University of Castilla-La Mancha

A publication of the

Department of Computer Science

Translating a Timed Process Algebra into Dynamic State Graphs.

by

Juan José Pardo Mateo, Valentín Valero Ruiz.

Technical Report **#DIAB-01-01-12** January 2001

A partial version has been submitted to CAV'01

DEPARTAMENTO DE INFORMÁTICA
ESCUELA POLITÉCNICA SUPERIOR
UNIVERSIDAD DE CASTILLA-LA MANCHA
Campus Universitario s/n
Albacete - 02071 - Spain
Phone +34.967.599200, Fax +34.967.599224

1 Introduction

We can find a wide range of systems for which time becomes important (real-time systems). We also find in these systems that some actions or tasks are critical, in the sense that they must be executed urgently once they become enabled. Thus, in the last years several extensions of the classical models to describe concurrent systems have been defined in order to incorporate this factor: we have timed extensions of CSP [17, 13, 19], CCS [11], and LOTOS [15]. Bolognesi and Lucidi [2] have defined another timed extension of LOTOS, which includes an operator for the specification of urgent interactions. Hansson and Jonsson have defined another timed and probabilistic extension of CCS [7, 6], by including a *time-out* operator. In fact, there are not yet too many proposals combining both factors on one single model. We can cite [3], but the syntax of the model there presented is very limited, and the model is restricted to finite processes. In [4] a testing semantics for an extension of LOTOS with time and probabilities is also presented.

The model we present is based on basic LOTOS, adding some new operators for the specification of timed behaviours and urgent interactions. Following the ideas of Quemada, Azcorra and de Frutos [15], we introduce a timed prefix operator, $a < t_1, t_2 >; P$, which establishes a time interval for the execution of the corresponding action, and once the deadline for the action has expired this action cannot be executed ever after. Other operators we have included are an internal action prefix, $\tau; P$, and a delay operator, $wait(t); P$. The first one establishes the urgent character of the τ action, in the sense that it must be executed immediately, without any delay (unless we have a conflict with another internal action), while the second one establishes a delay of t units of time, and then it behaves like $\tau; P$. Besides, we maintain the classical LOTOS operators for external choice, parallel composition, hiding and recursion.

We present an operational semantics for the language, which captures the aforementioned behaviour, and particularly the urgent character of the internal action. Nevertheless, in order to verify some properties of a model it is more convenient to have a graph representation, so the main goal of this paper is to define such a representation for a certain kind of terms, for which we obtain a finite graph. But, due to the special character of the internal action, which affects to the behaviour of the hiding operator, and of course, due to the parallel operator, we have that this graph cannot be static, in the sense that we cannot get an explicit finite representation of the reachable states which allows us to define correctly all the operators. Then, following similar ideas to that used in [12] we have defined a dynamic state graph, whose semantics is defined by means of a set of clocks, and a set of clock restrictions labelling each arc.

The relationship between process algebras and timed graphs has been studied in several previous works . Modecharts [8, 10] and timed automata [9, 1] are very well known classical tools to describe concurrent systems. Also, in [18] Communicating Real-Time Machines (CRSM) are defined to model real-time systems. In this model, a system is modeled by means of a collection of these machines interconnected by one-to-one communication channels. [16] provides a method to transform these machines into graphs whose nodes consists of the current location of each CRSM, the variable valuation and the time spent by each CRSM in its current location. Another timed model based on timed graphs that has been proposed are *Timed Transition Models* [14], in which time is modeled by means of a external and conceptual global clock, and a set of variables capture the local timing information of each process. Then, each transition consists of an enabling condition, a transformation function, and time bounds for its execution.

The technical report is structured as follows. In Section 2 we present the syntax and the semantics of the process algebra. In Section 3 the definition of the dynamic state graph is presented, showing the equivalence between both semantics. Finally, in Section 4 a classical example is modeled, the producer-consumer system, and even we show the possible use of the graph to conclude some properties of it.

2 Definition of the Process Algebra

2.1 Syntax

Let Vis be a finite set of actions, and $Act = Vis \cup \{\tau\}$. The syntax of the language TPA (Timed Process Algebra) is defined by the following BNF expression:

$$T ::= P \mid N$$
$$N ::= stop \mid a < t_1, t_2 >; T \mid \tau; T \mid wait(t); T \mid N \Box N \mid N \parallel_A N \mid N \backslash_a \mid X \mid \mu X.N$$
$$P ::= \sum_{i \in I} [q_i].N \mid P \parallel_A N \mid N \parallel_A P \mid P \parallel_A P \mid P \backslash_a \mid X \mid \mu X.P$$

where $A \subseteq Act$, $X \in idf$ (set of identifiers), $a \in Vis$, $t, t_1 \in \mathbb{R}_0^+ = \mathbb{R}^+ \cup \{0\}$, $t_2 \in \mathbb{R}_0^+ \cup \{\infty\}$, $q_i \in [0,1]$, $\sum_i q_i = 1$.

The informal interpretation of the operators follows:

stop .- It represents a deadlock, i.e. no action can be executed.

$a < t_1, t_2 >; T$.- The action $'a'$ can be executed after a delay in the time interval $[t_1, t_2]$. However, once the time interval has expired, we lose the ability to execute that action, i.e., we do not force its execution when its time-out is about to expire. As usual, once a has been executed, the process behaves like T.

$\tau; T$.- The internal action (τ) is executed immediately, and the process behaves like T.

$wait(t); T$.- The process waits for $'t'$ units of time, and then it behaves like $\tau; T$.

$N_1 \Box N_2$.- It is essentially the classical external choice operator. When an action is requested to be executed, it will be executed by the non-deterministic process that can execute it, and if both processes may execute that action, the choice is internal (the system chooses the process that will execute the action).

$N_1 \parallel_A N_2$.- It represents the parallel execution of P_1 and P_2 synchronizing on the actions in A.

$N\backslash a$.- All instances of the action a in P are hidden, and thus they become urgent (τ).

X .- It may represent any process of the language.

$\mu X.N$.- It is the classical recursion operator.

$\sum_{i\in I}[q_i].N_i$.- It represents a probabilistic choice i.e. the system selects in a probabilistic way the action that will be executed.

Terms obtained from this syntax will be denoted by TPA. However, in order to define the dynamic state graph for a timed process we will restrict to regular terms, for which terms $\mu X.P$ must be guarded (all appearances of X in P must be prefixed), and there is neither a parallel operator, nor a hiding operator in P). These terms will be denoted by RTPA.

2.2 Operational Semantics

As usual, we define an operational semantics for the closed terms of this language by a labelled transition system in the Plotkin and Milner's style. We have two kind of transition rules for timed processes (see table 1) and one kind for the probabilistic processes (see table 2).

$N \underset{t}{\longmapsto} N'$.- The process N behaves as P' after t units of time.

$N \xrightarrow{a} T'$.- The process N executes the action a and then behaves as T' (no time passes).

$P \rightarrow_q N \ where \ q \in [0,1]$.- The process evolves by a system choice according to probabilities.

With rule N1 we capture that for the process stop the only possible evolution is the passage of time. Nules R2a-c capture the semantics of the timed prefix. Concretely, rule N2c states that once the time to execute an action has expired, we lose the ability to execute that action. Rules N3 and N4a-b capture the urgent character of internal actions, and the semantics of the wait operator. Rules N5a-b gives the classical semantics for the external choice, while N7 states that the time can pass in an external choice only when both components are able to do that. Rules N8a-c are the classical ones for the parallel operator, while N8d is similar to N7. Semantics of the hiding operator is captured by rules N9a-c, the latter one capturing the fact that once an action has been hidden, it becomes urgent [1]. Finally rules N10a-b capture the semantics for the recursion operator.

[1] N9c introduces negative premises, however, by using the same procedure as in [5] consistency of the transition system can be proved.

N1)	$\text{stop} \underset{t}{\longmapsto} \text{stop} \quad \forall t \in \mathbb{R}_0^+$
N2a)	$a < 0, t >; T \xrightarrow{a} T \quad \forall t \in \mathbb{R}_0^+$
N2b)	$a < t_1, t_2 >; T \underset{t'}{\longmapsto} a < t_1 \dot{-} t', t_2 - t' >; T \quad 0 \leq t' \leq t_2,$
	where $x \dot{-} y = \max\{0, x - y\}$
N2c)	$a < 0, 0 >; T \underset{t'}{\longmapsto} \text{stop} \quad t' > 0$
N3)	$\tau; T \xrightarrow{\tau} T$
N4a)	$\text{wait}(t); T \underset{t'}{\longmapsto} \text{wait}(t - t'); T \quad 0 \leq t' \leq t$
N4b)	$\text{wait}(0); T \xrightarrow{\tau} T$
N5a)	$\dfrac{N_1 \xrightarrow{e} T_1'}{N_1 \Box N_2 \xrightarrow{e} T_1'}$
N5b)	$\dfrac{N_2 \xrightarrow{e} T_2'}{N_1 \Box N_2 \xrightarrow{e} T_2'}$
N6)	$\dfrac{N_1 \underset{t_1}{\longmapsto} N_2 \quad N_2 \underset{t_2}{\longmapsto} N_3}{N_1 \underset{t_1+t_2}{\longmapsto} N_3}$
N7)	$\dfrac{N_1 \underset{t'}{\longmapsto} N_1' \quad N_2 \underset{t'}{\longmapsto} N_2'}{N_1 \Box N_2 \underset{t'}{\longmapsto} N_1' \Box N_2'}$
N8a)	$\dfrac{N_1 \xrightarrow{e} T_1' \quad e \notin A}{N_1 \parallel_A N_2 \xrightarrow{e} T_1' \parallel_A N_2}$
N8b)	$\dfrac{N_2 \xrightarrow{e} T_2' \quad e \notin A}{N_1 \parallel_A N_2 \xrightarrow{e} N_1 \parallel_A T_2'}$
N8c)	$\dfrac{N_1 \xrightarrow{a} T_1', \quad N_2 \xrightarrow{a} T_2' \quad a \in A}{N_1 \parallel_A N_2 \xrightarrow{a} T_1' \parallel_A T_2'}$
N8d)	$\dfrac{N_1 \underset{t'}{\longmapsto} N_1', \quad N_2 \underset{t'}{\longmapsto} N_2'}{N_1 \parallel_A N_2 \underset{t'}{\longmapsto} N_1' \parallel_A N_2'}$
N9a)	$\dfrac{N \xrightarrow{e} T', \ e \in Act \quad e \neq a}{N \backslash a \xrightarrow{e} T' \backslash a}$
N9b)	$\dfrac{N \xrightarrow{a} T'}{N \backslash a \xrightarrow{\tau} T' \backslash a}$
N9c)	$\dfrac{N \underset{t}{\longmapsto} N', \neg N \xrightarrow{a} \wedge \neg \Phi(N, t, a)}{N \backslash a \underset{t}{\longmapsto} N' \backslash a}$ where
	$\Phi(N, t, a) \equiv [\exists\, t' < t, \ N \underset{t'}{\longmapsto} N'', \ N'' \xrightarrow{a}]$
N10a)	$\dfrac{N\{\mu x.N/x\} \xrightarrow{e} T'}{\mu x.N \xrightarrow{e} T'}$
N10b)	$\dfrac{N\{\mu x.N/x\} \underset{t}{\longmapsto} N'}{\mu x.N \xrightarrow{e} N'}$

Table 1: Non-Deterministic Rules

Rule P1 indicates the behaviour of the probabilistic choice operator. Rules P2a-c give the semantics for the probabilistic parallel operator. Rule P3 gives the semantics for the hiding operator for probabilistic processes and with rule P4 we capture the sematics for

$$
\begin{array}{ll}
\textbf{P1)} & \dfrac{}{\sum_{i\in I}[q_i]N_i \longrightarrow_{q_j} N_j} \\
\textbf{P2a)} & \dfrac{P_1 \longrightarrow_{q_1} N_1 \quad P_2 \longrightarrow_{q_2} N_2}{P_1 \parallel_A P_2 \longrightarrow_{q_1 \cdot q_2} N_1 \parallel_A N_2} \\
\textbf{P2b)} & \dfrac{P_1 \longrightarrow_q N_1}{P_1 \parallel_A N_2 \longrightarrow_q N_1 \parallel_A N_2} \\
\textbf{P2c)} & \dfrac{P_2 \longrightarrow_q N_2}{N_1 \parallel_A P_2 \longrightarrow_q N_1 \parallel_A N_2} \\
\textbf{P3)} & \dfrac{P \longrightarrow_q N}{P\backslash a \longrightarrow_q N\backslash a} \\
\textbf{P4)} & \dfrac{P\{\mu x.P/x\} \longrightarrow_q N}{\mu x.P \longrightarrow_q N}
\end{array}
$$

Table 2: Probabilistic Rules

the recursion operator.

Definition 1 Let $P,\ P' \in TPA$, $s \in Act^*$, $t \in \mathbb{R}_0^+$, $q \in [0,1]$.

We say that $P \stackrel{s}{\Longrightarrow}_{q,t} P'$ if and only if there is a sequence of rules from P to P'

$$P \longrightarrow_{q_1} \quad \cdots \quad P_i \xrightarrow{a_i} P_{i+1} \longrightarrow_{q_k} \quad \cdots \quad P_j \underset{t_j}{\longmapsto} P_{j+1} \quad \cdots \quad P'$$

such that $s = a_1 \cdots a_n,\ t = \sum_{j=1}^{m} t_j,\ q = \prod_{k=1}^{r} q_k$.

□

3 Dynamic State Graph

In this section, we only consider non-deterministic processes without probabilities, i.e., we only take into account timed processes. For this kind of processes we construct a Dynamic State Graph, which captures the behaviour of the process. The extension of this contructing to th full language, including probabilistic processes is straightforward, just considering on extension of the graph, but here we only present the timed version of these graphs.

These dynamic state graphs are essentially automata extended with timers, which capture the local time information of each process. We may find in the literature a related work [12], where timed graphs are defined for the language ATP (Algebra of Timed Processes). There are some differences between the dynamic state graphs here defined and the timed graphs presented in that paper. Some of these differences are related to the different operators that are considered in each model, but there are also some technical changes: Dynamic state graphs here defined include a global clock which capture the time evolution for the whole system, and local timers are not reset when local actions are executed. Instead, local timers synchronize with the global clock when it is necessary, thus maintaining the global timing information.

Then, nodes of these graphs are labelled by terms, and arcs are labelled by tuples (a, r, S) where a is an action, r is a set of time intervals specifying restrictions for some timers, and S is a set of timers that must synchronize with the global clock once executed this transition.

The formal definition follows:

Definition 2 A dynamic state graph is a tuple $G = (N, E, \lambda, n_0, \mathit{Clocks})$, where N is the set of nodes, each one labelled by a term (which we will denote by $\mathit{term}(n) \in \mathit{TPA}$, $n \in N$), $E \subseteq N \times N$, $n_0 \in N$, $\mathit{Clocks} = \{R_1, \ldots R_r\}$ is the set of timers where one of them is the global one (it will referred as R_g henceforth), and

$$\lambda : E \longrightarrow \mathit{Act} \times \mathit{Times} \times \mathcal{P}_F(\mathit{Clocks})$$

where

$$\mathit{Times} : \mathit{Clocks} \hookrightarrow \mathbb{R}_0^+ \times \mathbb{R}_0^+ \cup \{\infty\}$$

is a partial function that associates a time interval to some of the timers. We will denote each component of $\lambda(e)$ by $\lambda_i(e)$, $i = 1, 2, 3$.

□

Then given a dynamic state graph $G = (N, E, \lambda, n_0, \mathit{Clocks})$, we define a state of it by a tuple (n, C, t) where $n \in N$, $C : \mathit{Clocks} \longrightarrow \mathbb{R}_0^+$ (function that gives the timer values) and $t \in \mathbb{R}_0^+$ is the current instant, $t \geq C(R_g)$.

Given a state $s = (n, C, t)$ and an arc e leaving the node n with label: $\lambda(e) = (a, r, S)$, we define:

$$\mathit{Min}(n, C, e) = \max\{(C(R_j) + \alpha_j) \dot{-} C(R_g) \mid R_j \in \mathit{Clocks},\ r(R_j) \text{ defined},\ r(R_j) = [\alpha_j, \beta_j]\}$$

which represents the minimum delay until the action a can be executed according to this arc.

$$\mathit{Max}(n, C, e) = \min\{(C(R_j) + \beta_j) - C(R_g) \mid R_j \in \mathit{Clocks}, r(R_j) \text{ defined},\ r(R_j) = [\alpha_j, \beta_j]\}$$

which represents the maximum delay to execute this action a according to this arc. A negative value means that the time to execute that action has already expired, and therefore it cannot be executed.

We also define

$$\mathit{Min}_\tau(n, C) = \begin{cases} \infty & \text{if } \nexists\, e \text{ s.t. } n \xrightarrow{e},\ \lambda_1(e) = \tau \\ \min\{\mathit{Min}(n, C, e) \mid n \xrightarrow{e},\ \lambda_1(e) = \tau\} & \text{otherwise} \end{cases}$$

which represents the maximum delay that we have to execute the visible actions, due to the restrictions imposed by the urgent ones.

According to these definitions, we have that the action a can be executed at the state $s = (n, C, t)$ iff we have an arc e with label $\lambda(e) = (a, r, S)$ leaving n such that:

$$\mathit{Min}(n, C, e) \leq \Delta t \leq \mathit{Max}(n, C, e) \wedge \Delta t \leq \mathit{Min}_\tau(n, C)$$

where $\Delta t = t - C(R_g)$.

We will denote this predicate by $act(s,e)$.

In consequence, we may establish the following rules to capture the graph semantics:

G1) $$\frac{s \ , \ act(s,e)}{s \xrightarrow{e} (m, adjust(C,e,t), t)}$$

G2) $$\frac{s \ , \ t' \geq t \ , \ t' - C(R_g) \leq Min_\tau(n,C)}{s \underset{t'-t}{\longmapsto} (n, C, t')}$$

with $s = (n,C,t)$, $e = (n,m) \in E$ and

$$adjust(C,e,t)(R_i) = \begin{cases} t & \text{if } R_i \in \lambda_3(e) \\ C(R_i) & \textit{otherwise} \end{cases}$$

Let us observe that according to these rules the dynamic evolution of a graph does not affect to the clocks restrictions labelling the arcs, and even, the urgent character of the internal actions is captured by the graph semantics.

We will define the initial state of a graph by $s_0 = (n_0, C_0, 0)$, taking $C_0(R_j) = 0$, $\forall R_j \in \mathit{Clocks}$, and we will denote the reachable states by $\mathit{Reach}(G, s_0)$.

Lemma 1 Let $G = (N, E, \lambda, n_0, \mathit{Clocks})$ be a dynamic state graph, $s_0 = (n_0, C_0, 0)$ the initial state, as defined previously, and the state $s'_0 = (n_0, C'_0, t')$, for $t' \in \mathbb{R}_0^+$, with $C'_0(R_j) = t'$, $\forall R_j \in \mathit{Clocks}$.

We have that $(n, C, t) \in \mathit{Reach}(G, s_0)$ if and only if $(n, C', t+t') \in \mathit{Reach}(G, s'_0)$, with $C'(R_j) = C(R_j) + t'$, $\forall R_j \in \mathit{Clocks}$.

Proof: By induction on the number of rules (G1 or G2) that we apply to reach a state. Both the base case and the general one are very similar, by using the fact that $Min(n,C,e) = Min(n,C',e)$, $Max(n,C,e) = Max(n,C',e)$, $Min_\tau(n,C) = Min_\tau(n,C')$, and then checking that both rules preserve this property.

$\square$

3.1 Dynamic state graph for a timed process

Given a regular process $P \in RTPA$, we define in a structural way the Dynamic state graph associated to it. In this definition, for simplicity we name the nodes by the term that they represent, but it is obvious that we may have several nodes labelled with the same term.

- STOP
  The graph for the process STOP consists of only one node:

$$G[\![stop]\!] = (\{stop\}, \emptyset, \emptyset, stop, \{R_g\})$$

- Timed prefix
  Let $P = a < t_1, t_2 >; P_1$ be a process, and let $G[\![P_1]\!] = (N_1, E_1, \lambda_1, n_{01}, \mathit{Clocks}_1)$ be the graph associated to P_1. Then $G[\![P]\!] = (N, E, \lambda, P, \mathit{Clocks}_1)$, where

$$
\begin{array}{l}
N = N_1 \cup \{P\} \\
E = E_1 \cup \{P \xrightarrow{e} n_{01}\} \\
\lambda(\bar{e}) = \left\{ \begin{array}{ll} \lambda_1(\bar{e}) & \forall \bar{e} \in E_1 \\ (a, \mathit{Intg}, \mathit{Clocks}_1) & \text{if } \bar{e} = e \end{array} \right.
\end{array}
$$

  where Intg is defined only for R_g, being $\mathit{Intg}(R_g) = [t_1, t_2]$.

- Internal action prefix
  Let $P = \tau; P_1$ be a process, and let $G[\![P_1]\!] = (N_1, E_1, \lambda_1, n_{01}, \mathit{Clocks}_1)$ be the graph associated to P_1. Then $G[\![P]\!] = (N, E, \lambda, P, \mathit{Clocks}_1)$, where

$$
\begin{array}{l}
N = N_1 \cup \{P\} \\
E = E_1 \cup \{P \xrightarrow{e} n_{01}\} \\
\lambda(\bar{e}) = \left\{ \begin{array}{ll} \lambda_1(\bar{e}) & \forall \bar{e} \in E_1 \\ (a, \mathit{Intg}, \mathit{Clocks}_1) & \text{if } \bar{e} = e \end{array} \right.
\end{array}
$$

  where Intg is defined only for R_g, being $\mathit{Intg}(R_g) = [0, 0]$.

- Wait operator
  Let $P = \mathit{wait}(t); P_1$ be a process, and let $G[\![P_1]\!] = (N_1, E_1, \lambda_1, n_{01}, \mathit{Clocks}_1)$ be the graph associated to P_1. Then $G[\![P]\!] = (N, E, \lambda, P, \mathit{Clocks}_1)$, where

$$
\begin{array}{l}
N = N_1 \cup \{P\} \\
E = E_1 \cup \{P \xrightarrow{e} n_{01}\} \\
\lambda(\bar{e}) = \left\{ \begin{array}{ll} \lambda_1(\bar{e}) & \forall \bar{e} \in E_1 \\ (a, \mathit{Intg}, \mathit{Clocks}_1) & \text{if } \bar{e} = e \end{array} \right.
\end{array}
$$

  where Intg is defined only for R_g, being $\mathit{Intg}(R_g) = [t, t]$.

- External choice
  Let $P = P_1 \,\Box\, P_2$ be a process. We define $G[\![P]\!] = G[\![\mathit{Unfold}(P_1)]\!] \,\Box_G\, G[\![\mathit{Unfold}(P_2)]\!]$, where $\Box_G$ is a graph operation defined below that requires for both components to have an initial state without incoming arcs, and thus, in order to define the general behaviour of the external choice we need to *unfold* the outer recursion that could appear on the external choice components. This function is defined as follows:

$$
\begin{array}{l}
\mathit{Unfold}(\mathit{stop}) = \mathit{stop} \\
\mathit{Unfold}(a < t_1, t_2 >; P) = a < t_1, t_2 >; P \\
\mathit{Unfold}(\tau; P) = \tau; P \\
\mathit{Unfold}(\mathit{wait}(t); P) = \mathit{wait}(t); P \\
\mathit{Unfold}(P_1 \,\Box\, P_2) = \mathit{Unfold}(P_1) \,\Box\, \mathit{Unfold}(P_2) \\
\mathit{Unfold}(P_1 \,\|_A\, P_2) = \mathit{Unfold}(P_1) \,\|_A\, \mathit{Unfold}(P_2) \\
\mathit{Unfold}(P \backslash a) = \mathit{Unfold}(P) \backslash a \\
\mathit{Unfold}(\mu X.P) = \mathit{Unfold}(P\{\mu X.P/X\})
\end{array}
$$

Let us observe that this definition is correct, because we are considering regular terms.

Now, given two graphs, $G_i = (N_i, E_i, \lambda_i, n_{0i}, \mathit{Clocks}_i), i = 1, 2$, such that their initial states do not have incoming arcs, we define $G_1 \,\Box_G\, G_2 = (N, E, \lambda, n_0, \mathit{Clocks}_1 \cup \mathit{Clocks}_2)$ as follows:

$$\begin{array}{l}
N = (N_1 \cup N_2 \cup \{P\}) \setminus \{n_{01}, n_{02}\} \\
E = ((E_1 \cup E_2) \setminus E_r) \cup E_m\,, \quad \text{where} \\
\quad E_r = (\{(n_{01}, n_1) \,|\, (n_{01}, n_1) \in E_1\} \cup \{(n_{02}, n_2) \,|\, (n_{02}, n_2) \in E_2\})) \\
\quad E_m = (\{(n_0, n_1) \,|\, (n_{01}, n_1) \in E_1\} \cup \{(n_0, n_2) \,|\, (n_{02}, n_2) \in E_2\}) \\
\lambda(\bar{e}) = (\bar{\lambda_1}(\bar{e}),\ \bar{\lambda_2}(\bar{e}),\ \bar{\lambda_3}(\bar{e}) \cup \{R_g\}) \quad \forall \bar{e} \in E
\end{array}$$

where R_g is either the global clock of G_1 or the global clock of G_2, which becomes the global clock of G, and $\bar{\lambda}_j$, for $j = 1, 2, 3$, is defined by:

$$\bar{\lambda_j}(\bar{e}) = \begin{cases} \lambda_{ij}(\bar{e}) & \forall \bar{e} \in E_i \cap E \\ \lambda_{ij}(\bar{e_i}) & \text{if } \bar{e} = (n_0, n_i),\ \text{with } \bar{e_i} = (n_{0i}, n_i) \end{cases}$$

where λ_{ij} stands for the j-component of λ_i.

- Parallel Composition
  Let P_1, P_2 be two processes with graphs $G[\![P_i]\!] = (N_i, E_i, \lambda_i, n_{0i}, \mathit{Clocks}_i)$. Then we define the graph for the process $P = P_1 \|_A P_2$ as follows:

  $$G[\![P]\!] = (N, E, \lambda, n_0, \mathit{Clocks})$$

  where:

  $$\begin{array}{l} N = (N_1 \times N_2) \\ E = E_{11} \cup E_{12} \cup E_{13} \end{array}$$

  with

  $$\begin{array}{l}
  E_{11} = \{((n_1, n_2), (n_1', n_2)) \,|\, n_1 \xrightarrow{e_1}_1 n_1', \lambda_{11}(e_1) \notin A\} \\
  E_{12} = \{((n_1, n_2), (n_1, n_2')) \,|\, n_2 \xrightarrow{e_2}_2 n_2', \lambda_{21}(e_2) \notin A\} \\
  E_{13} = \{((n_1, n_2), (n_1', n_2')) \,|\, n_1 \xrightarrow{e_1}_1 n_1',\ n_2 \xrightarrow{e_2}_2 n_2', \lambda_{11}(e_1) = \lambda_{21}(e_2) \in A\} \\
  \mathit{Clocks} = \mathit{Clocks}_1 \cup \mathit{Clocks}_2 \cup \{R_g\}, \text{ s.t. } R_g \notin \mathit{Clocks}_1 \cup \mathit{Clocks}_2 \\
  n_0 = (n_{01}, n_{02}) \\
  \lambda(e) = \begin{cases} (\lambda_{11}(e_1), \lambda_{12}(e_1), \lambda_{13}(e_1) \cup \{R_g\}) & \text{for } e_1 \in E_1,\ e \in E_{11} \\ (\lambda_{21}(e_2), \lambda_{22}(e_2), \lambda_{23}(e_2) \cup \{R_g\}) & \text{for } e_2 \in E_2,\ e \in E_{12} \\ (a, \lambda_{12}(e_1) \cup \lambda_{22}(e_2), \lambda_{13}(e_1) \cup \lambda_{23}(e_2) \cup \{R_g\}) & \text{for } e \in E_{13}, a = \lambda_{11}(e_1) \end{cases}
  \end{array}$$

- Hiding
  Let $P = P_1 \backslash a$ be a process, and let $G[\![P_1]\!] = (N_1, E_1, \lambda_1, n_{01}, \mathit{Clocks}_1)$ be the graph associated to P_1. Then $G[\![P]\!] = (N_1 \backslash a, E_1, \lambda, n_{01}, \mathit{Clocks}_1)$, where

  $$\lambda(\bar{e}) = \begin{cases} (\tau, \lambda_{12}(e), \lambda_{13}(e)) & \forall e \in E_1, \text{ s.t. } \lambda_{11}(e) = a \\ \lambda_1(e) & \mathit{otherwise} \end{cases}$$

- X process
  The graph for the process X consists of only one node:

$$G[\![X]\!] = (\{X\}, \emptyset, \emptyset, X, \{R_g\})$$

- Recursion
  Let $P = \mu X.P_1$ be a process, and let $G[\![P_1]\!] = (N_1, E_1, \lambda_1, n_{01}, Clocks_1)$ be the graph associated to P_1. Then, $G[\![P]\!] = (N, E, \lambda, n_{01}, Clocks_1)$, where

$$N = N_1 \setminus \{n \in N_1 \mid term(n) = X\}, \text{ and also each occurrence of the identifier } X \text{ in the terms labelling the nodes of } N \text{ is replaced by } P.$$

$$E = (E_1 \cup \{(n, n_{01}) \mid (n, m) \in E_1 \mid term(m) = X\}) \setminus \{(n, m) \in E_1 \mid term(m) = X\}$$

$$\lambda(e) = \begin{cases} \lambda_1(e) & \text{if } e \in E_1 \\ (\lambda_{11}(e), \lambda_{21}(e), Clocks_1) & \text{if } e = (n, n_{01}) \wedge (n, m) \in E_1,\ term(m) = X \end{cases}$$

Proposition 1 $G[\![P]\!]$ is well defined for every $P \in RTPA$.

Proof: For finite terms is trivial. For infinite terms a structural induction is needed. The only cases requiring some comments are the external choice and the recursion.

- For the external choice, it is evident that given the graphs $G[\![P_1]\!]$, $G[\![P_2]\!]$, which can be constructed by induction hypothesis, we may also construct the graphs $G[\![Unfold(P_i)]\!]$, and they must have an initial state without incoming arcs, because all the identifiers must be prefixed in the recursion operator.
- If $P = \mu X.P_1$ we just need to observe that every appearance of X in P_1 must be prefixed, and neither a parallel operator nor a hiding operator can appear in P_1, so there cannot be any nodes labelled by terms like $X \backslash a$, $X \| Q$, $X \Box Q$ in $G[\![P_1]\!]$.

$\square$

In order to establish the correspondence between the terms that we can reach according to the operational semantics and the reachable states in the dynamic state graph associated to the given process, we need to capture the reachable states from a given node by the passage of time. However, we must take into account that every sequential component of a process has its own clock, and they are updated separately. In consequence, the evolution in time from a node must be defined by distinguishing the separate evolutions in time of each sequential component.

Definition 3 (Sequential components)
We define the set of sequential components by the following syntax:

$$S ::= stop \mid a < t_1, t_2 >; P \mid \tau; P \mid wait(t); P \mid S \Box S \mid S \|_A \mid {}_A\| S \mid S \backslash a$$

with $P \in TPA$. Then, we define the set of sequential components of a closed regular process P as follows:

$$
\begin{aligned}
&dex(stop) = \{stop\}\\
&dex(a < t_1, t_2 >; P) = \{a < t_1, t_2 >; P\}\\
&dex(\tau; P) = \{\tau; P\}\\
&dex(wait(t); P) = \{wait(t); P\}\\
&dex(P_1 \Box P_2) = dex(P_1) \Box dex(P_2)\\
&dex(P_1 \|_A P_2) = dex(P_1) \|_A \cup_A \| dex(P_2)\\
&dex(P \backslash a) = dex(P) \backslash a\\
&dex(\mu X.P) = dex(P\{\mu X.P/X\})
\end{aligned}
$$

□

Definition 4 Let P be a closed regular term. We define the complexity of P, $\gamma(P)$, as follows:

$$
\begin{aligned}
&\gamma(stop) = \gamma(a < t_1, t_2 >; P) = 0\\
&\gamma(P + Q) = \gamma(P \|_A Q) = 1 + \max\{\gamma(P), \gamma(Q)\}\\
&\gamma(P \backslash a) = \gamma(\mu X.P) = 1 + \gamma(P)
\end{aligned}
$$

γ is well defined, because $\gamma(\mu X.P) > \gamma(P)$, and P is guarded. As a consequence, we have that dex is also well defined.

□

Once we have defined the decomposition in sequential processes, we can define the function *"time_evol"*, which captures the possible terms that we may reach by the passage of time on each sequential component of a process, as follows:

Definition 5 Given two closed regular processes, P, Q. We say that $P \in time\_evol(Q)$ if and only if there is bijection $\varphi : dex(Q) \longrightarrow dex(P)$, s.t. $\forall S_Q \in dex(Q)\ \exists r \in \mathbb{R}_0^+$, $old(S_Q, r) = \varphi(S_Q)$, where the function old is defined as follows:

$$
\begin{aligned}
&old(stop, r) = stop\\
&old(a < t_1, t_2 >; P, r) = \begin{cases} stop & \text{if } r > t_2\\ a < t_1 \dot{-} r, t_2 - r >; P & \text{if } r \leq t_2 \end{cases}\\
&old(\tau; P, r) = \begin{cases} stop & \text{if } r > 0\\ \tau; P & \text{if } r = 0 \end{cases}\\
&old(wait(t); P, r) = \begin{cases} stop & \text{if } r > t\\ wait(t - r); P & \text{if } r \leq t \end{cases}\\
&old(S_1 \Box S_2, r) = old(S_1, r) \Box old(S_2, r)\\
&old(S \|_A, r) = old(S, r) \|_A\\
&old({}_A\| S, r) =_A \| old(S, r)\\
&old(S \backslash a, r) = old(S, r) \backslash a
\end{aligned}
$$

□

Now we may state our two main results, which establish the equivalence between the operational semantics and the graph semantics.

Theorem 1 Given two closed regular processes $P, P' \in RTPA$, if we have $P \stackrel{s}{\Longrightarrow}_t P'$,

then there is a term $\bar{P}' \in TPA$ and a node $n_{\bar{P}'} \in G[\![P]\!]$ such that $P' \in time\_evol(\bar{P}')$, and there is a reachable state $(n_{\bar{P}'}, C, t)$ in $G[\![P]\!]$, where s is the sequence of actions executed to reach that state.

Proof:

The complete proof is quite involved, because it is based on structural induction, and a lot of cases must be distinguished, so we only present the main points of it.

The base case is the process *stop*, for which the proof is immediate, because the only possible behaviour of the associated graph is the passage of time. For $a < t_1, t_2 >; P$, $\tau; P$, and $wait(t); P$ the reasoning is very similar, so we only mention the timed prefix, and then we see the main points of the proof for the remaining operators:

- $P = a < t_1, t_2 >; P_1$

  Let n be the number of rules that we are applying to reach P'. If $n = 1$ we have that either $P \xrightarrow{a} P' = P_1$, or $P \underset{t}{\longmapsto} P' = a < t_1 \dot{-} t, t_2 - t >; P_1$. It is straightforward to check that both rules can be simulated according to the graph semantics, and even that for a delay t greater than t_2 the graph semantics only allows the passage of time.

  If $n > 1$ we may distinguish two cases according to the initial rule that we are applying:

  1. $P \xrightarrow{a} P_1 \stackrel{s_1}{\Longrightarrow}_t P'$
     In this case, by the induction hypothesis for P_1 we have that there is a node $n_{\bar{P}'}$ in $G[\![P_1]\!]$ such that $P' \in time\_evol(n_{\bar{P}'})$, and there is a reachable state $(n_{\bar{P}'}, C, t)$ in $G[\![P_1]\!]$, by the sequence of actions s_1. Then, taking now $G[\![P]\!]$, once the action a has been executed we are at the initial node of $G[\![P_1]\!]$, and the timer values are still 0, so we may execute the same sequence of transitions to reach the state $(n_{\bar{P}'}, C, t)$.
  2. $P \underset{t_1'}{\longmapsto} a < t_1 \dot{-} t_1', t_2 - t_1' >; P_1 \stackrel{s_1}{\Longrightarrow}_t P'$
     We may assume without loss of generality that the first rule in s_1 is an action rule (the execution of a). Then, we need to check that $G[\![P]\!]$ allows the execution of that action at a time $t_1 \in [t_1, t_2]$, which is immediate, according to the graph semantics. Once executed this action, we reach the initial node of $G[\![P_1]\!]$, but now the all the timer values are t_1. In consequence, by applying the induction hypothesis for P_1 and Lemma 1, we may conclude easily the property.

- $P = P_1 \square P_2$

  The *Unfold* function does not affect to the process behaviour, because it only unfolds an outer recursion, and thus, from the point of view of the operational semantics we may evolve in the same way. Then, the application of the induction hypothesis for $Unfold(P_1)$ and $Unfold(P_2)^2$ allows us to conclude the property, because each

[2] The presence of *Unfold* does not affect to the application of the induction hypothesis, because it is purely syntactic.

sequence of transitions that we can execute in the new graph must be an executable sequence of transitions in one of the graphs $G[\![\mathit{Unfold}(P_1)]\!]$, $G[\![\mathit{Unfold}(P_2)]\!]$, with the only difference that the new graph always updates the global clock. If the global clock of the new graph is the global clock of the graph from which the sequence of transitions is taken there is no problem, because it was already updated on each transition of it, and otherwise all we are doing is updating a new clock on each transition.

Let us observe that also rule R7 is well captured by the graph semantics, because $P_1 \,\Box\, P_2 \underset{t}{\longmapsto} P_1' \,\Box\, P_2'$ iff $(\mathit{Unfold}(P_1) \underset{t}{\longmapsto} P_1' \wedge \mathit{Unfold}(P_2) \underset{t}{\longmapsto} P_2')$, and then, by the induction hypothesis, $\mathit{Min}_\tau(n_{0i}, C_{0i}) \geq t,\ i = 1, 2$, which occurs if and only if in the new graph we have $\mathit{Min}_\tau(n_0, C_0) \geq t$.

- $P = P_1 \,\|_A\, P_2$

  We may split the sequence of transitions into two subsequences, each one corresponding to the separate evolution of each component, the intersection being the parts that both execute simultaneously (the actions in A and the passage of time).

  Then, we have $P_i \overset{s_i}{\Longrightarrow}_t P_i'$, $i = 1, 2$. By the induction hypothesis for P_i we have a reachable state $(n_{\bar{P}_i'}, C_i, t)$ on $G[\![P_i]\!]$, with $P_i' \in \mathit{time\_evol}(\bar{P}_i')$, corresponding to the sequence s_i. Therefore, as the clocks of both graphs are disjoint, the complete sequence of transitions s can be also executed in $G[\![P]\!]$, with the only difference that now we are updating a new clock on each transition, and transitions with actions in A update the corresponding clocks from both graphs. Let us also note that the introduction of the new global clock does not affect to the evaluation of $\mathit{act}(s, e)$, because the value of this new clock would be the same as the value of the global clock of the involved component (or even of both components if it is a synchronization arc). Then, we will reach a state $(n_{\bar{P}_1' \|_A \bar{P}_2'}, C, t)$ in the new graph, and according to the definition of $\mathit{time\_evol}$ it is easy to see that $P_1' \|_A P_2' \in \mathit{time\_evol}(\bar{P}_1' \|_A \bar{P}_2')$.

- $P = P_1 \backslash a$

  Taking $G[\![P_1]\!]$ we obtain $G[\![P]\!]$ by changing all the actions a in it by τ. By the induction hypothesis and by the rule R13 it is easy to conclude that this change does not affect to any other actions which are executable in P, i.e., they cannot be disallowed in $G[\![P]\!]$ by the urgent character of the new $\tau's$ that appear. Then, if $P_1 \backslash a \overset{s}{\Longrightarrow}_t P' \backslash a$, we have the equivalent sequence in P_1: $P_1 \overset{s_1}{\Longrightarrow}_t P_1'$, and using the induction hypothesis we obtain that there is a reachable state in $G[\![P_1]\!]$, $(n_{\bar{P}_1'}, C, t)$, such that $P_1' \in \mathit{time\_evol}(\bar{P}_1')$, and the same sequence of transitions (where a is now τ) can be also executed in $G[\![P]\!]$.

- $P = \mu X.P_1$

  Note that according to the construction the initial node of $G[\![P]\!]$ becomes labelled with the term $P_1\{\mu X.P_1/X\}$, which is operationally equivalent to P.

  As P is regular, we may split s into some subsequences, in the following way:

$$P \overset{s_1}{\Longrightarrow}_{t_1} \bar{P}_1 \overset{s_2}{\Longrightarrow}_{t_2} \ldots \overset{s_n}{\Longrightarrow}_{t_n} \bar{P}_n = P'$$

  where $n \geq 1$, $t = \sum_{i=1}^{n} t_i$, and $\bar{P}_i = P$, $i = 1, \ldots, n-1$. Taking $G[\![P_1]\!]$ we have by the induction hypothesis that every s_i $(i < n)$ can be executed starting from the initial node, and leads us to a node labelled with X in $G[\![P_1]\!]$; s_n can be executed too, and it leads to a node $n_{\bar{P}'}$ of $G[\![P_1]\!]$ (which could be also X). According to the construction of $G[\![P]\!]$, the final edge of each one of these sequences s_i in $G[\![P_1]\!]$ $(i < n)$ has been changed to point at the initial node, and all the clocks are synchronized. Therefore, by Lemma 1 we may conclude that $s = s_1 \dots s_n$ can be executed in $G[\![P]\!]$, and leads us in time t either to the node $n_{\bar{P}'}$ or to the initial one if it was X too.

$\square$

Let us observe that the converse is not true in general, because not every $P' \in \mathit{time\_evol}(\bar{P}')$ can be reached by using the operational semantics. Instead, we have the following result for the converse.

Theorem 2 Given a closed regular processes $P \in \mathit{RTPA}$, if $(n_{\bar{P}'}, C, t)$ is a reachable state in $G[\![P]\!]$, then there is a closed regular process $P' \in \mathit{RTPA}$ such that $P \stackrel{s}{\Longrightarrow}_t P'$, where $P' \in \mathit{time\_evol}(\bar{P}')$, and s is the sequence of transitions executed in $G[\![P]\!]$ to reach that state.

Proof: Again by structural induction on P, it is very similar to the previous one:

- stop: Immediate.
- $a < t_1, t_2 >; P_1$:

  If $n_{\bar{P}'}$ is the initial node, we have that $(n_{\bar{P}'}, C, t)$ must have been obtained by the passage of time, and it is immediate to check the property in this case. Otherwise it must be a node of $G[\![P_1]\!]$, and then we may apply the induction hypothesis for the state $(n_{\bar{P}'}, C', t - t')$, where t' is the instant at which we execute the initial action a, and $C'(R_j) = C(R_j) - t'$, $\forall R_j \in \mathit{Clocks}_1$. Thus we obtain that there is a P' such that $P_1 \stackrel{s_1}{\Longrightarrow}_{t-t'} P'$, and then we may conclude that $P \stackrel{s}{\Longrightarrow}_t P'$.
- $\tau; P_1$ and $\mathit{wait}(t); P_1$:

  They are similar to the previous one, although in this case the initial passage of time is limited by the value of $\mathit{Min}_\tau(n_0, C_0)$ (0 and t respectively).
- $P = P_1 \,\square\, P_2$:

  According to the construction, if $n_{\bar{P}'}$ is again the initial node, we must have that the considered state must have been obtained by the passage of time. In this case both graphs $G[\![\mathit{Unfold}(P_1)]\!]$ and $G[\![\mathit{Unfold}(P_2)]\!]$ must enable the passage of t units of time at their initial nodes, and thus by using the induction hypothesis we obtain $P_i \underset{t}{\longmapsto} P_i'$, $i = 1, 2$, which implies $P_1 \,\square\, P_2 \underset{t}{\longmapsto} P_1' \,\square\, P_2'$, and also $P_1' \,\square\, P_2' \in \mathit{time\_evol}(\bar{P'}_1 \,\square\, \bar{P}_2')$.

  Otherwise, that node must belong to one of the graphs $G[\![P_i]\!]$, $i = 1, 2$. Taking the sequence of transitions leading us to that state, we may also execute it in the corresponding graph, and thus we may apply the induction hypothesis to get the process P'.

- $P = P_1 \parallel_A P_2$:

  We may split s into two subsequences s_1, s_2, taking the transitions executed by each component, as we did in Theorem 1. It is easy to check that both sequences can be executed by their corresponding graphs separately, and thus the induction hypothesis can be applied to get the processes P_i', such that $P_i \stackrel{s_i}{\Longrightarrow}_t P_i'$, with $P_i' \in \mathit{time\_evol}(\bar{P}_i')$. By the operational semantics we can obtain $P_1 \parallel_A P_2 \stackrel{s}{\Longrightarrow}_t P_1' \parallel_A P_2'$, and from the definition of *time_evol* we have that $P_1' \parallel_A P_2' \in \mathit{time\_evol}(\bar{P}_1' \parallel_A \bar{P}_2')$.

- $P = P_1 \backslash a$:

  If we take the sequence of transitions leading us to the state $(n_{\bar{P}' \backslash a}, C, t)$ in $G[\![P]\!]$, we have that the corresponding sequence s' in $G[\![P_1]\!]$ can also be executed, and thus we can apply the induction hypothesis to get a process P' such that $P_1 \stackrel{s'}{\Longrightarrow}_t P'$, and $P' \in \mathit{time\_evol}(\bar{P}')$. Then, by the operational semantics we can obtain that $P_1 \backslash a \stackrel{s}{\Longrightarrow} P' \backslash a$. The only point needing somme comments here is the aplication of rule R13, which must hold when delays are required along that sequence of rules. However, if an action a disables a required delay at a certain point, we can conclude that there is a possible sequence of transitions in $G[\![P_1]\!]$ including the execution of that action at an earlier instant than the action following the delay in the given sequence, and thus, we get a contradiction with the graph semantics in $G[\![P]\!]$.

- $P = \mu X.P_1$:

  The sequence of transitions s can be divided into some subsequences, in the same way as we did in Th. 1, $s = s_1 \ldots s_n$, where each s_i $(i < n)$ ends with an arc (n, n_0) which comes from an arc (n, X) in $G[\![P_1]\!]$.

  Then, by Lemma 1 we have that each s_i can be executed in $G[\![P_1]\!]$ separately, taking a time t_i, with $t = \sum_{i=1}^{n}$. Therefore, by applying the induction hypothesis we can obtain:

  $$P_1\{\mu X.P_1/X\} \stackrel{s_i}{\Longrightarrow}_{t_i} P_1\{\mu X.P_1/X\} \;, \quad \text{for } i = 1, \ldots, n-1$$

  and $P_1 \stackrel{s_n}{\Longrightarrow}_{t_n} P'$, with $P' \in \mathit{time\_evol}(\bar{P}')$, where $n_{\bar{P}'}$ could be also the initial node. Then, it is now immediate that $P \stackrel{s}{\Longrightarrow}_t P'$.

□

4 Example: The Producer/Consumer System

We consider the PC problem, with a buffer with capacity 2. The specification of this system consists of three processes in parallel:

System=(((Prod $\parallel_{\{\}}$Cons)$\parallel_{\{it,et\}}$Buffer)$\backslash$ it)$\backslash$ et

where:
Prod = pt$< 5, 10 >$;it$< 0, \infty >$;Prod
Cons = et$< 0, \infty >$;ct$< 12, 15 >$;Cons
Buffer = it$< 0, \infty >$;(it$< 0, \infty >$;Buffer1 $\Box$ et$< 0, \infty >$;Buffer) , where *Buffer1* is:
Buffer1 = et$< 0, \infty >$;(it$< 0, \infty >$;Buffer1 $\Box$ et$< 0, \infty >$;Buffer)

The meaning of the actions is the following:

pt:	Produce an item
ct:	Consume an item
it:	Insert an item into the buffer
et:	Extract an item from the buffer

By applying the graph construction we get the graphs in Fig. 1 for these processes, and for the complete system we get the graph shown in Fig. 2.

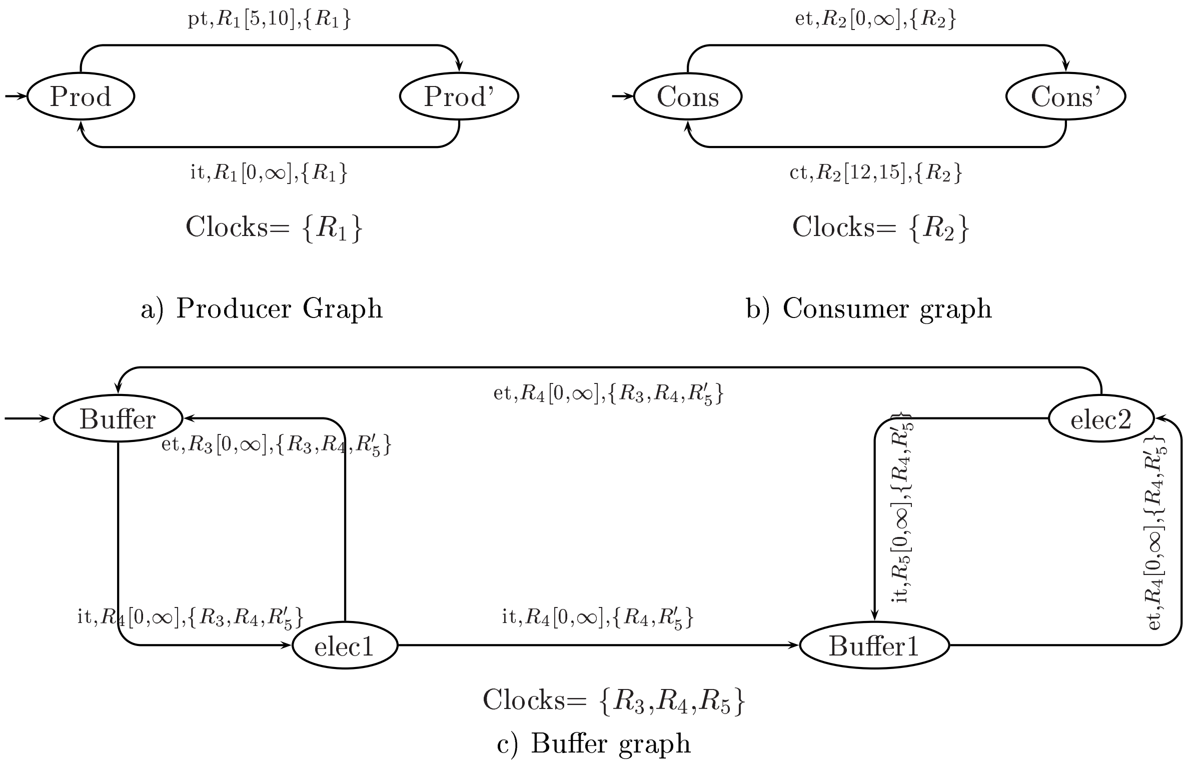
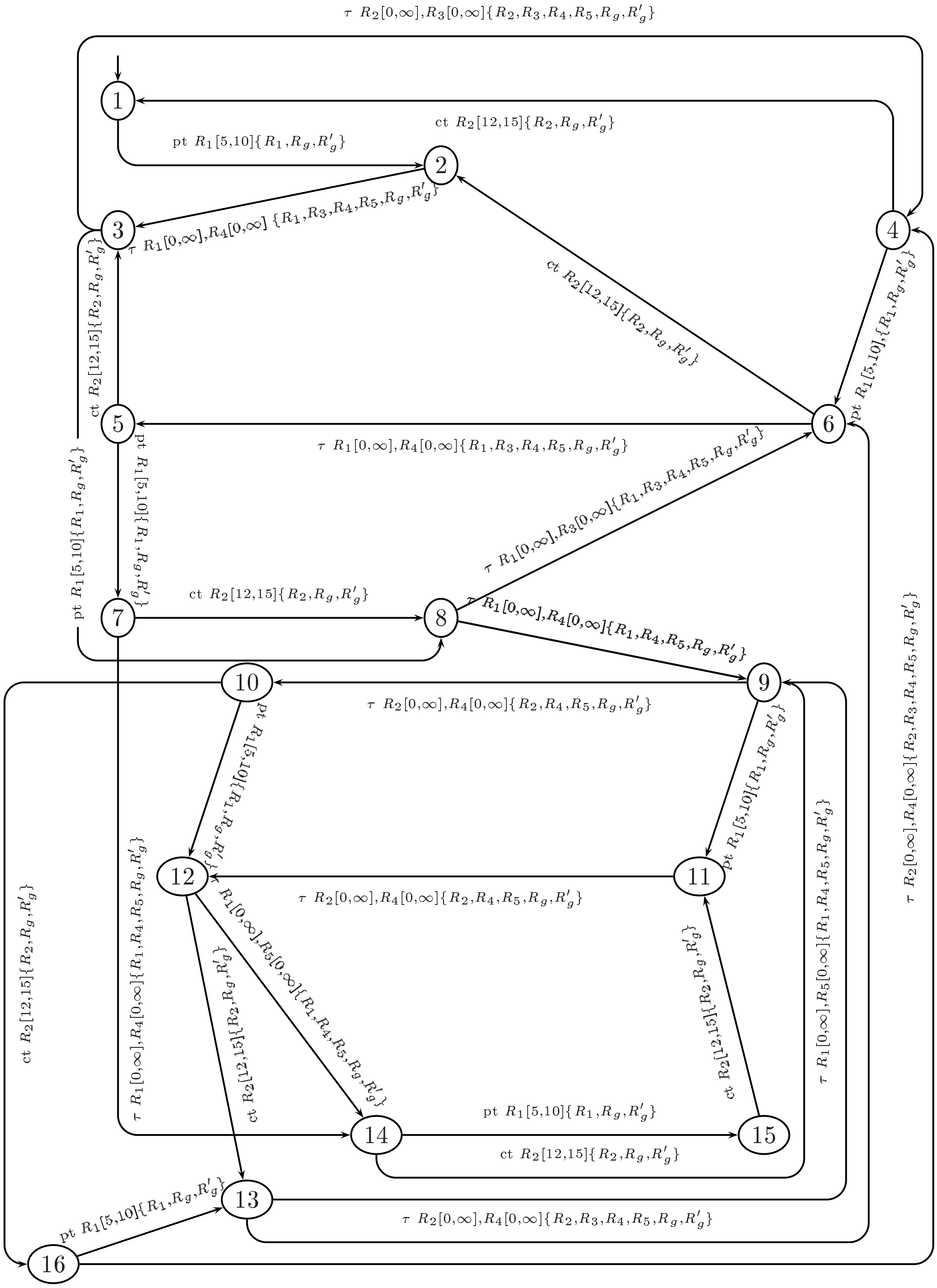


Figure 1: Dynamic state graphs for the producer, consumer and the buffer

Somme immediate properties can be concluded from that graph, for instance the system is deadlock free, considering that no time-out expires, because there are no terminal nodes. We can also compute some rough bounds for the times to reach the graph nodes, by the following method:

Let s be a sequence of transitions in $G[\![P]\!]$. We compute both a lower bound, $Min(s, R_j)$, and an upper one, $Max(s, R_j)$, for the timer R_j when reaching the final node of s, as follows:

- $Min(\epsilon, R_j) = 0,\ Max(\epsilon, R_j) = 0, \forall R_j \in Clocks$, where ϵ is the empty sequence.
- Given $Min(s, R_j)$ and $Max(s, R_j)$, we can compute $Min(s', R_j)$ and $Max(s', R_j)$, for $s' = s \cdot e$, with $\lambda(e) = (a, r, S)$, as follows:



Clocks= $\{R_1,R_2,R_3,R_4,R_5,R_g,R'_g\}$

Figure 2: Graph for the PC problem

Firstly, we compute a lower bound for the value of $Min(n, C, e)$ and an upper one for $Max(n, C, e)$, where n is the final node of s:

- $\alpha = max\{Min(s, R_j) + \alpha_j \dot{-} Max(s, R_g) \mid r(R_j) \text{ defined }, r(R_j) = [\alpha_j, \beta_j]\}$
- $\beta = min\{Max(s, R_j) + \beta_j - Min(s, R_g) \mid r(R_j) \text{ defined }, r(R_j) = [\alpha_j, \beta_j]\}$

The same procedure can be applied to compute an upper bound for $Min_\tau(n, C)$:

- $\Delta_\tau = \infty$, if $\nexists e'$ s.t. $n \xrightarrow{e}$, $\lambda_1(e') = \tau$, or
- $\Delta_\tau = \min\limits_{e} \{ \max\limits_{j} \{Max(s, R_j) + \alpha_j \dot{-} Min(s, R_g) \mid r(R_j) \text{ defined },$
  $r(R_j) = [\alpha_j, \beta_j]\} \mid n \xrightarrow{e}, \lambda_1(e) = \tau\}$, otherwise

Finally:

- $Min(s', R_j) = \begin{cases} Min(s, R_j) & \text{if } R_j \notin S \\ Min(s, R_g) + \alpha & \text{if } R_j \in S \end{cases}$

- $Max(s', R_j) = \begin{cases} Max(s, R_j) & \text{if } R_j \notin S \\ Max(s, R_g) + min\{\beta, \Delta_\tau\} & \text{if } R_j \in S \end{cases}$

For instance, we obtain $Min(1.2.3.8, R_g) = 5$, $Max(1.2.3.8, R_g) = 25$ for the PC problem, which are not very good bounds, because we know that 10 units of time must pass when the producer has generated two items, which can be computed by hand on the graph too.

5 Conclusions

In this technical report we have defined a Timed Probabilistic Process Algebra, which is based on basic LOTOS, by adding some new operators for the specification of timed and probabilistic behaviours, and urgent interactions. We have defined an operational semantics for the language, but, in order to verify some properties of a model it is more convenient to have a graph representation, so the main goal of this paper has been to define such a representation for a certain kind of terms, concretely, for regular terms, for which we obtain a finite graph. We have proved that both semantics are equivalent, in the sense that every sequence of transitions that we may execute according to the operational semantics can be also executed in the corresponding graph, and conversely. Although we have made this contruction only for timed processes, it can be easily extended to probabilistic ones, just by extending the kind of graphs that we consider.

Our future work will focus on the analysis of the obtained graphs, trying to obtain better time bounds for the clocks when reaching the graph nodes. We will also define some reduction algorithms for the generated graphs, by joining nodes and eliminating the superfluous ones.

References

[1] R. Alur and D. L. Dill, *Automata-theoretic verification of real-time systems.*, Formal Methods for Real-Time Computing (Constance Heitmeyer and Dino Mandrioli, eds.), Wiley, 1996, pp. 55–82.

[2] T. Bolognesi and F. Lucidi, *Timed process algebras with urgent interactions and a unique powerful binary operator*, Proc. of REX Workshop (Real-Time: Theory and Practice) (1991).

[3] R. Cleaveland, I. Lee, P. Lewis, and S.A. Smolka, *A theory of testing for soft real-time processes*, Proc. 18th Intern. Conference on Software Engineering and Knowledge Engineering (1996).

[4] C. Gregorio and M. Nú nez, *Specifying and verifying the alternating bit protocol with probabilistic-timed lotos*, Proc. of COST 247 International Workshop on Applied Formal Methods in System Design (1996), 38–50.

[5] Jan Friso Groote, *Transition system specifications with negative premises*, Theoretical Computer Science **118** (1993), 263–299.

[6] H. Hansson, *Modeling timeouts and unreliable media with a timed probabilistic calculus*, Formal Description Techniques IV (1992).

[7] H. Hansson and B. Jonsson, *A calculus for communicating systems with time and probabilities*, Proceedings Real-Time Systems Symposium (1990).

[8] F. Jahanian and Stuart D.A., *A method for verifying properties of modechart specifications*, IEEE Real-Time Systems Symposium, 1988.

[9] Inhye Kang and Lee Insup, *An efficient generation of the timed reachability graph for the analysis of real-time systems*, 1993.

[10] A. K. Mok, D. A. Stuart, and F. Jahanian., *Specification and analysis of real-time systems: Modechart language and toolset.*, Formal Methods for Real-Time Computing (Constance Heitmeyer and Dino Mandrioli, eds.), Wiley, 1996, pp. 33–54.

[11] F. Moller and C. Tofts, *A temporal calculus of communicating systems*, Proceedings CONCUR 90, Lecture Notes in Computer Science (1990), no. 458, 401–415.

[12] Xabier Nicollin, Joseph Sifakis, and Sergio Yovine, *From atp to timed graphs and hybrid systems*, Acta informatica (1993), no. 30, 181–202.

[13] Yolanda Ortega, *En busca del tiempo perdido (in spanish)*, PhD. Thesis, Univ. Complutense de Madrid, Dpto. Informática y Automática UCM, 1990.

[14] J.S. Ostroff, *Deciding properties of timed transition models*, IEEE Transactions on Parallel and Distributed Systems, 1(2), April 1990.

[15] J. Quemada, A. Azcorra, and D. de Frutos, *A timed calculus for lotos2*, Proc. FORTE 89 (1989).

[16] S.C.V. Raju, *An automatic verification technique for communicating real-time state machines*, Tech. Report 93-04-08, Universidad de Washington, April 1993.

[17] G. M. Reed and A. W. Roscoe, *A timed model for communicating sequential processes*, Theoretical Computer Science (1988), no. 58, 249–261.

[18] A.C. Sahw, *Communicating real-time state machines*, IEEE Transactions on Software Engineering, 18(9), September 1992.

[19] S. Schneider, *An operational semantics for timed csp*, Information and Computation (1995), no. 116, 193–213.