A Hoare-style Proof System for LOTOS

S. Gallouzi[a], L. Logrippo[a], A. Obaid[b]

[a]University of Ottawa, Protocols Research Group, Department of Computer Science, Ottawa, Ontario, Canada, K1N 6N5

[b]Université du Québec à Hull, Département d'informatique, Case postale 1250, succursale B, Hull, Québec, Canada, J8X 3X7

Abstract

A Hoare-style proof system for LOTOS, defined in terms of proof rules based on the structure of processes, is presented. Hoare's satisfaction relation is used to define these rules. This system is based on the failures model. It is shown to be adequate to allow proofs of correctness of compound processes to be constructed from proofs of correctness of its parts. An example, consisting of the proof of a property of the failure set of a two-slot buffer is presented.

1 INTRODUCTION

The established method for proving properties of LOTOS processes is bisimulation. This method, well-known in process algebras (see among others [Par81, Mil80, Mil89]) has been extended to LOTOS in a number of papers [BC89], among which one of the most complete is the PhD thesis of Brinksma [Bri88a]. By using bisimulation, one can prove a LOTOS process equivalent to another LOTOS process. An example of this way of proceeding is presented in papers by [Naj87] and by Shiratori et al. [SKTN90], where service processes are shown to be equivalent to the composition of protocol processes with the underlying service provider.

While the equivalence of processes is an important practical issue in the design of distributed systems, many properties of interest cannot be formulated or proven in this form. For example, one would like to be able to verify invariant properties of processes such as: "if there have been as many inputs as outputs, then a further input is possible". Of course, one could think of transforming in some way such a property into an equivalence property of processes, however it does not appear that this would be the most natural way of proceeding.

Assertions such as the one above correspond to a well-established way of thinking about concurrent processes. For example, in a well-known paper, Hailpern and Owicki

[HO83] introduce the concept of *histories*, i.e. sequences of actions of processes, and show how properties of histories can be formalized in temporal logic. Closer to LOTOS, Hoare et al. [BHR84, Hoa85] formalize the concepts of *traces* and *refusals* of processes and develop methods for proving properties of traces and refusals. Hoare's method is much more precise than the one of [HO83], since it is based on a complete formalization of the properties of his language. Furthermore, while the method of [HO83] is based on assertions on histories, [BHR84] adds to this concept the concept of refusal, which describes the fact that a certain event may be refused after a given sequence of events.

Another advantage of proof methods based on the idea of invariant properties is the compositionality they exhibit. Both [HO83] and [Hoa85] provide rules for proving properties of a process on the basis of the properties of its component processes. Compositionality is a very valuable characteristic of proof methods, especially to help formalizing the design process, by allowing the simultaneous construction of processes and proofs.

Finally, concepts of traces, refusals, and failures have been shown to be important for the development of a testing theory for LOTOS. [Bri88b] uses these concepts based on an operational model and without developing a semantic theory for them. Our paper is a contribution towards a logical and denotational model for these concepts.

A first attempt to use the failures model as an alternative semantic model for LOTOS is reported in [Wes86]. A more recent attempt to define semantic models based on traces, refusals, and failures for LOTOS can be found in [Gal89]. Furthermore, extensions of the failures model and some of its applications to LOTOS proof theory can be found in [Bri88b, Led90]. The latter work was developed at the same time as our work but independently. It proposes an extension of the failure model that distinguishes a behavior expression that cannot initially engage into an internal action from one that fails in exactly the same circumstances and can engage into an internal action. [Led90] does not, however, propose a complete logical proof system for LOTOS based on his extension of the failure model.

One problem with these methods is that they are intrinsically more complex than bisimulation methods. In the latter, the universe of discourse is limited to behavior expressions, and equivalence relations between them, nothing else needs to be mentioned. In methods based on the concept of traces and refusals, one needs to introduce the concepts of traces and refusals, operators on traces and refusals, and properties of traces and refusals. This requires the use of some type of predicate calculus and some arithmetic.

A problem specific to LOTOS is presented by the fact that no generally usable fixpoint induction principle for reasoning about failures has been formulated for this language yet. Although LOTOS is partially based on CSP, the failure fixpoint induction principle of CSP cannot be extended to LOTOS because of the presence of the internal action in LOTOS. While the use of fixpoints appears to be highly desirable, we show in this paper that some interesting types of proofs can be carried out without fixpoints.

2 BASIC DEFINITIONS AND NOTATIONS

2.1 Operators on traces

A *trace* is a finite-length sequence of *symbols*. Symbols are denoted by identifiers and may represent events in a process, often referred to as actions. Each trace may then be interpreted as a sequence of (atomic) interactions that may take place between a process and its environment. A trace will be denoted as follows

- $\langle\rangle$ is the empty trace containing no symbols
- $\langle a, b, \ldots\rangle$ is a trace containing a sequence of symbols where a is followed by b.

Every trace is associated with some *alphabet*, a finite set of symbols. For each alphabet A, A^* denotes the set of all traces, including the empty trace $\langle\rangle$, which are formed from symbols in A. We will need to handle a special symbol denoted by δ, which is used in LOTOS to indicate the successful termination of a process which engages in it, via the **exit** process. Therefore this symbol can appear only at the end of a trace. We will thus let A_δ be the alphabet $A \cup \{\delta\}$ and A_δ^* be the set of all traces of symbols of A_δ, where δ may only occur at the end of a trace.

Next, we introduce several operators on traces that will serve as a basis to develop our proof system. For a full treatment of these operators the reader can refer to [Gal89].

- $t \lfloor A$ denotes the *projection* of a trace t on a set of actions A, which can be obtained from t by omitting all symbols outside A.
- $|t|$ denotes the *length* of a trace t. The number of occurrences of symbols from A in a trace t, denoted $t \downarrow A$, can then be counted by $|t \lfloor A|$. If A contains a single symbol a, the number of its occurrences in the trace t is denoted $t \downarrow a$ (instead of $t \downarrow \{a\}$).
- The *concatenation* operator, denoted by "$\frown$", constructs a trace by putting two traces, u and v, together in this order.
- We introduce three partial orderings on traces whose least element is $\langle\rangle$.
  1. We write $u \preceq v$, to denote that u is an initial segment of v, often called *prefix*.
  2. We write u *in* v to express the fact that a trace u is a contiguous subsequence of a trace v (not necessarily initial).
  3. If u is a subsequence of v, not necessarily a contiguous one, we write $u \sqsubseteq v$.
- If the successful termination symbol δ does not occur at the end of a trace u (in which case it does not occur in u at all), the *sequential composition* of u and v, denoted $u \gg v$, is u. If δ does occur at the end of u, it is removed and the result is concatenated to v.
- The sequential composition uses δ as a glue which sticks two traces u and v together. We define another composition function, called *disruption*, that does not need the glue to stick a prefix of u and v together, and if the glue occurs, v cannot stick. It is denoted by $u[> v$, and can be defined as follows

$$u[> v = \{t \frown s | t \preceq u \wedge (if\ \neg(\langle\delta\rangle\ in\ t)\ then\ s = v\ else\ s = \langle\rangle)\}$$

- Let F be a function mapping symbols in an alphabet A to symbols in an alphabet B such that $F(a) = \delta$ iff $a = \delta$. We define the postfixed *renaming* operation over traces, as follows

  1. $\langle\rangle[F] = \langle\rangle$
  2. $(\langle a\rangle \frown t)[F] = \langle F(a)\rangle \frown (t[F])$

  We shall use convenient abbreviations in writing renamings explicitly. Thus

  $$b_1/a_1, \ldots, b_n/a_n$$

  (where $a_1, \ldots, a_n$ are distinct symbols) stands for the renaming

  $$F : \{a_1, \ldots, a_n\} \longrightarrow \{b_1, \ldots, b_n\}$$

  given by

  1. $F(a_i) = b_i$ if $a_i \in \{a_1, \ldots, a_n\}$
  2. $F(a) = a$ if $a \notin \{a_1, \ldots, a_n\}$

- We define a composition function, called *merging*, to describe the parallel composition of two or more processes. This includes their mutual communication embodied by the common actions specified in their "synchronization alphabet" and the successful termination action. Each communication requires the participation of both processes. The reader will recall that two LOTOS processes P and Q composed by means of the parallel composition operator $|[S]|$, where S is a sequence of actions will

  1. mutually interleave for actions not in S.
  2. mutually synchronize for the actions in S. This means that if P offers action $a \in \{S\}$, so must Q, and the result of their synchronization is a single offer of a.
  3. mutually synchronize on the "exit" action δ.

  If at any point a required synchronization is not possible, a deadlock occurs. This leads to the following definition. Let

  $$\mathcal{M}(v_1, A, v_2) = \{t | v_i \sqsubseteq t \wedge v_i \lfloor A_\delta = t \lfloor A_\delta \wedge |t| = \sum_j |v_j| - (v_i \downarrow A_\delta)\}$$

  where $i, j = 1, 2$. Furthermore, for a set of traces T, trace $t \in T$ is said to be *longest* in T if $|t'| \leq |t|$ for any $t' \in T$. The *merging* of two traces u_1 and u_2 in the order imposed by a synchronization alphabet A, denoted by $u_1|A|u_2$, is the set of longest traces t such that there exists two prefixes v_1 and v_2 of u_1 and u_2, respectively, for which $t \in \mathcal{M}(v_1, A, v_2)$. For example,

$$\langle a,b,a\rangle|\{b\}|\langle b,c\rangle = \{\langle a,b,a,c\rangle, \langle a,b,c,a\rangle\}$$

$$\langle a,b,\delta\rangle|\{\}|\langle c,\delta\rangle = \{\langle a,b,c,\delta\rangle, \langle a,c,b,\delta\rangle, \langle c,a,b,\delta\rangle\}$$

$$\langle a,b,c\rangle|\{b,d\}|\langle d,e\rangle = \{\langle a\rangle\}$$

2.2 Traces and failures of processes

A process definition in LOTOS describes the behavior pattern of a process, by defining the sequences of observable actions that may occur (be observed), over a finite set of actions. The latter is the *process alphabet*, denoted $\alpha(P)$ for a given process P. The main component of a process definition is the *behavior expression* which is built from the constants and operators of LOTOS. A behavior expression B can perform an action x and then behave like B' (another behavior expression). What we derive here are labeled transitions denoted $B \xrightarrow{x} B'$. We shall write $B \xrightarrow{s} B'$, where s denote a sequence of actions (not necessarily observable) $x_1 \ldots x_n$ $(n \geq 0)$, to mean that for some $\mathrm{B_i}$ $(0 \leq i \leq n)$

$$B = B_0 \xrightarrow{x_1} B_1 \xrightarrow{x_2} B_2 \cdots \xrightarrow{x_n} B_n = B'$$

In order to abstract from the invisible actions that are derived by the transition relation when applied to a behavior expression, we define the *trace relation* "$\Longrightarrow$" as follows

Definition 1 *Let t denote a trace $\langle a_1, a_2, \ldots, a_n\rangle$ then we have $B \stackrel{t}{\Longrightarrow} B'$ whenever there exists a sequence $i^{k_0}a_1 i^{k_1} a_2 i^{k_2} \ldots a_n i^{k_n}$ $(k_i \geq 0)$, denoted by s, such that $B \xrightarrow{s} B'$, where i^k denotes a sequence of k internal actions.*

This implies that $B \stackrel{\langle\rangle}{\Longrightarrow} B'$ whenever $B \xrightarrow{i^k} B'$ $(k \geq 0)$, and that, for any B, $B \stackrel{\langle\rangle}{\Longrightarrow} B$ (in this case $k = 0$).

We can now define the trace set and refusal set of a behavior expression B by means of the trace relation.

Definition 2 *Let S be a set of behavior expressions*

$$traces(B) = \{t | \exists B'.B \stackrel{t}{\Longrightarrow} B'\}$$
$$traces(S) = \bigcup_{B\in S} traces(B)$$

Note that the trace set of every process is *prefix-closed.*

Definition 3 *Let $\alpha(E)$ be the alphabet of an environment E in which B is placed and S is a set of behavior expressions*

$$refusals(B) = \{X \subseteq \alpha(E) | \exists B' \cdot B \stackrel{\langle\rangle}{\Longrightarrow} B' \wedge X \cap initials(B') = \{\}\}$$
$$refusals(S) = \bigcup_{B\in S} refusals(B)$$

where $initials(B) = \{a \in \alpha(B) | \exists B' \cdot B \stackrel{\langle a\rangle}{\Longrightarrow} B'\}$

The concept of refusals was first introduced in [BHR84] where the *failures model* is introduced. The latter takes into account what a process may refuse after engaging in an arbitrary trace of its behavior.

Definition 4 *Let B be a behavior expression. We define*

$$failures(B) = \{(t, X) | t \in traces(B) \wedge X \in refusals(B/t)\}$$

where $B/t = \{B' | B \stackrel{t}{\Longrightarrow} B'\}$.

In what follows we will use A and $\{A\}$ to denote a sequence of actions and a set of actions formed by the elements of A, respectively.

3 FAILURES SEMANTICS

In this section we present a set of rules and axioms that define the failures of compound processes in terms of the failures of their components. The result is a failures semantics for LOTOS, restricted to non-divergent behaviors, since failure sets describe all observable aspects of the behavior of a process, and the following axioms and rules provide a means for interpreting LOTOS text as a failure set.

We say that B is *i-prefixed*, and write $i\text{-}prefixed(B)$, If $\exists B'.B \xrightarrow{i} B'$. Obviously, the processes **stop**, **exit**, and any expression prefixed by an observable action are not i-prefixed. However,

1. $i\text{-}prefixed(\mathbf{i};\ B) \equiv true$
2. $i\text{-}prefixed(\mathbf{hide}\ A\ \mathbf{in}\ B) \equiv (initials(B) \cap \{A\} \neq \{\}) \vee i\text{-}prefixed(B)$
3. $i\text{-}prefixed(B_1 \gg B_2) \equiv i\text{-}prefixed(B_1) \vee \delta \in initials(B_1)$
4. $i\text{-}prefixed(B_1 * B_2) \equiv i\text{-}prefixed(B_1) \vee i\text{-}prefixed(B_2)$

where $*$ stands for $[]$, $|[A]|$, and $[>$. Note that a process instantiation is i-prefixed if the behavior expression defining that process is i-prefixed. If P is recursive and not guarded then $i\text{-}prefixed(P)$ is undefined. $i\text{-}prefixed(B)$ is also undefined whenever it is undefined for any of the component expressions of B.

Let E denote the environment which interacts with any of the indicated processes.

F 1 $failures(\mathbf{stop}) = \{(\langle\rangle, X) | X \subseteq \alpha(E)\}$

F 2 $failures(\mathbf{exit}) = \{(\langle\rangle, X) | X \subseteq (\alpha(E) - \{\delta\})\} \cup \{(\langle\delta\rangle, X) | X \subseteq \alpha(E)\}$

F 3 $failures(a;\ B) = \{(\langle\rangle, X) | X \subseteq (\alpha(E) - \{a\})\} \cup \{(\langle a\rangle \frown t, X) | (t, X) \in failures(B)\}$

F 4 $failures(\mathbf{i};\ B) = failures(B)$ *if B does not diverge.*

F 5 $failures(\mathbf{hide}\ A\ \mathbf{in}\ B) = \{(t \lfloor (\alpha(B) - \{A\}), X) | (t, X \cup \{A\}) \in failures(B)\}$
if $\forall t \in traces(B) \cdot \neg diverges(B/t, \{A\})$

where $diverges(B, S) \equiv (\forall n \cdot \exists t \in traces(B) \cap S^* \cdot |t| > n)$ and S is a set of behavior expressions. Note that in the latter definition B can denote a set of behavior expressions.

F 6 *If "***process** $P\ [a'_1, \ldots, a'_n] := B_p$ **endproc**" *is a process definition and F is a renaming given by $a_1/a'_1, \ldots, a_n/a'_n$ then* $failures(P\ [a_1, \ldots, a_n]) = \{(t[F], F(X)) | (t, X) \in failures(B_p)\}$

F 7 *If* $i\text{-}prefixed(B_j)$ $(j = 1, 2)$, *and* $\neg i\text{-}prefixed(B_k)$ $(k = 3, 4)$
then a) $failures(B_1 [] B_2) = failures(B_1) \cup failures(B_2)$

b)
$$failures(B_2 [] B_3) = failures(B_3 [] B_2)$$
$$= \{(t, X) | (t, X) \in failures(B_2) \vee (t \neq \langle\rangle \wedge (t, X) \in failures(B_2) \cup failures(B_3))\}$$

c)
$$failures(B_3 [] B_4)$$
$$= \{(t, X) | (t, X) \in (failures(B_3) \cap failures(B_4)) \vee (t \neq \langle\rangle \wedge (t, X) \in failures(B_3) \cup failures(B_4))\}$$

F 8 $failures(B_1|[A]|B_2)$
$= \{(t, X \uplus_{\{A\}_\delta} Y) | \exists u, v \cdot t \in u|\{A\}|v$
$\wedge (u, X) \in failures(B_1) \wedge (v, Y) \in failures(B_2)\}$

where $X \uplus_S Y = (X \cap Y) \cup ((X \cup Y) \cap S)$ and S is a set of actions.

F 9 $failures(B_1|||B_2)$
$= \{(t, X \uplus_{\{\delta\}} Y) | \exists u, v \cdot t \in u|\{\}|v$
$\wedge (u, X) \in failures(B_1) \wedge (v, Y) \in failures(B_2)\}$

F 10 $failures(B_1||B_2)$
$= \{(t, X \cup Y) | \exists u, v.t \in u|\alpha(B_1) \cup \alpha(B_2)|v$
$\wedge (u, X) \in failures(B_1) \wedge (v, Y) \in failures(B_2)\}$

F 11 $failures(B_1 \gg B_2) = \{(t, X) | (t, X \cup \{\delta\}) \in failures(B_1)\}$
$\cup \{(u \frown v, X) | u \frown \langle \delta \rangle \in traces(B_1) \wedge$
$(v, X) \in failures(B_2)\}$
if B_2 does not diverge.

F 12 $failures(B_1[> B_2) = \{(t, X) | \exists u, v \cdot t \in u[> v \wedge$
$(if\ v = \langle\rangle\ then\ (u, X) \in failures(B_1)$
$else\ u \in traces(B_1)) \wedge (v, X) \in failures(B_2)\}$
if $\neg i\text{-}prefixed(B_2)$
$= \{(t, X) | \exists u, v \cdot t \in u[> v \wedge$
$u \in traces(B_1) \wedge (v, X) \in failures(B_2)\}$
otherwise.

Note that contrary to CSP, the failures $\subseteq$-ordering is not monotonic for all LOTOS operators (i.e. [] is not monotonic for this ordering). Therefore, the general fixpoint theory, which implies in the case of a complete partial order (for which all operators are continuous) that any fixpoint equation has a unique minimal and maximal solution, cannot be applied in LOTOS by means of the failures $\subseteq$-ordering. This is also the case for CCS, because no complete partial order, that is continuous for all its operators, can be found. The implications of this fact is that failures of recursive behavior expressions cannot be computed, at least not by what appears to be the immediately obvious method of fixpoint induction.

However, a new preorder on processes, denoted by $\sqsubseteq$, can be defined which is based on the failures $\subseteq$-ordering but is preserved by all LOTOS operators. Such preorder is defined as follows

$B_1 \sqsubseteq B_2$ iff for every LOTOS context $\mathcal{C}[]$ we have $failures(\mathcal{C}[B_1]) \subseteq failures(\mathcal{C}[B_2])$

A direct characterization of this preorder can be given

$B_1 \sqsubseteq B_2$ iff $failures(B_1) \subseteq failures(B_2) \wedge$ $i\text{-}prefixed(B_1)$ implies $i\text{-}prefixed(B_2)$

The failures $\subseteq$-ordering and the preorder defined above coincide with some of the relations in [DH84] in the case of non-divergent processes.

Alternately, one can develop an adequate version of LOTOS which does not use the internal action combinator. Instead we replace the choice and disable operators by two new choice and two new disable operators respectively, where each pair of operators represents internal and external nondeterminism. The failures semantics of the resulting language would be much simpler and the failures $\subseteq$-ordering would be preserved by the new version of LOTOS operators. This point remains, however, to be elaborated. A similar approach was taken by [DH87] to develop Testing CCS (i.e. CCS without τ's).

4 PROOF SYSTEM

The required behavior of a process can be described in terms of some observable aspects of its behavior. The most relevant observations, that we have mentioned, are the trace of actions that occur at some moment in time and the sets of actions that might be refused (i.e. refusals) at that moment in time. This description is then a predicate containing two free variables t and r, that stand for an arbitrary trace and refusal set, respectively, of the process being defined, together with, possibly, some other variables.

A behavior requirement describes the characteristics of a process's failure set. Therefore, there is a need for a notion of a process P satisfying its behavior requirements S. In this context, P can be viewed as an implementation of S, since it gives a behavioral description of the system specified in S.

4.1 Satisfaction relation

The fact that a process P meets a behavior requirement S, can be modeled as a binary relation "$\models$". We write $\models$ in an infix manner and $P \models S$ may be read "P *satisfies* S", which means that S is a property that is true for every possible observation of the behavior of P; or, more formally

$$\forall t, r \cdot t \in traces(P) \wedge r \in refusals(P/t) \Rightarrow S(t, r)$$

In other words, S is true if its variables t and r take values observed from the process P. Following Hoare, we will sometimes write $S(t)$ or $S(t, r)$ to indicate that a behavior requirement S contains a free variable t, or two free variables t and r, respectively, whenever there is a need to show how t or r, or both, may be substituted by some more elaborated expression.

The satisfaction relation $\models$ is defined by means of the same rules governing Hoare's **sat** relation [Hoa85]. These rules provide the most general properties of the satisfaction relation. They apply to all kinds of processes and all kinds of specifications, and are based on the structure of the specification. Our objective is to design a framework for the verification of statements about the behavior of LOTOS processes, which requires additional rules based on their structure. Such rules should permit proof of the correctness of a compound process to be constructed from a proof of correctness of its parts. We shall give these rules in the next section and refer to them as the *proof rules*. This leads to a Hoare-style proof system for LOTOS.

4.2 Proof rules

The proof rules permit the use of formal reasoning to ensure that a LOTOS behavior expression B meets its specification S. They also provide other means of characterizing processes, in that the behavior of a process can be modeled by logical assertions on its traces and refusals, and possibly some other aspects of that.

The reader should beware that, in the list of rules given below, different occurrences of the symbols t and r in the same rule may refer to different traces and refusals, respectively. Also recall that, as stated at the beginning of section 4.1, the variables t and r are always supposed to be universally quantified in such a context. This shorthand is due to Hoare [Hoa85].

A behavior requirement will be written as S, $S(t)$, or $S(t, r)$, according to the context. Sometimes, we will use a single underline character (i.e. $S(t, \_)$) to denote a variable that is not relevant to the context.

The process **stop** is completely inactive, and so the only possible observation of its behavior is the empty trace since it refuses everything, including every subset of the alphabet of the environment in which this process is placed. Therefore, every refusal of **stop** is a subset of this alphabet. This is consistent with the fact that every process refuses every subset of actions that is not in its alphabet, and the alphabet of **stop** is empty.

S 1 $\mathbf{stop} \models t = \langle\rangle \wedge r \subseteq \alpha(E)$

The process **exit** refuses every set not containing the successful termination action δ, after which it behaves like **stop**

S 2 $\mathbf{exit} \models ((t = \langle\rangle \wedge \delta \notin r) \vee (t = \langle\delta\rangle \wedge r \subseteq \alpha(E)))$

Every trace of the expression $(a;\ B)$ is either empty, or has a as its head and its tail is a possible trace of B. Therefore, the behavior requirements of B must describe its tail. Initially, when $t = \langle\rangle$, the (observable) action a cannot be refused

S 3 *If* $B \models S(t)$
then $(a;\ B) \models ((t = \langle\rangle \wedge a \notin r) \vee (t = \langle a\rangle \frown t' \wedge S(t')))$

All rules below assume that the behavior expressions involved in the following properties do not diverge; they do not engage in an unbounded sequence of internal actions.

The behavior requirements of B must describe every trace and refusal of $(\mathbf{i};\ B)$, since the **i**-action is not observable

S 4 *If* $B \models S$
then $(\mathbf{i};\ B) \models S$

A behavior expression involving hiding can only interact with its environment when it reaches a state in which it cannot engage in any further hidden actions. The proof rule for hiding is restricted to the case where the behavior expression B does not diverge immediately on hiding actions in a set A

S 5 *If* $B \models (\neg diverges(B, A) \wedge S(t, r))$
then $(\mathbf{hide}\ A\ \mathbf{in}\ B) \models (\exists u \cdot t = u \lfloor (\alpha(B) - A) \wedge S(u, r \cup A))$

Every trace and refusal set of a process instantiation is the result of renaming a trace and a refusal set, respectively, described by the behavior requirements of the behavior expression defining that process

S 6 *Let "***process** $P\ [a'_1, \ldots, a'_n] := B_p$ **endproc***" be a process definition*
and F be a renaming given by $a_1/a'_1, \ldots, a_n/a'_n$
If $B_p \models S(t,r)$
then $P\ [a_1, \ldots, a_n] \models (\exists u \in traces(B_p) \cdot t = u[F] \wedge$
$\exists X \in refusals(B_p/u) \cdot r = F(X) \wedge S(u,X))$

Any observation of the behavior $(B_1[]B_2)$ must be a possible observation of either B_1 or B_2, and so it must be described by at least one of their behavior requirements. If both operands are i-prefixed, every possible observation of the behavior of $(B_1[]B_2)$ will be a possible observation for B_1 or B_2. However, if only B_1 is i-prefixed, initially, when $t = \langle\rangle$, $(B_1[]B_2)$ cannot refuse a set of actions X unless it is refused by B_1. Furthermore, if neither B_1 nor B_2 is i-prefixed, $(B_1[]B_2)$ cannot refuse X unless it is refused by both B_1 and B_2, whenever $t = \langle\rangle$. Therefore

S 7 *If $i\text{-}prefixed(B_j)$ $(j = 1,2)$, and $\neg i\text{-}prefixed(B_k)$ $(k = 3,4)$*
and $B_j \models S_j$ and $B_k \models S_k$
then a) $(B_1[]B_2) \models (S_1 \vee S_2)$
b) $(B_2[]B_3) \models (if\ t = \langle\rangle\ then\ S_2\ else\ (S_2 \vee S_3))$
c) $(B_3[]B_4) \models (if\ t = \langle\rangle\ then\ (S_3 \wedge S_4)\ else\ (S_3 \vee S_4))$

Note that the right-hand-side of rule *b)* applies also to $(B_3[]B_2)$.

The following describes the proof rules associated with the parallel composition, sequential composition, and disruption. This description is based on the fact that every trace and refusal of such compound processes can be described as equations between the traces and refusals , respectively of their components

S 8 *If* $B_1 \models S_1(t,r)$
and $B_2 \models S_2(t,r)$
then $(B_1|[A]|B_2) \models (\exists u, v, X, Y \cdot t \in u|A|v \wedge$
$r = X \uplus_{\{A\}_\delta} Y \wedge S_1(u,X) \wedge S_2(v,Y))$

S 9 *If* $B_1 \models S_1(t,r)$
and $B_2 \models S_2(t,r)$
then $(B_1 \gg B_2) \models (\exists u, v \cdot t = u \gg v \wedge (if\ \langle\delta\rangle\ in\ u\ then\ (S_1(u,\_) \wedge$
$S_2(v,r))\ else\ (S_1(u,r_\delta) \wedge S_2(v,\_))))$

The rule for [> needs to distinguish two cases. If its second operand is not i-prefixed, then every refusal of the compound expression is either a possible refusal of both operands or a possible refusal of the second operand if the first one was already interrupted. Otherwise, every refusal of the compound expression is a refusal of the second operand before or after disruption. Therefore

S 10 *If* $B_1 \models S_1(t,r)$
and $B_2 \models S_2(t,r)$
then $(B_1[> B_2) \models (\exists u, v \cdot t \in u[> v \wedge (if\ v = \langle\rangle\ then\ S_1(u,r)$
$else\ S_1(u,\_)) \wedge S_2(v,r))$
$if \neg i\text{-}prefixed(B_2)$
$\models (\exists u, v \cdot t \in u[> v \wedge S_1(u,\_) \wedge S_2(v,r))$
$otherwise.$

5 EXAMPLE

Consider the definition of a Two Slot Buffer (TSB) using two one slot simplex buffers (as defined in [ISO88]-tutorial), which can accept up to two inputs at a time, and perform one or two outputs depending on what was input. Its behavior can formally be described as

process TSB [$input$, $output$] :=
hide mid **in**
SB [$input$, mid]
|[mid]|
SB [mid, $output$]
endproc

where the one slot simplex buffer (SB) is defined as follows

process SB [$input$, $output$] :=
$input$; $output$; SB [$input$, $output$]
endproc

We wish to prove that
$TSB[input, output] \models S(t,r)$
where

$S(t,r) = (0 \leq t \downarrow input - t \downarrow output \leq 2)$
$\wedge$(if $t \downarrow input = t \downarrow output$
then $input \notin r$
else $output \notin r$)

In other words: at any given time there can be at most two undelivered messages; also, if there have been as many inputs as outputs, a further input is possible, else a further output is possible.

We first prove that the one slot buffer SB satisfies its behavior requirements.

Lemma

$SB[input, output] \models (0 \leq t \downarrow input - t \downarrow output \leq 1)$
$\wedge$(if $t \downarrow input = t \downarrow output$
then $input \notin r$
else $output \notin r$)

Proof. By induction on the length of t using S 3 and S 6. □

Let

$$\begin{aligned} R_{[x,y]}(t,r) =& (0 \le t\downarrow x - t\downarrow y \le 1) \\ & \wedge (\text{if } t\downarrow x = t\downarrow y \\ & \quad \text{then } x \notin r \\ & \quad \text{else } y \notin r) \end{aligned}$$

where x and y denote variables that are bound to some actions. Also let

$A = \{input, output\}$

$B' = SB[input, mid]|[mid]|SB[mid, output]$

$B =$ **hide** mid **in** B'

Note that $\alpha(B') = \{input, output, mid\}$

Proof. First Note that by S 6 and lemma above, we have

$SB[input, mid] \models R_{[input,mid]}(t,r)$

since the renaming is one-to-one. Similarly, we can show that

$SB[mid, output] \models R_{[mid,output]}(t,r)$

By S 8 it follows that

$$\begin{aligned} B' \models & \exists u, v \cdot t \in u|\{mid\}|v \wedge R_{[input,mid]}(u,w) \wedge R_{[mid,output]}(v,z) \\ & \wedge r = w \uplus_{\{mid\}_\delta} z \end{aligned}$$

Note that

$t \in u_1|S|u_2 \Rightarrow \exists v_1, v_2 \cdot v_i \preceq u_i \wedge t \in v_1|S|v_2$ such that

a) $t\downarrow S = v_i \downarrow S$

b) $t\downarrow S' = v_1 \downarrow S' + v_2 \downarrow S' - v_i \downarrow (S \cap S')$

where S and S' denote sets of actions. It follows that

$$\begin{aligned} B' \models & \exists u, v \cdot t \in u|\{mid\}|v \\ & \wedge 0 \le (u\downarrow input - u\downarrow mid + v\downarrow mid - v\downarrow output) \le 2 \\ & \wedge t\downarrow A = (u\downarrow A + v\downarrow A) \wedge (t\downarrow mid = u\downarrow mid = v\downarrow mid) \\ & \wedge R_{[input,mid]}(u,w) \wedge R_{[mid,output]}(v,z) \wedge r = w \uplus_{\{mid\}_\delta} z \end{aligned}$$

Also note that

if $t\downarrow input = t\downarrow output$

then $u\downarrow input = u\downarrow mid \wedge v\downarrow mid = v\downarrow output$

else $u\downarrow input \neq u\downarrow mid \vee v\downarrow mid \neq v\downarrow output$

and $u\downarrow output = v\downarrow input = 0$. Therefore

$$\begin{aligned} B' \models & 0 \le (t\downarrow input - t\downarrow output) \le 2 \\ & \wedge (\text{if } t\downarrow input = t\downarrow output \text{ then } input \notin r \\ & \quad \text{else } (input \notin r \wedge output \notin r) \vee mid \notin r \vee output \notin r) \end{aligned}$$

$\Rightarrow$ [by S 5, since clearly $traces(B') \cap \{mid\}^* = \{\langle\rangle\}$ and $mid \in r \cup \{mid\}$]

$$\begin{aligned} B \models & \exists u \cdot t = u\lfloor A \wedge 0 \le (u\downarrow input - u\downarrow output) \le 2 \\ & \wedge (\text{if } u\downarrow input = u\downarrow output \text{ then } input \notin r \cup \{mid\} \\ & \quad \text{else } output \notin r \cup \{mid\}) \end{aligned}$$

Note now that this last property is almost identical to $S(t,r)$.

To see that it can be reduced to $S(t,r)$, note that

$(u\lfloor A)\downarrow a = u\downarrow a$ if $a \in A$

is also true and that

$x \notin S \cup S' \Rightarrow x \notin S$

Therefore, $\{mid\}$ can be removed yielding $S(t,r)$. Hence

$B \models S(t,r)$ □

6 CONCLUSION

The work presented in this paper is closely related to CSP, and favors a proof theory for LOTOS based on failure equivalences, since two failure equivalent processes satisfy the same behavior requirements. It provides a useful mathematical framework for the analysis of LOTOS specifications and for developing logical systems to prove their properties. This is illustrated by the proof rules that permit a proof of correctness of a compound process to be constructed from a proof of the correctness of its parts.

Our proof system is restricted to strongly convergent processes. This is not regarded as a serious limitation, since divergence is never an intentional result in the attempted definition of a process. Also, the absence of some types of divergence (such as divergence caused by hiding) can be proven by using the method itself. It remains, however, to extend our system to cope with value expressions in LOTOS specifications.

Acknowledgment

This work was supported by the Natural Science Research Council of Canada, The Telecommunication Research Institute of Ontario, the Tunisian government, and the Canadian International Development Agency.

References

[BC89] Tommaso Bolognesi and Maurizio Caneve. Equivalence verification: Theory, algorithms and a tool. In P.H.J. van Eijk, C.A. Vissers, and M. Diaz, editors, *The Formal Description Technique LOTOS*. North-Holland, 1989.

[BHR84] S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential. *JACM*, 31(3):560–599, 1984.

[Bri88a] E. Brinksma. *On the Design of Extended LOTOS*. PhD thesis, University of Twente, The Netherlands, 1988.

[Bri88b] E. Brinksma. A theory for the derivation of tests. In S. Aggarwal and K. Sabnani, editors, *Protocol Specification, Testing, and Verification VIII*, pages 63–74. North-Holland, 1988.

[DH84] R. DeNicola and M. Hennessy. Testing equivalences for processes. *TCS*, 34:83–133, 1984.

[DH87] R. DeNicola and M. Hennesy. CCS without τ's. In *Proceedings of the International Joint Conference on Theory and Practice of Software Development*. LNCS volume 249, Springer-Verlag, 1987.

[Gal89] Souheil Gallouzi. Trace analysis of LOTOS behaviours. Master's thesis, University of Ottawa, Canada, 1989.

[HO83] Brent T. Hailpern and Susan S. Owicki. Modular verification of computer communication protocols. *IEEE Transactions on Communications*, COM-31(1):56–68, 1983.

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

[ISO88] ISO, IS 8807. *Information processing systems - Open systems interconnection - LOTOS - A formal description technique based on the temporal ordering of observational behaviour*, May 1988.

[Led90] Guy Leduc. *On the Role of Implementation Relations in the Design of Distributed Systems using LOTOS.* PhD thesis, Université de Liège, 1990.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science.* Sringer-Verlag, 1980.

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

[Naj87] E. Najm. A verification oriented specification in LOTOS of the transport protocol. In H. Rudin and C.H. West, editors, *Protocol Specification, Testing, and Verification VII*, pages 181–203. North-Holland, 1987.

[Par81] David Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Theoretical Computer Science, 5th GI-Conference.* LNCS volume 104, Springer-Verlag, 1981.

[SKTN90] N. Shiratori, H. Kaminaga, K. Takahashi, and S. Noguchi. A verification method for LOTOS specifications and its application. In E. Brinksma, G. Scollo, and C. Vissers, editors, *Protocol Specification, Testing, and Verification IX.* North-Holland, 1990. To appear.

[Wes86] A.J.G. Wester. CSP as an alternative model for LOTOS. Technical Report SEDOS/C2/, Technical University of Twente, The Netherlands, 1986.