From ATP to Timed Graphs and Hybrid Systems *

Xavier Nicollin Joseph Sifakis Sergio Yovine

Laboratoire de Génie Informatique
IMAG-Campus B.P. 53X
38041 Grenoble Cedex – FRANCE

Abstract

The paper presents results of ongoing work aiming at the unification of some behavioral description formalisms for timed systems.

We propose for the algebra of timed processes **ATP** a very general semantics in terms of a time domain.

It is then shown how **ATP** can be translated into a variant of timed graphs. This result allows the application of existing model-checking techniques to **ATP**.

Finally, we propose a notion of hybrid systems as a generalization of timed graphs. Such systems can evolve, either by executing a discrete transition, or by performing some "continuous" transformation.

The formalisms studied admit the same class of models: time deterministic and time continuous, possibly infinitely branching transition systems labeled by actions or durations.

1 Introduction

The paper presents a generic version of the Algebra of Timed Processes **ATP** and its relationships to timed graphs and hybrid systems. The aim of this section is to present the general framework and assumptions about timed systems that motivated the design choices in **ATP**.

A timed system is usually considered to be a system with a global parameter (state variable) called time, used to constrain the occurrences of the actions. Introducing time requires consistent assumptions about its progress with respect to the evolution of the system: correspondence between instants (domain of definition of the time parameter) and action occurrences, duration of actions...

The model for timed systems used for **ATP** is based on the following principles:

- A timed system is the parallel composition of communicating sequential components (processes). Each component has a state variable defined on an appropriate time domain D with a binary operation $+$ which has essentially the properties of addition on non negative numbers. A component may modify its state either by executing some (atomic) action or by increasing its time variable (letting time progress).

*Work supported by the ESPRIT BRA SPEC

- System time progresses synchronously in all processes, i.e., from a given global state, time increases by a quantity d if all the components accept to do so.
- An execution sequence is a sequence of steps, a step being determined by two successive time evolutions. Within a step, components may execute either independently or in cooperation, a finite though arbitrarily long sequence of actions after which they coordinate to let time progress. This corresponds to the beginning of a new step.

The functioning described combines both synchronous and asynchronous cooperation in two alternating phases: one where all the components agree for the time to progress, and an eventually terminating asynchronous computation phase during which the progress of time is blocked. Such a mode of two-phase functioning is quite appropriate and natural for modeling reactive systems. For instance, hardware and systems for real-time control function according to this principle. Considering such a mode of functioning, allows to correlate the speeds of a system's components and introduces a concept of duration. Direct consequences of these assumptions are:

- The assumption about atomicity of actions implies that actions take no time. This simplifies theoretical development and does not go against generality as non atomic actions can be modeled by sequences of atomic ones. It has been advantageously adopted by other formalisms like Esterel [BC84].
- The time considered is *abstract* in the sense that it is used as a parameter to express constraints about instants of occurrences of actions. The implementability of such constraints taking into account speeds and execution times of processors, is a separate though not independent issue. This distinction between abstract and concrete or physical time is an important one. It allows simplifications that are convenient at conceptual level as it leads to simpler and more tractable models. For instance, the assumption that an action may take zero time, though not realistic for physical time, is quite convenient for abstract time. However, such an abstraction can take into account realisability issues: in any correct implementation, the clock period should be taken greater than the longest duration of a step computed so as to take into account execution time of sequences of ideally zero time actions. The assumption about eventual progress of time guarantees that such a bound exists.

Following these ideas, the Algebra of Timed Processes **ATP** has been studied [NRSV90, NS90]. In these papers we considered that the time domain is discrete and this was implemented by assuming that processes can perform, apart from ordinary actions, a special "time action" represented by χ, whose synchronous execution makes time progress by one unit. Apart from standard operators of process algebras like prefixing by an action different from χ, non-deterministic choice and parallel composition, **ATP** has delay constructs allowing to describe timeouts and watchdogs.

This work has been motivated by the study of relationships between **ATP** and timed graphs, a model for the description and analysis of timed systems proposed in [ACD90]. This model is a state-transition graph extended with a mechanism that allows the expression of constraints on the delays between the state transitions. Constraints are expressed as predicates on state variables representing timers. The interesting fact about timed graphs is that there exist model

checking methods for temporal logics with quantitative temporal operators which are directly applicable to them.

A main result of the paper is a method for the translation of **ATP** into timed graphs. As the semantics of timed graphs has been defined for dense time domains, we provide a generic definition $\mathbf{ATP}_D$ of **ATP** in terms of an arbitrary time domain D. The proposed translation method allows to obtain from a process of $\mathbf{ATP}_D$ a timed graph whose timers are defined on D. Due to this result, the verification methods applicable to timed graphs can be extended to **ATP**. This work required the definition of a general notion of time domain and the generalization of well-known results about **ATP**.

Another important idea of the paper is the proposition of a general model for hybrid systems as a generalization of timed graphs. A hybrid system is described as an extended state-transition graph with a set of variables X whose values can be modified as follows:

- Transitions are labeled with guarded commands i.e., pairs $(b(X), X := f(X))$, where $b(X)$ is a condition on X and f is a generalized function. As usually, a transition is enabled for a given configuration of values of the variables X_0 if $b(X_0)$ evaluates to true. The resulting state is obtained by executing the assignment and moving from the current node of the graph to its successor by the transition.
- Nodes are labeled with functions φ representing transformations of the variables in terms of the time t elapsed and of their initial values X_0 when the node is entered. That is, the value of the state variables after sojourn time t at a node is $\varphi(X_0, t)$. The transformation stops when the node is left by executing a transition.

The functioning described is composed as for timed systems of two phases: One where time passes and state transformation follows a law depending on time progress, and another instantaneous where the state is submitted to a "discrete" change. A result of this work is that **ATP**, Timed Graphs and Hybrid Systems can be defined using the same class of models.

The paper is organized as follows:

- Section 2 is devoted to the presentation of the algebra $\mathbf{ATP}_D$. The semantic rules of the operators are parametrized by the time domain D. We present an axiomatization for sequential processes, and we study the properties of the underlying models.
- In section 3, we define the Action Timed Graphs and their operational semantics in the same class of models as **ATP**. Action timed graph are a variant of the timed graphs proposed in [ACD90] adapted so as to take into account features of **ATP**.
- We present then a method for translating an **ATP** process into an action timed graph with the same time domain, whose model is observationally equivalent to that of the process.
- Finally, in section 5, the issue of hybrid systems is discussed and illustrated by two examples. A definition is proposed, yielding a quite simple operational semantics.

2 $\mathbf{ATP}_D$: ATP for a time domain D

We present here a generic version of the algebra **ATP**, parametrized with an arbitrary time domain. For a complete theory of **ATP**, the reader should refer to [NS90].

The primitive constructs of **ATP** in this paper are different from those presented in [NS90]. We introduce the notion of *delayable action*, and the start-delay operator of [NS90] is replaced by a *timeout* operator. These modifications are motivated by the fact that the timeout operator is much easier to define over timed graphs than the start-delay operator. However, the class of models obtained, in the case where the domain is discrete, is the same.

2.1 Time domains

Definition: A *time domain* is a commutative monoid $(D, 0, +)$ satisfying the following requirements.

- For all d, d' in D, $d + d' = d$ if and only if $d' = 0$;
- The preorder $\leq$, defined by $d \leq d' \Leftrightarrow (\exists d'')\ d + d'' = d'$, is a total order.

The following properties can be easily proved.

- 0 is the least element of D;
- For any d, d', if $d \leq d'$, then the element d'' such that $d + d'' = d'$ is unique. It is denoted by $d' - d$.

We denote $D - \{0\}$ by D_*. We also write $d < d'$ instead of $d \leq d' \ \wedge \ d \neq d'$.

D is called *dense* if $(\forall d, d')\ [d < d' \Rightarrow (\exists d'')\ d < d'' < d']$.

D is called *discrete* if $(\forall d)\ (\exists d')\ [d < d' \wedge (\forall d'')\ d < d'' \Rightarrow d' \leq d'']$. Since the order is total, d' is unique and is called the *successor* of d, denoted by $\mathtt{succ}(d)$. $\mathtt{succ}(0)$ is denoted by χ. An important property of a discrete domain is that for any d, $\mathtt{succ}(d) = d + \mathtt{succ}(0)$. That is, any element of D can be obtained from 0 by adding as many $\mathtt{succ}(0)$ as necessary.

Examples of time domains are $\mathbb{N}$ (discrete), $\mathbb{Q}^+$ and $\mathbb{R}^+$ (dense), or even the singleton $\{0\}$.

2.2 $\mathbf{ASTP}_D$: Sequential ATP

2.2.1 Syntax

Let $\mathcal{A}$ be a vocabulary whose elements, called *actions*, are denoted by possibly indexed lowercase greek letters (except δ). Put

$$\dot{\mathcal{A}} \ \triangleq \ \{\dot{\alpha} \mid \alpha \in \mathcal{A}\}$$
$$\widetilde{\mathcal{A}} \ \triangleq \ \{\widetilde{\alpha} \mid \alpha \in \mathcal{A}\}$$

$\dot{\mathcal{A}} \cup \tilde{\mathcal{A}}$ is the set of prefixing operators of the algebra. $\dot{\alpha}$ is called an *immediate* action, while $\tilde{\alpha}$ is called a *delayable* action. $\underline{\alpha}$ denotes either $\dot{\alpha}$ or $\tilde{\alpha}$.

Let $\mathcal{V}$ be a set of *process variables*, whose elements are denoted by $X, Y, X_1, ...$ The syntax of sequential **ATP** with time domain D (noted $\mathbf{ASTP}_D$, or **ASTP** when the time domain is not relevant) is described by the following BNF grammar.

$$P ::= \mathbf{0} \quad | \quad \delta \quad | \quad X \quad | \quad \underline{\alpha}\, P \quad | \quad P \oplus P \quad | \quad P \overset{d}{\triangleright} P \quad | \quad \mathtt{rec} X \cdot P$$

where $d \in D_*$.

- $\mathbf{0}$ is a blocked process; it can do nothing, not even let time pass.
- δ is a terminated process; it can do nothing but letting time pass forever.
- $\underline{\alpha}\, P$ is prefixing of P by $\underline{\alpha}$. The process $\dot{\alpha}\, P$ performs the action α immediately, whereas the process $\tilde{\alpha}\, P$ may wait any amount of time before performing α.
- $P \oplus Q$ is the non-deterministic choice between processes P and Q.
- $P \overset{d}{\triangleright} Q$ is a *timeout* construct with delay d. It waits until time d for P to perform an action, in which case it behaves as P. If an action of P is not performed before time d, then it behaves as Q.
- $\mathtt{rec} X \cdot P$ is the classical recursion construct.

When writing processes, we consider that operators have the following decreasing priority: prefixing, timeout, non-deterministic choice, and recursion. Moreover, timeout is considered to be right-associative.

In this language, we consider only closed *well-guarded* terms. A term is well-guarded if in all its recursive sub-terms $\mathtt{rec} X \cdot P$, the free occurrences of X in P appear either under a prefixing operator, or in the second argument of a timeout operator.

2.2.2 Operational semantics

The operational semantics of $\mathbf{ASTP}_D$ associates labeled transition systems to processes. States are process expressions. Since the time domain is not necessarily IN, we cannot use the time action χ, as in [NS90], for temporal transitions. We use, as in [Wan90, MT90], transition systems labeled by $\mathcal{L} = \mathcal{A} \cup D_*$.

The transition relation is then a subset of $\mathbf{ASTP} \times \mathcal{L} \times \mathbf{ASTP}$. It is defined by structural induction, à la Plotkin [Plo81], by the following rules, where $d, d' \in D_*$ and $l \in \mathcal{L}$.

$$\delta \xrightarrow{d} \delta \qquad\qquad \frac{P[(\mathtt{rec} X \cdot P)/X] \xrightarrow{l} P'}{\mathtt{rec} X \cdot P \xrightarrow{l} P'}$$

$$\dot{\alpha}\,P \xrightarrow{\alpha} P \qquad \tilde{\alpha}\,P \xrightarrow{\alpha} P \qquad \tilde{\alpha}\,P \xrightarrow{d} \tilde{\alpha}\,P$$

$$\frac{P \xrightarrow{\alpha} P'}{P \oplus Q \xrightarrow{\alpha} P'} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \oplus Q \xrightarrow{\alpha} Q'} \qquad \frac{P \xrightarrow{d} P' \ , \ Q \xrightarrow{d} Q'}{P \oplus Q \xrightarrow{d} P' \oplus Q'}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \overset{d}{\triangleright} Q \xrightarrow{\alpha} P'} \qquad \frac{P \xrightarrow{d'} P'}{P \overset{d+d'}{\triangleright} Q \xrightarrow{d'} P' \overset{d}{\triangleright} Q} \qquad \frac{P \xrightarrow{d} P'}{P \overset{d}{\triangleright} Q \xrightarrow{d} Q} \qquad \frac{P \xrightarrow{d} P' \ , \ Q \xrightarrow{d'} Q'}{P \overset{d}{\triangleright} Q \xrightarrow{d+d'} Q'}$$

Strong equivalence
Transition systems are considered modulo the *strong equivalence* $\sim$, which is the largest binary relation $\mathcal{R}$ on processes such that

$$P\,\mathcal{R}\,Q \;\Rightarrow\; \begin{cases} \forall l, \forall P', P \xrightarrow{l} P' \Rightarrow \exists Q', Q \xrightarrow{l} Q' \;\wedge\; P'\,\mathcal{R}\,Q' \\ \forall l, \forall Q', Q \xrightarrow{l} Q' \Rightarrow \exists P', P \xrightarrow{l} P' \;\wedge\; P'\,\mathcal{R}\,Q' \end{cases}$$

We write $(P)_D$ to denote the model of P in the semantics of $\mathbf{ASTP}_D$.

2.2.3 Properties of the models

The transition relation satisfies the following properties, where d and d' range over D_* (the terminology used for (1) and (2) is due to Wang [Wan90]).

(1) time determinacy: $(P \xrightarrow{d} P') \wedge (P \xrightarrow{d} P'') \Rightarrow P' = P''$
(where $=$ is syntactic equality)

(2) time continuity: $(P \xrightarrow{d} P') \wedge (P' \xrightarrow{d'} P'') \Leftrightarrow P \xrightarrow{d+d'} P''$

(3) partial persistency: $P \xrightarrow{d} P' \Rightarrow \; (\exists d' > 0)(\forall \alpha)(\forall P'')(\forall d'' \in]0, d'[)(\exists P_1, P_2)$
$P \xrightarrow{\alpha} P'' \Rightarrow P \xrightarrow{d''} P_1 \xrightarrow{\alpha} P_2$

A process P is said to be *immediate* if it cannot let time progress, i.e., there is no transition $P \xrightarrow{d}$.

The last property asserts that, if a process is not immediate, then any action it can perform immediately remains possible during some non-empty interval (upto d'). Moreover, we have here $P_2 = P''$.

An important property of the models is that, if D' is a time domain such that $D' \subseteq D$, then for all P in $\mathbf{ASTP}_{D'}$, $(P)_{D'}$ can be obtained from $(P)_D$ by restricting its transition relation to $D' \cup \mathcal{A}$. However, even though $D' \subseteq D$, the behaviors described by the same term may be quite different. For instance, take $D' = \mathbb{N}$ and $D = \mathbb{R}^+$, and consider the process

$$P = \widetilde{\alpha}\,(\widetilde{\beta}\,\dot{\gamma}\,\delta \overset{1}{\rhd} \delta) \overset{1}{\rhd} \delta$$

- In $(P)_{\mathbb{N}}$, α may only be performed before time 1, that is at time 0; then β may also be performed only before time 1, and finally γ is performed immediately after β. So, γ *may only be performed before time 1* (in fact at time 0).
- In $(P)_{\mathbb{R}^+}$, α may be performed before time 1, so it is possible to wait for instance 0.7 time units before executing it. Afterwards, β may be executed until one time unit *after performing* α; hence, β may be performed at time 1.5 for instance. We find then that γ *may be performed at any time before time 2.*

The models are generally infinitely branching transition systems. Moreover the number of states is infinite if the time domain is dense, but it is finite in the discrete case. If D is discrete, it is possible to build a finite state, finitely branching, transition system, by considering only the transitions labeled by actions or by χ (remember that χ is $\mathtt{succ}(0)$). The time-continuity property is no longer valid, but the definition of strong equivalence and the equivalence classes remain the same in this semantics, which is the one given in [NS90].

For instance, the transition system of the process $\widetilde{\alpha}\,\dot{\beta}\,\delta \overset{3}{\rhd} \widetilde{\gamma}\,\delta$ in $\mathbf{ASTP}_{\mathbb{N}}$, given in figure 1, may be reduced into the model of figure 2.

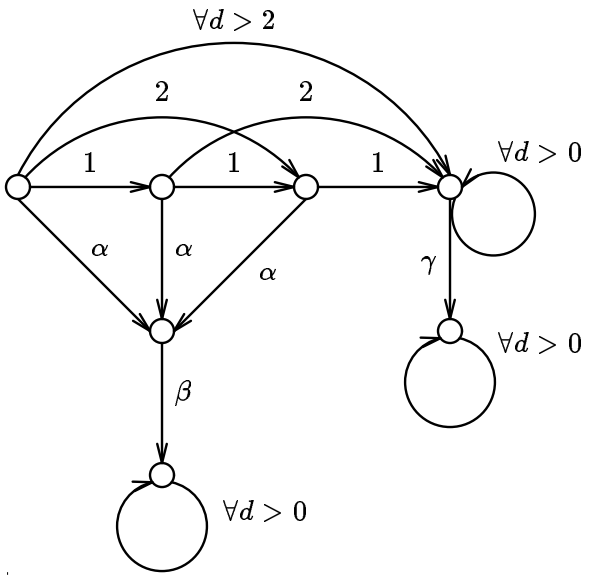

Figure 1: infinitely branching model

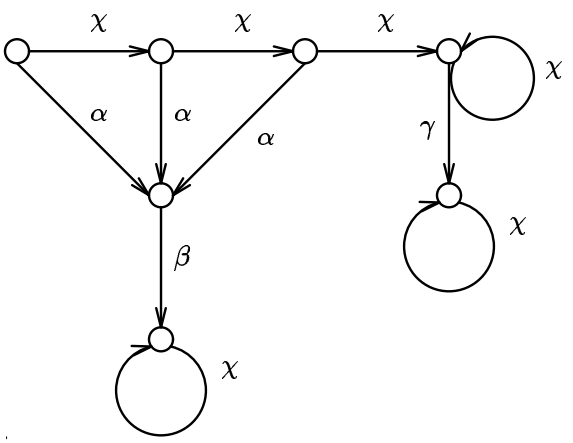

Figure 2: finitely branching model

2.2.4 Axiomatization and canonical form

The following equations, where $d, d' \in D_*$ and $e \in D$, define a sound axiomatization of **ASTP** for strong equivalence.

$$[\oplus 1] \qquad P \oplus Q \quad \equiv \quad Q \oplus P$$

$$
\begin{array}{lll}
[\oplus 2] & (P \oplus Q) \oplus R & \equiv \quad P \oplus (Q \oplus R) \\
[\oplus 3] & P \oplus P & \equiv \quad P \\
[\oplus 4] & P \oplus \delta & \equiv \quad P \\
[\oplus 5] & \mathbf{0} \ \oplus \ \dot{\alpha}\, P & \equiv \quad \dot{\alpha}\, P \\
[\oplus 6] & \mathbf{0} \ \oplus \ \widetilde{\alpha}\, P & \equiv \quad \dot{\alpha}\, P \\
[\oplus 7] & \mathbf{0} \ \oplus \ P \overset{d}{\triangleright} Q & \equiv \quad \mathbf{0} \ \oplus \ P \\
[\oplus 8] & P_1 \overset{d}{\triangleright} Q_1 \ \oplus \ P_2 \overset{d}{\triangleright} Q_2 & \equiv \quad (P_1 \oplus P_2) \overset{d}{\triangleright} (Q_1 \oplus Q_2) \\
[\triangleright 1] & \mathbf{0} \overset{d}{\triangleright} Q & \equiv \quad \mathbf{0} \\
[\triangleright 2] & \delta \overset{d}{\triangleright} \delta & \equiv \quad \delta \\
[\triangleright 3] & \widetilde{\alpha}\, P \overset{d}{\triangleright} \widetilde{\alpha}\, P & \equiv \quad \widetilde{\alpha}\, P \\
[\triangleright 4] & (P \overset{d+e}{\triangleright} Q) \overset{d}{\triangleright} R & \equiv \quad P \overset{d}{\triangleright} R \\
[\triangleright 5] & (P \overset{d}{\triangleright} Q) \overset{d+d'}{\triangleright} R & \equiv \quad P \overset{d}{\triangleright} Q \overset{d'}{\triangleright} R \\
[\mathrm{rec}1] & \mathtt{rec}X \cdot P & \equiv \quad P[(\mathtt{rec}X \cdot P)/X] \\
[\mathrm{rec}2] & \text{if} \quad P[Q/X] \equiv Q & \text{then} \quad \mathtt{rec}X \cdot P \equiv Q
\end{array}
$$

Proposition: Any process of **ASTP** is equivalent to a process in one of the following forms.

$$
\sum_{i \in I} \dot{\alpha}_i\, P_i \qquad\qquad \left(\sum_{i \in I} \widetilde{\alpha_i}\, P_i \right) \overset{d}{\triangleright} P \qquad\qquad \delta \overset{d}{\triangleright} P
$$

where I is finite. □

Moreover, the following result, already presented in [NS90], can be generalized if D is such that $(\forall d, d')\ (\exists n, m \in \mathbb{N})\ nd = md'$, where $nd = \underbrace{d + ... + d}_{n \text{ times}}$.

Proposition (canonical form): For any P in $\mathbf{ASTP}_D$, there exist $n \in \mathbb{N}$ and $P_1, ..., P_n$ in $\mathbf{ASTP}_D$, such that $P \equiv P_1$ and for any i in $[1, n]$,

$$
P_i \equiv \sum_{j \in J} \dot{\alpha}_j\, P_j \quad \text{or} \quad P_i \equiv \left(\sum_{j \in J} \widetilde{\alpha_j}\, P_j \right) \overset{d_i}{\triangleright} P_k \quad \text{or} \quad P_i \equiv \delta \overset{d_i}{\triangleright} P_k
$$

where the j's and k are in $[1, n]$. □

This proposition does not hold if, for instance, $D = \mathbb{R}^+$. Indeed, consider the process $P = Q \oplus R$, where

$$
\begin{array}{lll}
Q & = & \mathtt{rec}X \cdot \widetilde{\alpha}\, \delta \overset{1}{\triangleright} \delta \overset{1}{\triangleright} X \\
R & = & \mathtt{rec}Y \cdot \widetilde{\beta}\, \delta \overset{\pi}{\triangleright} \delta \overset{\pi}{\triangleright} Y
\end{array}
$$

P may perform α at any instant between $2k$ and $2k+1$, and β at any instant between $2k\pi$ and $(2k+1)\pi$, where $k \in \mathbb{N}$. Following the first proposition, we have

$$P \equiv \left(\widetilde{\alpha}\,\delta \oplus \widetilde{\beta}\,\delta\right) \overset{1}{\triangleright} \left(\delta \overset{1}{\triangleright} Q \;\oplus\; \widetilde{\beta}\,\delta \overset{\pi^{-1}}{\triangleright} \delta \overset{\pi}{\triangleright} R\right)$$

but it is not possible to find a *finite* set of equations giving a canonical form of P.

2.3 The algebra $\mathbf{ATP}_D$

2.3.1 Syntax

We present now the algebra $\mathbf{ATP}_D$, by adding three new operators. This presentation in two steps is motivated by the fact that we do not want recursion through these operators. The syntax of $\mathbf{ATP}_D$ is given by

$$\begin{array}{lllllll} P ::= & P \in \mathbf{ASTP}_D & | & \underline{\alpha}\,P & | & P \oplus P & | \quad P \overset{d}{\triangleright} P \\ & P \parallel P & | & \partial_H(P) & | & \lceil P \rceil^d(P) & \end{array}$$

where $d \in D_*$ and $H \subseteq \mathcal{A}$.

- $P \parallel Q$ is the *parallel composition* operator. It behaves as the parallel composition operator of ACP [BK84, BK86], as far as actions are concerned. Moreover, $P \parallel Q$ may delay for time d when and only when both P and Q may do so.

  As for ACP, a binary *communication function* $|$ is defined over $\mathcal{A}_\perp = \mathcal{A} \cup \{\perp\}$, where $\perp$ is a new symbol not belonging to $\mathcal{A}$. It is such that

  $$\alpha|\beta = \beta|\alpha \qquad \alpha|(\beta|\gamma) = (\alpha|\beta)|\gamma \qquad \alpha|\perp = \perp|\alpha = \perp|\perp = \perp$$

  If $\alpha|\beta = \perp$, then no communication is possible between the actions α and β.

- $\partial_H(P)$ is the *encapsulation* operator of ACP. It disallows P to perform actions of H.

- $\lceil P \rceil^d(Q)$ is the *execution delay* or *watchdog* operator. It behaves as P until time d; at time d, P is "aborted", and Q is started. However, if P performs a special action ξ, called *cancel action*, then the delay is cancelled, and the subsequent behavior is that of P after ξ. The cancel action is internal, that is, it cannot communicate with any other action. Its effect is restricted to the innermost delay construct only; outside this operator, it is viewed as a τ (internal) action (see [NS90] for more details).

2.3.2 Operational semantics

The semantics of the operators already present in **ASTP** remain the same. For parallel composition, encapsulation and execution delay, the transition relation is given by the following rules.

$$\frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'}$$

$$\frac{P \xrightarrow{\alpha} P' \ , \ Q \xrightarrow{\beta} Q' \ , \ \alpha|\beta \neq \bot}{P \parallel Q \xrightarrow{\alpha|\beta} P' \parallel Q'} \qquad \frac{P \xrightarrow{d} P' \ , \ Q \xrightarrow{d} Q'}{P \parallel Q \xrightarrow{d} P' \parallel Q'}$$

$$\frac{P \xrightarrow{l} P' \ , \ l \notin H}{\partial_H(P) \xrightarrow{l} \partial_H(P')}$$

$$\frac{P \xrightarrow{\alpha} P' \ , \ \alpha \neq \xi}{\lceil P \rceil^d(Q) \xrightarrow{\alpha} \lceil P' \rceil^d(Q)} \qquad \frac{P \xrightarrow{\xi} P'}{\lceil P \rceil^d(Q) \xrightarrow{\tau} P'}$$

$$\frac{P \xrightarrow{d'} P'}{\lceil P \rceil^{d+d'}(Q) \xrightarrow{d'} \lceil P' \rceil^d(Q)} \qquad \frac{P \xrightarrow{d} P'}{\lceil P \rceil^d(Q) \xrightarrow{d} Q} \qquad \frac{P \xrightarrow{d} P' \ , \ Q \xrightarrow{d'} Q'}{\lceil P \rceil^d(Q) \xrightarrow{d+d'} Q'}$$

The four properties of the models of sequential processes remain valid in **ATP**. However, an important difference is that in the last one (partial persistency), the behavior subsequent to the action α may depend on the instant when α is performed, as soon as a parallel or execution delay operator is involved. This difference makes impossible the formulation of an expansion theorem for these operators in the case where D is dense. In order to give such a theorem, we need to extend **ATP** by adding time parameters to processes, as, for instance, in [Wan91]. However, it is possible to prove that $\mathbf{ATP}_D$ is a conservative extension of $\mathbf{ASTP}_D$ when D is discrete (see [NS90] where this result is proven for $D = \mathbb{N}$).

3 Action Timed Graphs

We present in this section a variant of the timed graphs of Alur, Courcoubetis and Dill [ACD90]. Timed graphs are automata extended with variables used to count time passing, called *timers*. The values of all the timers increase at the same speed. A transition between two nodes of the automaton may be labeled by a predicate on the values of timers, telling when the transition may be fired. A transition is instantaneous, and it may be used to change discretely the timers values (usually by resetting some of them to zero).

We provide hereafter a general definition of timed graphs. We give then their operational semantics, which defines their models in terms of labeled transition systems of the same kind as those of $\mathbf{ATP}_D$ processes.

3.1 Definitions

Let A be an action vocabulary, and a denote an element of A.

A tuple $T = (t_1, ..., t_m)$ of *timers* is a set of variables taking their values in D. We represent by $\mathcal{B}(T)$ the set of the predicates over T, that is, the set of mappings from D^n into $\{\mathtt{tt}, \mathtt{ff}\}$. Furthermore, $\mathcal{F}(T)$ is the set of mappings from D^n into D^n.

An *Action Timed Graph* is a structure $G = (N, T, n_0, \mathtt{div}, \rightarrow)$, where

- N is the finite set of *nodes*;
- T is the tuple of *timers*;
- $n_0 \in N$ is the *initial node*;
- $\mathtt{div}$ is a mapping from N into $\{\mathtt{tt}, \mathtt{ff}\}$, called the *divergence function*; a node n is called *divergent* if $\mathtt{div}(n)$ is true;
- $\rightarrow \subseteq N \times A \times \mathcal{B}(T) \times \mathcal{F}(T) \times N$ is the *transition relation.*
  We write $n \xrightarrow{a,b,f} n'$ instead of $(n, a, b, f, n') \in \rightarrow$.

For any predicate $b \in \mathcal{B}(T)$, the predicate $\widehat{b}$ is defined by

$$\widehat{b}(T) = (\exists d \in D)\ \ b(T + d)$$

where $T + d = (t_1 + d, ..., t_n + d)$. Notice that $\widehat{b}$ is a predicate meaning "eventually b", i.e., "b is true or will be true in the future".

For instance, if $b = (2 < t_1 \leq t_2) \wedge (t_2 \leq 4)$, then

$$\widehat{b} = (\exists d)\ \ (2 < t_1 + d \leq t_2 + d) \wedge (t_2 + d \leq 4) = (t_1 \leq t_2 \leq 4) \wedge (t_2 \leq t_1 + 2)$$

For a node n in N, we define the *enabling condition* of n, $\mathtt{en}(n)$, characterizing the values of timers for which a transition can be executed from n:

$$\mathtt{en}(n) \triangleq \bigvee \left\{ b \mid n \xrightarrow{a,b,f} \right\}$$

We also define the *activity condition* of n, $\mathtt{act}(n)$, by

$$\mathtt{act}(n) \triangleq \widehat{\mathtt{en}(n)} \vee \mathtt{div}(n)$$

3.2 Operational semantics of Action Timed Graphs

The timers of an action timed graph are variables which are increased "continuously" and at the same speed. Initially, all the timers have value 0. The nodes represent the control states. From the node n, if the enabling condition b of a transition $n \xrightarrow{a,b,f} n'$ is satisfied by the values of the timers, then the transition *may* be executed. That is, the action a may performed, and the control state becomes n' after modification of the values of the timers according to the function f.

If the divergence condition $\mathtt{div}(n)$ is $\mathtt{tt}$, then it is possible to remain in node n, even though it will nevermore be possible to execute some transition. Conversely, if $\mathtt{div}(n) = \mathtt{ff}$, then the system *must* evolve so as to avoid such a situation. The predicate $\mathtt{act}$ is defined so that the system may remain at node n at the "instant" defined by the value of T only if $\mathtt{act}(n)(T) = \mathtt{tt}$.

Formally, the operational model of an action timed graph is a labeled transition system, where states are pairs (n, T), labels are elements of $(A \cup D_*)$, and the initial state is $(n_0, \vec{0})$, where n_0 is the initial node of the timed graph. The transition relation is defined by the following rules.

$$\frac{n \xrightarrow{a,b,f} n' \ , \ b(T)}{(n,T) \xrightarrow{a} (n', f(T))} \qquad \frac{\mathtt{act}(n)(T+d)}{(n,T) \xrightarrow{d} (n, T+d)}$$

The first rule asserts that a transition is executable if its enabling predicate evaluates to true. The second one formalizes the condition to satisfy for remaining in a node by waiting d time units.

Remark: the divergence condition is not quite necessary for defining timed graphs. In fact, one can consider action timed graphs without divergent nodes. To model then divergence of a node, it is enough to add a transition, labeled by some special internal action τ, whose predicate becomes true only when all the conditions of the other transitions will never more be true. The target of this transition is then a "sink" node, which in fact must also have a transition, consisting in a loop with some predicate which always eventually becomes true. The activity condition of a node n is then defined by $\widehat{\mathtt{en}(n)}$ only, and the semantics rules above remain the same.

4 Translation of ATP into timed graphs

The purpose of this section is to provide a method for translating an **ATP** process into an action timed graph such that the corresponding models are equivalent.

4.1 Translation of ASTP

We first show how a sequential process is translated into a *one-timer relative-time* action timed graph, that is, a timed graph with only one timer which is reset every time an action transition is executed. The reader should convince himself that such a translation is not possible for a process of **ATP**, as soon as the time domain is dense.

The method consists in fact in building the canonical form of the process. Due to the proposition about this form, a sufficient condition for its applicability is that the delay parameters be taken in a subdomain D' of D such that $(\forall d, d' \in D')\,(\exists n, m \in \mathbb{N})\, nd = md'$.

For technical reasons, the operators $._{|d}$, where d is an element of D_*, are added to **ASTP**. $P_{|d}$ is called the *time-derivative at d* of P. It is defined only when P may delay upto d, and it behaves as P behaves at time d. Its semantics is quite simple, it is given by the following rule.

$$\frac{P \xrightarrow{d} P' \ , \ P' \xrightarrow{l} P''}{P_{|d} \xrightarrow{l} P''}$$

For technical reasons again, the action vocabulary of the timed graphs we build is $A = \mathcal{A} \cup \{i\}$, where $i \notin \mathcal{A}$. The action i, called *invisible action*, will represent the expirations of timeouts. Using such an action i simplifies considerably the translation.

The nodes of the timed graphs represent elements of **ASTP**, the unique timer is represented by t, and the initial node of the timed graph of P is P.

For P in **ASTP**, we define $S(P)$ as the instant upto which it is possible to stay in node P. For this purpose, ω is added as an upper bound of D, satisfying $\omega - d = \omega$. The same kind of definition may be found in [BB90, MT90].

$$\begin{aligned}
S(\mathbf{0}) &\triangleq 0 \\
S(\delta) &\triangleq \omega \\
S(\dot{\alpha}\, P) &\triangleq 0 \\
S(\tilde{\alpha}\, P) &\triangleq \omega \\
S(P \oplus Q) &\triangleq \min(S(P), S(Q)) \\
S(P \overset{d}{\triangleright} Q) &\triangleq \min(S(P), d) \\
S(P_{|d}) &\triangleq S(P) - d \qquad \text{when } P_{|d} \text{ is defined} \\
S(\mathtt{rec}X \cdot P) &\triangleq S(P[\mathtt{rec}X \cdot P/X])
\end{aligned}$$

The divergence in node P is defined by

$$\mathtt{div}(P) \triangleq (S(P) = \omega)$$

For the transition relation, the function f of a transition $P \xrightarrow{a,b,f} Q$ is always the function which resets t to 0; it is omitted here.

The transition relation is given by the following rules. Notice that an expiring timeout is translated into a transition with an invisible action i.

$$\dot{\alpha}\, P \xrightarrow{\alpha, t = 0} P \qquad \tilde{\alpha}\, P \xrightarrow{\alpha, 0 \le t} P$$

$$\frac{P \xrightarrow{\alpha, b} P'}{P \oplus Q \xrightarrow{\alpha, b \wedge t \le S(Q)} P'} \qquad \frac{Q \xrightarrow{\alpha, b} Q'}{P \oplus Q \xrightarrow{\alpha, b \wedge t \le S(P)} Q'}$$

$$\frac{P \xrightarrow{i, t = d} P' \ , \ S(Q) > d}{P \oplus Q \xrightarrow{i, t = d} P' \oplus Q_{|d}} \qquad \frac{Q \xrightarrow{i, t = d} Q' \ , \ S(P) > d}{P \oplus Q \xrightarrow{i, t = d} P_{|d} \oplus Q'}$$

$$\frac{P \xrightarrow{i,t=d} P' \ , \ Q \xrightarrow{i,t=d} Q'}{P \oplus Q \xrightarrow{i,t=d} P' \oplus Q'}$$

$$\frac{P \xrightarrow{\alpha,b} P'}{P \overset{d}{\triangleright} Q \xrightarrow{\alpha,b \wedge t<d} P'} \qquad \frac{P \xrightarrow{i,t=d'} P' \ , \ d' < d}{P \overset{d}{\triangleright} Q \xrightarrow{i,t=d'} P' \overset{d-d'}{\triangleright} Q} \qquad \frac{S(P) \geq d}{P \overset{d}{\triangleright} Q \xrightarrow{i,t=d} Q}$$

$$\frac{P \xrightarrow{a,b} P'}{P_{|d} \xrightarrow{a,b[t+d/t]} P'} \qquad \frac{P[\mathtt{rec}X \cdot P/X] \xrightarrow{a,b} P'}{\mathtt{rec}X \cdot P \xrightarrow{a,b} P'}$$

4.2 Translation of ATP

In order to be able to translate **ATP** we present an *algebra of timed graphs*, by defining the operators of **ATP** over the action timed graphs. To obtain the timed graph corresponding to an element P of **ATP**, we proceed then as follows.

- if P is in **ASTP**, then the timed graph is obtained by using the rules above.
- if P in not in **ASTP**, and is in the form $op(P_1, ..., P_n)$, where op is an n-ary operator of **ATP**, then the timed graph of P is obtained by applying the operator op defined below to the timed graphs of $P_1, ..., P_n$.

4.2.1 Operations over action timed graphs

In the sequel, $G_1 = (N_1, T_1, n_1^0, \mathtt{div}_1, \rightarrow_1)$ and $G_2 = (N_2, T_2, n_2^0, \mathtt{div}_2, \rightarrow_2)$ are two timed graphs with the action vocabulary $A = \mathcal{A} \cup \{i\}$, such that $N_1 \cap N_2 = T_1 \cap T_2 = \emptyset$.

The function resetting a set of timers T is denoted by $\mathcal{R}(T)$.

Prefixing
Let α be an element of $\mathcal{A}$. We define $\underline{\alpha} G_1 = (N, T, n_0, \mathtt{div}, \rightarrow)$ as follows.

$N = N_1 \cup \{n_0\} \quad (n_0 \notin N_1)$
$T = T_1 \cup \{t\}$ where t may be in T_1
if $n \in N_1$ then $\mathtt{div}(n) = \mathtt{div}_1(n)$
$\mathtt{div}(n_0) =$ if $\underline{\alpha} = \dot{\alpha}$ then $\mathtt{ff}$ else $\mathtt{tt}$

$$\frac{\underline{\alpha} = \dot{\alpha}}{n_0 \xrightarrow{\alpha, t=0, \mathcal{R}(T_1)} n_1^0} \qquad \frac{\underline{\alpha} = \widetilde{\alpha}}{n_0 \xrightarrow{\alpha, t \geq 0, \mathcal{R}(T_1)} n_1^0} \qquad \frac{n \in N_1 \ , \ n \xrightarrow{a,R}_1 n'}{n \xrightarrow{a,R} n'}$$

Non-deterministic choice

$G_1 \oplus G_2 = (N, T, n_0, \mathtt{div}, \rightarrow)$ is defined as follows.

$$N = N_1 \cup N_2 \cup (N_1 \times N_2)$$
$$T = T_1 \cup T_2$$
$$n_0 = (n_1^0, n_2^0)$$
if $n \in N_i$ then $\mathtt{div}(n) = \mathtt{div}_i(n)$
$$\mathtt{div}(n_1, n_2) = \mathtt{div}_1(n_1) \wedge \mathtt{div}_2(n_2)$$

$$\frac{n \xrightarrow{a,b,f}_i n'}{n \xrightarrow{a,b,f} n'} \qquad \frac{n_1 \xrightarrow{\alpha,b,f}_1 n_1'}{(n_1,n_2) \xrightarrow{\alpha, b \wedge \mathtt{act}(n_2), f} n_1'} \qquad \frac{n_2 \xrightarrow{\alpha,b,f}_2 n_2'}{(n_1,n_2) \xrightarrow{\alpha, b \wedge \mathtt{act}(n_1), f} n_2'}$$

$$\frac{n_1 \xrightarrow{i,b,f}_1 n_1'}{(n_1,n_2) \xrightarrow{i, b \wedge \mathtt{act}(n_2), f} (n_1', n_2)} \qquad \frac{n_2 \xrightarrow{i,b,f}_2 n_2'}{(n_1,n_2) \xrightarrow{i, b \wedge \mathtt{act}(n_1), f} (n_1, n_2')}$$

Timeout

Let d be an element of D_*, and t be a timer not belonging to T_1.

We define $G_1 \overset{d}{\triangleright} G_2 = (N, T, n_0, \mathtt{div}, \rightarrow)$ as follows.

$$N = N_1 \cup N_2 \cup (N_1 \times \{n_2^0\})$$
$$T = T_1 \cup T_2 \cup \{t\}$$
$$n_0 = (n_1^0, n_2^0)$$
if $n \in N_i$ then $\mathtt{div}(n) = \mathtt{div}_i(n)$
$$\mathtt{div}(n_1, n_2^0) = \mathtt{ff}$$

$$\frac{n \xrightarrow{a,b,f}_i n'}{n \xrightarrow{a,b,f} n'} \qquad (n_1, n_2^0) \xrightarrow{i, (t=d) \wedge \mathtt{act}(n_1), \mathcal{R}(T_2)} n_2^0$$

$$\frac{n_1 \xrightarrow{\alpha,b,f}_1 n_1'}{(n_1,n_2^0) \xrightarrow{\alpha, b \wedge (t<d), f} n_1'} \qquad \frac{n_1 \xrightarrow{i,b,f}_1 n_1'}{(n_1,n_2^0) \xrightarrow{i, b \wedge (t<d), f} (n_1, n_2^0)}$$

Parallel composition

We define $G_1 \parallel G_2 = (N, T, n_0, \mathtt{div}, \rightarrow)$ as follows

$$N = N_1 \times N_2$$
$$T = T_1 \cup T_2$$
$$n_0 = (n_1^0, n_2^0)$$
$$\mathtt{div}(n_1, n_2) = \mathtt{div}_1(n_1) \wedge \mathtt{div}_2(n_2)$$

$$\frac{n_1 \xrightarrow{a,b,f}_1 n_1'}{(n_1,n_2) \xrightarrow{a,b \wedge \mathtt{act}(n_2),f} (n_1',n_2)} \qquad \frac{n_2 \xrightarrow{a,b,f}_2 n_2'}{(n_1,n_2) \xrightarrow{a,b \wedge \mathtt{act}(n_1),f} (n_1,n_2')}$$

$$\frac{n_1 \xrightarrow{\alpha_1,b_1,f_1}_1 n_1' \ , \ n_2 \xrightarrow{\alpha_2,b_2,f_2}_2 n_2' \ , \ \alpha_1|\alpha_2 = \alpha \neq \bot}{(n_1,n_2) \xrightarrow{\alpha,b_1 \wedge b_2,f_1 \star f_2} (n_1',n_2')}$$

where $f_1 \star f_2$ denotes the function acting as f_1 on T_1, and as f_2 on T_2.

Encapsulation
Let H be a subset of $\mathcal{A}$. $\partial_H(G_1) = (N,T,n_0,\mathtt{div},\rightarrow)$ is defined by

$$N = N_1$$
$$T = T_1$$
$$n_0 = n_1^0$$
$$\mathtt{div}(n) \ = \ \mathtt{div}_1(n)$$
$$\rightarrow = \rightarrow_1 \ - \{(n,a,b,f,n') | a \in H\}$$

Execution delay
Let d be an element of D_*, and t be a timer not belonging to T_1.
We define $\lceil G_1 \rceil^d G_2 = (N,T,n_0,\mathtt{div},\rightarrow)$ as follows.

$$N = N_1 \cup N_2 \cup (N_1 \times \{n_2^0\})$$
$$T = T_1 \cup T_2 \cup \{t\}$$
$$n_0 = (n_1^0,n_2^0)$$
if $n \in N_i$ then $\mathtt{div}(n) = \mathtt{div}_i(n)$
$$\mathtt{div}(n_1,n_2^0) = \mathtt{ff}$$

$$\frac{n \xrightarrow{a,b,f}_i n'}{n \xrightarrow{a,b,f} n'} \qquad (n_1,n_2^0) \xrightarrow{i,(t=d) \wedge \mathtt{act}(n_1),\mathcal{R}(T_2)} n_2^0$$

$$\frac{n_1 \xrightarrow{a,b,f}_1 n_1' \ , \ a \neq \xi}{(n_1,n_2^0) \xrightarrow{a,b \wedge (t<d),f} (n_1',n_2^0)} \qquad \frac{n_1 \xrightarrow{\xi,b,f}_1 n_1'}{(n_1,n_2^0) \xrightarrow{\tau,b \wedge (t<d),f} n_1'}$$

4.2.2 Correctness of the translation

The models of an **ATP** process and of the corresponding timed graph are certainly not strongly equivalent, since the latter has transitions labeled by i. However, the following proposition allows to suppress these transitions.

Proposition: Let M be the model of an action timed graph corresponding to a process P. M is *observationally equivalent* [Mil89] (with i as the internal action) to a transition system M' without i-transitions. □

If we saturate M' for d-transitions, that is, if we add a transition $q \xrightarrow{d+d'} q''$ for every pair of transitions $q \xrightarrow{d} q'$ and $q' \xrightarrow{d'} q''$, then the resulting transition system is *strongly equivalent* to the model of P.

4.3 Example: the login procedure

To illustrate the translation of an **ATP** process into a timed graph, we present the example of the *login procedure* of [NS90].

To start the procedure, the system sends a login prompt p. After this event, the user has l time units to enter a valid login response v to successfully finish the task and to start a session phase S. When the user supplies an invalid response n, or when time l elapsed, a new login procedure is started. This behavior can be modeled in **ATP** as follows:

$$L = \dot{p}\left((\tilde{v}\,S \oplus \tilde{n}\,L) \overset{l}{\triangleright} L\right)$$

L is a sequential process. Then, it is possible to construct a one-timer relative-time timed graph. Figure 3 shows the timed graph of L. The divergence condition in a node is written inside the node.

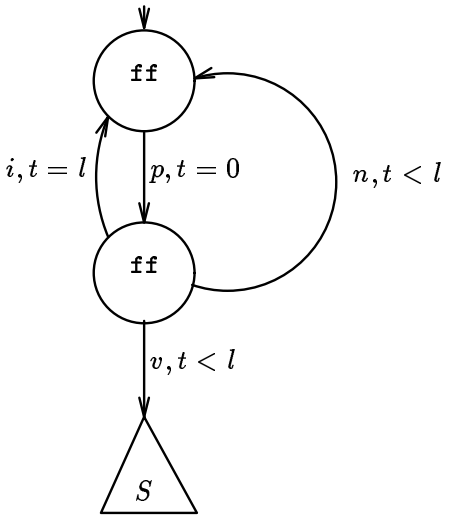

Figure 3: Simple login procedure' timed graph

In addition, the system usually limits the overall duration of the login procedure to a maximum time of d units and forces an exception E to be carried out after that. We can model this process

P using the execution delay operator and adding a cancel action ξ in L to quit when a successful response v is provided.

The complete procedure is then described by P, where

$$\begin{aligned} P &= \lceil L \rceil^{d}(E) \\ L &= \dot{p}\left((\widetilde{v}\,\dot{\xi}\,S \oplus \widetilde{n}\,L) \overset{l}{\triangleright} L\right) \end{aligned}$$

The whole login procedure P is not a sequential **ATP** process. Figure 4 shows the timed graph of P in which two timers are used: t_1, with an upper bound d, to limit the overall duration of L, and the timer t of L. Since the function f of a transition is always a resetting of some timers, we write the set of these timers instead of functions. For sake of simplicity, the reset part of a transition is omitted if it is empty.

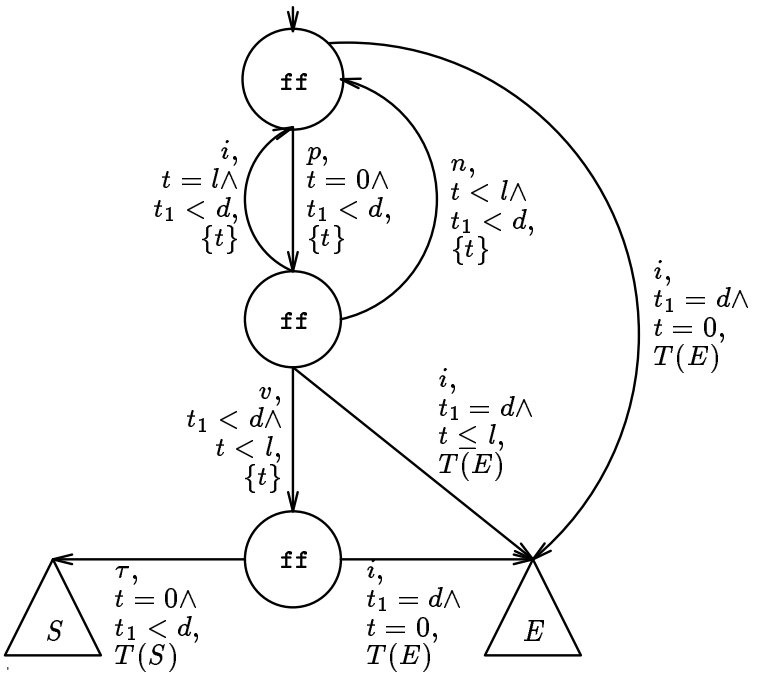

Figure 4: Complete login procedure' timed graph

5 Hybrid Systems

This section presents a model for *hybrid systems*, that is, systems that may evolve

- either by executing transitions, i.e., discrete state changes realised by modification of the state by performing atomic timeless operations;
- or by performing some transformation of the state according to a law described as a function of a time parameter.

The model presented here is a straightforward generalization of Action Timed Graphs obtained by adding to the set of timers T a set of state variables X that do not change linearly with respect to time, but according to an evolution function.

5.1 Definition

A *hybrid system* with action vocabulary A is a structure $(N, T, X, n_0, T_0, X_0, \rightarrow, \varphi)$, where

- N is the set of nodes;
- T is an m-uple $(t_1, ..., t_m)$ of timers, taking its value in D^m;
- X is a p-uple $(x_1, ..., x_p)$ of state variables taking its value in some domain D_X;
- $n_0 \in N$, $T_0 \in D^m$ and $X_0 \in D_X$ are the initial node, initial value of T and initial value of X respectively;
- $\rightarrow \subseteq N \times A \times \mathcal{B}(T,X) \times \mathcal{F}(T,X) \times N$ is the transition function. As for timed graphs, we write $n \xrightarrow{a,b,f} n'$ instead of $(n, a, b, f, n') \in \rightarrow$;
- $\varphi : (N \times D^m \times D_X \times D) \rightarrow D_X$ is the *evolution function* of the state variables. $\varphi(n, T_1, X_1, d)$ is written $\varphi_n[T_1, X_1](d)$ and is the value of X, d time units after the instant when the timers had value T_1 and the state variables had value X_1. The following requirements should be satisfied by the evolution function.

$$\begin{aligned} \varphi_n[T,X](0) &= X \\ \varphi_n[T,X](d+d') &= \varphi_n[T+d, \varphi_n[T,X](d)](d') \end{aligned}$$

The first one expresses the fact that the state cannot change within zero duration. The second one ensures that the models satisfy the time-continuity property, as it expresses the fact that the state $< T+d+d', \varphi_n[T,X](d+d') >$, reached from $< T, X >$ within time $d+d'$, is the same as the state reached by transforming successively $< T, X >$ during a time period d to reach a state $< T', X' > = < T+d, \varphi_n[T,X](d) >$, and $< T', X' >$ during a time period of d'.

5.2 Operational semantics

The model of a hybrid system is a labeled transition system, where states are pairs (node, values of timers and state variables), and labels are elements of $A \cup D_*$. The transition relation is given by the following rules.

$$\frac{n \xrightarrow{a,b,f} n' \ , \ b(T,X)}{(n, < T, X >) \xrightarrow{a} (n', f(T,X))} \qquad \frac{\exists d' \geq d \ \mathbf{en}(n)(T+d', \varphi_n[T,X](d'))}{(n, < T, X >) \xrightarrow{d} (n, < T+d, \varphi_n[T,X](d) >)}$$

The first rule is the same as the rule specifying discrete transitions of action timed graphs. The second rule says that from a given state $< T, X >$ the state resulting after evolution during d

is $< T + d, \varphi_n[T, X](d) >$. The enabling condition allows such an evolution only if at the state reached some transition is eventually enabled.

It is important to notice that hybrid systems have their models in the same class as those of **ATP**. Thus, all the operators of **ATP** can be defined for hybrid systems, in the sense that one can associate a model to any recursion-free expression built from hybrid systems and operators of **ATP**.

5.3 Examples of hybrid systems

5.3.1 A temperature regulator

The temperature θ of a room is controlled by a thermostat commanding a heater. The thermostat is provided with two parameter: θ_m and θ_M which determine the interval in which the temperature of the room is supposed to remain. The thermostat switches ON and OFF a heater, in order to satisfy this requirement. Initially the heater in is ON and the temperature of the room is 0.

Figure 5 describes the hybrid system controlling the room temperature. There is no timer, and only one state variable θ. When the heater is OFF the temperature decreases according to the law $\varphi_{OFF}[\theta](d) = \theta\, e^{-Kd}$. When the heater is ON we take $\varphi_{ON}[\theta](d) = \theta\, e^{-Kd} + \theta_h(1 - e^{-Kd})$ where θ_h is a constant depending on the power of the heater, and K is a constant.

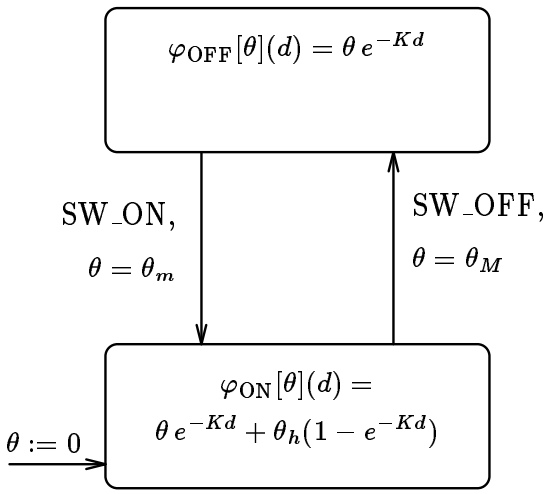

Figure 5: Hybrid system of the temperature regulator

5.3.2 A bouncing ball

We consider the two-dimensional motion of a ball in a room of width l (figure 6). Its position is determined by the coordinates x and y, and its velocity by the two components v_x and v_y. The ball is supposed to start from point $(0, h)$, with a horizontal speed v_0.

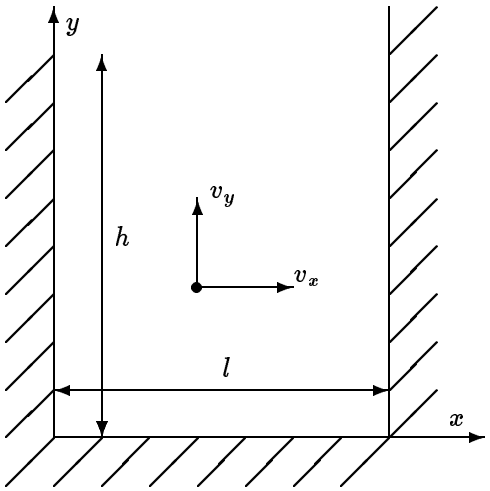

Figure 6: The room

Since the evolutions of x and v_x are independent from those of y and v_y, we can model this motion as the parallel composition of two hybrid systems presented in figure 7 (action names are omitted on the transitions). Both systems have not timer. The states variables of the left hand side system are x and v_x; those of the right hand side system are y and v_y.

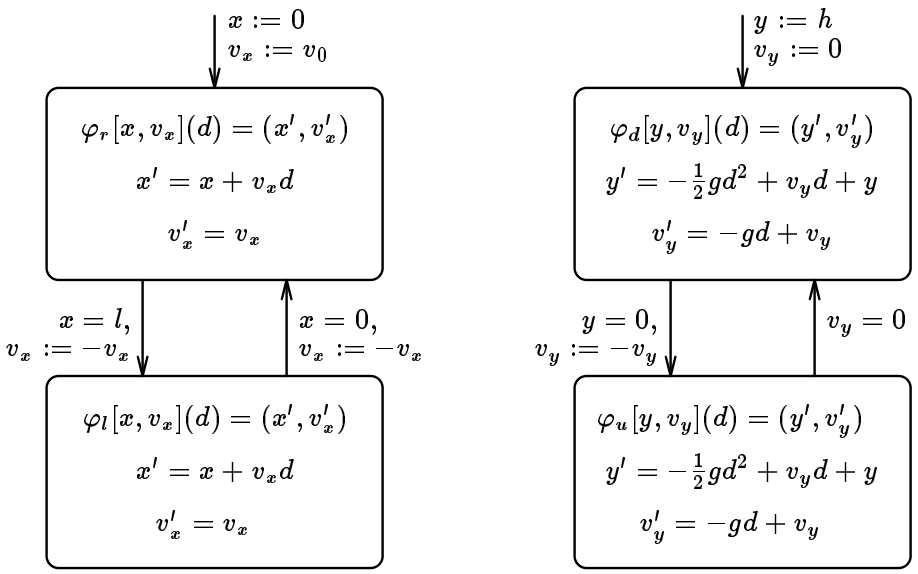

Figure 7: Hybrid systems of the ball's displacement

6 Conclusion

The paper presents results of ongoing work aiming at the unification of some behavioral description formalisms for timed systems.

- We propose for **ATP** a very general semantics in terms of a time domain. This semantics agrees with the semantics in [NS90] when the time domain is discrete.
- It is shown that **ATP** can be translated into a variant of timed graphs. This result allows the application of existing model-checking techniques to **ATP**.
- Finally, we propose a notion of hybrid systems as a generalization of timed graphs. Such systems can evolve (modify their state) either by executing a transition (discrete changes) or by performing some "continuous" transformation according to a law described by a function.

The formalisms studied admit the same class of models: time deterministic and time continuous possibly infinitely branching transition systems labeled by actions or durations.

Acknowledgement: This work has been initiated due to fruitful discussions between the authors and C. Courcoubetis when the latter visited IMAG, in September 1990.

References

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proceedings of the fifth annual IEEE symposium on Logics In Computer Science*, pages 414–425, Philadelphia, PA, USA, June 1990.

[BB90] J.C.M. Baeten and J.A. Bergstra. *Real Time Process Algebra.* Technical report CS-R9053, Centre for Mathematics and Computer Science, Amsterdam, The Netherlands, 1990.

[BC84] G. Berry and L. Cosserat. The ESTEREL synchronous programming language and its mathematical semantics. In *LNCS 197, proceedings CMU Seminar on Concurrency*, pages 389–448, Springer Verlag, 1984.

[BK84] J.A. Bergstra and J.W. Klop. *Algebra of Communicating Processes.* Technical report CS-R8420, Centre for Mathematics and Computer Science, Amsterdam, The Netherlands, 1984.

[BK86] J.A. Bergstra and J.W. Klop. Process algebra : specification and verification in bisimulation semantics. In M. Hazewinkel, J.K. Lenstra, and L.G.L.T. Meertens, editors, *Proceedings of the CWI Symposium Mathematics and Computer Science II (CWI Monograph 4)*, pages 61–94, North Holland, Amsterdam, 1986.

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

[MT90] F. Moller and C. Tofts. A Temporal Calculus of Communicating Processes. In J.C.M. Baeten and J.W. Klop, editors, *LNCS 458, proceedings of CONCUR '90 (Theories of Concurrency: Unification and Extension), Amsterdam, The Netherlands*, pages 401–415, Springer Verlag, August 1990.

[NRSV90] X. Nicollin, J.-L. Richier, J. Sifakis, and J. Voiron. ATP: an Algebra for Timed Processes. In *Proceedings of the IFIP TC 2 Working Conference on Programming Concepts and Methods*, Sea of Galilee, Israel, April 1990.

[NS90] X. Nicollin and J. Sifakis. The algebra of timed processes ATP: theory and application. *Information and Computation*, (to appear), December 1990.

[Plo81] G.D. Plotkin. *A Structural Approach to Operational Semantics.* Technical report DAIMI FN-19, Århus University, Computer Science Department, Århus, Denmark, 1981.

[Wan90] Wang Yi. Real-time behaviour of asynchronous agents. In J.C.M. Baeten and J.W. Klop, editors, *LNCS 458, proceedings of CONCUR '90 (Theories of Concurrency: Unification and Extension), Amsterdam, The Netherlands*, pages 502–520, Springer Verlag, August 1990.

[Wan91] Wang Yi. CCS + Time = an Interleaving Model for Real Time Systems. In *Proceedings of ICALP '91, Madrid, Spain*, July 1991.