A Denotational Semantics for LOTOS

Alessandro Fantechi, Martina Ferro, Stefania Gnesi, Giuliana Tamorri

Nota Interna B4-21
Luglio 1991
A Denotational Semantics for Full LOTOS

A. Fantechi*, M. Ferro**, S. Gnesi*, G. Tamorri**

* I.E.I. - C.N.R. Pisa
** Università di Pisa - Dipartimento di Informatica

ABSTRACT

A denotational semantics for LOTOS is proposed in order to provide a formal tool to verify properties of LOTOS specification and to verify equivalences of LOTOS processes in a simple way.

The defined semantics is a linear temporal one, given using a compositional approach, that relates a temporal logic formula to each language construct. Such semantics is proved fully abstract with respect to the trace equivalence defined on the operational semantics of the language. Also, we investigate on the capability of other semantics to be fully abstract with respect to stronger equivalences, because of the weakness of the trace equivalence.

1. INTRODUCTION

LOTOS is one of the "Formal Description Techniques" standardized by ISO for the formal specification of the services and protocols of the Open System Interconnection architecture; beyond that, LOTOS is a powerful formalism for the specification of concurrent systems in general.

Our interest in LOTOS is due to the fact that it has a few number of operators and at the same time it allows expressive power. So, LOTOS gives the possibility of describing user readable, concurrent programs.

LOTOS has been given a formal semantics operationally: this semantics describes the possible moves a LOTOS process can perform in its execution. Equivalences on LOTOS processes have been defined, so that processes behaving in the same way can be given the same formal meaning.

A more desirable situation for a programming a specification language is to have a denotational semantics, which allows to associate a mathematical object to every term of the language. Denotational semantics have, as an advantage over operational semantics, the possibility of giving a meaning to every single piece of a program in a modular way and without resorting to the need of having the entire program. In order to provide modularity in the specification of LOTOS processes, semantics may be compositional.

In particular, for a language as LOTOS, which is able to describe the behaviour of non-finite state processes, denotational semantics allows to compute the meaning of any process in a finite amount of time, due to its syntax-driven nature, what is not possible for operational semantics, which forces instead to execute every possible computation of the process in question.

In defining the denotational semantics of a language, the first step is the choice of a
suitable domain for the denotations of language terms. LOTOS terms describe the possible executions of concurrent processes, so suitable denotations for them should be mathematical objects retaining such information.

Also, in order to provide modularity in the specification of LOTOS processes, semantics must be compositional. A semantics $S$ is compositional with respect to an $n$-adic operator $Op$ if and only if

$$\exists f : D_S^n \rightarrow D_S \text{ such that } \forall p_1, \ldots, p_n \in \mathcal{P} . S(\text{Op}(p_1, \ldots, p_n)) = f(S(p_1), \ldots, S(p_n)),$$

where $\mathcal{P}$ is the object language, $D_S$ is the set representing the range of the semantic function $S$, $f$ is an operator (or a composition of operators) on the semantic domain chosen [Fis 87].

Sets of linear execution traces are the simplest objects that can embed the desired information on LOTOS processes; as we will see, this choice is not completely satisfactory, but is a good starting point for a definition of a denotational semantics for LOTOS. A pro for this choice is that there exists a powerful language able to describe sets of execution traces and to reason on them: linear Temporal Logic.

Each linear temporal logic formula can be seen as describing a set of execution traces: from here the idea of associating to each LOTOS term a temporal logic formula as its denotation.

The use of Temporal Logic for the description of concurrent systems and verification of their properties is well known: this gives us, as a side effect, the opportunity to analyze LOTOS specifications by means of the techniques developed for Temporal Logic. Indeed, Temporal Logic [Lam 80, MP 89, Pnu 86] is a semantic domain suitable to model the properties of concurrent systems, and it is also suitable in the process of deriving an implementation from a LOTOS specification, as a higher abstraction level description. Temporal Logic is a formalism able to express events occurring in different moments (termination, enabling, etc.) and which cannot be directly expressed by usual logics (Propositional Calculus, First Order Logic).

Temporal Logic can be used in order to describe both properties satisfied from LOTOS specification and the representations of each LOTOS construct.

Moreover, temporal semantics approach allows that:

i) the verification that a program satisfies a given property (expressed by a temporal logic formula) is a verification that the formula expressing the temporal meaning of the program logically implies the given property formula;

ii) the verification of the equivalence of programs is a verification of their logic equivalence.

We start choosing a linear version of Temporal Logic where at each moment only one future is possible, through which LOTOS behaviour expressions are modelled as finite or infinite, discrete in time, sequences of transactions. Then, since our temporal semantics is given using a linear time temporal logic, it should be able to capture typical properties which are expressible with linear temporal logic; these have been usually divided into two classes [Lam 83]:

2
- safety properties, which state that something bad never happens, i.e. that the program can never enter into an undesirable state (partial correctness, absence of deadlock, etc.); 
- liveness properties, which state that something good will eventually happen, i.e. that the program will eventually enter into a desirable state (termination, fairness, etc.).

The temporal semantics will be able to express all the properties in these classes if and only if it will be expressive with respect to a proper equivalence on the operational semantics. The temporal semantics for LOTOS presented in this paper is expressive with respect to its operational semantics modulo trace (or string) equivalence [Hoa 81]; this semantics allows us to prove those safety properties which can be stated for a LOTOS process, but not liveness properties.

In the end, we will see how we have attempted the extension of the present definition in order to be consistent with finer equivalences (maximal trace equivalence, bisimulation, ...) and the problems encountered in this direction, which has made such extension much harder then it might appear at a first sight.

2. SYNTAX AND OPERATIONAL SEMANTICS FOR LOTOS

LOTOS, as defined in [DIS 8807], describes the behaviour of processes, their synchronization and interprocess value communication.

The syntax of a LOTOS specification and of a LOTOS process is:

specification Id [Gatelist](Varlist) : functionality
  type definition
  behaviour behaviour expression
  where type definition, process definition endspec

process Id [Gatelist](Varlist) : functionality :=
  behaviour expression
  where type definition, process definition endproc

The representation of values, value expressions and data structures in LOTOS are derived from the specification language for abstract data types ACT ONE. ACT ONE is an algebraic specification method to write unparameterized as well as parameterized specifications.

However in this context, we do not give the semantics for the type definition; hence, every function of values management will be not defined, because we consider them defined in a previous phase of static analysis.

In table 1 we recall the syntax and the operational semantics for LOTOS, as defined in [DIS 8807].
The operational semantics of this language is based on the concept of "Labelled Transition Systems" (LTSs). A LTS is a 4-uple \((S, \Sigma, \rightarrow, s_0)\) such that \(S\) is a set of states, \(\Sigma\) is a set of actions, \(\rightarrow \subseteq (S \times \Sigma \times S)\) is the labelled transition relation, \(s_0 \in S\) is the initial state.

To describe the operational semantics for LOTOS we indicate with \(\mathbf{V}\) the set of definable values of LOTOS, \(G\) the set of user definable gates, \(i\) the unobservable action, \(\mathbf{Act}\) the set \(\{g < v^+ > | g \in G, v^+ \in \mathbf{V}\} \cup \{i\}\). Moreover, we use a fictitious gate \(\delta\), not user definable, to indicate the successful termination of a process, then \(\mathbf{Act}^+\) is the set \(\mathbf{Act} \cup \{\delta < v^* > | v^* \in \mathbf{V}\}\).

The function \textit{name}, defined on \(\mathbf{Act}^+\), returns the gate on which the action occurs.

Then if we indicate with \(\mathbf{BE}\) the universe of LOTOS behaviour expressions, our labelled transition relation is \(\rightarrow \subseteq (\mathbf{BE} \times \mathbf{Act}^+ \times \mathbf{BE})\).

<table>
<thead>
<tr>
<th>operator</th>
<th>syntax</th>
<th>operational semantics</th>
<th>informal meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inaction</td>
<td>stop</td>
<td>(i;B)</td>
<td>models an event internal to the process.</td>
</tr>
<tr>
<td>Unobservable</td>
<td>action</td>
<td>(i;B)</td>
<td>(i;B - i \rightarrow B)</td>
</tr>
<tr>
<td>Observable</td>
<td>action</td>
<td>(g\times{\text{cond}};B) \text{ with } n \times \cdots n \times {\text{act}}\text{ and } \mathbf{di} \text{ of the form}</td>
<td>(g\times B - g &lt; v^+ &gt; \rightarrow B)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\forall\text{ (accepts } x \in \mathbf{E})) \text{ if } {\text{open } \mathbf{E}}</td>
<td>(g\times B - g &lt; v^+ &gt; \rightarrow B \text{ and } {\text{cond}} = \text{true}) \text{ implies } g\times{\text{cond}};B - g &lt; v^+ &gt; \rightarrow B)</td>
</tr>
<tr>
<td>Parallel</td>
<td>composition</td>
<td>(B_1 [L] B_2)</td>
<td>(B_1 - a \rightarrow B_1' \text{ and } \text{name}(a) \notin L) \text{ implies } B_1[L]B_2 - a \rightarrow B_1'[L]B_2) (B_2 - a \rightarrow B_2' \text{ and } \text{name}(a) \notin L) \text{ implies } B_1[L]B_2 - a \rightarrow B_1'[L]B_2) \text{ and } \text{name}(a) \notin L \cup {\delta} \text{ implies } B_1[L]B_2 - a \rightarrow B_1'[L]B_2)</td>
</tr>
<tr>
<td>Choice</td>
<td>(B_1 [</td>
<td></td>
<td>] B_2)</td>
</tr>
<tr>
<td>Generalized</td>
<td>choice (x : t [</td>
<td></td>
<td>] B)</td>
</tr>
<tr>
<td>Syntax</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| choice $g \in \{g_1, \ldots, g_n\} \mid B$ | this operator allows a choice between the different instances of $B(g_i)$.
| $B[g_i] \vdash a^+ \rightarrow B' \land g_i \in \{g_1, \ldots, g_n\}$ implies $\ldots$ | |
| exit($E^*$) | this operator models the process able to emit a successful termination signal. |
| exit($E^*$) $\cdot \delta \in \{E^*, \} \rightarrow \text{stop}$ | |
| Enabling $B_1 \rightarrow B_2$ | this operator allows sequential process composition without value passing. |
| $B_1 \cdot a \rightarrow B_1'$ implies $B \cdot a \rightarrow B_1' \land B_2'$ | |
| $\delta \cdot \delta \rightarrow \text{stop}$ implies $\ldots$ | |
| $B \cdot i \rightarrow B_2$ | and this one with value passing, $\ldots$ |
| $B = B_1 \gg \text{accept } x_1 \cdot t_1, \ldots, x_m \cdot t_m$ in $B_2$ | $\ldots$ |
| $B_1 \cdot a \rightarrow B_1'$ implies $B_1 \cdot a \rightarrow B_1' \gg \text{accept } x_1 \cdot t_1, \ldots, x_m \cdot t_m$ in $B_2$ | |
| $B_1 \cdot i \rightarrow B_2 \{i \cdot t_1, \ldots, k \cdot t_m\}$ where $\forall i, k \in \mathbb{N}$ | $\ldots$ |
| Disabling $B_1 \gg B_2$ | this operator allows process $B_1$ to be disabled by process $B_2$. |
| $B_1 \cdot \delta \cdot \delta \rightarrow B_1'$ implies $B_1 \gg B_2' \cdot \delta \cdot \delta \rightarrow B_1'$ | |
| $B_1 \cdot a \rightarrow B_1'$ implies $B_1 \gg B_2' \cdot a \rightarrow B_1'$ | |
| $B_2 \cdot a^+ \rightarrow B_2$ implies $B_1 \gg B_2 \cdot a^+ \rightarrow B_2$ | $\ldots$ |
| $B_1 \gg B_2 \cdot a^+ \rightarrow B_2$ | $\ldots$ |
| Hiding $g_1, \ldots, g_n$ in $B$ | this operator allows actions at gates in the list $g_1, \ldots, g_n$ to be "hidden". |
| $B \cdot a^+ \rightarrow B'$ and $\text{name}(a^+) \notin \{g_1, \ldots, g_n\}$ implies $\ldots$ | |
| hide $g_1, \ldots, g_n$ in $B$ $\ldots$ | $\ldots$ |
| $B \cdot a^+ \rightarrow B'$ and $\text{name}(a^+) \notin \{g_1, \ldots, g_n\}$ implies $\ldots$ | $\ldots$ |
| Guarded expression $\{\text{cond}\} \rightarrow B$ | this operator allows the execution of $B$ if the guard evaluation is true. |
| $B \cdot a^+ \rightarrow B'$ and $\{\text{cond}\} = \text{true}$ implies $\{\text{cond}\} \rightarrow B$ | $\ldots$ |
| Let $x_1 \cdot t_1 = E_1, \ldots, x_m \cdot t_m = E_m$ in $B$ | this operator allows to define variables and give them values. |
| $B \{E_1, \ldots, E_m\} \cdot x_1 \ldots x_m \cdot a^+ \rightarrow B'$ implies $\ldots$ | $\ldots$ |
| (let $x_1 \cdot t_1 = E_1, \ldots, x_m \cdot t_m = E_m$ in $B$) $\cdot a^+ \rightarrow B'$ | $\ldots$ |
| Process instantiation $P[h_1, \ldots, h_n](E_1, \ldots, E_m)$ | the actions of a process instantiation are those of the process declaration body, with the substitution of formal parameters with actual ones. |
| $B \cdot p \{h_1, \ldots, h_n\} \{g_1, \ldots, g_n\} \{E_1, \ldots, E_m\} \cdot x_1 \ldots x_m \cdot a^+ \rightarrow B'$ implies $\ldots$ | $\ldots$ |
| $P[h_1, \ldots, h_n](E_1, \ldots, E_m) \cdot a^+ \rightarrow B'$ | $\ldots$ |
| **Table 1** | |

Note that we have used the unique notation, "|L|", for all parallel composition constructs (full synchronization, pure interleaving, communication).
With the attributes \( d_i \) we can define different kinds of offer that a process can do. "!E" is a value declaration and represent the value expression offered by the process on a gate; "?x:t" is a variable declaration and represent the set of values that the process is ready to accept on a gate. In LOTOS the variables are single assignment ones. The scope rules for the variable declaration are straightforward; the scope of a variable \( x \) that is declared in an action is the behaviour expression following that action in an action prefix construct.

Interaction between two processes can take place if both processes have enabled one or more identical events. In the follow we list the different synchronizations that can occur between two processes that, for simplicity, offer only one attribute.

<table>
<thead>
<tr>
<th>process A</th>
<th>process B</th>
<th>synchronization condition</th>
<th>interaction sort</th>
<th>effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>g!E1</td>
<td>g!E2</td>
<td>( g\in1 = g\in2 )</td>
<td>value matching</td>
<td>synchronization</td>
</tr>
<tr>
<td>g!E</td>
<td>g?x:t</td>
<td>( g\in x = t )</td>
<td>value passing</td>
<td>after synchronization ( x = g\in )</td>
</tr>
<tr>
<td>g?x:t</td>
<td>g?y:u</td>
<td>( t = u )</td>
<td>value generation</td>
<td>after synchronization ( x = y = v ) for ( v \in t )</td>
</tr>
</tbody>
</table>

3. A PARTICULAR TEMPORAL LOGIC FOR LOTOS

For the particular denotational semantics we are going to define, we need a proper Temporal Logic \( T \) able to describe the meaning of every LOTOS specification, that is which satisfies adequacy and expressiveness requirements.

To do this we need the definition of a binary relation \( \vdash \subseteq P \times T \), usually called satisfaction relation, written "\( p \vdash \phi \)" and read "\( p \) satisfies the property \( \phi \)"; through which a process \( p \) is identified with the set of properties (formulae in \( T \)) which it satisfies. The definition of this relation induces an equivalence between processes which have the same property. Formally, we define

\[
F(p) = \{ \phi : \phi \in T, p \vdash \phi \}
\]

hence

\[
p \equiv_T q \iff F(p) = F(q)
\]

Now if \( = \subseteq P \times P \) is the equivalence induced by the native operational semantics of the processes language \( P \), it is possible to define a relation between \( = \) and \( \equiv_T \).

Definition
A logic \( T \) is **adequate** [Hen 85] with respect to an equivalence \( ( = ) \) defined on a given processes language \( P \); if for every pair of processes \( p, q \in P \):

\[
p \equiv_T q \iff p = q
\]

Definition
A logic \( T \) is **expressive** [Pnu 85] with respect to an equivalence \( ( = ) \) defined on a given processes language \( P \); if for every process \( p \in P \) exists a characteristic
formula (or logic semantics) \( L(p) \in \mathcal{T} \) such that:

i) \( p \vdash L(q) \) iff \( p = q \)

ii) \( p \vdash \phi \) iff \( L(p) \Rightarrow \phi \)

The standard linear Temporal Logic (TL) results a logic adequate but not expressive w.r.t. the trace equivalence because of some LOTOS operators (parallel composition, enabling, disabling, hiding and recursive process instantiations). In fact TL is a decidable logic and even if we are able to give the semantics to processes that use this operators, this semantics is not able to imply every property (described by a logic formula) satisfied by those processes. For this reason we present a powerful formalism, \( TL_{\mathcal{FL}} \), that is the logic TL enriched by three operators (maximal fixed point, chop, relabelling), which is expressive w.r.t. the trace equivalence.

A model \( \sigma \) of \( TL_{\mathcal{FL}} \) is a sequence, finite or infinite, of actions; \( \sigma(i) \) denotes the \( i \)-th action in the sequence; an interpretation is a pair \((\sigma, i)\) where \( \sigma \) is a model and \( i \) is a discrete instant of time, that is, is a positive integer.

In table 2 we present the interpretation of \( TL_{\mathcal{FL}} \) operators. Here, we indicate with "a" a generic element of the set of logical action predicates which coincides with the set \( \text{Act}^+ \).

<table>
<thead>
<tr>
<th>Prepositional Calculus operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\sigma, i) \models \text{true} ) if ( \sigma(i) = a )</td>
</tr>
<tr>
<td>((\sigma, i) \not\models \text{false} ) if ( (\sigma, i) \not\models \phi )</td>
</tr>
<tr>
<td>((\sigma, i) \models a ) if ( (\sigma, i) \models \phi ) and ((\sigma, i) \models \psi )</td>
</tr>
<tr>
<td>((\sigma, i) \models \phi \land \psi ) if ( (\sigma, i) \models \phi ) or ((\sigma, i) \models \psi )</td>
</tr>
<tr>
<td>((\sigma, i) \models \phi \lor \psi ) if ( (\sigma, i) \models \phi ) and ((\sigma, i) \models \psi )</td>
</tr>
<tr>
<td>((\sigma, i) \models \neg \phi ) if ( (\sigma, i) \models \phi )</td>
</tr>
<tr>
<td>((\sigma, i) \models (\sigma \cdot \sigma) ) if ( (\sigma, i) \models \phi ) and ((\sigma, i) \models \psi )</td>
</tr>
<tr>
<td>((\sigma, i) \models (\sigma \cdot \sigma^+) ) if ( (\sigma, i) \models \phi )</td>
</tr>
<tr>
<td>((\sigma, i) \models (\sigma \cdot \sigma^+) ) if ( (\sigma, i) \models \phi )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Temporal operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\sigma, i) \models \text{Op} ) if ( (\sigma, i) \models \phi )</td>
</tr>
<tr>
<td>((\sigma, i) \models \text{Cl} ) if ( \text{for all } j \geq i : (\sigma, j) \models \phi )</td>
</tr>
<tr>
<td>((\sigma, i) \models \text{F} ) if ( \text{exists } j \geq i : (\sigma, j) \models \phi )</td>
</tr>
<tr>
<td>((\sigma, i) \models \text{F}_\infty \psi ) if ( \text{exists } k &lt; i : (\sigma, k) \models \psi ) for all ( j \geq k ) and ( (\sigma, i) \models \phi )</td>
</tr>
<tr>
<td>((\sigma, i) \models \text{F} \psi ) if ( (\sigma, i) \models \text{Cl} \psi ) or ( (\sigma, i) \models \text{F} \psi )</td>
</tr>
<tr>
<td>((\sigma, i) \models \text{F} \psi ) if ( (\sigma, i) \models \text{Cl} \psi ) or ( (\sigma, i) \models \text{F} \psi )</td>
</tr>
<tr>
<td>((\sigma, i) \models \text{V}_\infty \chi(\xi) ) if ( (\sigma, i) \models \chi(\xi) ) for all ( k &lt; i : (\sigma, k) \models \chi^k(\text{true}) )</td>
</tr>
<tr>
<td>((\sigma, i) \models \text{F} \psi ) if ( (\sigma, i) \models \psi ) and ( (\sigma^+, i) \models \phi )</td>
</tr>
<tr>
<td>((\sigma, i) \models \text{F} \psi ) if ( (\sigma, i) \models \psi ) and ( (\sigma^+, i) \models \phi )</td>
</tr>
</tbody>
</table>

| Let us see the informal meaning of the last three operators, which are unusual also to the reader acquainted with temporal logic. |

The maximal fixed point constructor, \( \text{vF}_\infty \chi(\xi) \), denotes the maximal solution to \( \xi = \chi(\xi) \), which exists if the function \( \chi(\xi) \) is monotonic and continuous.
The formula $\phi \land \psi$, using the chop operator [Bar 84], holds for a sequence $\sigma$ which can be decomposed in a finite prefix $\sigma'$ and a suffix $\sigma''$ respectively satisfying $\phi$ and $\psi$, or for an infinite sequence $\sigma$ satisfying $\phi$. More precisely we use the "weak" chop operator as defined in [Bar 84, Ros 86].

The formula $\phi[P/b]$ uses a logical operator which has the effect of substituting every occurrence of $b$ in the model sequence of the formula $\phi$ with $p$; this operator, which is different from a syntactic substitution of every occurrence of $b$ with $p$ in the formula $\phi$, has been named relabelling [FGL 89].

4. A DENOTATIONAL SEMANTICS FOR LOTOS

Our aim is to give a temporal semantics for LOTOS, studying what is the finest equivalence which allows to maintain full abstractness, and, at the same time, to preserve the compositionality constraint. In order to give the temporal semantics, we need some definitions.

Definition
A semantics $D$ is fully abstract with respect to operational semantics $O$, if and only if:

$$\forall p, q \in P . D[p] = D[q] \Rightarrow O[p] = O[q] \text{ and } \forall \text{ context } C . O[C(p)] = O[C(q)] \Rightarrow D[p] = D[q]$$

Definition
A semantics $D$ is simply abstract with respect to operational semantics $O$, if and only if:

$$\forall p, q \in P . D[p] = D[q] \iff O[p] = O[q].$$

Now, we give a temporal semantics for LOTOS defining the characteristic function "$L$" which associates a temporal formula to each language construct: models of each formula are (partial) computations of the relative construct. $L$ will be defined by means of a set of syntax directed clauses, as is usually done when giving denotational semantics [FGL 89]. Since we want to give a compositional semantics, we will refer to the approach developed by Barringer, Kuiper and Pnueli [Bar 84, Bar 85, Bar 87]. In this approach, the semantics of a process must be open in order to obtain compositionality, that is the process is described immersed in all possible (parallel) environments. The semantics is given by closing the process semantics with all possible environments. This allows us to describe the meaning of the interleaving, induced by parallel composition operator. In fact the meaning of a process is a sequence of its own actions possibly interleaved with environment actions. When one process is composed in parallel with another, its semantics is partially closed because part of the external environment of the first process becomes known, since some of the
environment actions of the first process are actions performed by the other.

To denote an unknown environment action, a special proposition "e" is added to the set of action predicates Act⁺. The resulting definition is:

\[ \text{ACT} = \{ g<v^+> \mid g \in G, v^+ \subseteq V \} \cup \{ \delta<v^*> \mid v^* \subseteq V \} \cup \{ i \} \cup \{ e \} \]

The generic action predicate "g<v^+>" is true when an action offers the not empty list of values v^+ on gate g; "v^*" indicates a possible empty list of values; "i" indicates the action predicate relative to the unobservable action.

Characteristic function L is defined by structural induction; its type is:

\[ L : \text{BE} \rightarrow \text{Env} \rightarrow TL_{\mathcal{F}\mathcal{L}} \]

where BE is the space of behaviour expressions of our language, TL_{\mathcal{F}\mathcal{L}} is the space of temporal formulae and Env is the space of environments, that is maps from process identifiers to functions from lists of gates and variables to temporal logic formulae:

\[ \text{Env} : \text{Id} \rightarrow (L^* \rightarrow (V^* \rightarrow TL_{\mathcal{F}\mathcal{L}})) \]

where L^* \subseteq G and V^* \subseteq \text{Var} can be empty sets.

Since LOTOS process identifiers can be user definable, it will be needed to remember a context in which we will investigate every construct. This context will be extended, when new process declarations will be met, through the semantic function D_1 that permits to define the function D that constructs the environment starting from the space of the processes declarations, its type is:

\[ D : \text{Decl}^* \rightarrow \text{Env} \]

where Decl^* is a process declarations list; the type of D_1 is:

\[ D_1 : (\text{Decl} \times \text{Env}) \rightarrow \text{Env} \]

Note that the semantic functions do not need a parameter "Store", that holds the variable-value bindings. In fact, we have chosen to present the simpler notation of our semantics, that allows to manipulate the values of variables directly, because of the LOTOS variables are single assignment variables. In effect, we could have proposed some semantic functions for the management of this store. In this manner we could have a "classic" denotational semantics, in which for every program fragment we know all variable-value bindings. So doing, we will loose the simplicity of our semantics, that, on the other hand, surely holds all the bindings when we give the semantics to a complete specification.

Finally, the semantics of LOTOS specifications is given by the following semantic function:

\[ P : \text{Spec} \rightarrow TL_{\mathcal{F}\mathcal{L}} \]

where Spec is the space of LOTOS specifications.

The clauses defining the semantic functions are listed below. Note that in the
following we distinguish between syntactic substitution and semantic substitution by different parentheses: "[...]" and "[...]" respectively.

**Inaction**

\[ \mathcal{L}_0(\text{stop}) = e \mathcal{W} \text{ false} \]

The meaning of an inaction behaviour is true for any (possibly infinite) sequence of environment actions. The \( \mathcal{W} \) operator is used in this clause because the process \text{stop} will execute actions only if the environment is able to perform them: the process will terminate when the environment is ready to terminate.

**Action Prefix**

**Unobservable action:**

\[ \mathcal{L}_0(i; B) = e \mathcal{W} (i \land \mathcal{L}_0(B)) \]

The meaning of an unobservable action followed by a behaviour \( B \) is true for every sequence of environment events followed by the unobservable action predicate and by sequences for which the meaning of \( B \) is true; the possibility that the environment performs an infinite sequence of actions is considered by the \( \mathcal{W} \) operator.

**Observable action:**

\[ \mathcal{L}_0(\text{g} \pi; B) = e \mathcal{W} (\mathsf{h}'^{\pi}(\pi) (g < \mathsf{h}'(\pi) > \land \mathcal{L}_0(B)) ) \]

\[ \mathcal{L}_0(\text{g} \pi \text{[cond]}; B) = e \mathcal{W} (\mathsf{h}'^{\pi}(\pi) ((g < \mathsf{h}'(\pi) > \land \beta(\text{cond}) \land \mathcal{L}_0(B)) ) \]

where \( \mathsf{h}' \), \( \mathsf{h}'' \) are functions that allow us to write the chosen notation for the structured action prefix; \( \mathsf{h}' \) builds a list of n-adic \textit{or} operators, \( \mathsf{h}'' \) builds a list of functions. Semantic domain and definitions of \( \mathsf{h}' \) and \( \mathsf{h}'' \) are given in table 3.

---

**Auxiliary functions**

\( \mathsf{h}' : \text{Attr} \to (\text{TL} \to \text{TL}) \)

"Attr "is the syntactic category of the attributes (?x1, !E)

\[ \mathsf{h}'(? x : t \pi) = \vee_{x \in t} \mathsf{h}'(\pi) \]

\[ \mathsf{h}'(? x : t) = \vee_{x \in t} \]

\[ \mathsf{h}'(! E \pi) = \mathsf{h}'(\pi) \]

\[ \mathsf{h}'(! E) = \text{nil} \]

"nil" semantically represents the identity, syntactically is the void string

\( \mathsf{h}'' : \text{Attr} \to \text{Val}^+ \)

\[ \mathsf{h}''(? x : t \pi) = x, \mathsf{h}''(\pi) \]

\[ \mathsf{h}''(? x : t) = x \]

\[ \mathsf{h}''(! E \pi) = \varepsilon(E), \mathsf{h}''(\pi) \]

\[ \mathsf{h}''(! E) = \varepsilon(E) \]

N.B. remember that in these contexts variables are used as placeholders for values.

---

**table 3**

The semantics of a structured action followed by a behaviour \( B \) is true for every sequences of environment events followed by the predicate that expresses the offers on
g of single values and sets of values (attributes), followed by sequences for which the meaning of B is true. The offers on g are translated by the logical n-adic "or "\(\lor_{x \in t}\)" : this is a metalinguistic notation that we used to condense the formula whose extended notation would have a number of operators equal to the cardinality of domain t \(^{(1)}\).

Moreover, the LOTOS language admits the possibility to have a condition that limits the execution of the action. This is translated by a logical and between the structured action predicate and the evaluation of the condition.

For clarity see the following example:

let be \(n = \{1,6\}\) and \(b = \{\text{true}, \text{false}\}\) and suppose \(B(x,y)\) be a notation that underlines the scopes of \(x\) and \(y\), then

\[
\begin{align*}
& L_\theta( g ?x:n !x+1 ?y:b [x<5] ; B(x,y) ) = \\
& = e \nu ((\forall x \in \text{Nat} \forall y \in \text{Bool} ( (g<x,e(x+1),y) \land \beta(x<5)) \land O L_\theta(B(x,y))) ) = \\
& = e \nu (((g<1,2,\text{true}) \land \text{true}) \land O L_\theta(B(1,\text{true})) \\
& \lor \\
& (((g<1,2,\text{false}) \land \text{true}) \land O L_\theta(B(1,\text{false})) \\
& \lor \\
& (((g<6,7,\text{true}) \land \text{false}) \land O L_\theta(B(6,\text{true})) \\
& \lor \\
& (((g<6,7,\text{false}) \land \text{false}) \land O L_\theta(B(6,\text{false}))) = \\
& = e \nu (((g<1,2,\text{true}) \land O L_\theta(B(1,\text{true})) \\
& \lor \\
& (g<1,2,\text{false}) \land O L_\theta(B(1,\text{false})) \\
& \lor \\
& \text{false} \\
& \lor \\
& \text{false} ) = \\
& = e \nu (((g<1,2,\text{true}) \land O L_\theta(B(1,\text{true})) \\
& \lor \\
& (g<1,2,\text{false}) \land O L_\theta(B(1,\text{false})))
\end{align*}
\]

**Parallel Composition**

\[
L_\theta(B1 \parallel B2 ) = \\
= (L_\theta(B1) [ (evi2 \lor \forall g \in L, g \in \alpha(B2), v^+ \in \text{Val} (g(2v^+))/e, i_1 /i, \forall g \in L, g \in \alpha(B1), \text{v}^+ \in \text{Val} : g(1v^+)/g(1v^+) ]
\]

\(^{(1)}\) t is a finite set of elements of the same type belonging to the set of LOTOS definable types. Defining t as a finite domain, in the temporal formulae every operator will have a finite number of operands, that is we manipulate a finitary logic. Hence with "t" we will indicate the generic finite set.
\[ L_\theta(B2) \left[ \left( \exists v \in \mathcal{V} \forall g \in L, g \in \alpha(B1), v^+ \subseteq \text{Val} : g^1_1 < v^+ > \right) /_{e_1}, i_{12} /_{i}, \forall g \in L, g \in \alpha(B2), v^+ \subseteq \text{Val} : g^2_2 < v^+ > /_{g^2_2 < v^+ >} \right] \]

\[ \left[ \forall g \in L, v^+ \subseteq \text{Val} : g^1_1 < v^+ > /_{g^1_1 < v^+ >}, g^2_2 < v^+ > /_{g^2_2 < v^+ >}, i_{11}, i_{12} \right] \]

The logic relabelling operator is here liberally extended to express a set of substitutions. The last relabelling applies to the result of the conjunction. Function \( \alpha \), defined on behaviour expressions, gives the set of actions that a behaviour expression may perform during its execution (this set of actions can be produced by a static analysis of the behaviour expression - actually, the static analysis produces a larger set than that strictly needed but, due to the use of subscripts, all the unnecessary \( g \), which are not present in the matching sequence, will be dropped by the conjunction operation).

Parallel composition partially closes the open semantics of the component processes. In order to explain the formula we must: consider carefully the exact meaning of the LOTOS parallel composition: actions of the composed processes are interleaved, except for those actions occurring at gates belonging to the set \( L \); these actions are performed simultaneously by both processes. Hence, given the various actions that the component processes can perform, we can distinguish the following cases:

1. If \( B1 \) performs an \( i \) action (\( \in \text{Act}^+ \)), it will be considered as an environment action for \( B2 \), thus an \( i \) in \( L_\theta(B1) \) (\( \in \text{ACT} \)) must be paired with an \( e \) in \( L_\theta(B2) \) (and vice versa).
2. If \( B1 \) performs an action on gate \( g \), with \( g \in L \), it will be considered as an environment action for \( B2 \), thus it must be paired with an \( e \) in \( L_\theta(B2) \) (and vice versa).
3. If \( B1 \) performs an action on gate \( g \), with \( g \in L \), \( B2 \) must also perform an action on the same gate. If two processes perform an action on different gates, even if in \( L \), the result of the conjunction will be false.
4. If \( B1 \) performs a successful termination action, \( B2 \) must also perform this action.
5. Since the parallel composition should continue to have an open semantics, it must be open to environment actions which are considered as such for both \( B1 \) and \( B2 \); thus, an \( e \) in \( L_\theta(B1) \) should be paired with an \( e \) in \( L_\theta(B2) \).

This pairing is performed by the conjunction of the component meanings. Suppose the next action performed by \( B1 \) is \( x \) (\( \in \text{Act}^+ \)), whose semantic translation is \( x(x \in \text{ACT}) \), and by \( B2 \) is \( y \) (\( y \in \text{Act}^+ \)), whose semantic translation is \( y(y \in \text{ACT}) \); this means that:

\[ L(B1) \Rightarrow x \text{ and } L(B2) \Rightarrow y \]

hence

\[ L(B1) \land L(B2) \Rightarrow x \land y \]

Now, if \( x \) and \( y \) are different, the last formula will have no model since our logic admits only models for which only an action predicate at a time is true. This is
satisfactory for cases 3,4,5. In the other cases we need a set of action predicates, ACT1, so defined:

\[
ACT1 = ACT \cup \{ i1, i2 \} \cup \{ \delta^*<v^*> : \delta<v^*> \in ACT \} \cup \\
\cup \{ g1<v^+>, g2<v^+> : g<v^+> \in ACT \}
\]

This indexing will be useful to indicate the actions performed by one of the two processes only, that is will be useful to realize the "ownership" on actions (the ownership concept is not a LOTOS concept, but is necessary to describe the interleaving).

So, we should substitute the e predicates in \( L_\theta(B2) \) (or in \( L_\theta(B1) \)) with an i1 (or i2) in the case1, or action predicate g1<v^+> (or g2<v^+>) in the case 2.

In conclusion we would produce the following simpler formula which expresses the temporal meaning of B1!|B2:

\[
(L_\theta(B1) [(e^{v^+} \lor g \in L, g \in \alpha(B2), v^+ \in \text{Val}g<v^+>)/e] ) \\
\land \\
(L_\theta(B2) [(e^{v^+} \lor g \in L, g \in \alpha(B1), v^+ \in \text{Val}g<v^+>)/e] )
\]

However this formula could confuse actions performed by B1 with the same actions by B2. To overcome this problem we must identify (by a substitution) the owner of the i-action and of the action offered on the gate g (g \( \in \) L). These substitutions are semantic ones, and for this we use the logical operator relabelling.

**Choice**

\( L_\theta(B1|B2) = L_\theta(B1) \lor L_\theta(B2) \)

The meaning of the choice operator is true if the meaning of one of the component behaviours is true.

**Generalized Choice**

\( L_\theta(\text{choice } x:t [ ] B) = \lor_{x \in t} (L_\theta(B)) \)

The meaning of this operator is the disjunction of the open semantics of B, where in every operand of the "\( \lor_{x \in t} \)" notation every occurrence of the symbol x is a value belonging to the set t.

\( L_\theta(\text{choice } g \text{ in } [g1, ..., gn] [ ] B) = \lor g \in \{g1, ..., gn\} (L_\theta(B)) \)

The meaning of this operator is the disjunction of the open semantics of B, where in every operand of "\( \lor g \in \{g1, ..., gn\} \)" notation every occurrence of g is a gate belonging to the set \( \{g1, ..., gn\} \).

**Successful Termination**
\[ L_\theta(\text{exit } (E^*)) = e_\omega (\delta < E^* >) \land O e_\omega \text{ false } \]

The meaning of the successful termination is the same as a "\( \delta \)-action" followed by the meaning of stop.

**Enabling**

\[ L_\theta(B1 >> B2) = \]

\[ = (L_\theta(B1) \ [(\text{evi}2 \forall g \in \alpha(B2), v^+ \subset \text{Val} \ g^2 < v^+ >)/c, i^{1}_{/12}, \forall g \in \alpha(B1), v^+ \subset \text{Val} : g^1 < v^+ >/g < v^+ >, \delta ^* < >/\delta < > ] \]

\[ \land \]

\[ (e_\omega (\delta ^* < >) \land O L_\theta(B2))) \ [(\text{evi}1 \forall g \in \alpha(B1), v^+ \subset \text{Val} \ g^1 < v^+ >)/c, i^{2}_{/i}, \forall g \in \alpha(B2), v^+ \subset \text{Val} : g^2 < v^+ >/g < v^+ >)] \]

\[ [\forall g \in \alpha(B1), v^+ \subset \text{Val} : g^< v^+ >/g^1 < v^+ >, \forall g \in \alpha(B2), v^+ \subset \text{Val} : g^< v^+ >/g^2 < v^+ >, i_{/i1}, i_{/i2}, i_{/\delta ^*} < >] \]

\[ L_\theta(B1 >> \text{accept } x1:t1, ..., xn:tn \text{ in } B2) = \]

\[ = (L_\theta(B1) \ [(\text{evi}2 \forall g \in \alpha(B2), v^+ \subset \text{Val} \ g^2 < v^+ >)/c, i^{1}_{/1}, \forall g \in \alpha(B1), v^+ \subset \text{Val} : g^1 < v^+ >/g < v^+ >, \delta ^* < v^1, ..., vn >/\delta < v^1, ..., vn >] \]

\[ \land \]

\[ (e_\omega (\forall \forall \forall \forall (\delta ^* < x1, ..., xn >) \land O L_\theta(B2))) \ [(\text{evi}1 \forall g \in \alpha(B1), v^+ \subset \text{Val} \ g^1 < v^+ >)/c, i^{2}_{/i}, \forall g \in \alpha(B2), v^+ \subset \text{Val} : g^2 < v^+ >/g < v^+ >)] \]

\[ [\forall g \in \alpha(B1), v^+ \subset \text{Val} : g^< v^+ >/g^1 < v^+ >, \forall g \in \alpha(B2), v^+ \subset \text{Val} : g^< v^+ >/g^2 < v^+ >, i_{/i1}, i_{/i2}, i_{/\delta ^*} < v^1, ..., vn >] \]

We consider the enabling operators similar to the parallel composition operator, since their meanings are analogue to the parallel composition between the process B1 (where the relabelling \[ \delta ^* < >/\delta < > \], or \[ \delta ^* < v^1, ..., vn >/\delta < v^1, ..., vn > \], is applied to the characteristic formula of B1 depending on the presence of value passing) and the process whose syntactical notation is "\( \delta ^* ; B2 \)" or "\( \delta ^* \cdot x1, ..., xn; B2 \)".

**Disabling**

\[ L_\theta(B1 [> B2) = \]

\[ = (L_\theta(B1) \land \delta < v^* >) \]
\( (L_\theta(B_1) \land \Box \neg \delta<v*>) \subseteq L_\theta(B_2) \)

The meaning of disabling is achieved by using the logical *chop* operator. The models which satisfy the formula \( L_\theta(B_1) \rightarrow B_2 \) are all those sequences which have a "\( \delta \)-action" and satisfy the formula \( L_\theta(B_1) \) or are all those models which have as prefix model (without a "\( \delta \)-action") which satisfies \( L_\theta(B_1) \) and as a postfix a model which satisfies \( L_\theta(B_2) \). We recall that a model of \( L_\theta(B_1) \) is any prefix of the complete behaviour of \( B_1 \).

**Hiding**

\( L_\theta(\text{hide } L \text{ in } B) = L_\theta(B) [ \forall g \in L, v^+ \subseteq \text{Val} : i^g_{\langle v^* \rangle} ] \)

The meaning of hiding is true for every sequence which satisfies the meaning of its operand, in which every occurrence of the actions is substituted by unobservable actions.

**Guarded Expression**

\( L_\theta(\text{[cond]} \rightarrow B) = e(\beta(\text{cond}) \land (L_\theta(B))) \)

The meaning of the guarded expression construct is the meaning of \( B \), if the valuation of condition is true, is the meaning of inaction otherwise.

**Let**

\( L_\theta(\text{let } x_1:=E_1, \ldots, x_n:=E_n \text{ in } B) = L_\theta(B[ E_1, \ldots, E_n ] / x_1, \ldots, x_n) \)

The meaning of let construct is the meaning of \( B \) in which every occurrence of the variables \( x_1, \ldots, x_n \) is substituted by the values of the corresponding expressions.

**Process Instantiation - Process Declaration**

\( L_\theta(P[L](E*)) = \theta(P) ((L)(\varepsilon(E*))) \)

The meaning of a process call is obtained by applying the environment to the identifier \( P \) and to the, possibly empty, actual parameters lists \( L \) the gate list, \( \varepsilon(E*) \) the evaluated expressions list.

Till now we present how to build the semantic clauses for the behaviour expressions; in the following we see how to build the environment:

\[ \mathcal{D} (\text{decls}) = \forall x. \bigcup \mathcal{D}_1(\text{decl}_i, x) \]

\[ \mathcal{D}_1(\text{process } P[Q](\text{V:Types}) := B_P \text{ endproc}, x) = \]

\[ = (P, \lambda YZ \cdot L_x( B_P[Y/Q] [Z/V] )) \]

The definition of \( \mathcal{D}_1 \) returns a pair for a single declaration. The pair consists of a process heading and the meaning of the corresponding behaviour expression, parameterized by the gates used and by the values of the given variables. Since process
calls can be recursive, the meaning of behaviours should be evaluated in the overall
environment.

The definition of \( P \) is:

\[
P(\text{decls}; B) = \mathcal{L}_D(\text{decls})(B)
\]

where \( \text{decls} \) is a possibly empty list of declarations.

The proposed semantics presents an advantage due to the structural simplicity of the
characteristic function \( \mathcal{L} \), which does not need a state parameter nor its management
functions to manipulate values. In fact value management is carried out by logical
operators of the characteristic formula [FT 91].

Moreover, this semantics can be proved fully abstract w.r.t. the strong trace
equivalence; in fact it is a compositional semantics and it is proved simply abstract (see
Appendix B).

5. ANALYZING THE PROPOSED SEMANTICS

In this context, we have proposed a denotational semantics for LOTOS language,
following a compositional approach and using a linear temporal logic as semantic
domain.

Compositional approach allows us to treat the LOTOS modularity, in other words
we are able to analyze concurrent programs properties even if they are developed
separately.

Linear temporal logic as semantic domain allows us to treat linear properties of
LOTOS processes. From this point of view, this is not satisfactory because trace
 equivalences [Hoa 81] are weak equivalences, in fact the branching structure of non-
deterministic processes is lost. Moreover our semantics allows us to prove safety
properties, but not liveness ones.

In order to prove liveness properties we should have a semantic which is fully
abstract w.r.t. the maximal trace equivalence (whose definition, together with
definitions of simulation, bisimulation and 2/3 bisimulation equivalences, are in
Appendix A), instead of one which is fully abstract w.r.t. only the trace equivalence,
such as ours. Looking at a semantic that results fully abstract w.r.t. the maximal trace
 equivalence, the problems arise when we try to give the meaning of LOTOS parallel
 composition construct. The \( \text{TFLF} \) operators, that permit to describe the semantics for
the total LOTOS language, are not so powerful to translate the parallel composition
construct. In fact, it can be proved that a semantic which is fully abstract w.r.t. the
maximal trace equivalence exists only for a subset of LOTOS, but not for the total
LOTOS language, and, for the moment, the possibility to find new logical operators
able to describe such construct is not yet verified.
In order to prove liveness properties of LOTOS specifications, we can then try to follow another approach looking at a bisimulation semantics.

In order to describe such semantics we need a branching temporal logic as semantic domain. That prevents us from using the approach proposed in [Bar 87] in which a method is given able to release the open semantics, which allows to express the interleaving of processes, but suitable for linear semantics only. Moreover, in [FGR 91] is proved that in the branching approach we loose compositional requirement. In that context the study is developped starting from a subset of Basic LOTOS (action prefix, non deterministic choice and recursion) : for this subset a compositional branching temporal semantics fully abstract w.r.t. the bisimulation semantics is provided. When this subset is enriched with the full synchronization, it is proved that it is not possible to define such a semantics, even using a standard branching temporal logic as the target logic. Rather, a weaker equivalence, that is the simulation, is reached. The simulation is an equivalence that results insensitive to computation interruptions; moreover it appears to be the strongest equivalence w.r.t. which it is possible to give fully abstract compositional temporal semantics using standard logics.

So, even if we could arrive to a branching semantics simply abstract w.r.t. the bisimulation, defining new logical operators, it will be never proved fully abstract w.r.t. bisimulation, since it does not satisfy the compositional requirement.

Then the semantics proposed in this context, even if is a weak one and so apparently not very interesting, actually is the only one able to reflect the initial constraint, i.e. to be fully abstract w.r.t. a particular equivalence defined on operational semantics of the language.

6. A PROTOTYPAL SUPPORT ENVIRONMENT

The presented temporal semantics has served as a basis for the realization of a prototypical verification environment, with the aim of testing experimentally the proposed verification techniques.

This environment is structured in two phases: the first is a semantic generation phase, the second is the valid formulae generation phase:

```
  LOTOS Process → Semantic Generator → Temporal Semantics → \mathcal{LFL}^* Properties → Valid Formulae
```

The semantic generator is based on the software tool "Synthesizer Generator", that permits to create specialized editors, tailored on a particular language. The Synthesizer Generator creates a language-specific editor from a specification of the language.
abstract syntax, context-sensitive relationships, display format, concrete input syntax, and transformation rules for restructuring objects. The feature that makes the Synthesizer Generator unique is its use of an immediate-computation paradigm to perform analysis, translation, and error reporting while an object is being edited [RT 89].

The Synthesizer Generation specifications are written in the functional language Synthesizer Specification Language (SSL).

The first phase [FT 91] is realized as

\[ L_\theta : \text{LOTOS Abstract Syntax} \to \mathcal{T}L_{FL} \text{ Abstract Syntax} \]

Its components are an editor and a translator whose definitions are strictly related:

A third component is necessary because the maximal fixed point \( \mathcal{T}L_{FL} \) operator allows one to express the semantics of recursive processes. Thus we must calculate the logic expression that has to give the maximal fixed point operator. Then, we have to take in mind that to synthesize the formula that expresses the semantics of a process instantiation, we must consider the possibility of recursive calls in the process body. This is done by a normalization function phase (Normalizator).

The final structure of the semantic generator is:

The valid formulae generator verifies the validity of a temporal formula \( \phi \rightarrow \psi \), where \( \phi \) is a formula that represents a system specification and \( \psi \) is the property that we want to verify.
The valid formulae generator included in the environment uses a decision procedure, that is an algorithm that determines the validity of a formula, in a finite number of the steps. We already know, however, that $\mathcal{TL}_L$ is not decidable; so the valid formulae generator works for a sublanguage $\mathcal{TL}_L^*$, which however allows us to express most of the interesting verification situations.

The implementation of such a tool is realized in COMMON LISP language [Cop 91].

REFERENCES


20
Appendix A

On LTSs several equivalences are defined; we will considered only "strong" equivalences, which do not distinguish between observable and unobservable actions.

**Definition 2.1**

$\sigma \in (\text{Act}^\dagger)^\ast$ is a partial computation (or trace) of a process $p \in \mathcal{P}$, if:

there exist $\alpha_1, \alpha_2, \ldots, \alpha_n \in \text{Act}^\dagger$ and $p_1, p_2, \ldots, p_n \in \mathcal{P}$ such that

\[
\sigma = \alpha_1 \alpha_2 \cdots \alpha_n \quad \text{and} \quad p \rightarrow \alpha_1 \rightarrow p_1 \rightarrow \alpha_2 \rightarrow \cdots \rightarrow p_{n-1} \rightarrow \alpha_n \rightarrow p_n
\]

Moreover we define the set of traces of $p$ as $\mathcal{ST}(p)$; then $p, q \in \mathcal{P}$ are trace equivalent ($p \sim q$) if their sets of traces are the same (i.e. $\mathcal{ST}(p) = \mathcal{ST}(q)$).

**Definition 2.2**

$\sigma \in (\text{Act}^\dagger)^\ast \cap (\text{Act})^0$ is a maximal trace of a process $p \in \mathcal{P}$, if:

the length of $\sigma$ is infinite, or

there exist $\alpha_1, \alpha_2, \ldots, \alpha_n \in \text{Act}$, $\alpha_n \in \text{Act}^\dagger$ and $p_1, p_2, \ldots, p_{n-1}, p_f \in \mathcal{P}$ such that

\[
\sigma = \alpha_1 \alpha_2 \cdots \alpha_n \quad \text{and} \quad p \rightarrow \alpha_1 \rightarrow p_1 \rightarrow \alpha_2 \rightarrow \cdots \rightarrow p_{n-1} \rightarrow \alpha_n \rightarrow p_f \quad \text{and do not exist} \quad \alpha \in \text{Act}^\dagger, q \in \mathcal{P} \quad \text{such that} \quad p_f \rightarrow \alpha \rightarrow q.
\]

Processes $p, q \in \mathcal{P}$ are maximal trace equivalent ($p \sim q$) if their sets of maximal traces are the same.

**Definition 2.3**

A simulation $\mathcal{R}$ is a binary relation on $\mathcal{P}$ ($\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$) such that whenever $p \mathcal{R} q$ and $\alpha \in \text{Act}^\dagger$ then:

\[
p \rightarrow \alpha \rightarrow p' \Rightarrow \exists q' \in \mathcal{P}. q \rightarrow \alpha \rightarrow q' \land p' \mathcal{R} q'.
\]

A process $q$ is said to simulate a process $p$ if and only if there exists a simulation $\mathcal{R}$ with $p \mathcal{R} q$ and we write $p \sim q$. Two processes are said simulation equivalent if there exist two simulations $\mathcal{R}$ and $\mathcal{R}'$ with $p \mathcal{R} q$ and $q \mathcal{R}' p$, written $p \sim q$.

**Definition 2.4**

A bisimulation $\mathcal{R}$ is a binary relation on $\mathcal{P}$ ($\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$) if and only if both $\mathcal{R}$ and $\mathcal{R}' = \{(q, p) \mid (p, q) \in \mathcal{R}\}$ are simulations. Two processes are said bisimulation (observational) equivalent if and only if there exists a bisimulation $\mathcal{R}$ with $p \mathcal{R} q$, written $p \sim q$.

**Definition 2.5**

A 2/3 bisimulation $\mathcal{R}$ is a binary relation on $\mathcal{P}$ ($\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$) such that whenever $p \mathcal{R} q$ and $\alpha \in \text{Act}^\dagger$ then:
i) \( p \rightarrow \alpha \rightarrow p' \Rightarrow \exists q \in P \cdot q \rightarrow \alpha \rightarrow q' \wedge p \mathcal{R} q' \)

ii) \( q \rightarrow \alpha \Rightarrow p \rightarrow \alpha \rightarrow \).

Two processes are said 2/3 bisimulation equivalent if and only if there exist 2/3 bisimulation \( \mathcal{R} \) with \( p \mathcal{R} q \) and a 2/3 bisimulation \( \mathcal{R}' \) with \( q \mathcal{R}' p \), written \( p \equiv q \).

Appendix B

Theorem 1

\( \mathcal{T}_{TL} \) is adequate w.r.t. the trace equivalence.

Proof.

We consider two processes \( p, q \in P \), we want to prove that:

\[
p \equiv \mathcal{T}_{TL} q \quad \text{iff} \quad ST(p) = ST(q)
\]

Let \( F(p) \) and \( F(q) \) be the sets of formulae satisfied respectively by \( p \) and \( q \), that is \( F(p) = \{ \phi : \phi \in \mathcal{T}_{TL}, p \models \phi \} \) and \( F(q) = \{ \psi : \psi \in \mathcal{T}_{TL}, q \models \psi \} \). Then the relationship between \( p \) and \( q \) is:

\[
p \equiv \mathcal{T}_{TL} q \quad \text{iff} \quad F(p) = F(q)
\]

So we can demonstrate the \( \mathcal{T}_{TL} \) adequacy as in the following:

\[
F(p) = F(q) \quad \text{iff} \quad ST(p) = ST(q)
\]

that is

\[
F(p) \neq F(q) \quad \text{implies} \quad ST(p) \neq ST(q) \quad \text{and} \quad ST(p) \neq ST(q) \quad \text{implies} \quad F(p) \neq F(q)
\]

Let us suppose that

\[
\exists \gamma \in \mathcal{T}_{TL} : \gamma \in F(p), \gamma \notin F(q), \text{i.e.} \ F(p) \neq F(q), \text{i.e.} \ \exists \gamma \in \mathcal{T}_{TL} : p \models \neg \gamma , \ q \not\models \gamma
\]

but considering the sets of traces of \( p \) and \( q \) we have the following relations

\[
p \models \gamma \iff \forall \sigma \in ST(p), (\sigma,0) \not\models \gamma \quad \text{and} \quad q \not\models \gamma \iff \exists \omega \in ST(q), (\omega,0) \not\models \gamma
\]

hence \( \omega \in ST(p), \omega \in ST(q) \), that is \( ST(p) \neq ST(q) \)

Viceversa we suppose that \( \exists T \in \text{Act}^* \), where \( T = \tau_1 \tau_2 ... \tau_n \) (\( T \) is finite by definition) and \( \forall \tau \in \text{Act}^+ \) and that \( ST(q) \cup \{ \tau_1, \tau_1 \tau_2, ... , \tau_1 \tau_2 ... \tau_1 \tau_2 ... \tau_n \} = ST(p) \), hence we must prove that

\[
\exists \gamma \in \mathcal{T}_{TL} : (\forall \sigma \in ST(q), \tau \in ST(p) \setminus ST(q) \cdot ((\sigma,0) \models \gamma, (\tau,0) \not\models \gamma))
\]

such a \( \gamma \) is \( \gamma = \neg \tau_1 \lor O(\neg \tau_2 \lor O(\neg \tau_3 ... \lor O(\neg \tau_n) ... ) \)

in fact every model of \( p \), different from \( (\tau,0) \), will satisfy \( \gamma \), because the models which cannot satisfy \( \gamma \) are \( (\tau,0) \) where \( \tau \in ST(p) \setminus ST(q) \), hence

\[
p \not\models \gamma , q \not\models \gamma , \text{i.e.} \ \gamma \in F(p), \gamma \notin F(q), \text{i.e.} \ F(p) \neq F(q)
\]

Theorem 2

\( \mathcal{T}_{TL} \) is expressive w.r.t. the trace equivalence.

Proof.

From the definitions of expressiveness of a logic and of simply abstractness of a
semantics, we can affirm that the equivalence $O(p) = O(q)$ indicates that the process $p$ and $q$ have the same Operational Semantics, while $p = q$ indicates that $p$ and $q$ are operationally equivalent, that is we have two notations for the same concept. Then, if $p = q$ (cioè $O(p) = O(q)$) by definition we have $p \vdash L(q)$, that is $L(p) \Rightarrow L(q)$ because $L(p)$ is the maximal formula of $p$; moreover $\Rightarrow$ is a symmetric relation, hence if $p = q$ then $q = p$, that is $q \vdash L(p)$, that is $L(q) \Rightarrow L(p)$. So we can affirm $L(q) \Leftrightarrow L(p)$, because of $L$ is the denotational semantic function and $D$ is the Denotational Semantics, we have $D(p) = D(q)$.

From these considerations we can say that we demonstrate the expressiveness of $TLF_L$ by means of its simply abstracness.

The proof of simply abstractness (given, for simplicity, considering actions which offer only one attribute) is carried on by comparing the set of traces originated by a LOTOS process following the string equivalence, and the set of sequences which are models for the formula expressing the temporal semantics of the same process. The main difference between the set of traces and the set of sequences is that the latter may contain sequences with any number of $e$ actions in between any pair of LOTOS-meaningful actions.

In order for the two equivalences to be the same, this presence of $e$ actions should be the only difference.

For this reason, we define a relation $R$, between a set of traces and a set of sequences. This relation corresponds to eliminate the $e$'s from the elements of the latter and obtain henceforth the same sequences of the former.

The heart of the proof, by structural induction, shows that for any LOTOS process its trace set is in relation $R$ with its model sequence set.

We indicate with "a" an element of $Act^+$, with $S = (ACT)^* \cup (ACT)^0$ the set of possible model sequences and with $T = (Act^+)^* \cup (Act^+)^0$ the set of possible traces. We use $\sigma$ for ranging over $S$ and $t$ for ranging over $T$, $e^*$ to denote any finite sequence of consecutive $e$'s and $e^0$ to represent the infinite sequence of consecutive $e$'s. $\lambda$ denotes the empty string. A dot (.) denotes the concatenation of two strings.

Some useful definitions:

$\text{Def.}$ The relation $R \subseteq S \times T$ is so defined:

$\sigma R t \iff ((\sigma = e^* \cdot a \cdot \sigma' \text{ and } t = a \cdot t' \text{ and } \sigma R t') \text{ or } (\sigma = e^0 \text{ and } t = \lambda) \text{ or } (\sigma = e^0 \text{ and } t = \lambda))$

This relation correspond to an elimination of the $e$'s present in the model sequences.

$\text{Def.}$ We extend the relation $R$ to the sets of sequences and races: $R \subseteq P(S) \times P(T)$

$S R T \iff (\forall \sigma \in S \exists t \in T : \sigma R t \text{ and } \forall t \in T \exists \sigma \in S : \sigma R t)$

$\text{Def.}$ A set $S \subseteq S$ is said $e$-maximal if:

$\sigma_1 . e^k . act . \sigma_2 \in S \Rightarrow \forall n \geq 0, \sigma_1 . e^0 . act . \sigma_2 \in S \text{ and } \sigma_1 . e^0 \in S$

$\sigma_1 . e^k \in S \Rightarrow \forall n \geq 0, \sigma_1 . e^0 \in S \text{ and } \sigma_1 . e^0 \in S$

with $\sigma_1, \sigma_2 \in S$.

$\text{Theorem 1}$

Let us denote with $Sp$ the set of model sequences of the temporal semantics of a LOTOS process $P$ (that is: $Sp = \{ \sigma \mid \sigma \vdash L_0(P) \}$) and with $T_p$ the set of traces of the same program.
Then $SpRTp$, for every LOTOS process $P$.

**Proof (by structural induction)**

1. $P = \text{stop}$
   
   $Sp = \{ \sigma \mid \sigma \vdash e \omega \text{false} \} = \{ e^* \} \cup \{ e^0 \}$
   
   $Tp = \{ \lambda \}$
   
   By definition of $R$, we have $SpRTp$. This is the basis of the induction.

2. $P = i:B$
   
   $Sp = \{ \sigma \mid \sigma \vdash e \omega (i \land \sigma L \theta (B)) \} = \{ e^*, i, \sigma \mid \sigma \in S_B \} \cup \{ e^0 \}$
   
   $Tp = \{ t \mid t \in T_B \} \cup \{ \lambda \}$
   
   By definition of $R$, we have $SpRTp$ if $S_BRT_B$.

3. $P = g?x\: \text{type}\: B(x)$
   
   $Sp = \{ \sigma \mid \sigma \vdash e \omega (\forall x \in \text{type} \cdot (g<\times > \land \sigma L \theta (B))) \} = \{ e^*, g<\times >, \sigma \mid \sigma \in S_B \} \cup \{ e^0 \}$
   
   $Tp = \{ g<\times >, t \mid t \in T_B \} \cup \{ \lambda \}$
   
   By definition of $R$, we have $SpRTp$ if $\forall x \in \text{type} \cdot S_BRT_B[\forall x\: L_B[\forall x\: L]]$.

4. $P = g!E:B$
   
   $Sp = \{ \sigma \mid \sigma \vdash e \omega (g<\epsilon (E) > \land \sigma L \theta (B)) \} = \{ e^*, g<\epsilon (E) >, \sigma \mid \sigma \in S_B \} \cup \{ e^0 \}$
   
   $Tp = \{ g<\epsilon (E) >, t \mid t \in T_B \} \cup \{ \lambda \}$
   
   By definition of $R$, we have $SpRTp$ if $S_BRT_B$.

5. $P = B!L \land B2$
   
   $Sp = \{ \sigma \mid \sigma \vdash (L \theta (B_1) [\text{subst1}] \land L \theta (B_2) [\text{subst2}] ) [\text{subst3}] \}$
   
   where:
   
   subst1 = $(\epsilon v1 \land \forall g \in \alpha (B_2), v \in \text{val} \cdot g2<\times > / g1<\times >, g \in \alpha (B_1) : g1<\times > / g2<\times >)
   
   subst2 = $(\epsilon v1 \land \forall g \in \alpha (B_1), v \in \text{val} \cdot g1<\times > / g2<\times >, g \in \alpha (B_2) : g1<\times > / g2<\times >)
   
   subst3 = $\forall g \in L, v \in \text{val} : g<\times > / g1<\times >, g2<\times >, i_1, i_2$
   
   The definition of $Tp$ is given by introducing a set which is defined recursively to contain the traces obtained by properly interleaving and synchronizing the traces of the component processes: $Tp = \{ t \mid t \in IST_L(t_1, t_2), t_1 \in T_B1, t_2 \in T_B2 \}$
   
   The following definition of $IST_L(t_1, t_2) \subseteq T$ is easily derived from the operational semantics:

   $IST_L(t_1, t_2) = \{ t \mid t = a.t', \text{name}(a) \in L, t' \in IST_L(t_1', t_2), t_1 = a.t' \}
   
   \cup \{ t \mid t = a.t', \text{name}(a) \in L, t' \in IST_L(t_1, t_2'), t_2 = a.t' \}
   
   \cup \{ t \mid t = a.t', \text{name}(a) \in L, t' \in IST_L(t_1', t_2'), t_1 = a.t', t_2 = a.t' \} \cup \{ \lambda \}$
   
   Let us now define an analogous set $ISS_L(\sigma_1, \sigma_2) \subseteq S$.

   $ISS_L(\sigma_1, \sigma_2) = \{ \sigma \mid \sigma = \sigma^n.a.\sigma', \text{name}(a) \in L, \sigma' \in ISS_L(\sigma_1', \sigma_2'), \sigma_1 = \sigma^n.a.\sigma_1', \sigma_2 = \sigma^n+1.\sigma_2', n \geq 0 \}$
\[ \cup \{ \sigma \mid \sigma = e^n.a.\sigma', \text{name}(a) \in L, \sigma' \in \text{ISS}_L(\sigma'_1, \sigma'_2), \sigma_1 = e^{n+1}.\sigma'_1, \sigma_2 = e^n.a.\sigma'_2, n \geq 0 \} \]
\[ \cup \{ \sigma \mid \sigma = e^n.a.\sigma', \text{name}(a) \in L, \sigma' \in \text{ISS}_L(\sigma'_1, \sigma'_2), \sigma_1 = e^n.a.\sigma_1', \sigma_2 = e^n.a.\sigma_2', n \geq 0 \} \]
\[ \cup \{ e^* \} \cup \{ e^n \} \]

**Lemma**

\( \sigma \in \text{ISS}_L(\sigma_1, \sigma_2) \) if \( \sigma = \sigma' \text{[subst3]} \) and \( \sigma' \in \sigma_1 \text{[subst1]} \) and \( \sigma' \in \sigma_2 \text{[subst2]} \)

where \( \sigma' \text{[subst3]} \) denotes the (only) sequence obtained from \( \sigma \) by the relabelling subst3, and \( \sigma_1 \text{[subst1]} \), \( \sigma_2 \text{[subst2]} \) denote the set of sequences obtained from, respectively, \( \sigma_1, \sigma_2 \) by the relabelling subst1, subst2.

**Proof** (by induction on the length of \( \sigma_1, \sigma_2 \))

We have several possible structures for \( \sigma_1 \) and \( \sigma_2 \) (cases 1,2,3 are the inductive steps; cases 4,5,6 are the basis of the induction):

1) \( \sigma_1 = e^n.a.\sigma_1', \sigma_2 = e^{n+1}.\sigma_2', n \geq 0, \text{name}(a) \in L \)

The only \( \sigma' \) such that: \( \sigma' \in \sigma_1 \text{[subst1]}, \sigma' \in \sigma_2 \text{[subst2]} \) is \( \sigma' = e^n.a.\sigma', \) where \( \sigma' \in \sigma_1 \text{[subst1]}, \sigma' \in \sigma_2 \text{[subst2]} \), since subst1 substitutes act with \( \sigma' \) and subst2 substitutes the \( n+1 \)-th e with \( \sigma' \).

If we take \( \sigma = \sigma' \text{[subst3]} \), we have: \( \sigma = e^n.a.\sigma' \), where \( \sigma' = \sigma' \text{[subst3]} \).

This means, that, since the inductive hypothesis is true for the shorter string \( \sigma' \), we have \( \sigma' \in \text{ISS}_L(\sigma'_1, \sigma'_2) \). Hence, by definition of ISS, \( \sigma \in \text{ISS}_L(\sigma_1, \sigma_2) \).

2) \( \sigma_1 = e^{n+1}.\sigma_1', \sigma_2 = e^n.a.\sigma_2', n \geq 0, \text{name}(a) \in L \)

This case is simply the converse of the previous one; the proof is obviously the same.

3) \( \sigma_1 = e^n.a.\sigma_1', \sigma_2 = e^n.a.\sigma_2', n \geq 0, \text{name}(a) \in L \)

The only \( \sigma' \) such that: \( \sigma' \in \sigma_1 \text{[subst1]}, \sigma' \in \sigma_2 \text{[subst2]} \) is \( \sigma' = e^n.a.\sigma', \) where \( \sigma' \in \sigma_1 \text{[subst1]}, \sigma' \in \sigma_2 \text{[subst2]} \); that is, subst1 and subst2 behave as the identity on the \( n+1 \)-long prefixes of \( \sigma_1 \) and \( \sigma_2 \).

If we take \( \sigma = \sigma' \text{[subst3]} \), we have: \( \sigma = e^n.a.\sigma' \), where \( \sigma' = \sigma' \text{[subst3]} \) (Again, subst3 behaves as the identity on the \( n+1 \)-long prefix of \( \sigma' \)).

This means, that, since the inductive hypothesis is true for the shorter string \( \sigma' \), we have \( \sigma' \in \text{ISS}_L(\sigma'_1, \sigma'_2) \). Hence, by definition of ISS, \( \sigma \in \text{ISS}_L(\sigma_1, \sigma_2) \).

4) \( \sigma_1 = e^n, \sigma_2 = e^n, n \geq 0. \)

The only \( \sigma' \) such that: \( \sigma' \in \sigma_1 \text{[subst1]}, \sigma' \in \sigma_2 \text{[subst2]} \) is \( \sigma' = e^n \), since subst1 and subst2 can be reduced to the identical substitution. In this case also subst3 is the identity, hence \( \sigma = \sigma' \text{[subst3]} = e^n \in \text{ISS}_L(\sigma_1, \sigma_2) \).

5) \( \sigma_1 = e^n, \sigma_2 = e^n. \)

By the same reasoning of the above case we obtain \( \sigma = e^n \in \text{ISS}_L(\sigma_1, \sigma_2) \).

6) In all the remaining cases:

- \( \sigma_1 = e^n, \sigma_2 = e^n.m, n \neq m; \)
- \( \sigma_1 = e^n.a.\sigma_1', \sigma_2 = e^n.a.\sigma_2', n \neq m, \text{name}(a) \in L; \)
- \( \sigma_1 = e^n.a.\sigma_1', \sigma_2 = e^n.a.\sigma_2', \text{name}(a1) \neq \text{name}(a2), \text{name}(a1) \in L, \text{name}(a2) \in L; \)
- \( \sigma_1 = e^n, \sigma_2 = e^n.m.\sigma_2', n \leq m, \text{and its converse}; \)
\(\sigma_1 = e^0, \sigma_2 = e^m\), and its converse; 
\(\sigma_1 = e^0, \sigma_2 = e^m.a, \sigma_2',\) and its converse 
we can find no \(\sigma^-\) such that: \(\sigma^- \in \sigma_1[\text{subst1}], \sigma^- \in \sigma_2[\text{subst2}]\); hence this cases satisfy the hypothesis trivially. 
\(\sqcup\)

(Proof of Theorem 1, case \(P = B1/lIB2\) continued) 
By lemma, we have: \(Sp \subseteq \{\sigma | \sigma \in ISS_L(\sigma_1, \sigma_2), \sigma_1 \in S_{B1}, \sigma_2 \in S_{B2}\}\) 

Suppose now that there exists a \(\sigma' \in ISS_L(\sigma_1, \sigma_2), \sigma' \in Sp, \sigma'\) could not be one of the elements for which the first three clauses of the definition of ISS hold, otherwise by reverse reasoning of clauses 1), 2), 3) of lemma it would belong to \(Sp\). 

Hence it should be: \(\sigma' = e^n, n \geq 0, or \sigma' = e^0\). Since \(S_{B1}\) and \(S_{B2}\) are e-maximal (by proposition 3 below), there exist \(\sigma_1' \in S_{B1}, \sigma_2' \in S_{B2}\) such that \(\sigma_1' = e^0 = \sigma_2'\) (respectively, \(\sigma_1' = e^0 = \sigma_2'\)), from which \(\sigma'\) can be obtained by identical substitutions. Hence \(\sigma'\) should belong to \(Sp\). 

Note that this part of the proof, similarly to lemma, is again by induction on the length of the sequences. Moreover, it is intertwined with the proof of e-maximality of all LOTOS processes (proposition 3). 

In conclusion, we have: \(Sp = \{\sigma | \sigma \in ISS_L(\sigma_1, \sigma_2), \sigma_1 \in S_{B1}, \sigma_2 \in S_{B2}\}\) 

and, by comparing the definition of the sets IST and ISS, 
\(S_{B1}RT_{B1}\) and \(S_{B2}RT_{B2}\) implies \(SpRTp\). 

\(P = B1/lIB2\) 
\(Sp = \{\sigma | \sigma \vdash L_\theta(B_2) \lor L_\theta(B_2)\} = \{\sigma | \sigma \vdash L_\theta(B_2) \lor L_\theta(B_2)\} = \{\sigma | \sigma \vdash L_\theta(B_2) \lor L_\theta(B_2)\} = S_{B1} \cup S_{B2}\) 

\(Tp = T_{B1} \cup T_{B2}\) 

Hence, we have: \(S_{B1}RT_{B1}\) and \(S_{B2}RT_{B2}\) implies \(SpRTp\). 

\(P = \text{choice } x:1 [ ] B\) 
\(Sp = \{\sigma | \sigma \vdash \forall x: \text{type } (L_\theta(B))\} = \{\sigma \in S_{B1}/x, \sigma \in S_{B2}/x\} = \cup_{v \in \text{type } S_{B1}/x}\) 

\(Tp = \cup_{v \in \text{type } S_{B1}/x} T_{B1}/x\) 

So, \(\forall v \in \text{type } S_{B1}/x, \forall RT_{B1}/x\) implies \(SpRTp\). 

\(P = \text{exit } (E^*)\) 
\(Sp = \{\sigma | \sigma \vdash eW(\delta(e(E^*) \land \lor \theta false))\} = \{e^*, \delta(\leq \theta false) \land \sigma | \sigma \in S_{stop}\} \cup \{e^0\}\) 

\(Tp = \{\delta(\leq \theta false) \land \sigma | \sigma \in T_{stop}\} \cup \{\lambda\}\) 

By definition of \(R\), we have \(SpRTp\) because \(S_{stop}RT_{stop}\). 

\(P = B1/lIB2\) 
\(Sp = \{\sigma | \sigma \vdash (L_\theta(B_1) [\text{subst1}] \land (eW(\delta(e<0) \land \lor \theta (B_2))))([\text{subst2}]) [\text{subst3}]\} \) 

where 
\(\text{subst1} = (e^2v \forall v \in \alpha(B_2), v \in Valg(2)v) \land i_1, i_2, i_3, \forall v \in \alpha(B_1): g^1v \forall v \geq \theta, \delta^* < \delta < >\) 

\(\text{subst2} = (e^2v \forall v \in \alpha(B_1), v \in Valg(2)v) \land i_1, i_2, i_3, \forall v \in \alpha(B_2): g^2v \forall v \geq \theta, \delta^* < \delta < >\) 

\(\text{subst3} = \forall v \in \alpha(B_1): g^1v \land i_1, i_2, i_3, \forall v \in \alpha(B_2): g^2v \forall v \geq \theta, \delta^* < \delta < >\) 

26
$T_p = \{ t \mid t \in \text{IST}_\delta(t_1, t_2), t_1 \in T_{B_1}, t_2 \in T_{B_2} \}$

where the set $\text{IST}_\delta(t_1, t_2) \subseteq T$ is defined:

$\text{IST}_\delta(t_1, t_2) = \{ t \mid t = a.t', \text{name(a)} \neq \delta, t' \in \text{IST}_\delta(t_1', t_2), t_1 = a.t' \}
\cup \{ t \mid t = a.t', \text{name(a)} \neq \delta, t' \in \text{IST}_\delta(t_1, t_2), t_2 = a.t' \}
\cup \{ t \mid t = a.t', \text{name(a)} = \delta, t' \in \text{IST}_\delta(t_1', t_2), t_1 = \delta \circ .t' \cup (\lambda) \}$

Also we define $\text{ISS}_\delta(s_1, s_2) \subseteq S$:

$\text{ISS}_\delta(s_1, s_2) = \{ s \mid s = e^n.a.s', \text{name(a)} \neq \delta, s' \in \text{ISS}_\delta(s_1', s_2'), s_1 = e^n.a.s_1', s_2 = e^n.s_1' \cdot s_2', n \geq 0 \}
\cup \{ s \mid s = e^n.a.s', \text{name(a)} = \delta, s' \in \text{ISS}_\delta(s_1', s_2'), s_1 = e^n.s_1', s_2 = e^n.a.s_2', n \geq 0 \}
\cup \{ s \mid s = e^n.i.s', \text{name(a)} = \delta, s' \in \text{ISS}_\delta(s_1', s_2'), s_1 = e^n.i.s_1', s_2 = e^n.i.s_2', n \geq 0 \}
\cup \{ e^0 \} \cup \{ e^0 \}

The demonstration is analogous to the parallel composition one.

- $P = \text{hide L in B}$
  $S_p = \{ \sigma \mid \sigma \vdash L_\theta(B) \mid \forall g \epsilon L_\theta, \mu_g \epsilon S \} = \{ \sigma \mid \sigma = \sigma' \mid \forall g \epsilon L_\theta, \mu_g \epsilon S \}$
  $T_p = \{ t \mid t = t' \mid \forall g \epsilon L_\theta, \mu_g \epsilon S \}$

By definition of $R$, we have $S_p \cup T_p$ if $S_{B_1} \cup T_{B_1}$.

- $P = \text{cond} \rightarrow B$
  $S_p = \{ \sigma \mid \sigma \vdash L_\theta(B) \mid \beta[(\text{cond} \land (L_\theta(B))) \epsilon S \} = \{ \sigma \mid \sigma = \sigma' \mid \forall g \epsilon L_\theta, \mu_g \epsilon S \}$
  $T_p = \{ t \mid t \epsilon T_{\text{stop}} \} \cup \{ \lambda \}$

If cond is true : $S_{B_1} \cup T_{B_1}$ implies $S_{B_1} \cup T_{B_1}$, otherwise $S_{B_1} \cup T_{B_1}$ is just proved, because of $S_{\text{stop}} \cup T_{\text{stop}}$.

- $P = \text{let } x_1 \epsilon E_1, ..., x_n \epsilon E_1 \text{ in B}$
  $S_p = \{ \sigma \mid \sigma \vdash L_\theta(B) \mid \epsilon \{E_1, ..., E_1\}/x_1, ..., x_n \} = \{ \sigma \mid \sigma = \sigma' \mid \epsilon \{E_1, ..., E_1\}/x_1, ..., x_n \}$
  $T_p = \{ t \mid t \epsilon T_{B_1} \mid \epsilon \{E_1, ..., E_1\}/x_1, ..., x_n \}$

So, we have : $S_{B_1} \cup T_{B_1} \mid \epsilon \{E_1, ..., E_1\}/x_1, ..., x_n \mid \epsilon \{E_1, ..., E_1\}/x_1, ..., x_n \}$ implies $S_{B_1} \cup T_{B_1}$

- $P = \text{Id}[a](E)$, where process $\text{Id}[g^-](x^*) := B \text{ endproc}$
  $S_p = \{ \sigma \mid \sigma \epsilon S_{B_1} \epsilon a^*/g^- \epsilon \epsilon \{E_1\}/x^* \} = \{ \sigma \mid \sigma = \sigma' \mid \epsilon \{E_1\}/x^* \}$
  $T_p = \{ t \mid t \epsilon T_{B_1} \epsilon a^*/g^- \epsilon \epsilon \{E_1\}/x^* \}$

By definition of $R$, we have $S_{B_1} \cup T_{B_1}$ if $S_{B_1} \cup T_{B_1}$.

\[ \square \]

Prop.1 $\forall \sigma \epsilon S \exists! t \in T: \sigma R t$

Proof (by contradiction)

Suppose that, given $\sigma \epsilon S$, we have $t_1, t_2 \epsilon T$, $t_1 \neq t_2$, $\sigma R t_1$ and $\sigma R t_2$. Three cases are possible:

1) $t_1 = \lambda$; hence, by definition of $R$, $\sigma$ should be $e^0$ for some non-negative $n$, or $e^0$

Since $t_1 \neq t_2$, for some $a \epsilon \text{Act}$, it should be $t_2 = a.t_2$. By definition of $R$ it should be $\sigma = a . \sigma'$, with $\sigma R t_2$. 

27
contradicting what said above.
2) \( t_1 = a.t_1', t_2 = b.t_2', a \neq b \);

by definition of \( R \) it should be both \( \sigma = a.\sigma' \) and \( \sigma = b.\sigma'' \). Contradiction.
3) \( t_1 = a.t_1', t_2 = a.t_2, t_1 \neq t_2' \);

the contradiction will be reached by applying recursively this process on \( t_1' \) and \( t_2' \). We are assured of the
termination by the fact that if \( t_1 \neq t_2' \), there will be a \( k \geq 0 \) such that either \( t_1 \) has only \( k-1 \) elements, and \( t_2 \) has at
least \( k \) elements, so the first clause is applicable, or the \( k \)-th element of \( t_1 \) is different from the \( k \)-th element of \( t_2 \).
hence the second clause is applicable.

Note that the contrary (\( \forall t \in T \exists! \sigma \in S: \sigma R t \)) is obviously not true, by definition of \( T \) and \( S \).

**Prop. 2** \( \forall S \subseteq S \exists T \subseteq T: S R T \)

**Proof** (by contradiction)

Suppose that, for a given \( S \subseteq S \), we have \( T, T' \subseteq T \), \( T \neq T' \): \( S R T \) and \( S R T' \). Since \( T \neq T' \) (suppose
\( T \subseteq T' \)), there exists at least an element \( t \in T, t \notin T' \). Since \( S R T \), there exists a \( \sigma \in S \) such that \( \sigma R t \). But since
\( S R T' \), there exist a \( t' \in T' \) such that \( \sigma R t' \). For **Prop. 1**, \( t = t' \), and hence the contradiction.

**Theorem 2**

This theorem gives one side of the abstractness proof, saying that if two programs have the same temporal
meaning then they have also the same operational meaning, that is:

\[ \forall P, Q \text{ LOTOS processes} : Sp = SQ \Rightarrow Tp = TQ \]

**Proof**

By **Theorem 1** we have \( Sp R Tp \) and \( SQ R TQ \). But \( Sp = SQ \), and by **Prop. 2** we have: \( Tp = TQ \).

**Prop. 3** \( \forall P \text{ LOTOS process} : Sp \text{ is } e\text{-maximal.} \)

**Proof** (by structural induction)

1) The meanings of : \textit{stop}, \textit{i:B}, \textit{g?x:B(x)}, \textit{g!E:B}, \textit{exit}, due to the use of \"e \( \wedge \) \" operations, have as model
sequences all those with subsequences of \( e \)'s of any length, also infinite. Hence it is not the case that the
sequence \( a.e.k .i.e.\sigma \) satisfies the meanings of these operators, while the sequence \( a.e.k+1.a.e.\sigma \) does not satisfy it. Hence \( S \textit{stop} \) and \( S \textit{exit} \) are \( e \)-maximal, and \( S \textit{oi}\|B \) is \( e \)-maximal if \( S_B \) is \( e \)-maximal (\( oi \) is any LOTOS action).

2) The meanings of guarded expression, choice and generalized choice do not introduce any more \( e \) in their
model sequences than those present in the model sequences of their component processes.

3) The meanings of hiding, let and process instantiation relabel only non-\( e \) actions in the model sequences of
their component processes. Hence \( S \textit{hide}\ g\ in\ B \) is \( e \)-maximal if \( S_B \) is \( e \)-maximal. \( S \textit{let} x1:1=t1, ... ,xn:tn=En\ in\ B \) is \( e \)-maximal if \( S_B \) is \( e \)-maximal and \( S [\text{Id}[a^*][E^*]] \) (where \( \text{process}\ id[a^*][x^*] := B endproc \)) is \( e \)-maximal if \( S_B \) is \( e \)-maximal.

4) For what concerns the parallel composition and enabling operators it is easy to see that by definition the \( ISS_L \)
and \( ISS_S \) sets are \( e \)-maximals. Hence \( S_B 1 \| B_2 \) and \( S_B 1 >> B_2 \) are \( e \)-maximals.

**Prop. 4** \( \forall T \subseteq T \exists! S \subseteq S \), \( S \text{ e-maximal: } S R T \)

**Proof** (by contradiction)

Suppose that, for a given \( T \subseteq T \), we have \( S, S' \subseteq S, S \neq S', S, S' \text{ e-maximal: } S R T \) and \( S' R T \).

Since \( S \neq S' \) (suppose \( S \subseteq S' \)), there exists at least an element \( \sigma \in S, \sigma \notin S' \). Since \( S R T \), there exists a \( t \in T \) such
that \( \sigma R t \). But since \( S' R T \), there exist a \( \sigma' \in S' \) such that \( \sigma R t \). By definition of \( R \), if \( \sigma R t \) and \( \sigma R t, \sigma \) and
σ' may differ only for the length of some of their sub-sequences of e. Since S' is e-maximal and σ'eS', we have also σeS', which contradicts the hypothesis.

Theorem 3

This theorem, saying that two programs with the same operational meaning have the same temporal meaning, gives the other side of the abstractness proof, that is:

∀ P, Q LOTOS processes:  \( T_P = T_Q \Rightarrow S_P = S_Q \)

Proof

By theorem 1 we have \( S_P \mathcal{R} T_P \) and \( S_Q \mathcal{R} T_Q \). But \( T_P = T_Q \), and by Prop. 3 \( S_P \) and \( S_Q \) are e-maximal. Hence by Prop. 4 we have: \( S_P = S_Q \).

In conclusion we can say our temporal semantics is simply abstract w.r.t. to the trace equivalence. In fact the theorems 2 and 3 demonstrate that the equality \( D(P) = D(Q) \) implies the equality \( O(P) = O(Q) \), and vice versa. Moreover, because of the compositionality of the proposed semantics we can say that it is fully abstract w.r.t. the trace equivalence.