Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

PISA

A Temporal Semantics for Basic LOTOS

Alessandro Fantechi, Stefania Gnesi, Cosimo Laneve

Nota Interna B4-41
Agosto 1989

A TEMPORAL SEMANTICS FOR BASIC LOTOS

A. Fantechi* , S. Gnesi*, C. Laneve**

* *Istituto di Elaborazione dell'Informazione - C.N.R. - Pisa - Italy*
** *Dipartimento di Informatica- Università di Pisa - Italy*

Abstract. A temporal semantics for Basic LOTOS, i.e. LOTOS without value passing, is presented. The semantics is provided using a compositional approach by which it is possible to associate temporal logic formulae to each language construct. The relationships between this semantics and the operational semantics of the language have been studied in detail, showing its simple abstractness with respect to the operational semantics modulo string equivalence. The capability of the given semantics to be used to verify interesting properties of programs is also investigated.

1. Introduction

The specification language LOTOS [DIS 8807] describes a system by defining the temporal relation between the events that constitute its externally observable behaviour. The language has several operators (sequential composition, non-deterministic choice, synchronized and non-synchronized parallelism, etc.) to shape the temporal ordering of the events. The communication mechanism between a process and its external environment is based upon the general idea of "interaction"; in an interaction, a LOTOS process and the external environment establish together the t-uple of values they want to exchange.

A LOTOS description of a system appears to be quite detailed and close to an implementation. Sometimes however a more abstract description of a system - describing its 'abstract' properties instead of the event it can perform - is desired. This can happen when a complete analysis of a specification is to be performed, for example in order to see whether a LOTOS specification of a protocol conforms to its informal standard definition, or whether it satisfies a property. In this paper we discuss an approach that can be used to provide the semantics of a LOTOS program by expressing in a logic formalism the properties that the program satisfies.

Temporal logic [Lam 83, Man 81], a particular modal logic which can be used to describe the occurrence of events in time, is considered a suitable tool for specifying and reasoning about concurrent programs at a higher abstraction level. This type of logic is needed in order to express properties that relate events occurring in different moments (termination, accessibility, etc.) and which cannot be expressed by usual logics (propositional calculus, first order logic). We need therefore a way to derive temporal logic specifications of properties by the behavioural LOTOS specification.

To achieve this goal we propose giving LOTOS programs a *temporal semantics* [Pnu 81], i.e. the temporal logic formula expressing the properties of their execution sequences.

The advantages of this type of approach when providing the semantics for concurrent programs can be summed up in the following observations:

i) verifying that a program satisfies a given property (expressed by a temporal logic formula) is reduced to verifying that the formula expressing the temporal meaning of the program logically implies the given property formula.

ii) verifying the equivalence of programs is reduced to verifying logic equivalence.

This makes it possible to unify these two verification problems, as they amount to checking the validity of a logic formula. This can be carried out using mechanic tools such as model checkers and decision procedures.

Our research in this area aims to produce a complete account on the use of temporal logics for reasoning about LOTOS program properties, also realizing automatic tools to generate the temporal semantics of a program and verify its properties.

The purpose of this paper is to evaluate the application of a known method to give the temporal semantics of Basic LOTOS, a simplified version of LOTOS without value passing, both in terms of its capability to define

an"abstract" semantics and of its ability to express interesting program properties, to form the basis for a subsequent extension to full LOTOS. In [Fan 88a] a preliminary proposal in this direction is presented, which gives the temporal semantics to a subset of LOTOS operators, but with value passing.

The temporal semantics to this language is defined using a linear version of temporal logic; this decision permits to directly exploit the compositional approach presented in [Bar84, Bar85], which defines the temporal semantics in a denotational style.

We demonstrate that the temporal semantics proposed is not in contrast with the operational semantics of chosen language. More precisely, we prove that it is simply abstract with respect to the operational semantics modulo the string equivalence.

We show also that in order to be able to verify interesting properties, such as fairness and justice, we need that the language definition is enriched to express them, and we show that the given temporal semantics can be modified accordingly.

2. Basic LOTOS

The considered language is Basic LOTOS as defined in [Bol 87] which only describes process synchronization, while full LOTOS also describes interprocess value communication. This allows to define the temporal semantics to all LOTOS operators without introducing cumbersome formulae due to the handling of values.

In Table 1 we present the actions and operators of Basic LOTOS and their related operational semantics. From the operational semantics we can observe that the transitions of programs are labelled by the observable actions (marked as o) and by the unobservable (i) actions, which only move the control point along the behaviour expression, without any other state change.

Table 1 gives the syntax of behaviour expressions; the syntax of programs in our language is the following:

Program := *process declaration list ; behaviour expression*

process declaration list := *process declaration ; process declaration list* | *nil*

process declaration := **process** *process identifier[gate*]*
':=' behaviour expression **endproc**

where *gate** is a possibly empty list of gates.

This introduces the possibility to write recursive processes.

We note that the form of Basic LOTOS we use is slightly different from the original one for the following features:

1. The static structure of Basic LOTOS programs is simplified.
2. We only allow one form of parallelism (that syntactically expressed by the operator |*set of gates*|). This is in any case the most general parallelism and includes the other two (syntactically expressed through || and |||) as degenerate cases (||≡|*all gates*| and |||≡|ø|).
3. The successful termination operator (**exit**) is not considered. The sequential composition is performed of behaviours which terminate with the inaction (**stop**). This choice allows to ignore the "d" action, making the definition of the temporal semantics more readable, without loosing significance: the presence of the distinguished "d" action requires separate treatment in most semantic clauses, while not presenting any further difficulty.

operator	syntax	operational semantics	informal meaning
Inaction	**stop**		denotes a process which cannot perform any action.
Unobservable action	i;B	i;B -i-> B	models an event internal to the process.
Observable action	g;B	g;B -g-> B	models a process which can perform the transition g.
Choice	B1[]B2	B1-oi-> B1' *implies* B1[]B2 -oi-> B1' B2-oi-> B2' *implies* B1[]B2 -oi-> B2'	the actions of the process are the set of possible actions of B1 and B2.
Parallel composition	B1|S|B2	B1-oi-> B1' *and* gates(oi)$\notin$ S *implies* B1|S|B2-oi-> B1'|S|B2 B2-oi-> B2' *and* gates(oi)$\notin$ S *implies* B1|S|B2-oi-> B1|S|B2' B1-o-> B1' *and* B2-o-> B2' *and* gates(o)$\in$ S *implies* B1|S|B2-o-> B1'|S|B2'	the parallel composition forces the subprocesses to interact at every gate in the set S.
Sequential composition (enabling)	B1>>B2	B1 -oi-> **stop** *implies* B1>>B2 -oi-> B2 B1 -oi->B1' , B1'$\neq$ **stop** *implies* B1>>B2 -oi-> B1'>>B2	this operator allows sequential process composition.
Disabling	B1[>B2	B1-oi->B1' *implies* B1[>B2 -oi-> B1'[>B2 B2-oi-> B2' *implies* B1[>B2 -oi-> B2'	this operator allows process B1 to be disabled by process B2.
Hiding	**hide S in B**	B-oi-> B' *and* gates(oi)$\notin$ S *implies* **hide S in** B-oi-> **hide S in** B' B-o->B' *and* gate(o) $\in$ S *implies* **hide S in** B-i-> **hide S in** B'	this operator allows transitions at gates in S to be internalized.
Process instantiation	p[g_1,...g_n]	Bp[g_1/h_1,...g_n/h_n] -oi-> B' *implies* p[g_1,...g_n] -oi-> B'	the transitions of a process instantiation are those of the body of the process declaration (Bp) which substitute formal parameters (h_i) with actual ones.

Table 1

3. A Discrete Bounded Linear Temporal Logic

Since we are interested in giving the temporal semantics to Basic LOTOS and the behaviour expressions in the above language can be modelled by finite or infinite, discrete in time, sequences of transitions, we will define a bounded linear time temporal logic [Fis 87], that is having as models finite or infinite sequences. In this logic

predicates specify events, i.e. transitions, and formulae contain usual first order logic and basic temporal operators. The atomic formulae are borrowed from the action set (we will call them "action predicates"). The operators that we will use are the usual first order logic connectives:

true, **false**, $\neg$, $\vee$, $\wedge$, $\Rightarrow$;

together with the temporal operators:

o (next), [] (always), $\diamond$ (eventually), $\mathcal{W}$ (unless), $\mathcal{U}$ (until),

[/] (relabelling), $\mathbb{C}$ (enabling), $[\!>$ (disabling);

and a maximal fixed point constructor: $\nu\xi.\ \chi(\xi)$.

Propositional Calculus operators

$\sigma,i \models$ **true** = **t**, $\sigma,i \models$ **false** = **f**.

$\sigma,i \models a$	iff	$\sigma(i) = a$
$\sigma,i \models \neg\phi$	iff	not $(\sigma,i \models \phi)$
$\sigma,i \models \phi \vee \psi$	iff	$(\sigma,i \models \phi)$ or $(\sigma,i \models \psi)$
$\sigma,i \models \phi \wedge \psi$	iff	$(\sigma,i \models \phi)$ and $(\sigma,i \models \psi)$
$\sigma,i \models \phi \Rightarrow \psi$	iff	$\sigma,i \models \neg\phi \vee \psi$

Temporal operators

$\sigma,i \models O\phi$	iff	lenght$(\sigma) \leq i$ or $\sigma,i+1 \models \phi$
$\sigma,i \models []\phi$	iff	for all $j \geq i$: $(\sigma,j \models \phi)$
$\sigma,i \models \diamond\phi$	iff	exists $j \geq i$: $(\sigma,j \models \phi)$
$\sigma,i \models \phi\,\mathcal{U}\,\psi$	iff	exists $k \geq i$: $(\sigma,k \models \psi)$ and for all j : $i \leq j < k$ and $(\sigma,j \models \phi)$
$\sigma,i \models \phi\,\mathcal{W}\psi$	iff	$\sigma,i \models []\phi$ or $\sigma,i \models \phi\,\mathcal{U}\,\psi$
$\sigma,i \models \nu\xi.\ \chi(\xi)$	iff	for all $k>0$ $\sigma,i \models \chi^k$ (**true**)
$\sigma,i \models \phi\,[a/b]$	iff	exists $\sigma',i \models \phi$ and for all $k>i$: $\sigma',k \models b$ implies $\sigma,k \models a$ and for all $g \neq b$: $\sigma',k \models g$ implies $\sigma,k \models g$
$\sigma,i \models \phi\,\mathbb{C}\,\psi$	iff	(exist σ',σ'', lenght$(\sigma')<\infty$ and $\sigma',i \models \phi$ and $\sigma'',i \models \psi$ and $\sigma' o\, \sigma'' = \sigma$) or $(\sigma,i \models \phi$ and lenght$(\sigma)=\infty)$
$\sigma,i \models \phi[\!>\psi$	iff	(exist σ',σ'', lenght$(\sigma')<\infty$ and $\sigma' o\, \sigma'' = \sigma$ and $\sigma'',i \models \psi$ and and exist σ''', $\sigma' o\, \sigma''', i \models \phi$) or $(\sigma,i \models \phi$ and lenght$(\sigma)=\infty)$

Table 2

In order to define the semantics of temporal formulae, we need to introduce the concept of a model. A model σ is a sequence, finite or infinite, of actions (corresponding to a particular process execution). In the following with $\sigma(i)$ we denote the i-th action in the sequence, with the infix operator "$_o$" the concatenation of sequences, and with *lenght(σ)* , finite or infinite, the number of actions in the sequence σ. An interpretation is a pair $<\sigma,i>$, where σ is a model, and i is a discrete instant of time, that is, a positive integer.

In Table 2 we define inductively what we mean by an interpretation $<\sigma,i>$ to satisfy a formula ϕ and this is expressed by the notation: $\sigma,i \models \phi$.

Note that by χ^k (**true**) we denote the temporal formula $\chi(\chi(...\chi(\textbf{true})...))$ k-times. The maximal fixed point constructor $\nu\xi.\chi(\xi)$ denotes the maximal solution to $\xi=\chi(\xi)$, which exists if the function $\chi(\xi)$ is monotonic and continuous. The monotonicity requirement can be ensured by the appearance of the temporal formula ξ in χ under an even number of negations [Ban 86].

The *relabelling* operator $\phi[a/b]$, where a and b are action predicates, denotes the logical operator which has the effect of substituting every occurrence of b in the model sequences of a formula ϕ with a. This operator, which is different from a syntactic substitution of every occurrence of b with a in the formula ϕ, has been named "relabelling"[Den 88] since it is analogous to the CCS operator with the same name.

The formula $\phi\mathbb{C}\psi$, using the "chop" operator [Bar 84], holds for a sequence σ which can be decomposed in a finite prefix σ' and a suffix σ'' respectively satisfying ϕ and ψ, or for an infinite sequence σ satisfying ϕ. More precisely we use the "weak" chop (or "combine") operator as defined in [Bar 84, Ros 86].

The formula $\phi[>\psi$, using the "disabling" operator, holds for a sequence σ which can be decomposed in a finite prefix σ' *partially* satisfying ϕ and a suffix σ'' satisfying ψ. Partially means that exists a sequence σ''' having as prefix σ' and satisfying ϕ.

The formulae of our logic are very close to behavioural specifications; in fact, these formulae are given on the sequence of actions performed by the system, using propositions which are considered true if the system is able to perform the corresponding action, and composing them with temporal operators to obtain formulae that can have an obvious interpretation in models which are sequences of actions. The idea is to characterize all behaviours of a program through one temporal logic formula. Consequently a behaviour will be a program execution if and only if it satisfies the formula. For example, the simple formula consisting of the proposition 'a' will be true for the processes which can perform immediately an 'a' action, while the formula $(a \vee b) \wedge O[]c$ will be true for the processes which can perform an action 'a' followed by an infinite sequence of 'c' actions or an action 'b' followed by an infinite sequence of 'c' actions.

We will use the temporal logic presented to provide the semantics for a language which admits only interleaved executions, that is, in which actions are performed one at a time. Therefore, we will deal only with model sequences in which only one action predicate is true at any time.

This means that the formula $a \wedge b$, where a and b are action predicates, can have no model and hence is equal to false. At the level of logic formulae, we can express this constraint on the models as the implicit assumption that every formula is put into conjunction with the predicate:

$$[] \bigwedge_{\substack{x,y \in Act \\ x \neq y}} \neg(x \wedge y)$$

4. A compositional temporal semantics for the example language

In order to achieve modularity in the specification and verification of concurrent programs, the semantics provided should be compositional, i.e. the meaning of a program is given by a composition of the meanings of its components.

An approach towards compositional temporal semantics has been developed by Barringer, Kuiper and Pnueli [Bar 84, Bar 85, Bar 86]. In this approach, to obtain compositionality the semantics of a process must be *open*, i.e. the process is described immersed in all possible (parallel) environments. The semantics is given by closing

the process in question with all possible environments. The meaning of a process in closed systems of this type 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, i.e. some of the environment actions of the first process will be actions performed by the other. To denote an unknown environment action, a special proposition 'e', is added to the set of meaningful language actions (in [Bar 86] the same proposition is named 'i'; we choose 'e' to avoid confusion with the 'i' (internal) action). Therefore the set of action predicates we will use is defined as :

$Act = Gates \cup \{e,i\}$.

Due to the compositionality of the approach, a temporal semantics can be given by defining a function $\mathcal{L}$ which associates a temporal formula to each construct of the language by means of a set of syntax-directed clauses, as is usual when giving denotational semantics.

The language constructs are in the example language behaviour expressions, and their meaning is taken to be a predicate on possible execution sequences spawning from that behaviour expression. The meaning is however dependent on the context in which the construct is inserted, i.e. an environment which associates a process denotation to every declared identifier is considered as a parameter of the function $\mathcal{L}$. We adopt the notation $\mathcal{L}_\theta$ to indicate the environment parameter.

The type of the semantic function is hence:

$$\mathcal{L} : S \rightarrow E \rightarrow TL$$

where S is the space of behaviour expressions of our language, TL is the space of temporal formulae and E is the space of environments, i.e. of mappings from process identifiers to temporal logic formulae:

$E : P \rightarrow TL$.

Note that in the following the word "environment" is overloaded with two meanings: the first denotes an element of E, the second distinguishes the "e" action as an environment action, in the typical sense of the compositional approach. The meaning is, however, obvious from the context each time.

The clauses defining the function $\mathcal{L}$ by structural induction are listed here; each is followed by a comment.

$$\mathcal{L}_\theta (\mathbf{stop}) = e \,\mathcal{W}\, \neg \bigcirc \mathbf{true}$$

The meaning of an inaction behaviour is true for any (possibly infinite) sequence of environment actions. he formulae '$\neg\bigcirc$ **true**' is valid for the empty sequence. The $\mathcal{W}$ operator is used in this clause because the process '**stop**' will execute transitions only if the environment is able to perform them: the process will terminate when the environment terminates.

$$\mathcal{L}_\theta (i;B) = e \,\mathcal{W}\, [i \wedge \bigcirc \mathcal{L}_\theta (B)]$$

The meaning of an unobservable action followed by a behaviour B is true for every sequence of environment events followed by the unobservable action and by sequences for which the meaning of B is true.

$$\mathcal{L}_\theta (g; B) = e \,\mathcal{W}\, [g \wedge \bigcirc \mathcal{L}_\theta (B)]$$

The meaning of an observable action followed by a behaviour B is true for every sequence of environment events followed by the observable action and by sequences for which the meaning of B is true. In this case it is possible that an interaction never occurs, since its occurrence is dependent on the readiness of the environment to perform the same action.

$$\mathcal{L}_\theta (B_1 [] B_2) = \mathcal{L}_\theta (B_1) \vee \mathcal{L}_\theta (B_2)$$

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

In the following formula we should have used two functions: $\underline{1}$ and $\underline{2}$ defined on the set of gates Γ which give values, respectively, in the sets $\Gamma\times\{1\}$ and $\Gamma\times\{2\}$, such that $\underline{1}(x) = \langle x,1\rangle$, and $\underline{2}(x) = \langle x,2\rangle$. However, for simplicity as notation, we will use the subscripts to the names of the gates (i.e. is, x_1, x_2) to refer to the application of these functions.

$$\mathcal{L}_\theta (B_1 |g| B_2) =$$

$$(\mathcal{L}_\theta (B_1) [(e \vee i_2 \vee \bigvee_{f\notin g, f\in \alpha(B_2)} f_2) / e,\ i_1 / i,\ \forall f\notin g, f\in \alpha(B_1)\colon f_1 / f] \wedge$$

$$\wedge \mathcal{L}_\theta (B_2) [(e \vee i_1 \vee \bigvee_{f\notin g, f\in \alpha(B_1)} f_1) / e,\ i_2 / i,\ \forall f\notin g, f\in \alpha(B_2)\colon f_2 / f])$$

$$[\forall f\notin g\ f/f_1, \forall f\notin g\ f/f_2,\ i/ i_1, i/ i_2]$$

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 α, defined on behaviour expressions, gives the set of transitions that a behaviour expression *may* perform during its execution (this set of transitions 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 f, i. e. those 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 'g': 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 B_1 performs an *i* action will be considered as an environment action for B_2, thus an *i* in B_1 must be paired with an *e* in B_2 (and viceversa).
2. If B_1 performs a f action, with $f\notin g$, this will again be considered as an environment action for B_2; thus it must be paired with an *e* in B_2 (and viceversa).
3. If B_1 performs a f action, with $f\in g$, B_2 must also perform this action, and thus the same actions must be paired for B_1 and B_2.
4. 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 B_1 and B_2; thus an *e* in B_1 should be paired with an *e* in B_2.

This pairing is performed by the conjunction of the component meanings. Suppose the next action performed by B_1 is 'x' and by B_2 is 'y'; this means that:

$$\mathcal{L}(B_1)\Rightarrow x \qquad \text{and} \qquad \mathcal{L}(B_2)\Rightarrow y$$

hence

$$\mathcal{L}(B_1)\wedge\mathcal{L}(B_2)\Rightarrow x\wedge y \qquad (2)$$

Now, if x and y are different, formula (2) will have no model due to the interleaving constraints. Hence we should restrict models to the cases when x and y are the same. This is satisfactory for *e* actions (case 4) and for actions on gates in 'g' (case 3). In the other cases, we should substitute one of the *e*'s in $\mathcal{L}(B_1)$ with an *i* (case 1) or with a *f*($f\notin g$) in case 2. This substitution amounts to partially closing the open semantics of B_1 with actions performed by B_2.

In conclusion we would produce the following simpler formula which expresses the temporal meaning of $B_1|g|B_2$:

$$\mathcal{L}_\theta(B_1)[(ev\,i \vee \bigvee_{f \notin g, f \in \alpha(B_2)} f)/e] \wedge \mathcal{L}_\theta(B_2)[(ev\,i \vee \bigvee_{f \notin g, f \in \alpha(B_1)} f)/e]$$

However this formula could confuse actions performed by B_1 with the same actions performed by B_2 (in LOTOS we have no "ownership" on actions: any process can perform any action at any gate). To overcome this problem we must identify (by a substitution) the owner of the (i and f, $f \notin g$) actions. The identity of the owner is then dropped from the formula describing the behaviour of the composed process.

$$\mathcal{L}_\theta(B_1 >> B_2) = \mathcal{L}_\theta(B_1)\, \mathbb{C}\, \mathcal{L}_\theta(B_2)$$

The meaning of enabling is true for every sequence that is the concatenation of a model of B_1 with a model of B_2. The concatenation of models is achieved by the use of the chop operator.

$$\mathcal{L}_\theta(B_1 [> B_2) = \mathcal{L}_\theta(B_1)\, [\!>\, \mathcal{L}_\theta(B_2)$$

The meaning of disabling is achieved by directly using the logic disabling operator.

$$\mathcal{L}_\theta(\textbf{hide } g \textbf{ in } B) = \mathcal{L}_\theta(B)[\,\forall f \in g: i/f\,]$$

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.

$$\mathcal{L}_\theta(p([a])) = \theta(p)\,[a/gates(p)]$$

The meaning of a process call is retrieved by the environment, with the obvious parameter substitutions.

It is now time to define how an environment θ is built for a program. This is achieved by a semantic function $\mathcal{D}$ which is defined on a list of process declarations. The type of $\mathcal{D}$ is:

$$\mathcal{D} : Decl^* \rightarrow E$$

and is defined in the following way (using an auxiliary function $\mathcal{D}_1 : Decl^* \times E \rightarrow E$):

$\mathcal{D}\,(decls) = \nu\xi.\, \mathcal{D}_1\,(decls)\,(\xi)$
$\mathcal{D}_1$ (**process** p[g] := B **endproc**; *decls*) (ξ) = $\{<p[g], \mathcal{L}_\xi(B)>\} \cup \mathcal{D}_1\,(decls)\,(\xi)$
$\mathcal{D}_1\,(nil)\,(\xi) = \varnothing$

The definition of $\mathcal{D}_1$ augments the mapping with a new pair for every declaration. The pairs consist of a process heading and the meaning of the corresponding behaviour expression. Since process calls can be recursive, the meaning of behaviours should also be evaluated in the overall environment. The existence of the maximal fixed point is guaranteed by the consideration that the temporal logic formulae present in ξ can only appear under an even number of negations. The choice of the maximal vs. minimal fixed point to express recursion is due to the fact that the maximal fixed point semantics of a guarded recursion has as models with infinite sequences of (non-e) actions, which are not models of the minimal one, as shown in [Fan 88b].

Finally, the program semantics is given by a semantic function $\mathcal{P}$ whose type is:

$$\mathcal{P} : Prog \rightarrow TL$$

with the following definition:

$$\mathcal{P}\ (\ \text{decls}\ ;\ \text{B}) = \mathcal{L}_{\mathcal{D}(\text{decls})}(\text{B})$$

5. Relationship between temporal and operational semantics

The temporal semantics of a process is a temporal logic formula; the possible executions of the process are all the ordered sequences of actions which satisfy this formula.

The use of such a semantics implies an equivalence relation: two processes have the same meaning if the associated formulae are logically equivalent. From the meaning of temporal logic formulae follows that two equivalent processes will have the same set of sequence models. This fact recalls of the "string equivalence" [Hoa 81] which considers two processes to be equivalent if they are able to perform the same sequences of actions (traces).

We should, however, remember that the "official" LOTOS operational semantics [DIS 8807] is given by observational equivalence. The fundamental difference between observational and string equivalence is that the equivalence classes are, respectively, classes of action labelled trees (the synchronization trees of [Mil 80]) and classes of sets of sequences of actions. This is due to the difference between the basic model for the process used by the operational semantics, i. e. the labelled transition system, and the sequence model used for linear time temporal logic. This causes the different discriminating power: the observational equivalence has a finer discriminating power with respect to the string one.

What is of interest in our case is to show that there is, however, a correlation between the temporal and the operational semantics. This correlation is found in the formal correspondence between the logical and the string equivalence. For this purpose, we give the definition of simple abstractness:

A semantics $\mathcal{D}$ of a language L is simply abstract with respect to the operational semantics $\mathcal{O}$ of the same language if and only if:

$$\forall\, P, Q \in L\ \ \mathcal{D}(P) = \mathcal{D}(Q) \Leftrightarrow \mathcal{O}(P) = \mathcal{O}(Q)$$

We prove the simple abstractness of our temporal semantics with respect to the operational semantics modulo string equivalence by comparing the set of sequences which are models for the formula expressing the temporal semantics of a process in our language, and the set of traces originated by the same process following the string equivalence.

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 between any pair of meaningful actions.

If the two equivalences are to be considered as the same, this presence of e actions must be the only difference.

For this reason, we define a relation $\mathcal{R}$ between a set of traces and a set of sequences. This relation corresponds to eliminating the e's from the elements of the latter and therefore obtaining the same traces as the former.

The heart of the proof shows, by structural induction, that in any program its trace set is in relation $\mathcal{R}$ with its model sequence set. This fact, together with other features of the model sequences originated by the given temporal semantics is sufficient to prove the simple abstractness. The formal proof is given in the appendix. The proof does not hold for cases where unguarded recursion is present - this topic is discussed in [Fan 88b].

6. Justice and fairness properties

We have given previously some relationships between the temporal semantics of the language and its operational semantics. However, the main interest in giving temporal semantics is to be able to express properties which cannot be expressed by the operational semantics. Interesting cases are the *fairness* and *justice* properties. Both properties refer to infinite computations: a process is fair if in performing infinitely often a choice among more alternatives, each alternative will eventually be executed; justice is instead the requirement that if a process is constantly able to choose in a set of actions (that is, the actions of the set are constantly enabled) then eventually one action of the set will be performed [Pnu 88].

The reason for which these properties are not expressible with the operational semantics is in the model which underlies it. Labelled transition systems cannot distinguish fair or unfair behaviours, or behaviours with or without justice since such systems are not sensitive to computations which may be different at infinity. For this reason Fair Transition Systems [Pnu 88] have been introduced by Pnueli, as a model for fair concurrent systems.

It is instead well known that temporal logic can express fairness and justice requirements; in Table 3 two temporal formulae expressing the fairness and justice constraints as defined above are presented.

Fairness

$$[]\Diamond\Big(\bigvee_{a\in S} a\Big) \Rightarrow \Big(\bigwedge_{a\in S} \Diamond a\Big)$$

Justice

$$[]\Big(\bigvee_{a\in S} a\Big) \Rightarrow \Big(\bigvee_{a\in S} []\Diamond a\Big)$$

where $S \subseteq Act$

Table 3.

LOTOS has no syntax nor semantics to express that a behaviour is, for example, fair and another is not; it is hence impossible to verify any similar property on LOTOS specifications. For this purpose it is necessary to augment the language by syntactic extensions or by semantic enhancements.

The first approach - the syntactic extension - is proposed for CCS in [Par 85], where the behaviour syntax is augmented with temporal logic formulae which allow to express such properties. This would be possible also for LOTOS.

Another approach is to enhance the semantics of the actions of the language. In the following we show how the temporal semantics can be modified to express different justice requirements for Basic LOTOS.

The first case is that of imposing the following reasonable justice requirements, coherent with the basic principles of action synchronization in LOTOS:

- if a process is able to perform an external action, since its occurrence will in general depend on the external environment, it may be the case that the action is never performed (suppose for example that the process is synchronized with a non-terminating process which does not perform the same action);
- if a process is able to perform an internal action, eventually it must perform it.

The second requirement is expressible by modifying the formula expressing the semantics of internal action, by changing the unless operator with an until operator; we recall that the until operator means that its second operand must eventually be true:

$$\mathcal{L}'_\theta(i;B) = e\,\mathcal{U}\,[\,i \wedge O\,\mathcal{L}'_\theta(B)\,]$$

The semantics of observable actions may remain unchanged due to the first requirement:

$$\mathcal{L}'_\theta(g;B) = e\,\mathcal{W}\,[\,g \wedge O\,\mathcal{L}'_\theta(B)\,]$$

Anyway, the unobservable actions are generated also by the hiding operator. Whenever an action is hidden, its occurrence is no more dependent from the environment and must eventually happen, for the second requirement above. In order to express this, the formula expressing the semantics of hiding should be modified so that its sequence models do not include sequences which have an infinite sequence of e starting at a point where an hidden action is possible. This is not expressible with the temporal operators till now defined: we need to introduce a new operator, which we call "constrain", which in its general form has the following definition:

$$\sigma,i \models \phi\,/_{a\ \text{before}\ b} \quad \text{iff} \quad \sigma,i \models \phi \ \underline{\text{and}}\ \underline{\text{not}}\ (\ \underline{\text{exists}}\ k{>}i{:}\ ,\ \sigma,k \models [\,]\ a\ \underline{\text{and}}\ \underline{\text{exists}}\ \sigma'{:}\ (\sigma',\ i \models e\,\mathcal{U}\ b\ \underline{\text{and}}\ \sigma|_k \circ \sigma',i \models \phi))$$

where a, b are action predicates, $\sigma|_k$ the sequence σ cut at the k-th action.
This operator has the effect to drop the model sequences of ϕ which at a certain point become infinite sequences of a's, while other model sequences of ϕ exist which at the same point have a b action.
Using this operator, the new meaning of hiding is the following:

$$\mathcal{L}'_\theta(\textbf{hide}\ g\ \textbf{in}\ B) = \mathcal{L}_\theta(\textbf{hide}\ g\ \textbf{in}\ B)\,/_{e\ \text{before}\ i}$$

Consider the process:

a; **stop** ▯ i; **stop**

its temporal meaning, using the modified definition for the unobservable action prefix would be:

$$e\,\mathcal{W}\,(a \wedge O\ (e\,\mathcal{W}\,\neg O\ \textbf{true}))) \vee (e\,\mathcal{U}\,(i \wedge O\ (e\,\mathcal{W}\,\neg O\ \textbf{true}))) =$$
$$= e\,\mathcal{W}\,((a \vee i) \wedge O\ (e\,\mathcal{W}\,\neg O\ \textbf{true}))$$

which means that either action a or i may occur, but they also may never occur. To enforce the second justice requirement in this case it is necessary that the meaning of nondeterministic choice is modified in order to "spread" justice to the possible observable actions. The desired meaning would therefore be:

$$e\,\mathcal{U}\,((a \vee i) \wedge O\ (e\,\mathcal{W}\,\neg O\ \textbf{true}))$$

This can be obtained by applying the following definition:

$$\mathcal{L}'_\theta(B_1 \,[]\, B_2) = \mathcal{L}_\theta(B_1 \,[]\, B_2)\,/_{e\ \text{before}\ i}$$

The same reasoning holds for the disabling operator, whose modified semantics is:

$$\mathcal{L}'_\theta(B_1 \,[{>}\, B_2) = \mathcal{L}_\theta(B_1 \,[{>}\, B_2)\,/_{e\ \text{before}\ i}$$

For what concerns the enabling operator, it is also necessary to consider the possibility that the first action of the second process is an unobservable action which has to be eventually performed; however, the preceding behaviour terminates with a **stop**, which has as a model the infinite sequence of e's. This possibility is dropped by using the following definition:

$$\mathcal{L}'_\theta (B_1 >> B_2) = \mathcal{L}_\theta (B_1 >> B_2) /_{e \text{ before } i}$$

The definition of $\mathcal{L}'$ on the other operators remains the same of $\mathcal{L}$, since they do not introduce any possibility of infinite e sequence before an unobservable action.

A different way to define a justice semantics for LOTOS can be to assume that the external environment "properly" favours the execution of a process. By "properly" we can intend that the environment is always ready to offer/accept an action that the process is able to do; this guarantees justice. We can consider this assumption, by modifying the semantics of the language so that non-environment actions must eventually occurr. The modified semantics of the action prefix operators is:

$$\mathcal{L}''_\theta (i;B) = e\,\mathcal{U}\,[\, i \wedge O\,\mathcal{L}''_\theta (B)\,]$$

$$\mathcal{L}''_\theta (g;B) = e\,\mathcal{U}\,[\, g \wedge O\,\mathcal{L}''_\theta (B)\,]$$

In the definition of the semantics of the other operators it is necessary to make use of the constrain operator defined above, not only on unobservable actions, but also on observable ones.

As a simple example, we show that using the $\mathcal{L}''$ semantics it is possible to verify that the simple process:

process P[a,b,c] := a; b; P **endproc**

performs a infinitely often.
This amounts to proof the following implication:

$$\nu\xi.\,(e\,\mathcal{U}\,(a \wedge O\,(e\,\mathcal{U}\,(b \wedge O\,\xi)))) \quad \Rightarrow \quad []\Diamond\, a$$

which is straightforward.
This verification is not possible with the $\mathcal{L}$ or $\mathcal{L}'$ semantics, where due to the use of unless operatorsit is possible to enter infinite sequences of e's, which corresponds to say that the external environment is always able to perform non-a actions.

The discussion above shows that the method chosen to give the temporal semantics to our language is flexible enough to express particular assumptions on the semantics of the language, which allow to verify certain classes of properties. It is part of our future research to review completely the range of possible semantics for LOTOS, studying their relationships and their application to the expression and verification of justice and fairness properties.

7. Conclusive remarks

In the previous sections, we have used the compositional method proposed in [Bar 84, Bar 85, Bar 86] to give a temporal semantics for Basic LOTOS, proving that the given semantics is simply abstract with respect to the operational semantics modulo string equivalence. However, the official LOTOS operational semantics is given by observational equivalence, which is finer than string equivalence. For this reason, the comparison of our temporal semantics with string equivalence cannot be considered as conclusive, due to the low discriminating power of the latter.

In order to be closer to the official semantics and to re-acquire the discriminating power of the observational equivalence we would use another version of temporal logic (branching time), in which models are trees of actions. This increases the complexity of the semantics of the language and of the abstractness proofs and is considered a valid direction for future research in the specific case of LOTOS.

Our main objective is however to define a logic semantics which can express properties referring to infinite computations, such as justice and fairness.

We have started with a simple form of temporal semantics, which is at the same time close to the operational

semantics of the language and the simplest one in terms of temporal logic operators needed to express LOTOS behaviours - only the disabling LOTOS operator has been modeled by an unusual disabling temporal operator.

Next we have shown that in order to give a different semantics in terms of program properties we need to introduce new logic operators: we have discussed how to enrich our semantics to treat justice requirements.

In our work, starting form a simple temporal logic, we have encountered the need to increase the expressiveness of the logic semantics, by introducing new specific logic operators. A different approach is to start by defining an expressive logic for the language. The latter approach has been followed in [Hen 85], where an expressive logic (with respect to the observational equivalence, but not to properties of infinite computations) is defined to give logic semantics to CCS programs.

References

[Ban 86] B. Banieqbal, H. Barringer: "A Study of an Extended Temporal Language and a Temporal Fixed Point Calculus", University of Manchester, Technical Report UMCS-86-11-1, October 1986.

[Bar 84] H. Barringer, R. Kuiper, A.Pnueli: "Now you may Compose Temporal Logic Specifications", Proc. 16th ACM Symposium on the Theory of Computing, 1984, pp. 51-63.

[Bar 85] H. Barringer, R. Kuiper, A.Pnueli: "A Compositional Temporal Approach to a CSP-like Language", in E.J. Neuhold and G. Chroust eds., Formal Models of Programming, IFIP, North-Holland, pp. 207-227.

[Bar 86] H. Barringer, "Using Temporal Logic in the Compositional Specifications of Concurrent Systems", University of Manchester, Technical Report UMCS-86-10-1, October 1986.

[Bol 87] T. Bolognesi, E. Brinskma, "Introduction to the ISO Specification Language LOTOS", Computer Networks & ISDN Systems, vol. 14, n. 1, January 1987, pp. 25-29.

[DIS 8807] ISO - Information Processing Systems - Open System Interconnection, LOTOS, a Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, DIS 8807, 1987.

[Den 88] R. De Nicola, D. Latella, "Basic Ideas for Temporal Logic Semantics of a Subset of TCCS", Draft, June 1988.

[Fan 88a] A. Fantechi, S. Gnesi, C. Laneve, "Principles for a Temporal Semantics of LOTOS", I.E.I. Internal Report B4-36, August 1988.

[Fan 88b] A. Fantechi, S. Gnesi, C. Laneve, "A Logic Approach to Guarded and Unguarded Recursion", November 1988, Submitted to 16th ICALP.

[Fis 87] M.D. Fisher, "Temporal Logics for Abstract Semantics", University of Manchester, Technical Report UMCS-87-12-4, December 1987.

[Hen 85] M. Hennessy, R. Milner, "Algebraic Laws for Nondeterminism and Concurrency", Journal of ACM, vol. 32, n. 1, January 1985, pp. 137-161.

[Hoa 81] C.A.R. Hoare, "A Model for Communicating Sequential Processes", Technical Monograph Prg-22, Computing Laboratory, University of Oxford, 1981.

[Lam 83] L. Lamport, "What Good is Temporal Logic?", Proceedings of IFIP'83, North Holland, 1983, pp.657-668.

[Man 81] Z. Manna, A. Pnueli, "Verification of Concurrent Programs: The Temporal Framework, in R.S. Boyer, J.S. Moore, eds., "Correctness Problem in Computer Science, Academic Press, 1981, pp.215-273.

[Mil 80] R. Milner, "A Calculus of Communicating Systems", Lecture Notes in Computer Science vol. 92, 1980.

[Par 85] J. Parrow, R. Gustavsson, "Modelling Distributed Systems in an Extension of CCS with Infinite Experiments and Temporal Logic", in Y. Yemini, R. Strom, S. Yemini, eds., "Protocol Specification, Testing, and Verification, IV", North-Holland, 1985, pp. 309-348.

[Pnu 81] A. Pnueli, "The Temporal Semantics of Concurrent Programs", Theoretical Computer Science, vol.13, 1981, pp.45-60.

[Pnu 88] A. Pnueli, "Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends", REX School /Workshop, Noordwijkerhout, May 1988.

[Ros 86] R. Rosner, "A Choppy Logic", Master Degree Thesis, Weizmann Institute of Science, January 1986.

APPENDIX

We need some preliminary definitions and notations:

$Act = \{g \mid g \in Gates\} \cup \{i\}$ -- the set of Basic LOTOS actions;

$\mathfrak{S} = (Act \cup \{e\})^* \cup (Act \cup \{e\})^\omega$ -- the set of possible model sequences;

$\mathfrak{T} = Act^* \cup Act^\omega$ -- the set of possible traces.

We use σ for ranging over $\mathfrak{S}$ and t for ranging over $\mathfrak{T}$.

We use e^* to denote any finite sequence of consecutive e's, and e^ω to represent the infinite sequence of consecutive e's. λ denotes the empty string. A dot (.) denotes the concatenation of two strings.

The notation $\sigma \models \phi$ is an abbreviation for $\sigma,1 \models \phi$.

Definition 1

The relation $\mathfrak{R} \subset \mathfrak{S} \times \mathfrak{T}$ is so defined:

$\sigma \mathfrak{R} t$ iff (($\sigma = e^*.a.\sigma'$ and $t = a.t'$ and $\sigma' \mathfrak{R} t'$) or ($\sigma = e^*$ and $t = \lambda$) or ($\sigma = e^\omega$ and $t = \lambda$)).

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

Definition 2

We extend the relation $\mathfrak{R}$ to the sets of sequences and traces:

$\mathfrak{R} \subset \mathbb{P}(\mathfrak{S}) \times \mathbb{P}(\mathfrak{T})$

$S \mathfrak{R} T$ iff $\forall \sigma \in S \ \exists t \in T$: $\sigma \mathfrak{R} t$ and $\forall t \in T \ \exists \sigma \in S$: $\sigma \mathfrak{R} t$

Proposition 1

$\forall \sigma \in \mathfrak{S} \quad \exists! \, t \in \mathfrak{T}$: $\sigma \mathfrak{R} t$

Proof (By contradiction)

Suppose that, given $\sigma \in \mathfrak{S}$, we have $t_1, t_2 \in \mathfrak{T}$, $t_1 \neq t_2$, $\sigma \mathfrak{R} t_1$ and $\sigma \mathfrak{R} t_2$. Four cases are possible:

1) $t_1 = \lambda$; hence, by definition of $\mathfrak{R}$, σ should be e^n for some non negative n, or e^ω.

Since $t_1 \neq t_2$, for some $a \in Act$, it should be $t_2 = a.t_2'$. By definition of $\mathfrak{R}$ it should be

$\sigma = a.\sigma'$, with $\sigma' \mathfrak{R} t'_2$, contradicting what said above;

2) $t_2 = \lambda$; is symmetrical of the above case;

3) $t_1 = a.t_1'$, $t_2 = b.t_2'$, $a \neq b$;

by definition of $\mathfrak{R}$ it should be both $\sigma = a.\sigma'$ and $\sigma = b.\sigma''$. Contradiction.

4) $t_1 = a.t_1'$, $t_2 = a.t_2'$, $t_1' \neq t_2'$;

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. The contrary ($\forall t \in \mathfrak{T} \quad \exists! \, \sigma \in \mathfrak{S}$: $\sigma \mathfrak{R} t$) is obviously not true, by definition of $\mathfrak{T}$ and $\mathfrak{S}$. ♦

Proposition 2

$\forall S \subseteq \mathfrak{S} \quad \exists! \, T \subseteq \mathfrak{T}$: $S \mathfrak{R} T$

Proof (By contradiction)

Suppose that, for a given $S \subseteq \mathfrak{S}$, we have $T, T' \subseteq \mathfrak{T}$, $T \neq T'$: $S \mathfrak{R} T$ and $S \mathfrak{R} T'$.

Since $T \neq T'$ (suppose $T' \subset T$), there exists at least an element $t \in T$, $t \notin T'$. Since $S \mathfrak{R} T$, there exists a $\sigma \in S$ such that $\sigma \mathfrak{R} t$. But since $S \mathfrak{R} T'$, there exists a $t' \in T'$ such that $\sigma \mathfrak{R} t'$. For Prop.1, $t = t'$, and hence the contradiction.

♦

Definition 3

Let us define the notion of e-maximal set of sequences on $Act \cup \{e\}$:

A set $S \subseteq \mathfrak{S}$ is said e-maximal if:

$\sigma_1.e^k.\sigma_2 \in S \Rightarrow \forall n \geq 0, \sigma_1.e^n.\sigma_2 \in S$ and $\sigma_1.e^\omega \in S$
with $\sigma_1, \sigma_2 \in \mathbb{S}$ and σ_2 possibly empty.

Theorem 1
Let us denote with S_P the set of model sequences of the temporal semantics of a Basic LOTOS program P (that is: $S_P = \{\sigma \mid \sigma \models \mathcal{L}_\theta(P)\}$) and with T_P the set of traces of the same program.
Then $S_P \mathcal{R} T_P$, for every Basic LOTOS program P.

Proof (By structural induction on the syntax of Basic LOTOS)
P = stop
$S_P = \{\sigma \mid \sigma \models e\mathcal{W} \neg\circ \mathbf{true}\} = \{e^*\} \cup \{e^\omega\}$
$T_P = \{\lambda\}$
By definition of $\mathcal{R}$, we have $S_P \mathcal{R} T_P$. This is the basis of the induction.

P = i;B
$S_P = \{\sigma \mid \sigma \models e\mathcal{W} [i \wedge \circ \mathcal{L}_\theta(B)]\} = \{e^*.i.\sigma' \mid \sigma' \in S_B\} \cup \{e^\omega\}$
$T_P = \{i.t \mid t \in T_B\} \cup \{\lambda\}$
By definition of $\mathcal{R}$, we have $S_P \mathcal{R} T_P$ if $S_B \mathcal{R} T_B$.

P = g;B
$S_P = \{\sigma \mid \sigma \models e\mathcal{W} [g \wedge \circ \mathcal{L}_\theta(B)]\}$
$= \{e^*.g.\sigma' \mid \sigma' \in S_B\} \cup \{e^\omega\}$
$T_P = \{g.t \mid t \in T_B\} \cup \{\lambda\}$
By definition of $\mathcal{R}$, we have $S_P \mathcal{R} T_P$ if $S_B \mathcal{R} T_B$.

P = $B_1 \,[]\, B_2$
$S_P = \{\sigma \mid \sigma \models \mathcal{L}_\theta(B_1) \vee \mathcal{L}_\theta(B_2)\} = \{\sigma \mid \sigma \models \mathcal{L}_\theta(B_1)$ or $\sigma \models \mathcal{L}_\theta(B_2)\} =$
$= S_{B1} \cup S_{B2}$ -- due to the interleaving constraint
$T_P = T_{B1} \cup T_{B2}$
Hence, we have: $S_{B1} \mathcal{R} T_{B1}$ and $S_{B2} \mathcal{R} T_{B2}$ implies $S_P \mathcal{R} T_P$.

P =B1/g/B2
$S_P = \{\sigma \mid \sigma \models (\mathcal{L}_\theta(B_1)\,[\text{subst1}] \wedge \mathcal{L}_\theta(B_2)\,[\text{subst2}])\,[\text{subst3}]\}$
where:
$\text{subst1} = (e \vee i_2 \vee \bigvee_{f \notin g, f \in \alpha(B_2)} f_2) / e, i_1 / i, \forall f \notin g, f \in \alpha(B_1): f_1 / f$
$\text{subst2} = (e \vee i_1 \vee \bigvee_{f \notin g, f \in \alpha(B_1)} f_1) / e, i_2 / i, \forall f \notin g, f \in \alpha(B_2): f_2 / f$
$\text{subst3} = \forall f \notin g\ f/f_1, \forall f \notin g\ f/f_2, i/i_1, i/i_2$

The definition of T_P 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.
$T_P = \{t \mid t \in IST_g(t_1,t_2), t_1 \in T_{B1}, t_2 \in T_{B2}\}$

The following definition of $IST_g(t_1,t_2)$ is easily derived from the operational semantics:

$IST_g(t_1,t_2) = \{ t \mid t = act.t',\ act \notin g,\ t' \in IST_g(t_1',t_2),\ t_1 = act.t_1' \}$

$\cup$ { t | t = act.t', act$\notin$ g, t'$\in$ $IST_g(t_1,t_2')$, t_2 = act.t_2' }
$\cup$ { t | t = act.t', act$\in$ g, t'$\in$ $IST_g(t_1',t_2')$, t_1 = act.t_1', t_2 = act.t_2' }
$\cup$ {λ}

Let us now define an analogous set $ISS_g(\sigma_1, \sigma_2) \subseteq \mathfrak{S}$.

$ISS_g(\sigma_1,\sigma_2)$ =
{σ | $\sigma = e^n.act.\sigma'$, act$\notin$ g, $\sigma'\in$ $ISS_g(\sigma_1',\sigma_2')$, $\sigma_1 = e^n.act.\sigma_1'$, $\sigma_2 = e^{n+1}.\sigma_2'$, n$\geq$0}
$\cup$ {σ | $\sigma = e^n.act.\sigma'$, act$\notin$ g, $\sigma'\in$ $ISS_g(\sigma_1',\sigma_2')$, $\sigma_1 = e^{n+1}.\sigma_1'$, $\sigma_2 = e^n$ act $.\sigma_2'$, n$\geq$0}
$\cup$ {σ | $\sigma = e^n.act.\sigma'$, act$\in$ g, $\sigma'\in$ $ISS_g(\sigma_1',\sigma_2')$, $\sigma_1 = e^n.act.\sigma_1'$, $\sigma_2 = e^n$ act $.\sigma_2'$, n$\geq$0}
$\cup$ {e^*} $\cup$ {e^ω}

In order to prove this part of the theorem, we need the following lemma:

Lemma 1.1
$\sigma \in ISS_g(\sigma_1,\sigma_2)$ if $\sigma = \sigma'[subst_3]$ and $\sigma' \in \sigma_1[subst_1]$ and $\sigma' \in \sigma_2[subst_2]$
where $\sigma[subst_3]$ denotes the (only) sequence obtained from σ by the relabelling $subst_3$, and $\sigma[subst_1]$, $\sigma[subst_2]$ denote the set of sequences obtained from, respectively, σ_1,σ_2 by the relabelling $subst_1$, $subst_2$.

Proof (By induction on the length of σ_1,σ_2)
We have several possible structures for σ_1 and σ_2 (cases 1,2,3 are the inductive steps; cases 4,5,6 are the basis of the induction):
1) $\sigma_1 = e^n.act.\sigma_1'$, $\sigma_2 = e^{n+1}.\sigma_2'$, n$\geq$0, act $\notin$ g
The only σ^- such that: $\sigma^- \in \sigma_1[subst_1]$, $\sigma^- \in \sigma_2[subst_2]$ is $\sigma^- = e^n.\underline{1}(act).\sigma^{-'}$, where $\sigma^{-'} \in \sigma_1'[subst_1]$, $\sigma^{-'} \in \sigma_2'[subst_2]$, since $subst_1$ substitutes act with $\underline{1}$(act) and $subst_2$ substitutes the n+1-th e with $\underline{1}$(act).
If we take $\sigma = \sigma^-[subst_3]$, we have: $\sigma = e^n.act.\sigma'$, where $\sigma' = \sigma^{-'}[subst_3]$.
This means that, since the inductive hypothesis is true for the shorter string σ', we have $\sigma' \in ISS_g(\sigma_1',\sigma_2')$. Hence, by definition of ISS, $\sigma \in ISS_g(\sigma_1,\sigma_2)$.

2) $\sigma_1 = e^{n+1}.\sigma_1'$, $\sigma_2 = e^n.act.\sigma_2'$, n$\geq$0, act $\notin$ g
This case is simply the symmetrical of the previous one; the proof is obviously the same.

3) $\sigma_1 = e^n.act.\sigma_1'$, $\sigma_2 = e^n.act.\sigma_2'$, n$\geq$0, act $\in$ g
The only σ^- such that: $\sigma^- \in \sigma_1[subst_1]$, $\sigma^- \in \sigma_2[subst_2]$ is $\sigma^- = e^n.act.\sigma^{-'}$, where $\sigma^{-'} \in \sigma_1'[subst_1]$, $\sigma^{-'} \in \sigma_2'[subst_2]$; that is, $subst_1$ and $subst_2$ behave as the identity on the n+1-long prefixes of $\sigma_1$and σ_2.
If we take $\sigma = \sigma^-[subst_3]$, we have: $\sigma = e^n.act.\sigma'$, where $\sigma' = \sigma^{-'}[subst_3]$. (Again, $subst_3$ behaves as the identity on the n+1-long prefix of σ^-).
This means that, since the inductive hypothesis is true for the shorter string σ', we have $\sigma' \in ISS_g(\sigma_1',\sigma_2')$. Hence, by definition of ISS, $\sigma \in ISS_g(\sigma_1,\sigma_2)$.

4) $\sigma_1 = e^n$, $\sigma_2 = e^n$, n$\geq$0.
The only σ^- such that: $\sigma^- \in \sigma_1[subst_1]$, $\sigma^- \in \sigma_2[subst_2]$ is $\sigma^- = e^n$, since $subst_1$ and $subst_2$ can be reduced to the identical substitution. In this case also $subst_3$ is the identity, hence
$\sigma = \sigma^-[subst_3] = e^n \in ISS_g(\sigma_1,\sigma_2)$.

5) $\sigma_1 = e^\omega$, $\sigma_2 = e^\omega$.
By the same reasoning of the above case we obtain $\sigma = e^\omega \in ISS_g(\sigma_1,\sigma_2)$.

6) In all the remaining cases:

$\sigma_1 = e^n,\ \sigma_2 = e^m,\ n \neq m;$

$\sigma_1 = e^n.act.\sigma_1',\ \sigma_2 = e^m.act.\sigma_2',\ n \neq m,\ act \in g;$

$\sigma_1 = e^n.act_1.\sigma_1',\ \sigma_2 = e^m.act_2.\sigma_2',\ act_1 \neq act_2,\ act_1, act_2 \in g;$

$\sigma_1 = e^n,\ \sigma_2 = e^m.act.\sigma_2',\ n \leq m,$ and its symmetrical;

$\sigma_1 = e^\omega,\ \sigma_2 = e^m,$ and its symmetrical;

$\sigma_1 = e^\omega,\ \sigma_2 = e^m.act.\sigma_2',$ and its symmetrical;

we can find no σ^- such that: $\sigma^- \in \sigma_1[subst_1],\ \sigma^- \in \sigma_2\ [subst_2]$; hence this cases satisfy the hypothesis trivially. ♦

(Proof of Theorem 1, case P =B1/g/B2 continued)

By lemma 1.1, we have:

$S_P \subseteq \{\sigma \mid \sigma \in ISS_g(\sigma_1,\sigma_2),\ \sigma_1 \in S_{B1},\ \sigma_2 \in S_{B2}\}$

Suppose now that there exists a $\sigma' \in ISS_g(\sigma_1,\sigma_2),\ \sigma' \notin S_P$;

σ' 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 1.1 it would belong to S_P.

Hence it should be: $\sigma' = e^n,\ n \geq 0,$ or $\sigma' = e^\omega$. 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 $\sigma_1' = e^n = \sigma_2'$ (respectively, $\sigma_1' = e^\omega = \sigma_2'$), from which σ' can be obtained by identical substitutions. Hence σ' should belong to S_P.

Note that this part of the proof, similarly to lemma 1.1, is again by induction on the length of the sequences. Moreover, it is intertwined with the proof of e-maximality of all Basic LOTOS programs (by proposition 3 below).

In conclusion, we have:

$S_P = \{\sigma \mid \sigma \in ISS_g(\sigma_1,\sigma_2),\ \sigma_1 \in S_{B1},\ \sigma_2 \in S_{B2}\}$

and, by comparing the definition of the sets IST and ISS,

$S_{B1} \mathcal{R} T_{B1}$ and $S_{B2} \mathcal{R} T_{B2}$ implies $S_P \mathcal{R} T_P$

$P = B_1 >> B_2$

$S_P = \{\sigma \mid \sigma \models \mathcal{L}_\theta(B_1)\ \mathbb{C}\ \mathcal{L}_\theta(B_2)\} =$

$= \{\sigma_1.\sigma_2 \mid \sigma_1 \models \mathcal{L}_\theta(B_1),\ \sigma_1$ finite, $\sigma_2 \models \mathcal{L}_\theta(B_2)\} \cup \{\sigma \mid \sigma \models \mathcal{L}_\theta(B_1),\ \sigma$ infinite$\} =$

$= \{\sigma_1.\sigma_2 \mid \sigma_1 \in S_{B1},\ \sigma_1$ finite, $\sigma_2 \in S_{B2}\} \cup \{\sigma \mid \sigma \in S_{B1},\ \sigma$ infinite$\}$

$T_P = \{t_1.t_2 \mid t_1 \in T_{B1},\ t_1$ finite, $t_2 \in T_{B2}\} \cup \{t \mid t \in T_{B1},\ t$ infinite$\}$

Hence, we have: $S_{B1} \mathcal{R} T_{B1}$ and $S_{B2} \mathcal{R} T_{B2}$ implies $S_P \mathcal{R} T_P$.

$P = B1 [> B2$

$S_P = \{\sigma \mid \sigma \models \mathcal{L}_\theta(B_1)\ [>\ \mathcal{L}_\theta(B_2)\} =$

$= \{\sigma_1.\sigma_2 \mid \sigma_1$ finite, $\sigma_2 \models \mathcal{L}_\theta(B_2),\ \exists\, \sigma_3\colon \sigma_1.\sigma_3 \models \mathcal{L}_\theta(B_1)\} \cup \{\sigma \mid \sigma \models \mathcal{L}_\theta(B_1),\ \sigma$ infinite$\} =$

$= \{\sigma_1.\sigma_2 \mid \sigma_1$ finite, $\sigma_2 \in S_{B2},\ \exists\, \sigma_3\colon \sigma_1.\sigma_3 \in S_{B1}\} \cup \{\sigma \mid \sigma \in S_{B1},\ \sigma$ infinite$\}$

$T_P = \{t_1.t_2 \mid t_1$ finite, $t_2 \in T_{B2},\ \exists\, t_3\colon t_1.t_3 \in T_{B1}\} \cup \{t \mid t \in T_{B1},\ t$ infinite$\} =$

Hence, we have: $S_{B1} \mathcal{R} T_{B1}$ and $S_{B2} \mathcal{R} T_{B2}$ implies $S_P \mathcal{R} T_P$.

*$P =$ **hide** g **in** B*

$S_P = \{\sigma \mid \sigma \models \mathcal{L}_\theta(B)\ [\forall f \in g\colon i/f]\} = \{\sigma \mid \sigma = \sigma'[\forall f \in g\colon i/f],\ \sigma' \in S_B\}$

$T_P = \{t \mid t = t'[\forall f \in g\colon i/f],\ t' \in T_B\}$

By definition of $\mathcal{R}$, we have $S_P \mathcal{R} T_P$ if $S_B \mathcal{R} T_B$.

P = p([a]), where process p[g] := B endproc

$S_P = \{\sigma \mid \sigma = \sigma'[a/g],\ \sigma' \in S_B\}$
$T_P = \{t \mid t = t'[a/g],\ \sigma' \in T_B\}$
By definition of $\mathcal{R}$, we have $S_P \mathcal{R} T_P$ if $S_B \mathcal{R} T_B$. ♦

Theorem 2
$\forall$ P, Q Basic LOTOS programs: $S_P = S_Q \Rightarrow T_P = T_Q$.
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.

Proof
By theorem 1 we have $S_P \mathcal{R} T_P$ and $S_Q \mathcal{R} T_Q$. But $S_P = S_Q$, and by prop. 2 we have: $T_P = T_Q$.

♦

Proposition 3
$\forall$ P Basic LOTOS program: S_P is e-maximal.

Proof (by structural induction on the syntax of Basic LOTOS)

1) The meanings of: **stop**, i;B, g;B(x), due to the use of "e unless..." operations, have as model sequences all those with subsequences of e's of any length, also infinite. Hence $S_{\mathbf{stop}}$ is e-maximal, and $S_{oi;B}$ is e-maximal if S_B is e-maximal (oi is any Basic LOTOS action).
2) The meaning of the nondeterministic choice does not introduce any more e in their model sequences than those present in the model sequences of their component processes. Hence $S_{[b]\rightarrow B}$ is e-maximal if S_B is e-maximal and $S_{B1[]B2}$ is e-maximal if S_{B1} and S_{B2} are e-maximal.
The same holds for enabling and for disabling.
3) The meanings of hiding and of process instantiation relabel only non-e actions in the model sequences of their component processes. Hence $S_{\mathbf{hide}\ g\ \mathbf{in}\ B}$ is e-maximal if S_B is e-maximal and $S_{p([a][v])}$ (where **process** p[g][v] := B **endproc**) is e-maximal if S_B is e-maximal.
4) For what concerns the parallel composition operator it is easy to see that by definition the ISS_g set is e-maximal. Hence $S_{B1|g|B2}$ is e-maximal. (Note that in the proof that $S_{B1|g|B2}$ depends on the ISS_g set we have used the e-maximality of its components - this is not a trick, rather it fits in the structural inductive nature of the present proof). ♦

Proposition 4
$\forall T \subseteq \mathcal{T}$ $\exists!\ S \subseteq \mathcal{S}$, S e-maximal: $S \mathcal{R} T$.

Proof (by contradiction)
Suppose that, for a given $T \subseteq \mathcal{T}$, we have S, $S' \subseteq \mathcal{S}$, $S \neq S'$, S,S' e-maximal: $S \mathcal{R} T$ and $S' \mathcal{R} T$.
Since $S \neq S'$ (suppose $S' \subset S$), there exists at least an element $\sigma \in S$, $\sigma \notin S'$. Since $S \mathcal{R} T$, there exists a $t \in T$ such that $\sigma \mathcal{R} t$. But since $S' \mathcal{R} T$, there exist a $\sigma' \in S'$ such that $\sigma' \mathcal{R} t$.
By definition of $\mathcal{R}$, if $\sigma \mathcal{R} t$ and $\sigma' \mathcal{R} t$, σ and σ' may differ only for the length of some of their subsequences of e. Since S' is e-maximal and $\sigma' \in S'$, we have also $\sigma \in S'$, which contradicts the hypothesis. ♦

Theorem 3
$\forall$ P, Q Basic LOTOS programs: $T_P = T_Q \Rightarrow S_P = S_Q$.
This theorem, saying that two programs with the same operational meaning have the same temporal meaning, gives the other side of the abstractness proof.

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: $S_P = S_Q$. ♦