Consiglio Nazionale delle Ricerche

IST. EL. INF.
BIBLIOTECA
Posiz. ARCHIVIO B4-36

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

PISA

Verification of Partial Properties
Temporal Semantics

Section 3.2 of "Correctness Preserving Transformation"
ESPRIT Project 2304 LOTOSPHERE

A. Fantechi, S. Gnesi, C. Laneve

Nota Interna B4-36
Agosto 1990

Section 3.2

Temporal Semantics

3.2.1 Introduction

In this section we propose giving LOTOS programs a temporal semantics [P81], i.e. the temporal logic formula expressing the properties of their execution sequences. The advantages of the temporal semantics 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.
These observations make it possible to unify these two verification problems, as they amount to checking the validity of a logic formula. Here mechanic tools can help, such as decision procedures and theorem provers.

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 to verify its properties.The purpose of this section is to present the temporal semantics of Basic LOTOS [BB87], a simplified version of LOTOS without value passing. A linear version of temporal logic and the compositional approach presented in [BKP84, BKP85], are used to define the temporal semantics of this language in a denotational style.

The official semantics of LOTOS is given operationally; we should hence study the relationships between the operational semantics and the temporal one proposed. Different relations (related to different choices of the equivalence on the operational semantics) will induce different refinement degrees in the class of properties captured by the temporal semantics. Since the 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 [L83]:
- 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.).

Actually, 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 Basic LOTOS presented in this section is expressive with respect to its operational semantics modulo trace (or string) equivalence [H81]; this semantics allows us to prove those safety properties which can be stated for a LOTOS process, but not liveness properties. To capture *liveness properties* only maximal computations must be considered. This means that we must distinguish between processes which have *different maximal traces*. The corresponding equivalence is *maximal trace equivalence* [P85].

In order to show the flexibility of the temporal semantics approach to the verification of partial properties, we give first some verification problems on simple processes (Sect.3.2.2). Then, after a formal description of the temporal semantics (Sect. 3.2.4), we will give the details of the solutions of the verification examples (Sect. 3.2.5). The examples will show the verification of some simple safety property, but will show also that it is possible to verify a particular class of interesting liveness properties, i.e. fairness, by adding proper constraints on the behaviours.

3.2.2 Examples of partial properties of LOTOS specifications

In order to show the use of the temporal semantics to prove properties of Basic LOTOS specifications, we give some example verifications on some simple Basic LOTOS processes. The first case is the process:

process P[a,b,c] := (a; P[a,b,c]) [] (b;c; P[a,b,c]) **end process**

which is able to perform the action a or the sequence of actions b,c and then transforms in itself.

A form of partial correctness that can be required for this process can be expressed by the statement: "whenever a *c* action is performed after an *a* action, surely a *b* action has been performed in between the *a* and *c* actions". This property can be expressed by the formula (for details on the temporal logic used see Sect. 3.2.3):

$$[] \neg (a \wedge O((\neg b) \mathcal{U} c))$$

read as "it is always the case that is not true that the process both performs an *a* action and subsequently does not perform a *b* action until a *c* action is performed".

If we indicate with $\mathcal{L}(P)$ the temporal semantics of the process P, which is the formula expressing all the properties enjoyed by P, in order to tell whether P[a, b, c] satisfies this safety property, we should prove:

$$\mathcal{L}(P[a,b,c]) \Rightarrow [] \neg (a \wedge O((\neg b) \mathcal{U} c))$$

In order to discuss an example closer to the real world, we consider a simplified, Basic LOTOS, specification of the Caller process of POTS (Plain Old Telephone System), presented in [FLS89]. Simplifications are as follows:
1) In order to use Basic LOTOS, we cannot use values; for this purpose, only one Caller process is defined, with no "number" parameter, in the assumption that several copies of it, with different names, would be present in a hypothetical basic LOTOS specification of the whole system.
2) For the same reason, the user is able to dial only two numbers.

3) We have eliminated the case in which the user can find the phone busy because another extension at the same number is engaged in a call.

The process Caller as specified below is responsible to interact with the user by getting the number after the user has lifted the receiver, and by letting him talk; the talk can be interrupted at any time by one of the partners hanging the receiver. The process Caller interacts with a process Controller and with other Caller processes which interact with the other users of the systems. Both Controller and the other Callers are not considered here.

```
process Caller
      [Usr_Offhook,Ctl_Originate,Ctl_Connect_Ok,Usr_Voice,Ctl_Rec_Voice,
       Ctl_Send_Voice,Usr_Hang_up,Ctl_Disconn,Ctl_Disind,
       Usr_Dial_1, Usr_Dial_2,Ctl_Tone_Ok,Ctl_Dial_1,Ctl_Dial_2] :    noexit :=
 (
  Usr_Offhook;
  Ctl_Originate;
  (( Dialing [Usr_Dial_1, Usr_Dial_2,Ctl_Tone_Ok,Ctl_Dial_1,Ctl_Dial_2]
     >>
        ( Normal_Connection [Ctl_Connect_Ok,Usr_Voice,Ctl_Rec_Voice,Ctl_Send_Voice]
          []
          Number_is_Busy
        )
   )
   [> User_Disconnect [Usr_Hang_up,Ctl_ Disconn,Ctl_ Disind]
  )
 )

where

process Dialing
     [Usr_Dial_1, Usr_Dial_2,Ctl_Tone_Ok,Ctl_Dial_1,Ctl_Dial_2] :    exit :=
 (Ctl_Tone_Ok;
  (Usr_Dial_1; Ctl_Dial_1;  exit
    []
   Usr_Dial_2; Ctl_Dial_2;  exit
 ))
endproc (*Dialing*)

process Normal_Connection
      [Ctl_Connect_Ok,Usr_Voice,Ctl_Rec_Voice,Ctl_Send_Voice] :   noexit :=
 (Ctl_Connect_Ok;
  Talking[Usr_Voice,Ctl_Rec_Voice,Ctl_Send_Voice]
 )
endproc (*Normal_Connection*)

process Number_is_Busy : noexit :=
  stop
endproc (*Number_is_Busy*)

process Talking [Usr_Voice,Ctl_Rec_Voice,Ctl_Send_Voice] : noexit :=
  (Ctl_Rec_Voice; Usr_Voice; Talking[Usr_Voice,Ctl_Rec_Voice,Ctl_Send_Voice]
  []
  Usr_ Voice; Ctl_Send_Voice; Talking[Usr_Voice,Ctl_Rec_Voice,Ctl_Send_Voice]
  )
endproc (* Talking *)

process User_Disconnect [Usr_Hang_up,Ctl_Disconn,Ctl_Disind] : noexit :=
```

```
  (Usr_ Hang_up; Ctl_Disconn;  stop
  []
  Ctl_Disind; Ctl_Disconn; Usr_Hang_up;  stop
  )
endproc (* User_Disconnect *)

endproc (* Caller *)
```

A safety property that can be asked to this system is:" the user cannot talk if he has not lifted the receiver " expressed by the temporal logic formula:

$$\neg (\neg \text{Usr\_Offhook } \mathcal{U} \text{ Usr\_Voice}).$$

So proving that the previous specification satisfies this safety property amounts to verify the validity of the following implication:

$$\mathcal{L}(\text{Caller}) \Rightarrow \neg (\neg \text{Usr\_Offhook } \mathcal{U} \text{ Usr\_Voice})$$

The same reasoning can be done to prove that it also satisfies other properties having a similar structure, like for example, "the calling user cannot talk if he has not dialed a number before", and so on.

The third example is drawn from the same POTS specification and regards the Check_In_Use process, which we present in a simplified Basic LOTOS form.

```
process Checking_00 [Ctl_1_Orig_Ok,Ctl_2_Orig_Ok,Ctl_1_Orig_Busy,Ctl_2_Orig_Busy,
                     Ctl_1_Dial_Ok,Ctl_2_Dial_Ok,Ctl_1_Dial_Busy,Ctl_2_Dial_Busy,
                     Ctl_1_Disconn,Ctl_2_Disconn,Ctl_Not_Constr_Signal] :  noexit :=
  ( Ctl_1_Orig_Ok; Checking_10 [...]
  [] Ctl_2_Orig_Ok; Checking_01 [...]
  [] Ctl_1_Dial_Ok; Checking_10 [...]
  [] Ctl_2_Dial_Ok; Checking_01 [...]
  [] Ctl_1_Disconn; Checking_00 [...]
  [] Ctl_2_Disconn; Checking_00 [...]
  [] Ctl_1_Not_Constr_Signal; Checking_00 [...]
  )
endproc (* Checking_00 *)

process Checking_01 [Ctl_1_Orig_Ok,Ctl_2_Orig_Ok,Ctl_1_Orig_Busy,Ctl_2_Orig_Busy,
                     Ctl_1_Dial_Ok,Ctl_2_Dial_Ok,Ctl_1_Dial_Busy,Ctl_2_Dial_Busy,
                     Ctl_1_Disconn,Ctl_2_Disconn,Ctl_Not_Constr_Signal] :  noexit :=
  ( Ctl_1_Orig_Ok; Checking_11 [...]
  [] Ctl_2_Orig_Busy; Checking_01 [...]
  [] Ctl_1_Dial_Ok; Checking_11 [...]
  [] Ctl_2_Dial_Busy; Checking_01 [...]
  [] Ctl_1_Disconn; Checking_01 [...]
  [] Ctl_2_Disconn; Checking_00 [...]
  [] Ctl_1_Not_Constr_Signal; Checking_01 [...]
  )
endproc (* Checking_01 *)

process Checking_10 [Ctl_1_Orig_Ok,Ctl_2_Orig_Ok,Ctl_1_Orig_Busy,Ctl_2_Orig_Busy,
                     Ctl_1_Dial_Ok,Ctl_2_Dial_Ok,Ctl_1_Dial_Busy,Ctl_2_Dial_Busy,
                     Ctl_1_Disconn,Ctl_2_Disconn,Ctl_Not_Constr_Signal] :  noexit :=
  ( Ctl_1_Orig_Busy; Checking_10 [...]
  [] Ctl_2_Orig_Ok; Checking_11 [...]
  [] Ctl_1_Dial_Busy; Checking_10 [...]
```

```
  [] Ctl_2_Dial_Ok; Checking_11 [...]
  [] Ctl_1_Disconn; Checking_00 [...]
  [] Ctl_2_Disconn; Checking_10 [...]
  [] Ctl_1_Not_Constr_Signal; Checking_10 [...]
  )
endproc (* Checking_10 *)

process Checking_11 [Ctl_1_Orig_Ok,Ctl_2_Orig_Ok,Ctl_1_Orig_Busy,Ctl_2_Orig_Busy,
                     Ctl_1_Dial_Ok,Ctl_2_Dial_Ok,Ctl_1_Dial_Busy,Ctl_2_Dial_Busy,
                     Ctl_1_Disconn,Ctl_2_Disconn,Ctl_Not_Constr_Signal] : noexit :=
  ( Ctl_1_Orig_Busy; Checking_11 [...]
  [] Ctl_2_Orig_Busy; Checking_11 [...]
  [] Ctl_1_Dial_Busy; Checking_11 [...]
  [] Ctl_2_Dial_Busy; Checking_11 [...]
  [] Ctl_1_Disconn; Checking_01 [...]
  [] Ctl_2_Disconn; Checking_10 [...]
  [] Ctl_1_Not_Constr_Signal; Checking_11 [...]
  )
endproc (* Checking_11 *)
```

The Check_In_Use process enforces the global constraint that "at any time, a number is used at most once"; the Basic LOTOS version of this process takes in account only two numbers and uses different process identifiers to maintain the status of the used numbers. The global constraint can be seen as a mutual exclusion property (a safety property) and can be expressed, for the number one, by the temporal logic formula:

$$\Box\,((Ctl\_1\_Orig\_Ok \vee Ctl\_1\_Dial\_Ok) \Rightarrow O\,((\neg\, Ctl\_1\_Orig\_Ok \wedge \neg\, Ctl\_1\_Dial\_Ok)\; \mathcal{U}\; Ctl\_1\_Disconn))$$

Verifying that the Check_In_Use process correctly enforces the constraint amounts to verify that the temporal semantics of Check_In_Use implies the formula above.

We have to note however that only the properties that can be stated for Basic LOTOS, can be considered. Absence of deadlock, traditionally classified as a safety property, cannot be expressed for Basic LOTOS processes since deadlock is not distinguishable from the termination of a process.

It is interesting to note that, although our temporal semantics is not powerful enough to capture *liveness* properties, it is still possible to discuss whether a given process satisfies a given property when some constraints are imposed on the execution of the process itself. An interesting cases of liveness properties are the *fairness* properties. These 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 [P88]; actually, different forms of fairness properties can be defined, such as strong and weak fairness, etc. These properties are not expressible with the operational semantics, since the models which underlie it, (labelled transition systems) cannot distinguish fair or unfair behaviours, since such systems are not sensitive to computations which may be different at infinite. For this reason such properties cannot be expressed in LOTOS, in the sense that LOTOS has no syntax nor semantics to express that a behaviour is, for example, fair and another is not.

A general fairness constraint on the semantics of a process can be imposed by a conjunction of the formula expressing the proper constraint with the temporal semantics of the process. This means to cancel from the set of infinite sequences which are the models of a program those sequences which do not respect the constraint, and corresponds to give constraints on the implementation of the process. We can verify hence that a process satisfies a given fairness property by analyzing the remaining set of (infinite) sequences.

As an example, we can prove that, if we assume to be able to constrain the process P[a,b,c] defined above to perform fair choices among possible actions, and to execute forever, the process will perform infinitely often the action c. The proof can be expressed by the verification of the validity of the formula:

$$(\mathcal{L}(P[a,b,c]) \wedge ([]\Diamond (a \vee b) \Rightarrow []\Diamond a \wedge []\Diamond b) \wedge []\neg \bigcirc false \wedge []\Diamond \neg e \wedge []\Diamond (a \vee b)) \Rightarrow []\Diamond c$$

temporal semantics	fairness constraint	infinite sequences of actions performed by the process	a,b actions are repeatedly offered by the env.	desired property

As we have said, we will show in section 3.2.5 how this property, as well as the other defined on the previous examples, can be proved using the temporal semantics defined for Basic LOTOS.

3.2.3 A Linear Temporal Logic for Basic LOTOS

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 actions, we will define a bounded linear time temporal logic $\mathcal{TL}$, 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 of $\mathcal{TL}$ 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**, $\neg$, $\vee$, $\wedge$;

together with the usual temporal operators:

$\bigcirc$ (next), [] (always), $\Diamond$ (eventually), $\mathcal{W}$ (unless), $\mathcal{U}$ (until), $\mathcal{C}$ (chop),

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

In order to define the semantics of $\mathcal{TL}$ 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 length(σ), finite or infinite, the number of actions in the sequence σ, and with the infix operator "$\circ$" the concatenation of a finite sequence with a finite or infinite one. When the second sequence is empty then the concatenation is also defined if the first sequence is infinite. 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 3.2.3.1 we define inductively what we mean by an interpretation $<\sigma,i>$ to satisfy a formula ϕ and this is expressed by the notation: $\sigma,i \models \phi$.

Table 3.2.3.1

Propositional Calculus operators

$\sigma,i \models$ **true** = true

$\sigma,i \models$ **false** = false

$\sigma,i \models a$ 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 $(\sigma,i \models \neg\phi)$ or $(\sigma,i \models \psi)$

Temporal operators

$\sigma,i \models O\phi$ iff $\text{length}(\sigma) \leq i$ 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**)

$\sigma,i \models \phi \mathbb{C} \psi$ iff (exist σ',σ'', $\text{length}(\sigma') < \infty$ and $\sigma',i \models \phi$ and $\sigma'',i \models \psi$ and $\sigma' \circ \sigma'' = \sigma$) or $(\sigma,i \models \phi$ and $\text{length}(\sigma) = \infty)$

Informally, $O\phi$ means that the formula ϕ is true in the next state, $[]\phi$ means that the formula ϕ is true for all subsequent states, $\Diamond\phi$ means that the formula ϕ is true in one of the subsequent states, $\phi \mathcal{U} \psi$ means that the formula ϕ is true in all subsequent states until a state is reached in which ψ is true, $\phi \mathcal{W} \psi$ is the same as $\phi \mathcal{U} \psi$, apart from the fact that ϕ is enabled to hold indefinitely, so not allowing ψ to hold at any future state.

Note that by χ^k (**true**) we denote the temporal formula $\chi(\chi(\ldots \chi(\textbf{true})\ldots))$ 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 [BB89].

The formula $\phi \mathbb{C} \psi$, using the "chop" operator [BKP84], 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 [BKP84, R86].

The Relabelling Operator

When we try to write the meaning for every Basic LOTOS process, we fail because TL is not powerful enough to describe behaviours of some of them. We need to enrich the TL expressiveness introducing a new temporal operator. The semantics of this operator is given over the same models used for the standard ones.
The formula $\phi[p/b]$, where b is an action predicate and p is a propositional calculus predicate, uses a logical operator which has the effect of substituting every occurrence of b in the model sequences of a formula ϕ with p. This operator, which is different from a syntactic substitution of every occurrence of b with a in the formula ϕ, has been named "relabelling" since it is analogous to the CCS operator with the same name.

Table 3.2.3.2

$\sigma,i \models \phi\ [p/b]$ iff exists $\sigma',i \models \phi$ and for all $k>i$: [$\sigma',k \models b$ implies $\sigma,k \models p$ and for all $g \neq b$: $\sigma',k \models g$ implies $\sigma,k \models g$]

Adequacy and expressiveness

In order to define a temporal semantics for Basic LOTOS and to relate it to the operational semantics, we need some preliminary definitions. To give a logic semantics to a process language $\mathcal{P}$, with an associated operational semantics, in a particular (temporal) logic $\mathcal{T}$ means to identify a process p ($p \in \mathcal{P}$) with the set of all properties (formulae in $\mathcal{T}$) which it satisfies. This requires the definition of a binary relation $\models \subseteq \mathcal{P} \times \mathcal{T}$, usually called *satisfaction relation*, written: $p \models \phi$ and read "p satisfies the property ϕ". The definition of this relation induces an equivalence between processes which enjoy the same properties. Formally, we define:

$$\mathcal{F}(p) = \{ \phi : \phi \in \mathcal{T} \wedge p \models \phi \}$$

hence:

$$p \equiv_{\mathcal{T}} q \quad \Leftrightarrow \quad \mathcal{F}(p) = \mathcal{F}(q)$$

Now, if $\approx \subseteq \mathcal{P} \times \mathcal{P}$ is the equivalence induced by the native operational semantics of the process language $\mathcal{P}$, it is possible to define a relation between $\approx$ and $\equiv_{\mathcal{T}}$:

A logic $\mathcal{T}$ is **adequate** [HM85] w.r.t. an equivalence ($\approx$) defined on a given process language $\mathcal{P}$, if for every pair of processes p and q:

$$p \equiv_{\mathcal{T}} q \quad \Leftrightarrow \quad p \approx q$$

In order to show that the temporal logic we will use is *adequate* w.r.t. the *trace equivalence* it is enough to define a satisfaction relation $\vdash \subseteq \mathcal{P} \times \mathcal{TL}$, that uses the notion of partial computation of a process p ($p \in \mathcal{P}$). If Σ is the set of Basic LOTOS gates and $\sigma \in \Sigma^*$ then, given a process p, we will write:

$$p \rightarrow^{\sigma} \quad \text{iff} \quad \sigma=\varepsilon \text{ or } (\sigma=a \circ \sigma' \wedge \exists p'.\ p \text{-}a\rightarrow p' \wedge p' \rightarrow^{\sigma'}).$$

Consequently, we define the set of traces $\mathcal{ST}(p) = \{\sigma : p \rightarrow^{\sigma}\}$ and the satisfaction relation:

$$p \vdash \phi \quad \text{iff} \quad \forall \sigma \in \mathcal{ST}(p).\ \sigma,0 \models \phi.$$

However we may require a stronger relationship between a process language and a logic: a way to characterize with a unique finite formula in $\mathcal{T}$ all the properties satisfied by a given process p ($p \in \mathcal{P}$):

A logic $\mathcal{T}$ is **expressive** [P85] w.r.t. an equivalence relation ($\approx$) defined on a given process language $\mathcal{P}$, if for every process p ($p \in \mathcal{P}$) there exists a *characteristic formula* (or *logic semantics*) $\mathcal{L}(p) \in \mathcal{T}$ such that:

i. $p \models \mathcal{L}(q)$ iff $p \approx q$

ii. $p \models \phi$ iff $\mathcal{L}(p) \Rightarrow \phi$.

In the next section we will give the characteristic formula for every Basic LOTOS process by giving the temporal semantic function so that we can prove its *expressiveness* w.r.t. the trace equivalence.

3.2.4 A compositional temporal semantics for Basic LOTOS

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 [BKP84, BKP85, B87a]. 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 [B87a] the same proposition is named 'i'; we choose 'e' to avoid confusion with the LOTOS 'i' (internal) action). Therefore the set of action predicates we will use is defined as :

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

We will use also Σ to denote the set Act - {e}.

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 in Basic LOTOS are 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. Note that we are overloading the word "environment" with two meanings: the first denotes an element of *Env*, the second distinguishes the "*e*" action as an environment action, in the typical sense of the compositional approach. The appropriate meaning is, however, obvious from the context each time.

The definition of the environment should consider the static structure of Basic LOTOS specifications. We take in account a simplified syntax Basic LOTOS specifications, which still admits recursive process definitions:

specification := *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.
The type of the semantic function is hence:

$$\mathcal{L} : \mathcal{P} \rightarrow Env \rightarrow \mathcal{TL}$$

$$Env : Id \rightarrow (Gates^* \rightarrow \mathcal{TL})$$

The clauses defining the function $\mathcal{L}$ by structural induction are given in Table 3.2.4.1. For detailed comments we refer to [FGL89].

Note that we consider only the syntactic expression |*set of gates*| for parallel composition. This is the general case and includes the other two (syntactically expressed through || and |||) as degenerate cases (||≡|*all gates*| and |||≡|∅|).

Table 3.2.4.1

Inaction

$\mathcal{L}_\theta$ (**stop**) = e $\mathcal{W}$ **false**

Nondeterministic choice

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

Action prefixing

$\mathcal{L}_\theta$ (i;B) = e $\mathcal{W}$ (i $\wedge$ O $\mathcal{L}_\theta$ (B))

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

Parallel composition

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

Successful termination

$\mathcal{L}$ (**exit**) = e $\mathcal{W}$ (d $\wedge$ O (e $\mathcal{W}$ **false**)

Enabling

$$\mathcal{L}_\theta (B_1 >> B_2) = (\mathcal{L}_\theta (B_1)[i_1/i, \forall f \notin g, f \in \alpha(B_1) - \{d\}: f_1/f, d^*/d] \wedge$$
$$\wedge (e\, \mathcal{W} (d^* \wedge O\, \mathcal{L}_\theta (B_2)[(e \vee i_1 \vee \vee_{f \neq d, f \in \alpha(B_1)} f_1)/e, i_2/i, \forall f \notin g, f \in \alpha(B_2): f_2/f]))$$
$$[\forall f \notin g\ f/f_1, \forall f \notin g\ f/f_2, i/d^*, i/i_1, i/i_2]$$

Disabling

$$\mathcal{L}_\theta (B_1 [> B_2) = (\mathcal{L}_\theta(B_1) \wedge \Diamond d) \vee ((\mathcal{L}_\theta(B_1) \wedge [] \neg d)\, \mathcal{C}\, \mathcal{L}_\theta(B_2))$$

Hiding

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

Process instantiation

$\mathcal{L}_\theta$ (p([a])) = θ(p)(a)

An environment θ is built for a specification on the basis of its process declarations, and will include recursive definitions. This is achieved by a semantic function $\mathbb{D}$ which is defined on a list of process declarations. The type of $\mathbb{D}$ is:

$$\mathbb{D} : Decl^* \rightarrow Env$$

and is defined in the following way (using an auxiliary function $\mathbb{D}_1 : Decl \times Env \rightarrow Env$):

$$\mathbb{D}\ (\text{decls}) = \nu\, x.\ \cup_i\, \mathbb{D}_1(\text{decl}_i, x)$$

$$\mathbb{D}_1(\text{ process } p[g] := B_p \text{ endproc}, x) = \ (p, \lambda y.\ \mathbb{L}_x\,(B_p)[y/g]\ )$$

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

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

with the following definition:

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

The temporal semantics given induces an equivalence which can be proven coinciding with trace equivalence; this implies that $\mathbb{TL}$ is *expressive* with respect to trace equivalence on Basic LOTOS terms. The proof, reported in [FGL88], can be done 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 the maximal traces originated by the same process following the operational semantics. 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. The heart of the proof shows, by structural induction, that in any program its trace set corresponds to its model sequence set, when e actions are eliminated from the latter.

3.2.5 Some example verifications

Now we show the use of the temporal semantics to prove the properties of the Basic LOTOS specifications that we have given in section 3.2.2. First we consider the process:

process P[a,b,c] := (a; P[a,b,c]) [] (b;c; P[a,b,c]) **end process**
which is able to perform the action a or the sequence of actions b,c and then transforms in itself.
Its temporal semantics is the following:

$$\mathbb{L}(P[a,b,c]) = \nu\, x.\ ((e\,\mathcal{W}\ (a \wedge \bigcirc x)) \vee (e\,\mathcal{W}\ (b \wedge \bigcirc e\,\mathcal{W}\ (c \wedge \bigcirc x))))$$

The property we want to prove it satisfies is expressed by the formula:

$$[]\ \neg\,(a \wedge \bigcirc((\neg b)\,\mathcal{U}\, c))$$

In order to tell whether P[a, b, c] satisfies this safety property, we should prove:

$\mathcal{L}(P[a,b,c]) \Rightarrow [] \neg (a \wedge O((\neg b)\, \mathcal{U}\, c))$

which, since $\neg[]\neg\phi = \Diamond\phi$, and for propositional calculus reasoning is the same as:

$\neg\ (\mathcal{L}(P[a,b,c]) \wedge \Diamond (a \wedge O((\neg b)\, \mathcal{U}\, c)))$

hence, we should prove that the argument of the negation is false, i.e., that the following formula has no model:

$\nu x.((e\ \mathcal{W}\ (a \wedge Ox)) \vee (e\ \mathcal{W}\ (b \wedge Oe\ \mathcal{W}\ (c \wedge Ox)))) \wedge \Diamond (a \wedge O((\neg b)\, \mathcal{U}\, c))$

which is the same, by unfolding the fixed point, as:

$[]\,(a \Rightarrow O(e\ \mathcal{W}\ a) \vee O(e\ \mathcal{W}\ (b \wedge O(e\ \mathcal{W}\ c)))) \wedge$

$[]\,(b \Rightarrow O(e\ \mathcal{W}\ (c \wedge O((e\ \mathcal{W}\ a) \vee (e\ \mathcal{W}\ b))))) \wedge \Diamond (a \wedge O((\neg b)\, \mathcal{U}\, c))$

which implies:

$[]\,(a \Rightarrow O(e\ \mathcal{W}\ a) \vee O(e\ \mathcal{W}\ (b \wedge O(e\ \mathcal{W}\ c)))) \wedge \Diamond (a \wedge O((\neg b)\, \mathcal{U}\, c))$

which implies:

$\Diamond\,(a \wedge (O(e\ \mathcal{W}\ a) \vee O(e\ \mathcal{W}\ (b \wedge O(e\ \mathcal{W}\ c)))) \wedge O((\neg b)\, \mathcal{U}\, c))$

which implies:

$\Diamond\,(a \wedge O(\Diamond\ (e\ \mathcal{W}\ (b \wedge O(e\ \mathcal{W}\ c) \wedge (\neg b)\, \mathcal{U}\, c))))$

which implies:

$\Diamond\,(b \wedge \neg b \wedge O\Diamond c) = \mathbf{false}$

which clearly has no satisfying model; hence also the antecedent is false; q.e.d.

The second example is the Caller process of the Plain Old Telephone System. The temporal semantics of the Caller process is the following:

$\mathcal{L}_\theta$ (Caller) = $\mathcal{L}_\theta$ (
  Usr_Offhook;
  Ctl_Originate;
  ((Dialing [Usr_Dial_1, Usr_Dial_2,Ctl_Tone_Ok,Ctl_Dial_1,Ctl_Dial_2]
    >>
      (Normal_Connection [Ctl_Connect_Ok,Usr_Voice,Ctl_Rec_Voice,Ctl_Send_Voice]
        []
        Number_is_Busy
)
)
  [> User_Disconnect [Usr_Hang_up,Ctl_ Disconn,Ctl_ Disind])

$= e\ \mathcal{W}\ (\ Usr\_Offhook \wedge O(e\ \mathcal{W}\ (\ Ctl\_Originate\ \wedge O($
$(\mathcal{L}_\theta(Dialing)[d^*/d] \wedge (\ e\ \mathcal{W}\ (d^* \wedge O\,\mathcal{L}_\theta(Normal\_Connection)$
$[(e \vee i_1 \vee \vee f\ )/e])))\ [i/d^*]$
$f \in \{Usr\_Dial\_1,Usr\_Dial\_2,Ctl\_Tone\_Ok,Ctl\_Dial\_1,Ctl\_Dial\_2\}$
$\mathcal{C}\ \mathcal{L}_\theta(User\_Disconnect))))$

* comments on the application of the enabling formula:
* 1) (Normal_Connection [.....] [] Number_is_Busy) = Normal_Connection [.....]
* since Number_is_Busy = **false**
* 2) since the two component processes work on disjointed set of gates, the relabelling
* with owner identified actions is useless
* comments on the application of the disabling formula:
* 1) we have applied the simplified formula:
* $\mathcal{L}_\theta(B_1\ [>\ B_2) = \mathcal{L}_\theta(B_1)\ \mathcal{C}\ \mathcal{L}_\theta(B_2)$
* since in this case B_1 does not perform any d action.

$\mathcal{L}_\theta$ (Dialing) = $\mathcal{L}_\theta$ (Ctl_Tone_Ok; (Usr_Dial_1; Ctl_Dial_1; **exit** [] Usr_Dial_2; Ctl_Dial_2; **exit**))
= e $\mathcal{W}$ (Ctl_Tone_Ok $\wedge$ O(

$$e\mathcal{W}\ (\text{Usr\_Dial\_1} \wedge O(e\mathcal{W}\ (\text{Ctl\_Dial\_1} \wedge O(e\mathcal{W}\ (d \wedge O(e\mathcal{W}\ \textbf{false})))))) \vee e\mathcal{W}\ (\text{Usr\_Dial\_2} \wedge O(e\mathcal{W}\ (\text{Ctl\_Dial\_2} \wedge O(e\mathcal{W}\ (d \wedge O(e\mathcal{W}\ \textbf{false}))))))$$

$$\mathcal{L}_\theta(\text{Normal\_Connection}) = \mathcal{L}_\theta(\text{Ctl\_Connect\_Ok; Talking}) = e\mathcal{W}\ (\text{Ctl\_Connect\_Ok} \wedge O\mathcal{L}_\theta(\text{Talking})) =$$
$$\mathcal{L}_\theta(\text{Talking}) =$$
$$\mathcal{L}_\theta(\text{Ctl\_Rec\_Voice; Usr\_Voice; Talking [] Usr\_ Voice; Ctl\_Send\_Voice; Talking}) =$$
$$e\mathcal{W}\ (\text{Ctl\_Rec\_Voice} \wedge O(e\mathcal{W}\ (\text{Usr\_Voice} \wedge O\mathcal{L}_\theta(\text{Talking})))) \vee e\mathcal{W}\ (\text{Usr\_Voice} \wedge O(e\mathcal{W}\ (\text{Ctl\_Send\_Voice} \wedge O\mathcal{L}_\theta(\text{Talking}))))$$

$$\mathcal{L}_\theta(\text{User\_Disconnect}) =$$
$$\mathcal{L}_\theta(\text{Usr\_ Hang\_up; Ctl\_Disconn; } \textbf{stop} \text{ [] Ctl\_Disind; Ctl\_Disconn; Usr\_Hang\_up; } \textbf{stop}) =$$
$$e\mathcal{W}\ (\text{Usr\_Hang\_up} \wedge O(e\mathcal{W}\ (\text{Ctl\_Disconn} \wedge O(e\mathcal{W}\ \textbf{false})))) \vee e\mathcal{W}\ (\text{Ctl\_Disind} \wedge O(e\mathcal{W}\ (\text{Ctl\_Disconn} \wedge O(e\mathcal{W}\ (\text{Usr\_Hang\_up} \wedge O(e\mathcal{W}\ \textbf{false})))))$$

The previous formulae give the temporal semantics of the single processes; they can be combined (implicitly using the rules for handling the process definitions and declarations, and simplifying the enabling rule) in the following unique formula which gives the temporal semantics of the process Caller:

$$\begin{aligned}
&\mathcal{L}_\theta(\text{Caller}) = \\
&\quad e\mathcal{W}\ (\text{Usr_Offhook} \wedge O(e\mathcal{W}\ (\text{Ctl_Originate} \wedge O(\\
&\qquad e\mathcal{W}\ (\text{Ctl_Tone_Ok} \wedge O(\\
&\qquad\quad e\mathcal{W}\ (\text{Usr_Dial_1} \wedge O(e\mathcal{W}\ (\text{Ctl_Dial_1} \wedge O(e\mathcal{W}\ (i \wedge O(\\
&\qquad\qquad e\ \mathcal{W}(\text{Ctl_Connect_Ok} \wedge O \\
&\qquad\qquad\quad \vee x.(e\mathcal{W}\ (\text{Ctl_Rec_Voice} \wedge O(e\mathcal{W}\ (\text{Usr_Voice} \wedge Ox))) \\
&\qquad\qquad\qquad \vee e\mathcal{W}\ (\text{Usr_Voice} \wedge O(e\mathcal{W}\ (\text{Ctl_Send_Voice} \wedge Ox)))))))))) \\
&\qquad\quad \vee e\mathcal{W}\ (\text{Usr_Dial_2} \wedge O(e\mathcal{W}\ (\text{Ctl_Dial_2} \wedge O(e\mathcal{W}\ (i \wedge O(\\
&\qquad\qquad e\ \mathcal{W}(\text{Ctl_Connect_Ok} \wedge O \\
&\qquad\qquad\quad \vee x.(e\mathcal{W}\ (\text{Ctl_Rec_Voice} \wedge O(e\mathcal{W}\ (\text{Usr_Voice} \wedge Ox))) \\
&\qquad\qquad\qquad \vee e\mathcal{W}\ (\text{Usr_Voice} \wedge O(e\mathcal{W}\ (\text{Ctl_Send_Voice} \wedge Ox))))))))))) \\
&\qquad \mathcal{C}(e\mathcal{W}\ (\text{Usr_Hang_up} \wedge O(e\mathcal{W}\ (\text{Ctl_Disconn} \wedge O(e\mathcal{W}\ \textbf{false})))) \\
&\qquad\quad \vee e\mathcal{W}\ (\text{Ctl_Disind} \wedge O(e\mathcal{W}\ (\text{Ctl_Disconn} \wedge O(e\mathcal{W}\ (\text{Usr_Hang_up} \wedge \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad O(e\mathcal{W}\ \textbf{false})))))))))))
\end{aligned}$$

We want to prove the safety property "the user cannot talk if he has not lifted the receiver", expressed by the temporal logic formula:

$$\neg(\neg \text{Usr\_Offhook}\ \mathcal{U}\ \text{Usr\_Voice}).$$

The proof is based on the following consideration:
We can distinguish in the set of the sequence models of the temporal semantics of the process Caller those sequences which never perform either the action Usr_Offhook or the action Usr_Voice, from those sequences which eventually perform the action Usr_Offhook without performing before any action Usr_Voice. This amounts to prove that $\mathcal{L}_\theta$ (Caller) implies the formula:

$$[](\neg \text{Usr\_Offhook} \wedge \neg \text{Usr\_Voice}) \vee (\neg \text{Usr\_Voice}\ \mathcal{U}\ \text{Usr\_Offhook})$$

This proof can be done by formal reasoning on the models, or applying axioms and known theorems of the logic: a basic theorem repetitively used in this context is:

$e\mathcal{W}\ f \Rightarrow [](\neg k \wedge \neg f) \vee (\neg k\ \mathcal{U}\ f)$, where $k \neq f$

which is proved in the following way:

$e\ \mathcal{W}\ f \Rightarrow [] e \vee (e\ \mathcal{U}\ f) \Rightarrow [] (\neg k \wedge \neg f) \vee (\neg k\ \mathcal{U}\ f)$ since $e \Rightarrow \neg k$ and $e \Rightarrow \neg f$

Now, we should prove:

$[] (\neg \text{Usr\_Offhook} \wedge \neg \text{Usr\_Voice}) \vee (\neg \text{Usr\_Voice}\ \mathcal{U}\ \text{Usr\_Offhook}) \Rightarrow$
$\neg (\neg \text{Usr\_Offhook}\ \mathcal{U}\ \text{Usr\_Voice})$

We prove separately that:

1) $[] (\neg \text{Usr\_Offhook} \wedge \neg \text{Usr\_Voice}) \Rightarrow \neg (\neg \text{Usr\_Offhook}\ \mathcal{U}\ \text{Usr\_Voice})$

2) $(\neg \text{Usr\_Voice}\ \mathcal{U}\ \text{Usr\_Offhook}) \Rightarrow \neg (\neg \text{Usr\_Offhook}\ \mathcal{U}\ \text{Usr\_Voice})$

Proof:

1) $[] (\neg \text{Usr\_Offhook} \wedge \neg \text{Usr\_Voice}) \Rightarrow [] \neg \text{Usr\_Voice} \Rightarrow$
$\neg (\neg \text{Usr\_Offhook}\ \mathcal{U}\ \text{Usr\_Voice})$,
since $a\ \mathcal{U}\ b \Rightarrow \Diamond b$, which means $\neg \Diamond b \Rightarrow \neg (a\ \mathcal{U}\ b)$

2) The formula in 2) above is the same as:
$\neg (\neg \text{Usr\_Voice}\ \mathcal{U}\ \text{Usr\_Offhook}) \vee \neg (\neg \text{Usr\_Offhook}\ \mathcal{U}\ \text{Usr\_Voice}) =$
$\neg ((\neg \text{Usr\_Voice}\ \mathcal{U}\ \text{Usr\_Offhook}) \wedge (\neg \text{Usr\_Offhook}\ \mathcal{U}\ \text{Usr\_Voice})) =$
$\neg \Diamond (\text{Usr\_Voice} \wedge \text{Usr\_Offhook}) = \neg \Diamond \mathbf{false} =$
(since our models can have only an action at a time)
$[]\ \mathbf{true} = \mathbf{true}$

The third example is the Check_In_Use process of the Plain Old Telephone System. Its temporal semantics is the following, assuming the abbreviations defined below:

Ctl_1_Orig_Ok $\rightarrow$ a	Ctl_2_Orig_Ok $\rightarrow$ b	Ctl_1_Orig_Busy $\rightarrow$ c
Ctl_2_Orig_Busy $\rightarrow$ d	Ctl_1_Dial_Ok $\rightarrow$ j	Ctl_2_Dial_Ok $\rightarrow$ f
Ctl_1_Dial_Busy $\rightarrow$ g	Ctl_2_Dial_Busy $\rightarrow$ h	Ctl_1_Disconn $\rightarrow$ k
Ctl_2_Disconn $\rightarrow$ l	Ctl_Not_Constr_Signal $\rightarrow$ m	

$\mathcal{L}_\theta$(Check_In_Use) =
$\nu x_0 . (e\ \mathcal{W}\ ((k \vee l \vee m) \wedge Ox_0) \vee$
$e\ \mathcal{W}\ ((a \vee j) \wedge O \nu x_1 . (e\ \mathcal{W}\ ((c \vee g \vee l \vee m) \wedge Ox_1) \vee$
$e\ \mathcal{W}\ (k \wedge Ox_0) \vee$
$e\ \mathcal{W}\ ((b \vee f) \wedge O \nu x_3 . (e\ \mathcal{W}\ ((c \vee g \vee d \vee m \vee h) \wedge Ox_3) \vee$
$e\ \mathcal{W}\ (l \wedge Ox_1) \vee$
$e\ \mathcal{W}\ (k \wedge O \nu x_2 . (e\ \mathcal{W}\ ((a \vee j) \wedge Ox_3) \vee$
$e\ \mathcal{W}\ (l \wedge Ox_0) \vee$
$e\ \mathcal{W}\ ((d \vee k) \wedge Ox_2)))))) \vee$
$e\ \mathcal{W}\ (b \wedge O \nu x_2 . (e\ \mathcal{W}\ ((a \vee j) \wedge O \nu x_3 . (e\ \mathcal{W}\ ((c \vee g \vee d \vee m \vee h) \wedge Ox_3) \vee$
$e\ \mathcal{W}\ (k \wedge Ox_2) \vee$
$e\ \mathcal{W}\ (l \wedge O \nu x_1 . (e\ \mathcal{W}\ ((c \vee g \vee l \vee m) \wedge Ox_1) \vee$
$e\ \mathcal{W}\ (k \wedge Ox_0) \vee$
$e\ \mathcal{W}\ ((b \vee f) \wedge Ox_3)))))) \vee$
$e\ \mathcal{W}\ (l \wedge Ox_0) \vee$
$e\ \mathcal{W}\ ((d \vee k) \wedge Ox_2))))))$

We want to prove the safety property "at any time, the number 1 is used at most once",

expressed by the temporal logic formula:

$$[]\,((Ctl\_1\_Orig\_Ok \vee Ctl\_1\_Dial\_Ok) \Rightarrow O\,((\neg\, Ctl\_1\_Orig\_Ok \wedge \neg\, Ctl\_1\_Dial\_Ok)\ \mathcal{U}\ Ctl\_1\_Disconn))$$

which with the abbreviations defined above becomes:

$$[]\,((a \vee j) \Rightarrow O\,((\neg\, a \wedge \neg\, j)\ \mathcal{U}\ k)) \;=\; []\,((a \vee j) \Rightarrow O\,(\neg(a \vee j)\ \mathcal{U}\ k))$$

It is straightforward that the temporal semantics of Check_In_Use implies the above formula; the formal proof can be done along the lines of the previous example.

The last example goes back to the simple process P[a,b,c] defined above: we prove that, if we assume to be able to constrain the process P[a,b,c] defined above to perform fair choices among possible actions, and to execute forever, the process will perform infinitely often the action c.
We have to prove the following implication:

$$(\mathcal{L}(P[a,b,c]) \wedge ([]\diamond(a \vee b) \Rightarrow []\diamond a \wedge []\diamond b) \wedge []\neg Ofalse \wedge []\diamond \neg e \wedge []\diamond(a \vee b)) \Rightarrow []\diamond c$$

The left member of the implication above implies:

$$\mathcal{L}(P[a,b,c]) \wedge []\diamond b \wedge []\neg Ofalse \wedge \diamond \neg e$$

expanding, we obtain:

$$\nu x.\,((e\ \mathcal{W}\ (a \wedge O x)) \vee (e\ \mathcal{W}\ (b \wedge O\,e\ \mathcal{W}\ (c \wedge O x)))) \wedge []\diamond b \wedge []\neg Ofalse \wedge []\diamond \neg e$$

since, in the case of guarded recursion, $\nu x.\phi(x) \wedge [] \xi = \nu x.(\phi(x) \wedge [] \xi)$, we have:

$$\nu x.\,(((e\ \mathcal{W}\ (a \wedge O x)) \vee (e\ \mathcal{W}\ (b \wedge O\,e\ \mathcal{W}\ (c \wedge O x)))) \wedge []\diamond b) \wedge []\neg Ofalse \wedge []\diamond \neg e$$

which, since $(\phi(x) \Rightarrow \psi(x)) \Rightarrow (\nu x.\phi(x) \Rightarrow \nu x.\psi(x))$ and $[]\,\xi \Rightarrow \xi$ implies:

$$\nu x.\,((e\ \mathcal{W}\ (a \wedge O x)) \vee (e\ \mathcal{W}\ (b \wedge O\,e\ \mathcal{W}\ (c \wedge O x))) \wedge \diamond b) \wedge []\neg Ofalse \wedge []\diamond \neg e$$

which, since $(\phi\ \mathcal{W}\ \psi) \wedge \diamond \psi = \phi\ \mathcal{U}\ \psi$, is equal to:

$$\nu x.\,((e \vee a)\ \mathcal{U}\ (b \wedge O\,e\ \mathcal{W}\ (c \wedge O x)))) \wedge []\neg Ofalse \wedge []\diamond \neg e =$$
$$\nu x.\,((e \vee a)\ \mathcal{U}\ (b \wedge O\,e\ \mathcal{U}\ (c \wedge O x)))) \wedge []\neg Ofalse$$

which, since $\phi_1\ \mathcal{U}\ \phi_2 \Rightarrow \diamond\phi_2$, implies

$$\nu x.\,(\diamond\,(b \wedge O \diamond\,(c \wedge O\ x))) \wedge []\neg Ofalse$$

which implies:

$$[]\diamond\ c$$

since the infinite models of the fixed point infinitely often perform c. q.e.d.

3.2.6 Conclusive remarks

In this section, we have used the compositional method proposed in [BKP84,BKP85, B87a] to give a temporal semantics for Basic LOTOS. This work can be considered inside a research context in which several attempts to provide expressive logics for CCS-like process languages [BGS88, HM85, P85] have been developed.

Our work addresses a real specification language, rather than the simple languages considered in the literature; this fact has required a stepwise approach, in which problems related to the complexity of the language have been solved first for a simpler semantics. This section actually presents the first step of our research: a temporal logic expressive w.r.t. the trace equivalence. This is clearly not satisfactory: with the given semantics we are able to prove safety properties, but not liveness ones. For this purpose we are studying an expressive logic w.r.t. maximal trace congruence, of which a tentative definition on a subset of Basic LOTOS has been given. Our aim is to produce a range of temporal semantics with different expressive power for full LOTOS. This range of different semantics will constitute a powerful analysis tool for LOTOS specifications, since they will allow to relate them in different way with partial properties expressed in temporal logic.

References

[A87] S. Abramsky, "Observation Equivalence as Testing equivalence", *Theoretical Computer Science*, Vol.53, 1987.

[B85] G. Boudol, Notes on Algebraic Calculi of Processes", in K. Apt ed., "Logics and Models of Concurrent Systems", Nato ASI Series F13, 1985, pp 261-303.

[B87a] H. Barringer, "The Use of Temporal Logic in the Compositional Specifications of Concurrent Systems", in "Temporal Logics and their applications", A. Galton, ed., Academic Press, London, 1987, pp.53-90.

[B87b] G. Boudol, "Communication is an abstraction", INRIA, RR636, March 1987.

[B89] E. Brinskma, "Constraint-oriented specification in a constructive formal description technique" in "Stepwise Refinement of Distributed Systems", Proc. REX Workshop, Mook, 29 May - 2 June 1989, W.P. de Roever (ed.), *Lecture Notes in Computer Science*, Springer Verlag.

[BB87] T. Bolognesi, E. Brinskma, "Introduction to the ISO Specification Language LOTOS", *Computer Networks & ISDN Systems*, vol. 14, n. 1, January 1987, pp. 25-29.

[BB89] B. Banieqbal, H. Barringer: "Temporal Logic Fixed Point Calculus", Proceedings Colloquium on Temporal Logic and Specification, Altrincham, England, 1987. *Lecture Notes in Computer Science*, vol.398, 1989, pp.62-74.

[BC88] T. Bolognesi, M. Caneve, "Squiggles - a Tool for the Analysis of LOTOS Specifications", Proceedings 1st International Conference on Formal Description Techniques (FORTE'88), K.J. Turner, ed., North-Holland, 1989, pp. 201-216.

[BGS88] A. Bouajjani, S. Graf, J. Sifakis, "A Logic for the Description of Behaviours and Properties of Concurrent Sistems", REX School /Workshop, *Lecture Notes in Computer Science* vol. 354, May 1988, pp.398-410.

[BKP84] 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.

[BKP85] 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.

[BL 90] G. Boudol, K. Larsen, "Graphical vs. Logical Specifications", to appear in Proceedings CAAP '90, Copenhagen, May 1990.

[BRSV89] G. Boudol, V. Roy, R. de Simone, D. Vergamini, "Process Calculi, from Theory to Practice: Verification Tools", in Automatic Verification Methods for Finite State Systems, *Lecture Notes in Computer Science*, vol.407, pp.1-10.

[CE81] E.M Clarke, E.A Emerson,"Synthesis of synchronization skeletons for branching time temporal logic", Proceedings of the Workshop on Logic Programs, Yorktown Heights New York, *Lectures Notes in Computer Science*, Volume 131, 1981, pp. 52-71.

[CES86] E. M. Clarke, E.A. Emerson, A.P. Sistla, "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specification", *ACM Transactions on Programming Languages and Systems*, vol. 8, n. 2, pp. 244-263, 1986.

[CPS89] R. Cleaveland, J. Parrow, B. Steffen, "A Semantics Based Verification Tool for Finite State Systems", Ninth IFIP Symposium "Protocol Specification, Testing and Verification", Enschede (1989), North Holland.

[D86] A. Dicky, "An algebraic and algorithmic method for analysing transitions systems, *Theoretical Computer Science*, Vol. 46, n.2-3, 1986, pp. 285-303.

[D87] R. De Nicola, "Extensional Equivalences for Transition Systems", *Acta Informatica*, 24, pp. 211-237.

[D89] E. Dubuis, "An algorithm for translating LOTOS Behavior expressions into Automata and Ports", Proceedings 2nd International Conference on Formal Description Techniques (FORTE'89), Vancouver, Dec. 1989, pp. 215-229.

[dSV89] R. de Simone, D. Vergamini, "Aboard Auto", INRIA Technical Report 11, 1989

[EH86] E.A. Emerson, J.Y. Halpern, ""Sometimes' and 'Not Never' Revisited: On Branching versus Linear Time Temporal Logic", *Journal of the ACM*, vol. 33, n. 1, Jan. 1986, pp. 151-178.

[ES89] E.A Emerson, J. Srinivasan, "Branching Time Temporal Logic", in J.W. de Bakker, W.P. de Roever, G. Rozenberg, eds., "Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency", *Lecture Notes in Computer Science*, vol. 354, 1989, pp. 123-172.

[FGL88] A. Fantechi, S. Gnesi, C. Laneve, "A Proof of Simple Abstractness for the Temporal Semantics of LOTOS", I.E.I. Internal Report B4-38, August 1988.

[FGL89] A. Fantechi, S. Gnesi, C. Laneve, "An Expressive Temporal Logic for Basic LOTOS", Proceedings 2nd International Conference on Formal Description Techniques (FORTE'89), Vancouver, Dec. 1989, pp. 383-399.

[FLS89] M. Faci, L. Logrippo, B. Stépien, "Formal Specification of Telephone Systems in LOTOS", in E. Brinskma, G. Scollo, C.A. Vissers eds., Proc. of 9th IFIP WG 6.1 International Symposium on Protocol Specification, Testing and Verification, North-Holland 1989.

[H81] C.A.R. Hoare, "A Model for Communicating Sequential Processes", Technical Monograph Prg-22, Computing Laboratory, University of Oxford, 1981.

[HM85] M. Hennessy, R. Milner, "Algebraic Laws for Nondeterminism and Concurrency", *Journal of ACM*, vol. 32, n. 1, January 1985, pp. 137-161.

[K63] S.A Kripke, "Semantical Analysis of Modal Logic, *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, Vol. 9, 1963.

[L81] L. Lamport, "'Sometime' is Sometimes 'Not Never'", Proceedings of 7th Annual Symp. on Principles of Programming Languages, ACM, 1980, North Holland, 1983, pp.174-185.

[L83] L. Lamport, "What Good is Temporal Logic?", Proceedings of IFIP'83, North Holland, 1983, pp.657-668.

[L85] K. Larsen, "A Context Dependent Equivalence between Processes", Proc. ICALP'85, *Lecture Notes in Computer Science*, vol. 194, pp. 373-382.

[L90] K. Larsen, "Compositional Theories Based on an Operational Semantics of Contexts" in "Stepwise Refinement of Distributed Systems", Proc. REX Workshop, Mook, 29 May - 2 June 1989, W.P. de Roever (ed.), *Lecture Notes in Computer Science*, Springer Verlag.

[LT88] K. Larsen, B. Thompson, "A Modal Process Logic", Proc. LICS'88, pp. 203-210.

[M80] R. Milner, "A Calculus of Communicating Systems", *Lecture Notes in Computer Science* vol. 92, 1980.

[M81] R. Milner, "A modal characterization of observable machine-behaviour", Proceedings CAAP'81, *Lectures Notes in Computer Science*, Vol. 112, 1981, pp.25-34.

[MP81] 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.

[MP89] Z. Manna, A. Pnueli, "The Anchored Version of the Temporal Framework", in J.W. de Bakker, W.P. de Roever, G. Rozenberg, eds., "Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency", *Lecture Notes in Computer Science*, vol. 354, 1989, pp.201-284.

[MV89] E. Madeleine, D. Vergamini, "AUTO: A Verification Tool for Distributed Systems Using Reduction of Finite Automata Networks", Proceedings 2nd International Conference on Formal Description Techniques (FORTE'89), Vancouver, Dec. 1989, pp. 77-83.

[P79] R. Parikh, "Propositional Dynamic Logics of Programs: A survey" in "Logics of Programs", *Lecture Notes in Computer Science*, vol. 125, 1979, pp.102-144.

[P81] A. Pnueli, "The Temporal Semantics of Concurrent Programs", *Theoretical Computer Science*, vol.13, 1981, pp.45-60.

[P85] A. Pnueli, "Linear and Branching Structures in the Semantics and Logic of Reactive Systems" Proceedings of 12th ICALP, *Lecture Notes in Computer Science* vol. 194, July 1985, pp.15-32.

[P86] A. Pnueli : "Specification and Development of Reactive Systems", Proceedings IFIP Congress, North-Holland, 1986, pp. 845-858.

[P88] 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.

[R86] R. Rosner, "A Choppy Logic", Master Degree Thesis, Weizmann Institute of Science, January 1986.

[S82] J.Sifakis : "A unified approach for studying the properties of transition systems", *Theoretical Computer Science*, Vol 18, 1982, pp.227-258.

[S83] J.P Schwartz, "Quasar: une réalisation du système CESAR: Description, Spécification et Analyse des applications réparties, Thèse de Docteur Ingénieur 1983, Université Scientifique et Médicale de Grenoble.

[VGG89] R. J. Van Glabbeek, U. Goltz, "Equivalence Notions for Concurrent Systems and Refinement of Actions" MFCS 89, *Lecture Notes in Computer Science*, vol.379, pp. 237-248.

[WC89] J.P.Wu, S.T. Chanson, "Translation from LOTOS and Estelle Specifications to Extended Transitin Systems and its Verification", Proceedings 2nd International Conference on Formal Description Techniques (FORTE'89), Vancouver, Dec. 1989, pp. 677-697.