HIPPO: A LOTOS Simulator. In P.H.J. van Eijk, C.A. Vissers, and M. Diaz, editors, *The Formal Description Technique LOTOS*, pages 391–396. North-Holland, 1989.

[vdS92] Hans van der Schoot. Validation Activities for LOTOS Based on Static Data Flow Analysis. Master's thesis, University of Enschede, The Nehterlands, november 1992. Carried out at the University of Ottawa.

[vE88] P.H.J. van Eijk. *Software Tools for the Specification Language LOTOS*. PhD thesis, University of Twente, Netherlands, 1988.

[vE89] P.H.J. van Eijk. The Design of a Simulator Tool. In P.H.J. van Eijk, C.A. Vissers, and M. Diaz, editors, *The Formal Description Technique LOTOS*, pages 351–390. North-Holland, 1989.

[vEE91] Peter van Eijk and Henk Eertink. Design of the LOTOSPHERE symbolic LOTOS simulator. In J.Quemada, J.Manas, , and E.Vazquez, editors, *The Formal Description Technique, III*, pages 577–580. North-Holland, Amsterdam, 1991. Elsevier Science Publishers B.V.