Denotational Semantics for Timed Testing

Luis Fernando Llana Díaz and David de Frutos Escrig

Dpto. Informática y Automática
Universidad Complutense de Madrid
Ciudad Universitaria. 28040 Madrid. Spain.
e-mail: {llana,defrutos}@eucmax.sim.ucm.es

Abstract. In this paper we present a denotational semantics for a timed process algebra, which is fully abstract with respect to the must testing semantics previously developed [Lla96,LdFN96]. The domain of semantic processes is made up of consistent sets of barbs, which generalize the notion of acceptance sets, in such a way that the actions that are offered but not taken in each state are also recorded. the main difficulty when defining this denotational semantics has been that the natural ordering between semantic processes cannot be proved to be complete. So an alternative stronger complete ordering has to be considered, which is proved to be consistent with the original one, in the sense that lubs of chains with respect to the new ordering are also lubs with respect to the original one.
Keywords: process algebra, time, must testing semantics, denotational semantics.

1 Introduction

Process algebras have been widely used in recent years as high level languages for specifying concurrent systems [Mil89,Hoa85,BK84]. Time is an important aspect of the description of a concurrent system that cannot be directly represented in such process algebras. The introduction of aspects of time has received much attention in the recent past, and there have been many proposals, including [RR86,OM91,Yi91,BB93].

In [HR95] we can find a testing semantics for a timed process algebra, where there is a simple notion of time: it is expressed by introducing a σ (tic) action. The execution of this action by a process suggests that it is idling until next clock cycle. In a previous paper [LdFN96] we have presented testing semantics for a process algebra in which time is introduced in a more abstract way: we have transitions labeled with timed actions similar to that in [Sch95,Yi91,BB93,QdFA93]. The operational semantics we have considered has transitions of the form $P \xrightarrow{et} P'$, meaning that the process P performs the event e at time t and then becomes P'. In this operational semantics, internal actions (denoted by τ) are considered to be urgent; so we have

$$P \xrightarrow{\tau t} P' \Rightarrow P \xrightarrow{et'} \not\rightarrow \ t' > t.$$

The results could be easily adapted to process algebras with timed transitions and action transitions as proposed also in [NS91,dFLL+95].

In [LdFN96] a characterization of the must testing semantics is presented in an explicit way (not depending on tests) similar to that in [Hen88,dNH84]. This characterization is made in terms of barbs. A barb is a generalization of an acceptance, i.e., it is a sequence

$$A_1 a_1 t_1 A_2 a_2 t_2 \cdots A_n a_n t_n A_{n+1},$$

whose intuitive meaning is that the actions a_i have been executed at time t_i (the time is local, relative to the previous action), in a state where the process has previously offered to the environment the actions of the set A_i, and finally the process reach a state that offers the actions in the set A_{n+1}. Urgency is the reason because we have to take care of the actions that the process has offered to the environment before each action is executed. If the environment could have executed any action in the set A_i, both the process and the environment, should have synchronized on that action, and then the action a_i at time t_i would no longer be possible. So if finally the action a_i is executed at time t_i, this means that no action in the set A_i is possible in the environment.

2 Syntax

In this section we describe the syntax of the language we will consider. In order to focus on the main characteristics and problems of timed algebras, we introduce a simple timed process algebra which however contains the main operators that characterize such an algebra. More exactly, we are talking about those we consider the main operators of a *high level* timed process algebra. So, we neither consider *tic* actions measuring time in a explicit way, nor delay operators. Nevertheless is possible to translate our high level operators to a low level language containing such kind of operators, and thus similar results in this paper could be obtained for a language such as the one in [HR95]. In our language time is introduced via the prefix operator; actions must be executed at the indicated time, and we will consider a discrete time domain $\mathcal{T}$. We consider a finite set of actions Act, $a, a', \ldots$ range over Act. and an internal event $\tau \notin Act$; then we consider the set of events $\mathcal{E} = Act \cup \{\tau\}$, $e, e', \ldots$ range over $\mathcal{E}$. We also consider a set of process variables Var, $x, x', \ldots$ range over Var. Then we consider the set of processes Proc as the set of closed terms generated by the following B.N.F. expression:

$$P ::= \mathtt{STOP} \mid \mathtt{DIV} \mid et\,; P \mid P \,\square\, Q \mid P \sqcap Q \mid P \parallel_G Q \mid P \setminus a \mid x \mid \mathtt{REC}\, x.P.$$

We will denote by FProc the set of finite processes, i.e., those without recursion.

3 Operational Semantics

In order to define the operational semantics of the language we need an auxiliary function $\mathsf{Upd}(t, P)$, which represents the pass of t units of time on the process P. This function is defined in table 1. Looking at the operational semantics,

we observe that the function $\mathsf{Upd}(P,t)$ is only used when $P \xrightarrow{rt}\!\!\!\!\!\!/\;$; so, the way $\mathsf{Upd}(it\,;P,t')$ is defined when $t < t'$ is not important, and it is only included for completeness.

$$\mathsf{Upd}(t,P) = \begin{cases} P & \text{if } P = \mathtt{STOP} \text{ or } P = \mathtt{DIV} \\ \mathtt{STOP} & \text{if } P = et'\,;P_1 \text{ and } t' < t \\ e(t'-t)\,;P_1 & \text{if } P = et'\,;P_1 \text{ and } t' \geq t \\ \mathsf{Upd}(t,P_1)\ op\ \mathsf{Upd}(t,P_2) & \text{if } P = P_1\ op\ P_2,\ op \in \{\Box, \sqcap, \|_A\} \\ \mathsf{Upd}(t,P_1) \setminus A & \text{if } P = P_1 \setminus A \\ \mathtt{DIV} & \text{if } P = x \\ \mathsf{Upd}(t,P_1)[\mathtt{REC}\,x.P_1/x] & \text{if } P = \mathtt{REC}\,x.P_1 \end{cases}$$

[DIV] $\mathtt{DIV} \xrightarrow{\tau 0} \mathtt{DIV}$

[PRE] $et\,;P \xrightarrow{et} P$

[ELI] $P \sqcap Q \xrightarrow{\tau 0} P \qquad P \sqcap Q \xrightarrow{\tau 0} Q$

[CH1] $$\frac{P \xrightarrow{at} P',\ \forall t' < t:\ Q \xrightarrow{\tau t'}\!\!\!\!\!\!/\;}{P \Box Q \xrightarrow{at} P'} \qquad \frac{P \xrightarrow{at} P',\ \forall t' < t:\ Q \xrightarrow{\tau t'}\!\!\!\!\!\!/\;}{Q \Box P \xrightarrow{at} P'}$$

[CH2] $$\frac{P \xrightarrow{\tau t} P',\ \forall t' < t:\ Q \xrightarrow{\tau t'}\!\!\!\!\!\!/\;}{P \Box Q \xrightarrow{\tau t} P' \Box \mathsf{Upd}(Q,t)} \qquad \frac{P \xrightarrow{\tau t} P',\ \forall t' < t:\ Q \xrightarrow{\tau t'}\!\!\!\!\!\!/\;}{Q \Box P \xrightarrow{et} \mathsf{Upd}(Q,t) \Box P'}$$

[INT] $$\frac{P \xrightarrow{et} P',\quad \forall t' < t:\ Q \xrightarrow{\tau t'}\!\!\!\!\!\!/\;}{P \|_A Q \xrightarrow{et} P' \|_A \mathsf{Upd}(t,Q)}\ e \notin A \qquad \frac{P \xrightarrow{et} P',\quad \forall t' < t:\ Q \xrightarrow{\tau t'}\!\!\!\!\!\!/\;}{Q \|_A P \xrightarrow{et} \mathsf{Upd}(t,Q) \|_A P'}\ e \notin A$$

[SYN] $$\frac{P \xrightarrow{at} P',\quad Q \xrightarrow{at} Q'}{P \|_A Q \xrightarrow{at} P' \|_A Q'}\ a \in A$$

[HD1] $$\frac{P \xrightarrow{et} P',\quad \forall t' < t, a \in A:\ P \xrightarrow{a't'}\!\!\!\!\!\!/\;}{P \setminus A \xrightarrow{et} P' \setminus A}\ e \notin A$$

[HD2] $$\frac{P \xrightarrow{at} P',\quad \forall t' < t, a \in A:\ P \xrightarrow{a't'}\!\!\!\!\!\!/\;}{P \setminus A \xrightarrow{rt} P' \setminus A}\ a \in A$$

[REC] $\mathtt{REC}\,x.P \xrightarrow{\tau 0} P[\mathtt{REC}\,x.P/x]$

Table 1. Operational Semantics.

The operational semantics of Proc is given by the relation $\rightarrow \subseteq \mathsf{Proc} \times (\mathcal{E} \times \mathcal{T}) \times \mathsf{Proc}$ defined by the rules in table 1. Since some rules have negative premises we have to provide a way to guarantee that th generated transition system is consistent. This is achieved by defining a *stratification*, as detailed in [Gro93]. We consider the following function

$$f(P \xrightarrow{et} Q) = \textit{Number of operators in } P$$

that is indeed a stratification.

Definition 1. Let P be a process, we define the set of timed actions that P can execute as

$$\mathsf{TA}(P) = \{at \mid \exists P':\ P \xrightarrow{at} P'\}$$

4 Testing Semantics

In this section we define the testing semantics induced by the operational semantics above. Tests are just processes but defined by an extended grammar where we add a new process, OK, which expresses that the test has been passed. The operational semantics of tests is defined in the same way as for processes, but only adding a rule for the test OK*:

$$\textbf{[OK]} \qquad \mathtt{OK} \xrightarrow{\mathtt{OK}} \mathtt{STOP}.$$

Finally we define the composition of a test and a process in the following way:

$$P \mid T = (P \parallel_{Act} T) \setminus Act.$$

Definition 2. Given a computation of $P \mid T$

$$P \mid T = P_1 \mid T_1 \xrightarrow{\tau t_1} P_2 \mid T_2 \cdots P_k \mid T_k \xrightarrow{\tau t_k} P_{k+1} \mid T_{k+1} \cdots$$

we say that it is

- *Complete* if it is finite and blocked (no step is allowed), or infinite.
- *Successful* if there exists some k such that $T_k \xrightarrow{\mathtt{OK}}$.

Definition 3.

- We say that P *must pass* the test T (P must T) iff any complete computation of $P \mid T$ is successful.
- We say that $P \sqsubseteq Q$ iff whenever P must T we also have Q must T.

4.1 Operational Characterization

States, b-traces and barbs In order to characterize the testing semantics we will consider some kind of sets of timed actions which we call *states*, that represent any of the possible *local configurations* of a process. In a state we have the set of timed actions offered, and the time, if any, at which the process becomes divergent†. So, basically, a state is a set of timed actions: if a process P is in a state A such that $at \in A$ then P can execute action a in time t. In order to capture divergence with a simple notation, we introduce a new element $\Omega \notin Act$ that represents undefinition: if $\Omega t \in A$ then the process will become divergent at time t. Then we consider the sets $Act_\Omega = Act \cup \{\Omega\}$, $TAct = Act \times \mathcal{T}$ and $Act_\Omega \times \mathcal{T} = Act_\Omega \times \mathcal{T}$. So we have

Definition 4. We say that $A \subseteq Act_\Omega \times \mathcal{T}$ is a state if:

---

* To be exact, we should extend the definition of the operational semantics for processes and tests to mixed terms defining their composition, since these mixed terms are neither processes nor tests, but since this extension is immediate we have preferred to avoid this formal definition.

† A process is divergent if it can execute in a row an infinite number of internal actions, all of them at time 0. Note that divergent processes only pass trivial tests in the must sense.

- There is at most a single $\Omega t \in A$, i.e., $\Omega t,\ \Omega t' \in A \Rightarrow t = t'$.
- If $\Omega t \in A$ then t is the maximum time in A, i.e., $\Omega t,\ at' \in A \Rightarrow t' < t$.

We will denote by $\mathcal{ST}$ the set of states.

Now we give some auxiliary definitions:

Definition 5.

- We define the function $\mathsf{nd}(\cdot) : \mathcal{ST} \mapsto \mathcal{T} \cup \{\infty\}$, which give us the time at which a state becomes undefined (*not defined* function), by:
$$\mathsf{nd}(A) = \begin{cases} t & \text{if } \Omega t \in A, \\ \infty & \text{otherwise.} \end{cases}$$
- Given a state $A \in \mathcal{ST}$ and a time $t \in \mathcal{T}$, we define:
$$A + t = \{a(t + t') \mid at' \in A\}, \qquad A\rceil t = \{at' \mid at' \in A \text{ and } t' < t\}.$$
- If $A \in \mathcal{ST}$, we define its set of timed actions: $TAct(A) = A\rceil\mathsf{nd}(A)$.
- If $A \in \mathcal{ST}$ and $t \in \mathcal{T}$, we will say that $A < t$ (resp. $A \leq t$) iff for all $at' \in A$ we have $t' < t$ (resp. $t' \leq t$).

A barb is a generalization of an acceptance set [Hen88], but additional information must be included to record the actions that the process offers *before* any action has been executed. First we introduce the concept of b-trace, which is a generalization of the notion of trace. A b-trace, bs, is a sequence $A_1a_1t_1A_2a_2t_2 \cdots A_na_nt_n$ that represents the execution of the sequence of timed actions $a_1t_1a_2t_2 \cdots a_nt_n$ in such a way that after the execution of each prefix $a_1t_1 \cdots a_{i-1}t_{i-1}$ the timed actions in A_i were offered before accepting a_it_i. Then a barb is a b-trace followed by a final state, that represents the reached configuration of the process after executing the b-trace.

Definition 6.

- *b-traces* are finite sequences, $bs = A_1a_1t_1 \cdots A_na_nt_n$, where $n \geq 0$, $a_it_i \in TAct$, $A_i \subseteq TAct$, and if $a't' \in A_i$ then $t' < t_i$. We take $\mathsf{lon}(b) = n$; if $n = 0$ we have the empty b-trace denoted by ϵ.
- A *barb* b is a sequence $b = bs \cdot A$ where bs is a b-trace and A is a state. We will represent the barb $\epsilon \cdot A$ by A, and so we will consider that states are also (initial) barbs.

Definition 7.

- Given $t \in T$, a b-trace $bs = A_1a_1t_1 \cdot bs_1$ and a set of timed actions $A \subseteq TAct$ such that $A < t$, we define
$$(A, t) \sqcup bs = (A \cup (A_1 + t))a(t_1 + t) \cdot bs_1.$$
- If bs is a b-trace we define its duration as
$$\mathsf{Time}(bs) = \begin{cases} 0 & \text{if } bs = \epsilon, \\ t + \mathsf{Time}(bs') & \text{if } bs = Aat \cdot bs'. \end{cases}$$
- If $b = bs \cdot A$ is a barb we define its time of undefinition
$$\mathsf{nd}(b) = \mathsf{Time}(bs) + \mathsf{nd}(A).$$

We will use barbs and b-traces to characterize the testing semantics; this will be done by defining a pre-order between sets of barbs. In order to define this pre-order, we need first the following ordering relations:

Definition 8.

- We define the relation $\ll$ [‡] between b-traces as the least relation that satisfies: 1. $\epsilon \ll \epsilon$, 2. If $bs' \ll bs$ and $A' \subseteq A$ then $A'at \cdot bs' \ll Aat \cdot bs$.
- We define the relation $\ll$ between barbs as the least relation that satisfies:
  1. If bs, bs' are b-traces such that $bs' \ll bs$, and A, A' are states such that $\mathsf{nd}(A') \leq \mathsf{nd}(A)$ and $TAct(A') \subseteq A$, then $bs' \cdot A' \ll bs \cdot A$.
  2. If A' is a state, $b = A_1 a_1 t_1 \cdot b'$ is a barb such that $\mathsf{nd}(A') \leq t_1$ and $TAct(A') \subseteq A_1$, and $bs' \ll bs$ then $bs' \cdot A' \ll bs \cdot (A_1 a_1 t_1 \cdot b')$.

Intuitively, a b-trace bs is *worse* than another one bs', if the actions that appear in both b-traces are the same, and the intermediate sets A_i that appear in bs are smaller than those A'_i appearing in bs'. For barbs, we must notice that whenever a process is in an undefined state, which means $t = \mathsf{nd}(A) < \infty$, it can pass no test *after* that time in the must sense. Barbs and b-traces are introduced to characterize the testing semantics. As shown in [LdFN96], to characterize the must testing semantics it is enough to extend the preorder to sets of barbs:

Definition 9. Let B_1 and B_2 be sets of barbs, we define:

$$B_1 \ll B_2 \qquad \Longleftrightarrow \qquad \forall b_2 \in B_2 \; \exists b_1 \in B_1 : \; b_1 \ll b_2.$$

States, b-traces, barbs and processes The states of a process P are computed from its complete computations of internal actions. For any computation we record the information about the actions offered *before* the execution of any internal action. For divergent computations we also record the *time of divergence.*

Definition 10. For a process P, the set $\mathcal{A}(P)$ is the set of states $A \in \mathcal{ST}$ that are generated from the *complete* computations of internal actions of P as described below.

- Given an infinite computation,
$$P = P_1 \xrightarrow{\tau t_1} P_2 \xrightarrow{\tau t_2} \cdots P_k \xrightarrow{\tau t_k} P_{k+1} \cdots$$
generates the state $A \in \mathcal{A}(P)$ given by
$$A = \bigcup_{i \in \mathbb{N}} (\mathsf{TA}(P_i) \rceil t_i + t^i) \cup \begin{cases} \{\Omega t\} & \text{if } t = \sum_{i=1}^{\infty} t_i < \infty, \\ \varnothing & \text{otherwise.} \end{cases}$$
- Each finite blocked computation,
$$P = P_1 \xrightarrow{\tau t_1} P_2 \xrightarrow{\tau t_2} \cdots P_{n-1} \xrightarrow{\tau t_{n-1}} P_n$$
generates the state $(\mathsf{TA}(P_n) + t^n) \cup \bigcup_{i=1}^{n} (\mathsf{TA}(P_i) \rceil t_i + t^i) \in \mathcal{A}(P)$.

where, in both cases, we take $t^i = \sum_{j=1}^{i-1} t_j$.

[‡] Note that the symbol $\ll$ is overloaded, it is used for both b-traces and barbs.

Definition 11. Let P, P' be processes and bs a b-trace. We define the relation $P \stackrel{bs}{\Longrightarrow} P'$ as follows:

- $P \stackrel{\epsilon}{\Longrightarrow} P$.
- If $P \stackrel{\tau t}{\longrightarrow} P_1$, and $P_1 \stackrel{bs'}{\Longrightarrow} P'$ with $bs' \neq \epsilon$ then $P \stackrel{(\mathsf{TA}(P)\rceil t, t) \sqcup bs'}{\Longrightarrow} P'$.
- If $P \stackrel{at}{\longrightarrow} P_1$, and $P_1 \stackrel{bs'}{\Longrightarrow} P'$ then $P \stackrel{(\mathsf{TA}(P)\rceil t) at \cdot bs'}{\Longrightarrow} P'$.

Finally we define the set of b-traces of a process by

$$\mathsf{Btraz}(P) = \{bs \mid \exists Q :\ P \stackrel{bs}{\Longrightarrow} Q\}.$$

States and b-traces are closely related as the following proposition shows

Proposition 12.

- *If* $P \stackrel{Aat}{\Longrightarrow} Q$ *there exists a state* $A' \in \mathcal{A}(P)$ *such that* $A'\rceil t = A\rceil t$.
- *If* $A \in \mathcal{A}(P)$ *and* $at \in A$ *then there exists a process* Q *such that*

$$P \stackrel{(A\rceil t) at}{\Longrightarrow} Q.$$

Definition 13. Let $b = bs \cdot A$ be a barb and P a process; we say that b is a barb of P ($b \in \mathsf{Barb}(P)$) iff there exists a process P' such that $P \stackrel{bs}{\Longrightarrow} P'$ and $A \in \mathcal{A}(P')$.

The preorder defined between sets of barbs can be used to characterize the preorder induced by the testing semantics. This equivalence has been proved in detail in [LdFN96].

Theorem 14. *Let* P, Q *be processes then,* $P \sqsubsetneq Q \iff \mathsf{Barb}(P) \ll \mathsf{Barb}(Q)$.

5 Denotational Semantics

In this section we will give a denotational semantics to our language that will be proved fully abstract with respect to the testing semantics in section 6. Next we define the domain we will use to define the operators; first we need some auxiliary definitions:

Definition 15.

- A set of states $\mathcal{A}$ is *t-compact* iff

$$\forall A \in \mathcal{ST}\, (\forall t \in \mathcal{T}\, \exists A_t \in \mathcal{A} :\ A_t = A\rceil t) \ \Rightarrow\ A \in \mathcal{A}.$$

  This property can be seen as a kind of *temporal continuity* on sets of states: whenever every temporal restriction of a state is in a set of states we also have that the state itself is in the set.
- Let B be a set of barbs, we define the b-traces of B as

$$\mathsf{Btraz}(B) = \{bs \mid \exists A :\ bs \cdot A \in B\}.$$

  If $bs \in \mathsf{Btraz}(B)$ we define the barbs of B *after* bs and the states *reached after* bs, respectively by

$$\mathsf{Barb}(B, bs) = \{b \mid bs \cdot b \in B\}, \qquad \mathcal{A}(B, bs) = \{A \mid bs \cdot A \in B\}.$$

  As a particular case, we write $\mathcal{A}(B, bs) = \mathcal{A}(B)$ when $bs = \epsilon$.

Now we can define the semantic domain: the consistent sets of barbs $\mathcal{B}_{\mathrm{con}}$.

Definition 16. A set of barbs B is *consistent*, and then we write $B \in \mathcal{B}_{\text{con}}$, iff

- $B \neq \varnothing$.
- *Prefix closed*: if $bs \cdot Aat \in \mathsf{Btraz}(B)$ there exists some state A' such that $bs \cdot A' \in B$ and $A'\rceil t = A\rceil t$.
- *Continuation closed*: if $bs \cdot A \in B$ and $at \in A$ then $bs \cdot (A\rceil t)at \in \mathsf{Btraz}(B)$.
- *t-compact*: for each $bs \in \mathsf{Btraz}(B)$ the set of states $\mathcal{A}(B, bs)$ is t-compact.

All the above conditions are quite natural: first we require that a consistent set of barbs not to be empty; the *prefix closed* condition indicates that if a computation has been executed then all intermediate states are reachable; next in the *continuation closed* condition we establish that if a action is in a state there is a computation from that action; finally we require the temporal continuity property for the states *after* each computation.

Before going on we have to notice that the relation $\ll$ is not a partial order over consistent sets of barbs; is just a pre-order, since it does not verify the anti-symmetric property. So we have to deal instead with the equivalence induced by the pre-order:

$$B_1 \approx B_2 \quad \Longleftrightarrow \quad B_1 \ll B_2 \text{ and } B_2 \ll B_1$$

Then for every operator *op* we have to check that it is *well defined*, that is

- If $B_1, \ldots, B_i, \ldots B_n$ are consistent sets of barbs, then $op(B_1, \ldots, B_i, \ldots B_n)$ is too.
- It is congruent with respect the relation $\approx$ or, equivalently, we have to check $op(B_1, \ldots, B_i, \ldots B_n) \ll op(B_1, \ldots, B'_i, \ldots B_n)$ whenever $B_i \ll B'_i$.

congruent In the following we give the semantic meaning of each operator of the language. Due to lack of space, it is not possible to include the proofs of the well definition of the operators, they can be found in [Lla96].

Divergence and Stop

None of these operators can execute any visible action, the difference between them is that while `STOP` will allow the execution of any action of another process in the context of the external choice or parallel operators, `DIV` will not. This is reflected in the denotational semantics in the following way:

$$\mathcal{B}_{\text{con}}[\![\texttt{STOP}]\!] = \{\varnothing\}, \qquad \mathcal{B}_{\text{con}}[\![\texttt{DIV}]\!] = \Big\{\{\Omega 0\}\Big\}.$$

while `STOP` has a unique empty state, `DIV` has a unique undefined state undefined at time 0.

Prefix

Although we have a unique prefix operator, we have two cases depending on the prefixing event. First we consider the case where the prefixing event is not visible, the effect of this operator is just to delay the execution of the process the indicated units of time; so we have

$$\mathcal{B}_{\text{con}}[\![\tau t;]\!](B) = \{b + t \mid b \in B\}.$$

When the prefixing action is visible, the computations of the resulting process are the same of the ones of the old process but beginning with the prefixing action, so we have

$$\mathcal{B}_{\mathrm{con}}[\![at;]\!](B) = \{\varnothing at \cdot b \mid b \in B\} \cup \Big\{\{at\}\Big\}.$$

Internal Choice

A process built with this operator behaves in a non-deterministic way, choosing between its two components. Thus we take:

$$\mathcal{B}_{\mathrm{con}}[\![\sqcap]\!](B_1, B_2) = B_1 \cup B_2$$

External Choice

The definition of this operator is quite more complex. Now it is the environment who solve the choice by selecting the first action to be executed; then the process behaves like one of the processes. So the computations of the composition are again the computations of the processes in the choice operator. But now, the initial states are obtained by the union of states of the processes, although one has to take care of the time of undefinition. As a similar concept will be needed in the parallel operator, we give next a more general definition.

Definition 17. If A_1 y A_2 are states and $G \subseteq Act$ we define

$$\begin{aligned} A_1 \sqcup_G A_2 = {} & (A_1 \cap A_2 \cap G)\rceil t \\ & \cup((A_1 \setminus G)\rceil t) \cup ((A_2 \setminus G)\rceil t) \\ & \cup \begin{cases} \{\Omega t\} & \text{if } t < \infty, \\ \varnothing & \text{otherwise.} \end{cases} \end{aligned}$$

where $t = min\{\mathsf{nd}(A_1), \mathsf{nd}(A_2)\}$.

The set G stands for a synchronization alphabet: the actions in G will have to be executed by the two arguments in the parallel composition. At the moment we only have to consider the case when $G = \varnothing$. Then, taking $t = min\{\mathsf{nd}(A_1), \mathsf{nd}(A_2)\}$, we have

$$A_1 \sqcup_\varnothing A_2 = (A_1\rceil t) \cup (A_2\rceil t) \cup \begin{cases} \{\Omega t\} & \text{if } t < \infty, \\ \varnothing & \text{otherwise.} \end{cases}$$

Now the semantic meaning of the external choice operator is as follows:

$$\begin{aligned} \mathcal{B}_{\mathrm{con}}[\![\Box]\!](B_1, B_2) = & \Big\{ A_1 \sqcup_\varnothing A_2 \;\Big|\; A_1 \in B_1 \text{ and } A_2 \in B_2 \Big\} \\ & \cup \Big\{ (A_1 \cup A_2\rceil t)at \cdot b \;\Big|\; A_1 at \cdot b \in B_1 \; A_2 \in B_2 \text{ and } \mathsf{nd}(A_2) > t \Big\} \\ & \cup \Big\{ (A_2 \cup A_1\rceil t)at \cdot b \;\Big|\; A_2 at \cdot b \in B_2 \; A_1 \in B_1 \text{ and } \mathsf{nd}(A_1) > t \Big\}. \end{aligned}$$

Parallel

The definition of this operator is more complex. First the parallel composition of two barbs yield to a set of barbs by interleaving, what is formally defined as follows:

Definition 18. Given b_1 and b_2 barbs and $G \subseteq Act$, we define the set of barbs $b_1 \parallel_G b_2$ as the least set satisfying:

$$
\begin{array}{l}
b_1 = A_1 \text{ and } b_2 = A_2 \;\Rightarrow\; b_1 \parallel_G b_2 = \{A_1 \sqcup_G A_2\},\\
\left.\begin{array}{l} b_1 = A_1 a t \cdot b_1',\ b_2 = A_2,\ a \notin G,\\ \mathsf{nd}(A_2) \geq t \text{ and } b' \in b_1' \parallel_G (A_2 - t) \end{array}\right\} \Rightarrow (A_1 \sqcup_G A_2 \rceil t) a t \cdot b' \in b_1 \parallel_G b_2,\\
\left.\begin{array}{l} b_1 = A_1,\ b_2 = A_2 a t \cdot b_2',\ a \notin G,\\ \mathsf{nd}(A_1) \geq t \text{ and } b' \in (A_1 - t) \parallel_G b_2' \end{array}\right\} \Rightarrow (A_1 \sqcup_G A_2 \rceil t) a t \cdot b' \in b_1 \parallel_G b_2,\\
\left.\begin{array}{l} b_1 = A_1 a_1 t_1 \cdot b_1',\quad b_2 = A_2 a_2 t_2 \cdot b_2',\\ t_1 \leq t_2,\quad a_1 \notin G \text{ and}\\ b' \in b_1' \parallel_G (A_2 - t_1) a_2 (t_2 - t_1) \cdot b_2' \end{array}\right\} \Rightarrow (A_1 \sqcup_G A_2 \rceil t_1) a_1 t_1 \cdot b' \in b_1 \parallel_G b_2,\\
\left.\begin{array}{l} b_1 = A_1 a_1 t_1 \cdot b_1',\quad b_2 = A_2 a_2 t_2 \cdot b_2',\\ t_1 \geq t_2,\quad a_2 \notin G \text{ and}\\ b' \in (A_1 - t_2) a_1 (t_1 - t_2) \cdot b_1' \parallel_G b_2' \end{array}\right\} \Rightarrow (A_1 \rceil t_2 \sqcup_G A_2) a_2 t_2 \cdot b' \in b_1 \parallel_G b_2,\\
\left.\begin{array}{l} b_1 = A_1 a t \cdot b_1',\quad b_2 = A_2 a t \cdot b_2',\\ a \in G \text{ and } b' \in b_1' \parallel_G b_2' \end{array}\right\} \Rightarrow (A_1 \sqcup_G A_2) a t \cdot b' \in b_1 \parallel_G b_2.
\end{array}
$$

Now we can define the semantic meaning of the parallel operator by considering the set of all the possible combinations by interleaving of barbs of the two processes.

Definition 19. Let B_1 and B_2 be set of barbs and $G \subseteq Act$, we define

$$\mathcal{B}_{\mathrm{con}}[\![\|_G]\!](B_1, B_2) = \left\{ b \mid \exists\ b_1 \in B_1,\ b_2 \in B_2 :\ b \in b_1 \parallel_G b_2 \right\}.$$

Hiding

When applying this operator the hidden action becomes the not visible one (τ), and then becomes urgent. To compute the b-traces of the obtained process we need the following

Definition 20. We say that an action a can be hidden in a b-trace bs, and we will write $\mathsf{ocul}(bs, a)$, taking

- $\mathsf{ocul}(\epsilon, a)$, and then we will take $\epsilon \setminus a = \epsilon$.
- If $\mathsf{ocul}(bs_1, a)$ and $a \notin A_1$ then $\mathsf{ocul}(A_1 a_1 t_1 \cdot bs_1)$ holds if one of the following conditions is fulfilled:
  - $a \neq a_1$. In this case we will take $(A_1 a_1 \cdot bs_1) \setminus a = A_1 a_1 t_1 \cdot (bs_1 \setminus a)$.
  - $a = a_1$ y $bs_1 \neq \epsilon$. In this case we will take $(A_1 a_1 \cdot bs_1) \setminus a = (A_1, t_1) \sqcup (bs_1 \setminus a)$.

To make the reading easier, whenever we write $bs \setminus a$, we understand that we also have $\mathsf{ocul}(bs, a)$.

Then the b-traces of $P \setminus a$ are those $bs \setminus a$ where bs is a b-trace of P and $\mathsf{ocul}(bs, a)$. The urgent condition is reflected by the second condition since in the definition we require $a \notin A_1$. Now the states of $P \setminus a$ are obtained from the barbs of P where the executed action is the hidden one. These states are computed *step by step*, and so the final set of states is obtained as the limit of the (possible infinite) sequence of *steps*.

Definition 21. Let B be a consistent set of barbs and $a \in Act$, for each $k \in \mathbb{N}$ we define the set of states $\mathsf{Focul}(B, a, k)$, as follows:

- $\mathsf{Focul}(B, a, 0) = \left\{\{\Omega 0\}\right\}$.
- For $k > 0$ we have
  - If $A \in B$ and $a \notin A$ then $A \in \mathsf{Focul}(B, a, k)$.
  - If $Aat \in \mathsf{Btraz}(B)$, $a \notin A$ and $A_1 \in \mathsf{Focul}(\mathsf{Barb}(B, Aat), a, k-1)$, then
$$A \cup (A_1 + t) \in \mathsf{Focul}(B, a, k).$$

Definition 22. If B is a consistent set of barbs and $a \in Act$, we have $bs \cdot A \in \mathcal{B}_{\mathrm{con}}[\![\backslash a]\!](B)$ iff one of the following conditions holds:

- $\mathsf{nd}(A) < \infty$ and for each $k \in \mathbb{N}$ there exists some $l \geq k$ and a b-trace bs' verifying $A \in \mathsf{Focul}(\mathsf{Barb}(B, bs'), a, l)$ and $bs' \setminus a = bs$.
- $\mathsf{nd}(A) = \infty$ and for each $t \in \mathcal{T}$ there exist $l \in \mathbb{N}$, a state A_t and a b-trace bs' verifying $A_t]t = A]t$, $A_t \in \mathsf{Focul}(\mathsf{Barb}(B, bs'), a, l)$ and $bs' \setminus a = bs$.

Recursion

Once we have given the semantic definition of each operator, we have to deal with recursion. We want to use the classical theory of fixed points to give a meaning to recursive terms. Unfortunately we have a problem since we have not been able to prove that the pre-order[§] $\ll$ is complete. So we have decided to find an alternative pre-order $\prec$, that we have called *definition pre-order*:

Definition 23.

- Let b and b' be barbs, we define $b \prec b'$ iff one of the following conditions holds:
  - $b = A \wedge b' = A' \wedge \mathsf{nd}(A) \leq \mathsf{nd}(A') \wedge A]\mathsf{nd}(A) = A']\mathsf{nd}(A)$,
  - $b = A \wedge b' = A'_1 a_1 t_1 \cdot b'_1 \wedge \mathsf{nd}(A) \leq t_1 \wedge A]\mathsf{nd}(A) = A'_1]\mathsf{nd}(A)$,
  - $b = Aa_1 t_1 \cdot b_1 \wedge b' = Aa_1 t_1 \cdot b'_1 \wedge b_1 \prec b'_1$.
- Let B and B' be consistent sets of barbs, we write $B \prec B'$ whenever the following properties are fulfilled
  - for all $b' \in B'$ there exists $b \in B$ such that $b \prec b'$.
  - for all $bs \cdot A \in B$ there exists some state A' such that $bs \cdot A' \in B'$ and $A \prec A'$.

First we prove that $\prec$ is complete, so that we can compute least upper bound of chains $B_1 \prec B_2 \prec \cdots B_k \prec B_{k+1} \prec \cdots$

Definition 24. Let $\mathcal{B} = \{B_i \mid i \in \mathbb{N}\}$ be a chain, we define $\mathsf{lub}(\mathcal{B})$ by taking $b \in \mathsf{lub}(\mathcal{B})$ iff the following conditions hold:

- If $\mathsf{nd}(b) < \infty$ then $\forall k \in \mathbb{N}\ \exists l \geq k :\ b \in B_l$.
- If $\mathsf{nd}(b) = \infty$ then $\forall t \in \mathcal{T}\ \exists l \in \mathbb{N}, b_l \in B_l :\ b_l]t = b_k]t$.

[§] We are using pre-orders instead of partial orders, so we have to take account that the identity is modulo the equivalence relation induced by the pre-order.

Now we have that $\mathsf{lub}(\mathcal{B})$ is, in fact, the least upper bound of the chain $\mathcal{B}$.

Proposition 25. *If $\mathcal{B} = \{B_i \mid i \in \mathbb{N}\}$ is a chain then*

- $\forall i \in \mathbb{N} :\ B_i \prec \mathsf{lub}(\mathcal{B})$.
- $\Big(\forall i \in \mathbb{N} :\ B_i \prec B'\Big) \ \Rightarrow\ \mathsf{lub}(\mathcal{B}) \prec B'$.

The proof of the previous proposition is quite easy having account that we are using a finite alphabet and a discrete time domain. We also can prove that $\prec$ has good properties with respect to $\ll$:

Proposition 26.

- *If $B_1 \prec B_2$ then $B_1 \ll B_2$.*
- *If $\mathcal{B} = \{B_i \mid i \in \mathbb{N}\}$ is a chain*[¶] *then $\Big(\forall i \in \mathbb{N} :\ B_i \ll B'\Big) \ \Rightarrow\ \mathsf{lub}(\mathcal{B}) \ll B'$.*

The first property establishes that $\prec$ is stronger than $\ll$, so any chain with respect to $\prec$ is also a chain with respect $\ll$; the second one establishes that least upper bounds with respect to $\prec$ are also least upper bounds with respect to $\ll$.

The new pre-order $\prec$ also has *good* properties with respect to the defined operators. First we have that all the operators are monotonic with respect to $\prec$.

Proposition 27. *Let $B_1, \ldots B_n, B'_1, \ldots, B'_n$ be consistent sets of barbs such that $B_i \prec B'_i$, then we have:*
$\mathcal{B}_{\mathrm{con}}[\![op]\!](B_1, \ldots, B_n) \prec \mathcal{B}_{\mathrm{con}}[\![op]\!](B'_1, \ldots, B'_n)$ for each $op \in \{et;, \Box, \sqcap, \|_G, \backslash a\}$.

To be able to define a denotational semantics by means of fixed points we also need to check that all the semantic operators are continuous:

Proposition 28. *Let $\mathcal{B} = \{B_i \mid i \in \mathbb{N}\}$ be a chain of consistent set of barbs and B a consistent set of barbs, then we have:*

- *For $op \in \{et;, \backslash a\}$ we have $\mathcal{B}_{\mathrm{con}}[\![op]\!](\mathcal{B}) \prec \mathsf{lub}\Big(\{\mathcal{B}_{\mathrm{con}}[\![op]\!](B_i) \mid i \in \mathbb{N}\}\Big)$.*
- *For $op \in \{\Box, \sqcap, \|_G\}$ we have $\mathcal{B}_{\mathrm{con}}[\![op]\!](B, \mathcal{B}) \prec \mathsf{lub}\Big(\{\mathcal{B}_{\mathrm{con}}[\![op]\!](B, B_i) \mid i \in \mathbb{N}\}\Big)$ and $\mathcal{B}_{\mathrm{con}}[\![op]\!](\mathcal{B}, B) \prec \mathsf{lub}\Big(\{\mathcal{B}_{\mathrm{con}}[\![op]\!](B_i, B) \mid i \in \mathbb{N}\}\Big)$.*

Then we can define the denotational semantics in the classical way:

Definition 29. Let $ENV = \{\rho : \mathsf{Var} \mapsto \mathcal{B}_{\mathrm{con}}\}$ the set of environments, we define the semantic meaning of open terms $\mathcal{B}_{\mathrm{con}}[\![\cdot]\!] : \mathsf{Term} \times ENV \mapsto \mathcal{B}_{\mathrm{con}}$ as follows

$$\mathcal{B}_{\mathrm{con}}[\![P]\!]_\rho \begin{cases} \mathcal{B}_{\mathrm{con}}[\![op]\!]\Big(\mathcal{B}_{\mathrm{con}}[\![P_1]\!]_\rho, \cdots, \mathcal{B}_{\mathrm{con}}[\![P_n]\!]_\rho\Big) & \text{if } P = op(P_1, \cdots, P_n), \\ \rho(x) & \text{if } P = x \in \mathsf{Var}, \\ \mathrm{fix}(\lambda B.\mathcal{B}_{\mathrm{con}}[\![P]\!]_{\rho[B/x]}) & \text{if } P = \mathtt{REC}\, x.P. \end{cases}$$

[¶] $\mathcal{B}$ is a chain with respect the pre-order $\prec$, i.e., $B_1 \prec B_2 \prec \cdots B_k \prec B_{k+1} \cdots$

The fixed point of a function f can be reached by iterating the application of the function to the bottom element of the semantic domain:

$$\bot = f^0(\bot), f^1(\bot), \ldots, f^n(\bot), \ldots$$

As the bottom element of our domain is $\left\{\{\Omega 0\}\right\}$ while is also the meaning of the process DIV, fixed points can be obtained as the limit of the semantics of the corresponding finite approximations:

Definition 30. We define the finite approximations of a process P by

$$\mathsf{ap}(P,k) = \begin{cases} \mathtt{DIV} & \text{if } k = 0, \\ x & \text{if } P = x \in \mathsf{Var}, \\ op(\mathsf{ap}(P_1,k),\ldots,\mathsf{ap}(P_n,k)) & \text{if } P = op(P_1,\ldots,P_n) \text{ and } k > 0, \\ \mathsf{ap}(P_1,k-1)[\mathsf{ap}(P,k-1)/x] & \text{if } P = \mathtt{REC}\, x.P_1 \text{ and } k > 0. \end{cases}$$

Now we have

Proposition 31. *For each process P we have*

- $\mathcal{B}_{\mathrm{con}}[\![\mathsf{ap}(P,k)]\!] \prec \mathcal{B}_{\mathrm{con}}[\![\mathsf{ap}(P,k+1)]\!]$.
- $\mathcal{B}_{\mathrm{con}}[\![P]\!] = \mathsf{lub}\{\mathcal{B}_{\mathrm{con}}[\![\mathsf{ap}(P,k)]\!] \mid k \in \mathbb{N}\}$.

6 Full Abstraction of our Denotational Semantics

Now we relate of denotational semantics developed in the previous section with the testing semantics presented in section 4. We will show that the denotational semantics is fully abstract with respect to the testing semantics:

$$\mathcal{B}_{\mathrm{con}}[\![P]\!] \ll \mathcal{B}_{\mathrm{con}}[\![Q]\!] \iff P \sqsubsetneq Q.$$

For, we will use the operational characterization in section 4.1:

$$\mathcal{B}_{\mathrm{con}}[\![P]\!] \ll \mathcal{B}_{\mathrm{con}}[\![Q]\!] \iff \mathsf{Barb}(P) \ll \mathsf{Barb}(Q).$$

First we prove

Proposition 32. *Let B_1 and B_2 be consistent sets of barbs and P and Q be processes. We have*

- *for each* $op \in \{\mathtt{STOP}, \mathtt{DIV}, et, ;, \sqcap, \Box, \|_G, \backslash a\}$, *if* $B_i \ll \mathsf{Barb}(P_i)$ *then*
  $$\mathcal{B}_{\mathrm{con}}[\![op]\!](B_1,\ldots,B_n) \ll \mathsf{Barb}(op(P_1,\ldots,P_n)).$$
- *for each* $op \in \{\mathtt{STOP}, \mathtt{DIV}, et, ;, \sqcap, \Box, \|_G, \backslash a\}$, *if* $\mathsf{Barb}(P_i) \ll B_i$ *then*
  $$\mathsf{Barb}(op(P_1,\ldots,P_n)) \ll \mathcal{B}_{\mathrm{con}}[\![op]\!](B_1,\ldots,B_n).$$

The main difficulty in the proof of the previous proposition has been that the operational states of a process can be obtained by an infinite computation, so reasoning by induction on the length on that computation cannot be applied.

Then we have the fully abstraction theorem for finite processes

Theorem 33. *If P is a finite process we have* $\mathsf{Barb}(P) \ll \mathcal{B}_{\mathrm{con}}[\![P]\!]$ *and* $\mathcal{B}_{\mathrm{con}}[\![P]\!] \ll \mathsf{Barb}(P)$.

Now we have to proof the abstraction theorem for general processes (also those including recursion). First we have the following

Lemma 34. *For each process P we have:*

- *If $b \in \mathsf{Barb}(P)$ and $t \leq \mathsf{nd}(b)$ there exist $k \in \mathbb{N}$ and a barb b' such that $b_k \in \mathsf{Barb}(\mathsf{ap}(P,k))$ and $b_k \rceil t = b \rceil t$.*
- *If $b \in \mathsf{Barb}(P)$, $\mathsf{nd}(b) < \infty$ there exists $k \in \mathbb{N}$ such that $\forall k' \geq k : \; b \in \mathsf{Barb}(\mathsf{ap}(P,k'))$.*
- *If $b \in \mathsf{Barb}(\mathsf{ap}(P,k))$ and $\mathsf{nd}(b) \geq t$ there exists $b' \in \mathsf{Barb}(P)$ such that $b \rceil t = b' \rceil t$.*
- *If b is a barb verifying $\mathsf{nd}(b) < \infty$ and $\forall k \in \mathbb{N}\ \exists l \geq k : b \in \mathsf{Barb}(\mathsf{ap}(P,k'))$ then $b \in \mathsf{Barb}(P)$.*

Proof. It is enough to notice that any finite computation of a process can be simulated by an approximation $\mathsf{ap}(P,k)$ for a large enough k; and, on the other hand, any computation of any finite approximation $\mathsf{ap}(P,k)$ can be simulated by the process P itself. □

Proposition 35. *For each process P then $\mathcal{B}_{\mathrm{con}}[\![P]\!] \ll \mathsf{Barb}(P)$ and $\mathsf{Barb}(P) \ll \mathcal{B}_{\mathrm{con}}[\![P]\!]$.*

Proof. It is quite easy since $\mathcal{B}_{\mathrm{con}}[\![P]\!] = \mathsf{lub}\{\mathsf{ap}(P,k) \mid k \in \mathbb{N}\}$ and the the set of states of a process is t-compact. □

Finally, by consequence of the previous proposition we have the theorem that shows the abstraction of the denotational semantics.

Theorem 36. *Given P and Q processes, then $P \sqsubseteq Q \iff \mathcal{B}_{\mathrm{con}}[\![P]\!] \ll \mathcal{B}_{\mathrm{con}}[\![Q]\!]$.*

7 Conclusions

In this paper we have presented a denotational semantics for a timed process algebra. This denotational semantics is fully abstract with respect to a must testing semantics defined and characterized in a previous paper [LdFN96] by the same authors. Here we have a first consequence: since any denotational semantics is (by definition) compositional, the must testing semantics is a congruence.

The results in this paper are part of the Ph.D. thesis of the first author, where the must testing semantics for timed process algebras have been deeply studied. In that work one can also find an axiomatic semantics that will be presented in a companion paper. This semantics is proved to be sound and complete with respect to the denotational semantics presented in this paper, therefore it is sound and complete with respect to the must testing semantics.

Acknowledgments

I would like to mention in this section all the people that are working around the world to make it better, specially those that have recently been killed in Zaire and Rwanda. I would like to remark that almost all first world countries have not carried out their pledges in Rio de Janeiro to *refund* the 0.7% of their respective gross national product for the *real* development of third world countries.

References

[BB93] J. C. M. Baeten and J. A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3:142–188, 1993.

[BK84] J. A. Bergstra and J. W. Klop. Process algebra for syncronous communication. *Information and Control*, 60:109–137, 1984.

[dFLL+95] D. de Frutos, G. Leduc, L. Léonard, L. F. Llana-Díaz, C. Miguel, J. Quemada, and G. Rabay. Time Extended LOTOS. In J. Quemada, editor, *Working Draft on Enhancements to LOTOS*. ISO/IEC JTC1/SC21/WG1, November 1995.

[dNH84] R. de Nicola and M. C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[Gro93] J. F. Groote. Transition system specifications with negative premises. *Theoretical Computer Science*, 118:263–299, 1993.

[Hen88] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.

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

[HR95] M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117:221–239, 1995.

[LdFN96] L. F. Llana-Díaz, D. de Frutos, and M. Núñez. Testing semantics for urgent process algebras. In *Third AMAST Workshop in Real Time Programming*, March 1996. To appear in World Sientific: AMAST Series in Computing.

[Lla96] L. F. Llana-Díaz. *Jugando con el Tiempo*. PhD thesis, Universidad Complutense de Madrid, 1996.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[NS91] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In *Computer Aided Design*, pages 376–398, 1991. LNCS 575.

[OM91] Y. Ortega-Mallén. *En Busca del Tiempo Perdido*. PhD thesis, Departamento de Informática y Automática. Universidad Complutense de Madrid, 1991.

[QdFA93] J. Quemada, D. de Frutos, and A. Azcorra. Tic: A timed calculus. *Formal Aspects of Computing*, 5:224–252, 1993.

[RR86] G. M. Reed and A. W. Roscoe. A timed model for communicating sequential processes. In *ICALP '86*, pages 314–323. Springer-Verlag, 1986. LNCS 226.

[Sch95] S. Schneider. An operational semantics for timed CSP. *Information and Computation*, 116(2):193–213, 1995.

[Yi91] W. Yi. *A Calculus of Real Time Systems*. PhD thesis, Department of Computer Science. Chalmers University of Technology, 1991.