Automatic Translation of a Timed Process Algebra into Dynamic State Graphs *

J. José Pardo, Valentín Valero, Fernando Cuartero, Diego Cazorla
Departamento de Informática
Escuela Politécnica de Albacete
Univ. de Castilla-La Mancha
Campus Univ. s/n. 02071. Albacete, Spain.
ph: +34 967 599200
fax +34 967 599224
{jpardo, valentin, fernando, dcazorla}@info-ab.uclm.es

Abstract

In this paper we consider a Timed Process Algebra on the basis of the classical LOTOS operators, in order to specify the behaviour of concurrent systems and, concretely, those systems for which time becomes an important factor to be taken into account, such as real-time systems. One of the main goals of this paper is to define a translation into a kind of dynamic state graph, which is currently supported by a tool (TPAL), which allows us to simulate the execution of a specification by means of these dynamic state graphs.

1 Introduction

Process algebras have been introduced to specify the behaviour of concurrent systems, due to their simplicity and conciseness. In this way, a system is described by using a finite set of operators, which makes the specifications more readable, simpler and easier to modify than other classical notations, such as automata.

We can, however, find a wide range of systems for which time becomes a very important factor to consider in the specifications (real-time systems, fault-tolerant systems, etc.). Some common characteristics of these models can be identified: first of all, we have to associate time to actions, but we may also find some critical actions, which must be executed urgently, once they become enabled.

Thus, in the last few years several extensions of the classical models have been defined in order to incorporate these factors: we have timed extensions of CSP [15, 17], CCS [9, 18], and LOTOS [13], and [2] where the authors include an operator for the specification of urgent interactions. There are some other proposals, which combine both factors, time and probabilities; concretely [4] is an extension of CCS.

One of the drawbacks of timed process algebras is that it is not at all easy to form a complete picture of the possible states into which a system may evolve, a translation into a state graph would therefore be very useful. Furthermore, with this translation we can analyse some interesting properties of the system, like timed reachability of states, deadlock freeness, and so on.

The model we consider (TPA) has been defined on the basis of the classical operators of basic LOTOS. We have maintained the external choice, the parallel composition and hiding and recursion operators, at the same time adding some new operators for the specification of timed behaviours, as well as urgent interactions. Specifically, following up the ideas of Quemada, Azcorra and de Frutos [13], we have introduced a timed prefix operator, $a < t_1, t_2 >; P$, which establishes a time interval for the execution of the corresponding action, but this action can never be executed once its deadline has expired. Other operators we have included are: an internal action prefix, $\tau; P$, which establishes the urgent character of the τ action, and a delay operator, $wait(t); P$, which establishes a delay of t units of time for the process, and after that it behaves like $\tau; P$.

We define an operational semantics for the language, which captures the aforementioned behaviour and particularly the urgent character of the internal action. We also define a characterisation of TPA processes by means of a kind of dynamic state graph, which allow us to simulate the execution of TPA processes, as well as analysing their behaviour. Our tool (TPAL) currently includes both a syntax and semantics checker, graph generation and graph visuali-

*This work has been supported by the CICYT project "Formal Development of System Based on Mobile Agents and Performance Evaluation", TIC2000-0701-c02-02.

zation. Also, by using these dynamic state graphs we can make simulations of the specification, which can be either manual or automatic and in this latter case we can even indicate whether time-locks should be avoided or not.

With respect to the graph model used, it bears certain similarities to that presented in [10], where the semantics is defined by means of a set of timers and a set of timer restrictions labelling each edge. One difference between the model there presented and ours is the increasing character of the timers in our model[1], which are synchronised with a global clock instead of resetting then. The relationship between process algebras and timed graphs has been studied in several previous works . Modecharts [5, 8] and timed automata [1, 6, 7] are very well known classical tools for describing concurrent systems. Also, in [16] Communicating Real-Time Machines (CRSM) are defined to model real-time systems. In this particular model, a system is described by means of a collection of these machines interconnected by one-to-one communication channels. [14] provides a method to transform these machines into graphs whose nodes consist 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 is TTM (*Timed Transition Models*) [11], in which time is modelled by means of an external global clock, and a set of variables capture the local time information of each process. Thus, each transition consists of an enabling condition, a transformation function, and time bounds for its execution.

The paper is structured as follows. In Section 2 we present the syntax and semantics of the language, while the translation into dynamic state graphs is presented in Section 3, also showing the equivalence between both semantics. In Section 4, we present an algorithm to compute bounds for the time to reach each graph node and finally, in Section 5, we briefly describe TPAL, a tool that we are implementing to support this translation and the analysis of the generated graphs.

2 Syntax and Semantics of TPAL

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:

$$P ::= stop \mid a < t_1, t_2 >; P \mid \tau; P \mid wait(t); P \mid P \,\Box\, P \mid P \parallel_A P \mid P \backslash\, a \mid X \mid \mu X.P$$

where $A \subseteq Vis$, $X \in Idf$ (set of identifiers), $a \in Vis$, $t, t_1 \in \mathbb{R}_0^+ = \mathbb{R}^+ \cup \{0\}$ and $t_2 \in \mathbb{R}_0^+ \cup \{\infty\}$.

[1]Thus, we have preferred to rename them. They will be referred to as "clocks" instead of "timers".

The informal interpretation of the operators follows:

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

$a < t_1, t_2 >; P$.- 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. As usual, once a has been executed, the process behaves as P.

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

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

$P_1 \,\Box\, P_2$.- This is essentially the classical external choice operator.

$P_1 \parallel_A P_2$.- This represents the parallel execution of P_1 and P_2 synchronising on the actions in A.

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

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

$\mu X.P$.- This is the classical recursion operator.

Terms obtained from this syntax will be denoted by TPA. However, in order to define the corresponding dynamic state graph for any term we will need to restrict our attention 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 Plotkin and Milner's style (see table 1). We have two types of transition rules.

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

$P \xrightarrow{a} P'$.- The process P executes the action a and then behaves as P' (no time elapses).

We will also use the following notation, $P \xrightarrow{a}$, to represent that there is at least one process Q such that P may evolve into Q by executing the action a, i.e. $P \xrightarrow{a} Q$.

With rule R1 we capture that for the process stop the only possible evolution is the passage of time. Rules R2a-c capture the semantics of the timed prefix. Concretely, rule

R1) $\mathrm{stop} \underset{t}{\longmapsto} \mathrm{stop} \quad \forall t \in \mathbb{R}_0^+$

R2a) $a<0,t>;P \overset{a}{\longrightarrow} P \quad \forall t \in \mathbb{R}_0^+$

R2b) $a<t_1,t_2>;P \underset{t'}{\longmapsto} a<t_1 \dot{-} t', t_2 - t'>;P \quad 0 \le t' \le t_2$, where $x \dot{-} y = \max\{0, x-y\}$

R2c) $a<0,0>;P \underset{t'}{\longmapsto} \mathrm{stop} \quad t'>0$

R3) $\tau;P \overset{\tau}{\longrightarrow} P$

R4a) $\mathrm{wait}(t);P \underset{t'}{\longmapsto} \mathrm{wait}(t-t');P\,,\ 0 \le t' \le t$

R4b) $\mathrm{wait}(0);P \overset{\tau}{\longrightarrow} P$

R5a) $\dfrac{P_1 \overset{e}{\longrightarrow} P_1'}{P_1 \Box P_2 \overset{e}{\longrightarrow} P_1'}$ **R5b)** $\dfrac{P_2 \overset{e}{\longrightarrow} P_2'}{P_1 \Box P_2 \overset{e}{\longrightarrow} P_2'}$

R6) $\dfrac{P_1 \underset{t_1}{\longmapsto} P_2 \;\; P_2 \underset{t_2}{\longmapsto} P_3}{P_1 \underset{t_1+t_2}{\longmapsto} P_3}$ **R7)** $\dfrac{P_1 \underset{t'}{\longmapsto} P_1' \;\; P_2 \underset{t'}{\longmapsto} P_2'}{P_1 \Box P_2 \underset{t'}{\longmapsto} P_1' \Box P_2'}$

R8a) $\dfrac{P_1 \overset{e}{\longrightarrow} P_1' \quad e \notin A}{P_1 \parallel_A P_2 \overset{e}{\longrightarrow} P_1' \parallel_A P_2}$ **R8b)** $\dfrac{P_2 \overset{e}{\longrightarrow} P_2' \quad e \notin A}{P_1 \parallel_A P_2 \overset{e}{\longrightarrow} P_1 \parallel_A P_2'}$

R8c) $\dfrac{P_1 \overset{a}{\longrightarrow} P_1',\ \ P_2 \overset{a}{\longrightarrow} P_2' \ \ a \in A}{P_1 \parallel_A P_2 \overset{a}{\longrightarrow} P_1' \parallel_A P_2'}$ **R8d)** $\dfrac{P_1 \underset{t'}{\longmapsto} P_1',\ \ P_2 \underset{t'}{\longmapsto} P_2'}{P_1 \parallel_A P_2 \underset{t'}{\longmapsto} P_1' \parallel_A P_2'}$

R9a) $\dfrac{P \overset{e}{\longrightarrow} P' \ \ e \in Act \ \ e \neq a}{P \backslash a \overset{e}{\longrightarrow} P' \backslash a}$ **R9b)** $\dfrac{P \overset{a}{\longrightarrow} P'}{P \backslash a \overset{\tau}{\longrightarrow} P' \backslash a}$

R9c) $\dfrac{P \underset{t}{\longmapsto} P',\ \neg P \overset{a}{\longrightarrow} \wedge \neg[\exists\, t' < t,\ P \underset{t'}{\longmapsto} P''\,,\ P'' \overset{a}{\longrightarrow}]}{P \backslash a \underset{t}{\longmapsto} P' \backslash a}$

R10a) $\dfrac{P\{\mu x.P/x\} \overset{e}{\longrightarrow} P'}{\mu x.P \overset{e}{\longrightarrow} P'}$ **R10b)** $\dfrac{P\{\mu x.P/x\} \underset{t}{\longmapsto} P'}{\mu x.P \underset{t}{\longmapsto} P'}$

Table 1. Operational Semantics

R2c states that once the time to execute an action has expired, we lose the ability to execute that action. Rules R3 and R4a-b capture the urgent character of internal actions, and the semantics of the wait operator. Rules R5a-b give the classical semantics for the external choice, while R7 states that the time can pass in an external choice only when both components are able to do so. Rules R8a-c are the classical ones for the parallel operator, while R8d is similar to R7. Semantics of the hiding operator is captured by rules R9a-c, the latter one capturing the fact that once an action has been hidden, it becomes urgent [2]. Finally rules R10a-b capture the semantics for the recursion operator.

Definition 1 Let $P,\ P' \in TPA,\ s \in Act^*,\ t \in \mathbb{R}_0^+$. We say that $P \overset{s}{\Longrightarrow}_t P'$ if and only if there is a sequence of transitions from P to P' :

$$P \ \cdots \ P_i \overset{a_i}{\longrightarrow} P_{i+1} \ \cdots \ P_j \underset{t_j}{\longmapsto} P_{j+1} \ \cdots \ P' \text{ such}$$

that $s = a_1 \cdots a_n,\ t = \displaystyle\sum_{j=1}^{m} t_j$. □

3 Dynamic State Graphs

In this section we shall see how to construct a dynamic state graph capturing the semantics of each TPA process.

[2] R13 introduces negative premises, however, by using the same procedure as in [3] consistency of the transition system can be proved.

These dynamic state graphs are essentially automata extended with clocks, which capture the local time information of each process. We may find in the literature a related work [10], 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 captures the time evolution for the whole system, and local clocks are not reset when local actions are executed. Instead, local clocks synchronise with the global clock when necessary, and we thus keep the local time information for each component (i.e., the time spent on it) since the clocks are defined in such a way that they are associated to each component.

Then, nodes of these graphs are labelled by terms, and the edges are labelled by tuples (a, r, S) where a is an action, r is a set of time intervals that establish restrictions for some clocks, and S is a set of clocks that must synchronise with the global clock when this transition is executed.

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 clocks, where one of them is the global one (it will be henceforth referred to as R_g), and $\lambda : E \longrightarrow Act \times Times \times \mathcal{P}_F(\mathit{Clocks})$, where $Times : \mathit{Clocks} \hookrightarrow \mathbb{R}_0^+ \times \mathbb{R}_0^+ \cup \{\infty\}$ is a partial function that associates a time interval to some of the clocks, and $\mathcal{P}_F(\mathit{Clocks})$ denotes the finite power set of *Clocks*. We will denote each component of $\lambda(e)$ by $\lambda_i(e)$, $i = 1, 2, 3$. $\Box$

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 $n \in N$, $C : \mathit{Clocks} \longrightarrow \mathbb{R}_0^+$ (clock 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 edge e leaving the node n with label: $\lambda(e) = (a, r, S)$, we define:

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

which represents the minimum delay until this edge can be executed, with respect to the instant at which we reached the node *n*.

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

which represents the maximum delay to execute this edge with respect to the instant at which we reached the node *n*. A negative value means that the time to execute that action has already expired, and it cannot therefore be executed.

We also define

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

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, it follows that the action a can be executed at the state $s = (n, C, t)$ if and only if we have an edge 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 $act(s, e)$.

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

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

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

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

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

Note that according to these rules the dynamic evolution of a graph does not affect the clock restrictions labelling the edges, and furthermore, 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 $C_0'(R_j) = t'$, $\forall R_j \in \mathit{Clocks}$.

It follows 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 $C'(R_j) = C(R_j) + t'$, $\forall R_j \in \mathit{Clocks}$. $\Box$

3.1 Dynamic state graph for a 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 : $G[\![stop]\!] = (\{stop\}, \emptyset, \emptyset, stop, \{R_g\})$, where R_g is the global clock.
- 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 $N = N_1 \cup \{P\}$, $E = E_1 \cup \{(P, n_{01})\}$, and
$\lambda(\bar{e}) = \begin{cases} \lambda_1(\bar{e}) & \forall \bar{e} \in E_1 \\ (a, \mathit{Intg}, \mathit{Clocks}_1) & \text{if } \bar{e} = e \end{cases}$
where *Intg* is defined only for R_g, being $\mathit{Intg}(R_g) = [t_1, t_2]$.
- Internal action prefix and Wait operators : These are defined as the timed prefix, but the action labelling the new edge e is now τ in both cases, and $\mathit{Intg}(R_g) = [0, 0]$ for the internal action, and $\mathit{Intg}(R_g) = [t, t]$ for $wait(t); P$.
- External choice : Let $P = P_1 \Box P_2$ be a process.
Then $G[\![P]\!] = G[\![\mathit{Unfold}(P_1)]\!] \Box_G G[\![\mathit{Unfold}(P_2)]\!]$, where $\Box_G$ is a graph operation defined below that requires both components to have an initial state without incoming edges, thus, in order to define the general behaviour of the external choice we need to *unfold* the outer recursion that could appear on the ex-

ternal choice components. This function is defined as follows:

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

Note that this definition is correct, because we are considering regular terms. Now, given two graphs, $G_i = (N_i, E_i, \lambda_i, n_{0i}, \mathit{Clocks}_i)$, for $i = 1, 2$, such that their initial states do not have incoming edges, we define $G_1 \Box_G G_2 = (N, E, \lambda, n_0, \mathit{Clocks}_1 \cup \mathit{Clocks}_2)$ as follows:

$$
\begin{array}{ll}
N = & (N_1 \cup N_2 \cup \{P\}) \setminus \{n_{01}, n_{02}\} \\
E = & (E_1 \cup E_2) \backslash \\
 & (\{(n_{01}, n_1) \mid (n_{01}, n_1) \in E_1\} \cup \\
 & \{(n_{02}, n_2) \mid (n_{02}, n_2) \in E_2\})) \cup \\
 & (\{(n_0, n_1) \mid (n_{01}, n_1) \in E_1\} \cup \\
 & \{(n_0, n_2) \mid (n_{02}, n_2) \in E_2\}) \\
\multicolumn{2}{l}{\lambda(\bar{e}) = (\bar{\lambda_1}(\bar{e}),\ \bar{\lambda_2}(\bar{e}),\ \bar{\lambda_3}(\bar{e}) \cup \{R_g\})\ \ \forall \bar{e} \in E}
\end{array}
$$

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 $G[\![P_i]\!] = (N_i, E_i, \lambda_i, n_{0i}, \mathit{Clocks}_i)$. We then define the graph for the process $P = P_1 \|_A P_2$ as follows:

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

where $N = N_1 \times N_2$, $n_0 = (n_{01}, n_{02})$, $E = E_{11} \cup E_{12} \cup E_{13}$, being:

$$
\begin{array}{rl}
E_{11} = & \{((n_1, n_2), (n_1', n_2)) \mid n_1 \xrightarrow{e_1}_1 n_1',\ \lambda_{11}(e_1) \notin A\} \\
E_{12} = & \{((n_1, n_2), (n_1, n_2')) \mid n_2 \xrightarrow{e_2}_2 n_2',\ \lambda_{21}(e_2) \notin A\} \\
E_{13} = & \{((n_1, n_2), (n_1', n_2')) \mid n_1 \xrightarrow{e_1}_1 n_1',\ n_2 \xrightarrow{e_2}_2 n_2', \\
 & \quad \lambda_{11}(e_1) = \lambda_{21}(e_2) \in A\}
\end{array}
$$

$\mathit{Clocks} = \mathit{Clocks}_1 \cup \mathit{Clocks}_2 \cup \{R_g\}$, with $R_g \notin \mathit{Clocks}_1 \cup \mathit{Clocks}_2$, and finally λ is defined as follows:

$$
\lambda(e) = \begin{cases}
(\lambda_{11}(e_1), \lambda_{12}(e_1), \lambda_{13}(e_1) \cup \{R_g\}) \\
\qquad\qquad\qquad\qquad e \in E_{11},\ e_1 \in E_1 \\
(\lambda_{21}(e_2), \lambda_{22}(e_2), \lambda_{23}(e_2) \cup \{R_g\}) \\
\qquad\qquad\qquad\qquad e \in E_{12},\ e_2 \in E_2 \\
(\lambda_{11}(e_1), \lambda_{12}(e_1) \cup \lambda_{22}(e_2), \lambda_{13}(e_1) \cup \\
\ \cup\ \lambda_{23}(e_2) \cup \{R_g\}) \qquad\qquad e \in E_{13}
\end{cases}
$$

Note that with this definition we increase the number of clocks in the graph, since we have a clock for each component of the parallel composition as well as a new clock for the whole system (global clock).

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

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

- X process : $G[\![X]\!] = (\{X\}, \emptyset, \emptyset, X, \{R_g\})$, where R_g is the global clock.

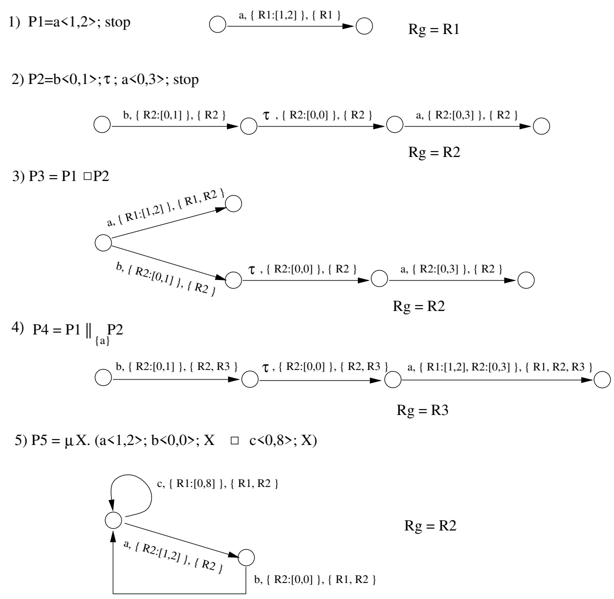
- Recursion : Let $P = \mu X.P_1$ be a process, and $G[\![P_1]\!] = (N_1, E_1, \lambda_1, n_{01}, \mathit{Clocks}_1)$ be the graph associated to P_1. Then, $G[\![P]\!] = (N, E, \lambda, n_{01}, \mathit{Clocks}_1)$, where:

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

$$
\begin{array}{l}
E = (E_1 \cup \{(n, n_{01}) \mid (n, m) \in E_1 \mid \mathit{term}(m) = X\}) \\
\qquad \backslash \{(n, m) \in E_1 \mid \mathit{term}(m) = X\}
\end{array}
$$

$$
\lambda(e) = \begin{cases} \lambda_1(e) & \text{if } e \in E_1 \\ (\lambda_{11}(e), \lambda_{21}(e), \mathit{Clocks}_1) & \text{if } e = (n, n_{01}) \wedge \\ & \quad (n, m) \in E_1,\ \mathit{term}(m) = X \end{cases}
$$

Examples: Let us see some TPA processes, and their corresponding dynamic state graphs:



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

□

3.2 Equivalence

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 first need to capture the reachable states from a given node by time elapsing. However, we must take into account that every sequential component of a process has its own clock, and they are updated separately. Consequently, 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 as follows:

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

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

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

□

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 time elapsing on each sequential component of a process, as follows:

Definition 4 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 *old* is defined as follows:

$$old(a < t_1, t_2 >; P, r) = \begin{cases} stop & \text{if } r > t_2 \\ a < t_1 \dot{-} r, t_2 - r >; P & \text{if } r \le t_2 \end{cases}$$

$$old(wait(t); P, r) = \begin{cases} stop & \text{if } r > t \\ wait(t - r); P & \text{if } r \le t \end{cases}$$

$$old(stop, r) = stop$$

$$old(\tau; P, r) = \begin{cases} stop & \text{if } r > 0 \\ \tau; P & \text{if } r = 0 \end{cases}$$

$$\begin{aligned}
&old(S_1 \Box S_2, r) = old(S_1, r) \Box old(S_2, r) \\
&old(S\|_A, r) = old(S, r)\|_A \\
&old({}_A\|S, r) = {}_A\|old(S, r) \\
&old(S \backslash a, r) = old(S, r) \backslash a
\end{aligned}$$

□

We may now 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. □

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

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

The proof of both theorems is a somewhat involved and it is omitted in this paper (it can be found in [12]).

4 Computing Time-Bounds

Once we have generated the graph associated to any given RTPA process, we can compute some bounds for the times to reach the graphs nodes along a concrete path. We use 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 every clock R_j for 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:

  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 $\nexists e'$, $n \stackrel{e}{\longrightarrow}$, $\lambda_1(e') = \tau$, or
  - $\Delta_\tau = \ \min_e \{ \ \max_j \ \{Max(s, R_j) + \alpha_j \dot{-} Min(s, R_g) \mid r(R_j) \text{ defined}, r(R_j) = [\alpha_j, \beta_j]\} \mid n \stackrel{e}{\longrightarrow}, \ \lambda_1(e) = \tau\}$, otherwise

Finally:

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

It is a straightforward task to check that these computed values are really bounds for the clocks, by using the operational semantics of the graph.

5 TPAL

We have implemented a tool (TPAL) that accepts specifications written in a language very similar to LOTOS according to our extended operators, and which allows us to check both the syntax and semantics correctness, as well as to generate and simulate the corresponding dynamic state graph.

The current version of this tool is only available for a Linux machine at the *ftp* address (*ftp://penelope.info-ab.uclm.es/Anonymous/tpal*). Its main features are the following:

- Full support for the creation and edition of specifications in TPAL.
- Syntax and semantics analysis of TPAL specifications.
- Generation of Dynamic State Graphs for TPAL specifications.
- Reduction of the generated graphs.
- Graph browser, and simulation by using the graphs.
- Configuration (graph browser and simulation).
- Calculation of bounds for the times to reach the graph nodes, according to a concrete path.

A snapshot of the tool is shown in Fig.1, where both the main form of the tool and the simulation form are shown.

5.1 Syntax and Semantics Analysis

Once we have introduced the specification, we may analyse the syntax and semantics correctness, generating in this process a syntactic tree. This syntactic tree will be later used for several purposes: first of all to make some additional semantic checks, such as those concerning the types of the involved processes, and the analysis of recursive specifications.

5.2 Graph Generation

Starting from the syntactic tree generated by the syntax checker, the dynamic state graph for the given specification can be generated. This task is accomplished by using a different graph constructor for each operator of the language. These graphs have the following features:

- Each graph is, as usual, a collection of nodes and edges, as well as a set of clocks.
- There is a global clock, which is updated with every transition.
- Nodes are annotated with algebraic terms (each node corresponds to a term).
- Edges are annotated with tuples (a, r, S), indicating the name of the action executed in this transition (a), some clock restrictions that we must obey for its firing (r), and a set of clocks that must synchronise with the global clock (S).

5.3 Graph Visualisation and Simulation

Once we have generated a graph, we can show it either in a text file or graphically. As the graph size can be very large, the graph is drawn partially, by using a canvas whose size can be configured, as well as some other parameters, like the radius and the distance between nodes. Thus, there are some nodes with adjacents that are not pictured, being coloured in yellow in order to identify them easily. We may click on a node to fix it as the current node, and the graph is redrawn starting from it. The current node is coloured in red, so it can be easily identified.

We also have the ability to simulate the behaviour of the system, either manually or automatically. In the latter case, we can set the option of avoiding time-locks, which tries to avoid the deadlock of the system due to the expiration of some deadlines. If we make an automatic simulation, both delays and edges are chosen randomly by using a configurable *seed*, which allows us to repeat the experiments. There are also some other configurable parameters related to the simulation, such as the maximum time of simulation, the maximum number of steps, etc.

5.4 Analysis of Properties

The analysis of properties is a matter of work in progress, so the current version of TPAL only allows us to compute both an upper and a lower bound for the times to reach the graph nodes. We are working on two techniques of verification, the simplest one being based on the transformation of the generated graph into timed automata, while the second is to make the verification directly by using the graphs, in a similar way to that followed in timed automata by defining a region graph.

6 Conclusions

In this paper we have defined a Timed Process Algebra, which is based on basic LOTOS, by adding some new operators for the specification of timed behaviours and urgent

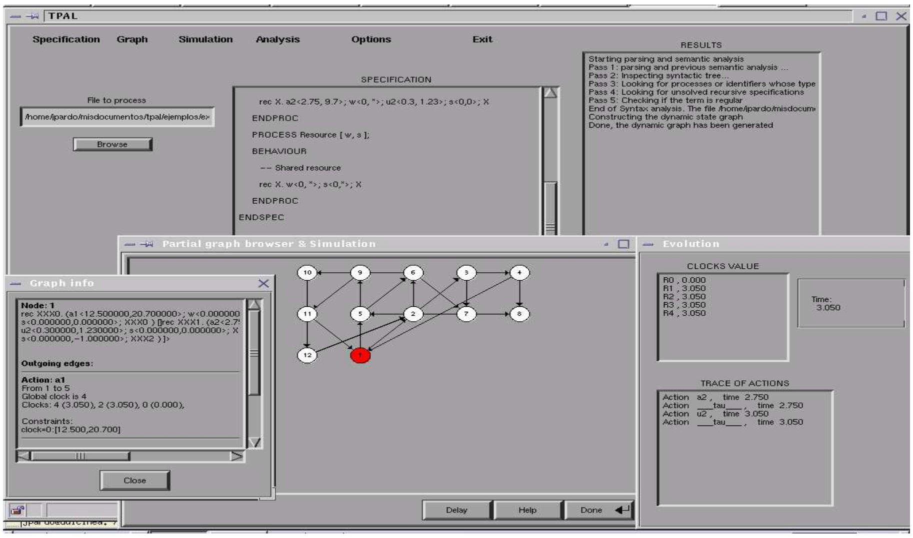


Figure 1. A picture of TPAL

interactions. We have defined an operational semantics for the language and a translation into dynamic state graphs, which allows us to simulate the specifications in an easy way, as well as to verify some properties of interest. We have implemented a tool that supports this language, in order to generate and simulate these graphs. This translation works fine 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 also be executed in the corresponding graph, and conversely.

Our future work will focus on the analysis of properties by using the graphs. Another aspect that we intend to develop is the modular specification of components, in order to reduce the complexity of the obtained graphs.

References

[1] R. Alur and D. L. Dill. Automata-theoretic verification of real-time systems. In *Formal Methods for Real-Time Computing*, pages 55–82. Wiley, 1996.

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

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

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

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

[6] I. Kang and L. Insup. An efficient generation of the timed reachability graph for the analysis of real-time systems. 1993.

[7] K. G. Larsen. Timed and Hybrid Systems in UPPAAL2k. *Proc. of MOVEP'2k (Modelling and Verification of Parallel Processes), Nantes, 2000.*

[8] A. K. Mok, D. A. Stuart, and F. Jahanian. Specification and analysis of real-time systems: Modechart language and toolset. In *Formal Methods for Real-Time Computing*, pages 33–54. Wiley, 1996.

[9] F. Moller and C. Tofts. A temporal calculus of communicating systems. *Proceedings CONCUR 90, Lecture Notes in Computer Science*, (458):401–415, 1990.

[10] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to Timed Graphs and Hybrid Systems. *Acta informatica*, (30):181–202, 1993.

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

[12] J. Pardo and V. Valero. Translating a timed process algebra into dynamic state graphs. Technical Report Computer Science Dept., Univ. of Castilla-La Mancha, Spain. DIAB-01-01-12, 2001.

[13] J. Quemada, A. Azcorra, and D. de Frutos. A Timed Calculus for LOTOS. *Proc. FORTE 89*, 1989.

[14] S. Raju. An automatic verification technique for communicating real-time state machines. Technical Report 93-04-08, Universidad de Washington, April 1993.

[15] G. M. Reed and A. W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, (58):249–261, 1988.

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

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

[18] W. Yi. CCS + Time = An Interleaving Model for Real Time Systems. *Proc. ICALP'91, Lecture Notes in Computer Science*, 510, 1991.