Basic Research in Computer Science

BRICS NS-01-5 Corradini & Vogler (eds.): MTCS '01 Proceedings

Preliminary Proceedings of
the 2nd International Workshop on

Models for Time-Critical Systems

MTCS '01

Aalborg, Denmark, August 25, 2001

Flavio Corradini
Walter Vogler
(editors)

BRICS Notes Series **NS-01-5**

ISSN 0909-3206 **August 2001**

**See back inner page for a list of recent BRICS Notes Series publications.
Copies may be obtained by contacting:**

**BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark**

**Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk**

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `NS/01/5/`

Second International Workshop on Models for Time-Critical Systems

MTCS 2001

Aalborg, Denmark
August 25, 2001

Editors:
Flavio Corradini, University of L'Aquila, Italy
Walter Vogler, University of Augsburg, Germany

This is a preliminary version.
The final version is considered for publication in
Electronic Notes in Theoretical Computer Science
http://www.elsevier.nl/locate/entcs

Table of Contents

Foreword

A large class of systems can be specified and verified by abstracting away from the temporal aspects. This is the class of systems where time affects the performance but not the functional behaviour. In time-critical systems, instead, time issues become essential. Their correctness depends not only on which actions a system can perform but also on their execution time. Due to their importance, time-critical systems have attracted the attention of a considerable number of computer scientists from various research areas.

This volume contains the *preliminary proceedings* of the 2nd International Workshop on Models for Time-Critical Systems (MTCS 2001); MTCS 2001 was held on 25 August 2001 as one of five satellite workshops co-located with the 12th International Conference on Concurrency Theory (CONCUR 2001), held in Aalborg (Denmark) 21-24 August 2001.

The first workshop, MTCS 2000, was held in State College (Pennsylvania, USA) on 26 August 2000, co-chaired by Flavio Corradini and Paola Inverardi. As for MTCS 2000, the objectives of MTCS 2001 were (i) to validate the more promising proposals on models for time-critical systems, ranging from theory to practice and (ii) to promote interaction between different research areas in the field of time-critical systems. Despite its focus on time-critical systems, MTCS 2001 was also open for more general time-related issues.

The seven papers in this volume were selected for presentation by the Program Committee from submissions received in response to a Call for Papers. The final versions of these papers are considered for publication in Electronic Notes in Theoretical Computer Science: http://www.elsevier.nl/locate/entcs. The volume includes the contribution by the Invited Speaker Jos Baeten (Eindhoven University of Technology).

We would like to thank Kim G. Larsen and Mogens Nielsen (CONCUR 2001 Conference Chairs) and Hans Hüttel (Satellite Workshops Chair) for the opportunity they gave us to organize MTCS 2001 and for their support. Many thanks are due to Jos Baeten (Invited Speaker), and to the members of the Program Committee as well as their sub-referees for their accurate work. We would also like to thank Michael Mislove for his help during the proceedings editorial process and to BRICS for the publication of the preliminary proceedings.

Flavio Corradini, University of L'Aquila, Italy
Walter Vogler, University of Ausburg, Germany

MTCS 2001 - Program Committee

Rajeev Alur (USA)	Jos Baeten (NL)	Frank de Boer (NL)
Flavio Corradini (IT)	Paola Inverardi (IT)	Jeff Kramer (UK)
Gerald Lüttgen (UK)	Jeff Magee (UK)	Andrea Maggiolo-Schettini (IT)
Bran Selic (CA)	Joseph Sifakis (FR)	Walter Vogler (DE)

Timed Process Algebras [1]

Jos Baeten [2]

Computing Science Department
Eindhoven University of Technology
P.O.Box 513, 5600 MB
Eindhoven, The Netherlands

We present an integrated framework of several process algebras dealing with timing. The timing of actions is either relative (to the time at which the preceding action is performed) or absolute, the algebraic format can include time-stamped actions or be two-phase (delays and actions separate), and the time scale on which time is measured is either discrete or continuous.

The presented theories are all extensions of ACP with explicit termination. All theories are given by operational rules and are axiomatized. We have an integrated framework in the following sense. All theories are generalizations of ACP without timing, and the theories with relative timing and absolute timing in which time is measured on a continuous time scale are generalizations of the theories with relative timing and absolute timing in which time is measured on a discrete time scale. Besides, the theories with absolute timing can easily be extended with a mechanism for parametric absolute timing which provides an alternative way to deal with relative timing. That is, the extended theories are generalizations of the corresponding theories with relative timing. In all cases, time-stamped and two-phase versions are interdefinable.

If one theory is a generalization of another theory, this roughly means that the processes considered in the former theory essentially include the processes considered in the latter theory. That is the reason why abstraction from timing is possible in a theory with timing and discretization is possible in a theory with continuous timing. Abstraction from timing enables analysis of systems without carrying the timing details wherever they are not needed, discretization enables analysis of systems at a level where time is measured with a finite precision wherever that is sufficient.

In this framework, we can define many relevant features: strong choice vs. weak choice, maximal progress, time outs, state operator, process creation.

[1] Joint work with C.A. Middelburg and M.A. Reniers.

[2] Email: jos@win.tue.nl

This is a preliminary version. The final version is considered for publication in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

References

[1] J.C.M. Baeten and C.A. Middelburg. Process algebra with timing: real time and discrete time. In: Handbook of Process Algebra, eds. J.A. Bergstra, A. Ponse and S.A. Smolka, North-Holland 2001, pp. 627–684.

[2] J.C.M. Baeten and M.A. Reniers. Termination in timed process algebra. Report CSR 00-13, Dept. of Comp. Sci., Technische Universiteit Eindhoven 2000.

Timed Automata with Urgent Transitions

Roberto Barbuti and Luca Tesei [1,2]

Dipartimento di Informatica
Università di Pisa
Corso Italia, 40
56125 Pisa - Italy

Abstract

In this paper we propose an extension to the formalism of timed automata by allowing urgent transitions. A urgent transition is a transition which must be taken within a fixed time interval from its enabling time. We give a set of rules formally describing the behaviour of urgent transitions and we show that, from a language theoretic point of view, the addition of urgency does not improve the expressive power of timed automata. However, from a specification point of view, the use of urgent transitions is crucial, especially in modular specification of systems.

Keywords: real-time systems, timed automata, modular specification, parallel composition.

1 Introduction

Timed automata are widely recognized as a standard model for describing systems in which the time plays a fundamental role [6,7]. Since their introduction, timed automata have been widely studied from different points of view [4,5,8,9], in particular for their possible use in the verification of real-time systems [1,2,3,10,19,20,22].

Usually the expressiveness of timed automata is given in terms of accepted timed languages, but, because of their use as a specification formalism, also the ease to describe real-time systems must be taken into account. For this purpose many extensions to the basic model have been proposed (see for example [11,16,17,18,21]). All these extensions have been discussed with respect to the expressiveness of the original model.

In this paper we present a further extension: *timed automata with urgent transitions*. The notion of urgency in timed automata was already introduced

[1] Email: barbuti@di.unipi.it
[2] Email: tesei@di.unipi.it

This is a preliminary version. The final version is considered for publication in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

in [13,14,15], where the urgency of transitions outgoing from a state is induced by the impossibility (due to the failure of a condition) to stay in the state while the time elapses. In this paper we consider a slightly different notion of urgency. Urgent transitions are transitions which must be performed within a given time interval starting from their enabling.

From the expressiveness point of view, both the approaches are suitable for specifying timed systems. The approach in [13,14,15] allows, in some cases, more general urgency conditions, while, in other cases, such as "as soon as possible" transitions, our approach allows more general time constraints.

In this preliminary version of the paper we impose that at most one urgent transition can exit from a state. We precisely define such a behavior by means of an operational semantics.

We show that, from the language theoretic point of view, timed automata and timed automata with urgent transitions are equivalent. This is proved by defining a transformation from a timed automaton with urgent transitions to a timed automaton which accepts the same language. However, the transformation is not a congruence with respect to parallel composition, thus it cannot be applied to a component which is supposed to be used in different environments. Thus, the use of urgent transitions is fundamental for the modular specification of systems.

2 Timed automata

We recall the definition of timed automata [7]. In the following, $\mathbb{R}$ is the set of real numbers and $\mathbb{R}^+$ the set of non-negative real numbers. $\mathbb{Q}$ is the set of rational numbers and $\mathbb{Q}^+$ is the set of positive rational numbers. A *clock* takes values from $\mathbb{R}^+$. Given a set $\mathcal{X}$ of clocks, a *clock valuation* over $\mathcal{X}$ is a function assigning a non-negative real number to every clock. The set of valuations of $\mathcal{X}$, denoted $\mathcal{V}_\mathcal{X}$, is the set of total function from $\mathcal{X}$ to $\mathbb{R}^+$. Given $\nu \in \mathcal{V}_\mathcal{X}$ and $\delta \in \mathbb{R}^+$, with $\nu + \delta$ (resp. $\nu - \delta$) we denote the valuation that maps each clock $x \in \mathcal{X}$ into $\nu(x) + \delta$ (resp. $\nu(x) - \delta$). Note that if there exists $x \in \mathcal{X}$ such that $\nu(x) - \delta < 0$, $\nu(x) - \delta$ is not a clock evaluation.

Given a set $\mathcal{X}$ of clocks, a *reset* γ is a subset of $\mathcal{X}$. The set of all resets of $\mathcal{X}$ is denoted by $\Gamma_\mathcal{X}$. Given a valuation $\nu \in \mathcal{V}_\mathcal{X}$ and a reset γ, with $\nu\backslash\gamma$ we denote the valuation

$$\nu\backslash\gamma(x) = \begin{cases} 0 & \text{if } x \in \gamma \\ \nu(x) & \text{if } x \notin \gamma \end{cases}$$

Given a set $\mathcal{X}$ of clocks, the set $\Psi_\mathcal{X}$ of *clock constraints* over $\mathcal{X}$ are defined by the following grammar:

$$\psi ::= true \mid false \mid \psi \wedge \psi \mid x\#t$$

where $x, y \in \mathcal{X}$, $t \in \mathbb{N}$ is a natural number, and $\#$ is a binary operator in $\{<, >, \leq, \geq, =\}$. Note that the negation operator is not needed because the negation of an atomic constraint $x\#t$ ($\#$ different from $=$) can be expressed as another constraint of the same kind. The negation of a constraint $x = t$ can be expressed by $x < t \vee x > t$. The disjunction can be simulated, as usual, by duplicating the edges in the automaton. Actually, in this version of the work, we do not allow more than one urgent action outgoing from a state, thus we have a restriction on the constraints of such actions.

Clock constraints are evaluated over clock valuations. The satisfaction by a valuation $\nu \in \mathcal{V}_{\mathcal{X}}$ of the clock constraint $\psi \in \Psi_{\mathcal{X}}$, denoted $\nu \models \psi$, is defined as follows:

$\nu \models true$ and $\nu \not\models false$

$\nu \models \psi_1 \wedge \psi_2$ iff $\nu \models \psi_1 \wedge \nu \models \psi_2$

$\nu \models x\#t$ iff $\nu(x)\#t$

Definition 2.1 [Timed automaton] A timed automaton T is a tuple $(Q, \Sigma, \mathcal{E}, I, R, \mathcal{X})$, where: Q is a finite set of states, Σ is a finite alphabet of actions, $\mathcal{E}$ is a finite set of edges, $I \subseteq Q$ is the set of initial states, $R \subseteq Q$ is the set of repeated states, $\mathcal{X}$ is a finite set of clocks. Each edge $e \in \mathcal{E}$ is a tuple in $Q \times \Psi_{\mathcal{X}} \times \Gamma_{\mathcal{X}} \times \Sigma \times Q$.

If $e = (q, \psi, \gamma, \sigma, q')$ is an edge, q is the *source*, q' is the *target*, ψ is the *constraint*, σ is the *label*, γ is the *reset*.

The semantics of a timed automaton T is given in terms of accepted timed language. The definition of such a language is based on an infinite transition system $\mathcal{S}(T) = (S, \rightarrow)$, where S is a set of states and $\rightarrow$ is the transition relation. The states S of $\mathcal{S}(T)$ are pairs (q, ν), where $q \in Q$ is a state of T, and ν is a valuation. An initial state of $\mathcal{S}(T)$ is a state (q, ν), where $q \in I$ is an initial state of T and ν is the valuation which assigns 0 to every clock in $\mathcal{X}$. At any state q, given a valuation ν, T can stay idle or it can perform an action labeling an outgoing edge e. If T stays idle, a transition is possible to a state of $\mathcal{S}(T)$ where the state of T is the same, but the valuation has been modified according to the elapsed time. If T moves along an outgoing edge $e = (q, \psi, \gamma, \sigma, q')$, this corresponds to a transition, labeled by σ, of $\mathcal{S}(T)$ from the state (q, ν) to the state $q', \nu\backslash\gamma$. This transition is possible only if the current clock valuation respects the constraint ψ of e. The rules to derive the transitions of $\mathcal{S}(T)$ are the following:

$$1.\ \frac{\delta \in \mathbb{R}^+}{(q, \nu) \xrightarrow{\delta} (q, \nu + \delta)} \qquad 2.\ \frac{(q, \psi, \gamma, \sigma, q') \in \mathcal{E}, \nu \models \psi}{(q, \nu) \xrightarrow{\sigma} (q', \nu\backslash\gamma)}$$

Rule 1. represents the case in which T stays idle in a state and the time passes, while Rule 2. corresponds to the occurrence of an action.

Definition 2.2 [run, action sequence] Given a timed automaton $T = (Q, \Sigma, \mathcal{E}, I, R, \mathcal{X})$, a *run* of the automaton is an infinite sequence of states and transitions of $\mathcal{S}(T)$

$s_0 \xrightarrow{l_0} s_1 \xrightarrow{l_1} \ldots$

where

- $s_0 = (q, \nu)$ where $q \in I$ and $\nu(x) = 0$ for every $x \in \mathcal{X}$
- a state $q \in R$ exists such that q occurs infinitely often in the pairs of the sequence $\{s_i\}$

Note that, given a run $s_0 \xrightarrow{l_0} s_1 \xrightarrow{l_1} \ldots$, for each i, $l_i \in (\Sigma \cup \mathbb{R}^+)$. Let r be a run.

- The *time sequence* $\overline{t_j}$ of the time elapsed from state s_0 to state s_j in r is defined as follows:

  $t_0 = 0$

  $$t_{i+1} = t_i + \begin{cases} 0 \text{ if } l_i \in \Sigma \\ l_i \text{ otherwise} \end{cases}$$

- The *event sequence* of the events occurring during r, including the elapsed times, is defined as follows:

  $(l_0, t_0)(l_1, t_1) \ldots$

- The *action sequence* of r is the projection of the event sequence of r on the pairs $\{(l, t) | l \in \Sigma\}$

Definition 2.3 [timed word, timed language] Let Σ be an alphabet. A *timed word* over Σ is an infinite sequence of pairs $(\sigma_0, t_0)(\sigma_1, t_1) \ldots$ such that $\sigma_i \in \Sigma$, and $t_i \in \mathbb{R}^+$, $t_i \leq t_i + 1$, for all i.

A *timed language* over Σ is a subset of the set of all timed words over Σ.

Definition 2.4 [acceptance] Given a timed automaton $T = (Q, \Sigma, \mathcal{E}, I, R, \mathcal{X})$, a timed word w over Σ is *accepted* by T if a run r of T exists such that $w = v$, where v is the action sequence of r. The set of timed words accepted by T is called the *accepted language* of T.

Note that we use the Büchi acceptance condition for the runs.

A parallel composition is defined on timed automata. Here we recall the definition given in [7].

Definition 2.5 [Product] Let $T_1 = (Q_1, \Sigma_1, \mathcal{E}_1, I_1, \mathcal{X}_1)$ and $T_2 = (Q_2, \Sigma_2, \mathcal{E}_2, I_2, \mathcal{X}_2)$ be two timed transition tables with $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$. The product of T_1 and T_2, denoted by $T_1 \parallel T_2$, is given as follows:

$$T_1 \| T_2 = \langle Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \mathcal{E}, I_1 \times I_2, R', \mathcal{X}_1 \cup \mathcal{X}_2 \rangle$$

where $\mathcal{E}$ is defined by:

(i) **Synchronization actions**
$\forall \sigma \in \Sigma_1 \cap \Sigma_2, \forall (q_1, \psi_1, \gamma_1, \sigma, q_1') \in \mathcal{E}_1, \forall (q_2, \psi_2, \gamma_2, \sigma, q_2') \in \mathcal{E}_2$
$\mathcal{E}$ contains $((q_1, q_2), \psi_1 \wedge \psi_2, \gamma_1 \cup \gamma_2, \sigma, (q_1', q_2'))$

(ii) T_1 **actions**
$\forall \sigma \in \Sigma_1 \backslash \Sigma_2, \forall (q, \psi, \gamma, \sigma, q') \in \mathcal{E}_1, \forall s \in Q_2$
$\mathcal{E}$ contains $((q, s), \psi, \gamma, \sigma, (q', s))$

(iii) T_2 **actions**
$\forall \sigma \in \Sigma_2 \backslash \Sigma_1, \forall (q, \psi, \gamma, \sigma, q') \in \mathcal{E}_2, \forall s \in Q_1$
$\mathcal{E}$ contains $((s, q), \psi, \gamma, \sigma, (s, q'))$

This definition shows what we expect in parallel behaviors:

- Common symbols of the alphabets are synchronization actions. A synchronization action can be executed if and only if all the component automata involved can execute it. The action must be executed synchronously by all of them.
- Other symbols can be executed by each component independently according to its original specification.

Defining the set of repeated states, R', of a parallel composition according to the Büchi acceptance condition requires a slight different construction with a lot of details. We refer to [7] for this.

3 Timed automata with urgent transitions

In this section we extend the model of timed automata with a new feature which is useful in the specification of a real-time systems. The idea is to provide, in each state of the automaton, the possibility of labeling one of the outgoing edges as *urgent*. Intuitively the labeled edge must be taken with higher priority with respect to the others, provided that its constraint is satisfied by the current clock valuation.

To be more precise, we introduce a constant $\ell \in \mathbb{Q}^+$ which represents the length of a time interval in which an enabled urgent action must be executed. The time interval is $[t, t + \ell)$ where t is the instant in which the constraint associated to the urgent transition becomes satisfied by the current clock valuation.

Choosing this notion of urgency allow us to define precisely the behavior of urgent actions. The intuitive idea "urgent transitions must be taken as soon as possible" introduces some problems when applied in a model with a dense time domain. Consider a state of a timed automaton in which the current value of clock x is in $[0, 1]$ and there is an outgoing urgent transition with a clock constraint $x > 1$. Letting the time to elapse, at which time would the urgent transition be executed? It is not possible to answer precisely to this question since the time domain is dense. To avoid this problem we introduced the constant ℓ and the interval $[0, \ell)$ the action must be executed within. The

choice of a right-open interval is also related to the denseness. Suppose, now, $\ell = 1$. If we chose an interval $[0, \ell]$ the upper bound of the interval within the urgent transition must be executed would not be precisely defined. This is because the lower bound is not precisely defined. The right-open interval allow us to express the upper bound as $x \leq 2$. If the transition has a constraint in the form of $x \geq 1$ then the upper bound is expressed by $x < 2$. The denseness also imposes the constant ℓ be greater than 0. The choice $\ell = 0$ could be interpreted as "immediately", but this leads to the problem discussed above. However, since $\ell \in \mathbb{Q}^+$, it can be chosen as small as we want. In other words, the "as soon as possible" limit behavior can be approximated with arbitrary precision.

If the constraint of the urgent transition is already satisfied when a state is entered then the interval starts at the instant in which the state is entered. We have chosen this approach for a uniform treatment of urgent transitions.

Note that the specification of ℓ could be *local* to each urgent transition. For the sake of simplicity we discuss the case in which ℓ is a global parameter. The case of local specification can be caught by a slight modification of the definition, the semantics and the transformation.

If a state has no urgent outgoing edges then the behavior is the usual one of timed automata. This also happens when a state is entered and the urgent transition is not enabled.

Moreover, when an urgent transition is enabled in a state, the unique way to continue the run is to execute it, both while the associated constraint is satisfied and within the interval specified above.

Definition 3.1 [Timed automaton with urgent transitions] Let $\ell \in \mathbb{Q}^+$ be a constant. A timed automaton with urgent transitions T_u^ℓ is a tuple $(Q, \Sigma, \mathcal{E}, \mathcal{U}, I, R, \mathcal{X})$, where: Q is a finite set of states, Σ is a finite alphabet of actions, $\mathcal{E}$ and $\mathcal{U}$ are finite sets of edges, the non-urgent and the urgent ones, $I \subseteq Q$ is the set of initial states, $R \subseteq Q$ is the set of repeated states, $\mathcal{X}$ is a finite set of clocks. Each edge $e \in \mathcal{E} \cup \mathcal{U}$ is a tuple in $Q \times \Psi_{\mathcal{X}} \times \Gamma_{\mathcal{X}} \times \Sigma \times Q$. $\mathcal{U}$ is the set of urgent transitions. To impose that at most one urgent transition can exit from a state we require that $(q_1, \psi_1, \gamma_1, \sigma_1, q_1'), (q_2, \psi_2, \gamma_2, \sigma_2, q_2') \in \mathcal{U}$ iff $q_1 \neq q_2$.

The class of all timed automata with urgent transitions will be denoted by $\mathcal{T}_u^\ell$.

In the following the superscript ℓ could be omitted and, when this happens, it should be considered implicitly defined.

For timed automata with urgent transitions T_u^ℓ we define an infinite transition system $\mathcal{S}(T_u^\ell) = (S_u, \rightarrow)$ as for timed automata. The states S_u are triples (q, ν, δ_q) such that $q \in Q$ is the current state of the automata T_u^ℓ, ν is the current clock valuation and $\delta_q \in \mathbb{R}^+ \cup \{0\}$ is a number recording the time elapsed since the state q has been entered. The rules to derive the transitions of $\mathcal{S}(T_u^\ell)$ are the following:

(Time) $$\frac{\delta \in \mathbb{R}^+}{(q,\nu,\delta_q) \xrightarrow{\delta} (q,\nu+\delta,\delta_q+\delta)}$$

(Non-Urgent 1) $$\frac{(q,\psi,\gamma,\sigma,q') \in \mathcal{E},\ \nu \models \psi,\ (\neg\exists (q,\psi_u,\gamma_u,\sigma_u,q'_u) \in \mathcal{U})}{(q,\nu,\delta_q) \xrightarrow{\sigma} (q',\nu\backslash\gamma,0)}$$

(Non-Urgent 2) $$\frac{\begin{array}{c}(q,\psi,\gamma,\sigma,q') \in \mathcal{E},\ \nu \models \psi \\ (q,\psi_u,\gamma_u,\sigma_u,q'_u) \in \mathcal{U},\ (\neg\exists \delta.\ 0 \le \delta < \delta_q \wedge \nu-\delta \models \psi_u)\end{array}}{(q,\nu,\delta_q) \xrightarrow{\sigma} (q',\nu\backslash\gamma,0)}$$

(Urgent) $$\frac{(q,\psi_u,\gamma_u,\sigma_u,q') \in \mathcal{U},\ \nu \models \psi_u, (\nu-\ell \not\models \psi_u \vee \delta_q < \ell)}{(q,\nu,\delta_q) \xrightarrow{\sigma_u} (q',\nu\backslash\gamma_u,0)}$$

Rule **(Time)** lets the time elapse in a state and updates both the clock valuation and the time elapsed in the state.

Rule **(Non-Urgent 1)** is used when T_u is in a state without outgoing urgent edges. In this case the behavior is the same as timed automata. Note that when a new state is entered the time elapsed is set to 0.

Rule **(Non-Urgent 2)** manages the case in which T_u is in a state with a (unique by definition) urgent transition. The "$\neg\exists$" condition in the rule requires that the urgent transition has never been enabled since the current state was entered. If this is false the rule is not applicable.

Rule **(Urgent)** executes an urgent action σ. The condition $(\nu-\ell \not\models \psi \vee \delta_q < \ell)$ ensures that an urgent transition is taken either before a time ℓ is elapsed after its enabling time, or the time elapsed in the state is less than ℓ. Without this guard an urgent transition could be fired after the expiry time expressed by ℓ. This would not be sound with respect to the notion of urgency.

Note that if $S(T_u)$ is in a state in which an urgent transition can be executed by the rule **(Urgent)** it cannot be postponed until its constraint becomes false. This because the "$\neg\exists$" condition of rule **(Non-Urgent 2)** will never be true and the run could not proceed.

The notion of run for a timed automata with urgent transitions is defined in the same way as for timed automata using the transition system $S(T_u)$. Similarly for the accepted timed language.

Example 3.2 Figure 1 shows an example of a timed automaton with a urgent transition, indicated by the letter u. In this example we consider $\ell = 1$. The automaton can execute the action b when the value of the clock x is in the interval $(0,1]$. When the value of x becomes greater than 1, b cannot be performed any longer and the urgent action a must be executed. Moreover, because of the urgency, a must be performed while the value of x is in the

interval $(1, 2]$.

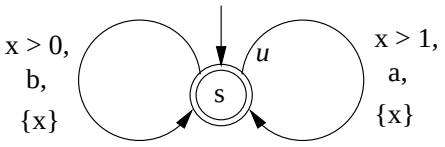

Fig. 1. An automaton with urgent transitions, T_u^1

4 The expressive power of timed automata with urgent transitions

In this section we show that, from a language theoretic point of view, the expressive power of timed automata with urgent transitions is equivalent to the one of timed automata. This is shown by providing a transformation which preserves the accepted language. Because timed automata are special cases of timed automata with urgent transitions, the transformation is only given starting from the latter ones.

In the next section we show that the transformation is not compositional, thus, for specification purposes, the use of urgent transitions is fundamental.

4.1 The region form of a timed automata

Let T_u^l be a timed automaton with urgent transitions. We give a transformation that builds a timed automata accepting the same timed language.

Note that if $\ell = \frac{a}{b}$ with a and b natural numbers, it is always possible to transform a $T_u^{\frac{a}{b}}$ automaton to an isomorphic one T_u^a by multiplying all the constants in the clock constraints by b. So we can assume without loss of generality that ℓ is a positive natural number. For the sake of simplicity we assume in this section that $\ell = 1$, but the transformation can be easily defined for any positive natural number.

Given a set of clocks $\mathcal{X}$, a clock region, as defined in [7], is an equivalence class of clock evaluations such that, given two clock evaluations ν and ν' belonging to it, for every clock constraint ψ, $\nu \models \psi$ iff $\nu' \models \psi$. Note that, given a timed automaton T and a set of clocks $\mathcal{X}$, the clock regions are finite. Let us denote such a set by $\mathtt{Reg}(T, \mathcal{X})$. We denote the equivalence class of a clock evaluation ν as $[\nu]$. A clock region $\alpha \in \mathtt{Reg}(T, \mathcal{X})$ can be uniquely identified by specifying, for every clock $x \in \mathcal{X}$, one clock constraint of the set $C_x = \{x = c | c = 0, 1, \ldots, c_x\} \cup \{c - 1 < x < c | c = 1, 2, \ldots c_x\} \cup \{x > c_x\}$ where c_x is the greatest constant to which x is compared in the constraints of T. Moreover, for every pair of clocks x and y such that we specified $c - 1 < x < c$ and $d - 1 < y < d$, for some c, d, an inequality of type $\mathtt{fract}(x) \# \mathtt{fract}(y)$

where $\# \in \{<, =, >\}$ must be specified. Here $\mathtt{fract}(x)$ is the fractional part of the value of clock x.

Given a clock region $\alpha \in \mathtt{Reg}(T, \mathcal{X})$ and $x \in \mathcal{X}$ we denote by $R_T(\alpha, x)$ the unique clock constraint in C_x specifying α.

In [7] it is shown how to construct, given a clock region $\alpha \in \mathtt{Reg}(T, \mathcal{X})$, the ordered set of clock regions that are *time successors* of α. We denote such set by $\mathtt{succ}(\alpha)$. The order $\leq_\alpha$ of the clock regions in the set $\mathtt{succ}(\alpha)$ is total and such that $\alpha \leq_\alpha \alpha'$ iff α' is a time successor of α.

Given a clock region $\alpha \in \mathtt{Reg}(T, \mathcal{X})$ and a reset $\gamma \subseteq \mathcal{X}$, we denote by $[\gamma \to 0]\alpha$ the clock region such that, for all $x \in \gamma$, the constraint in α for x is substituted by $x = 0$.

In the following we need a transformation of clock constraints ψ which gives a *logically equivalent* constraint $\mathtt{min}(\psi)$ such that it does not contain redundancies. Essentially the transformation drop from ψ the atomic constraints which are implied by others, yielding a minimal conjunction of constraints. In other words, for each clock $x \in \mathcal{X}$, there is only one constraint in $\mathtt{min}(\psi)$ of the forms $x = c$, $x \# c$, $c \# x \# 'd$ or $c \# x$, where $\#, \#' \in \{<, \leq\}$. Let us denote by $\mathtt{select}(\mathtt{min}(\psi), x)$ such unique constraint.

The following definition describes a first transformation, in region form, of a timed automaton with urgent transitions. To this purpose a state of the transformed automaton records both the state of the original one and the equivalence class (clock region) of the values of clocks when the state is entered.

Definition 4.1 Let $T_u = (Q, \Sigma, \mathcal{E}, \mathcal{U}, I, R, \mathcal{X})$ be a timed automaton with urgent transitions.
The corresponding timed automaton *in region form*,
$T_u^r = (Q^r, \Sigma, \mathcal{E}^r, \mathcal{U}^r, I^r, R^r, \mathcal{X})$
is defined as follows:

- the states in Q^r and R^r are of the form $\langle q, \alpha \rangle$ where $q \in Q$ and α is a clock region,
- the states in I^r are of the form $\langle q, [\nu_0] \rangle$ where $q \in I$ and $\nu_0(x) = 0$ for all $x \in \mathcal{X}$
- $(\langle q, \alpha \rangle, \mathtt{min}(\psi) \wedge \bigwedge_{x \in \mathcal{X}} R_{T_u}(\alpha'', x), \gamma, \sigma, \langle q', [\gamma \to 0]\alpha'' \rangle) \in \mathcal{E}^r$ (resp. $\mathcal{U}^r$) iff $(q, \psi, \gamma, \sigma, q') \in \mathcal{E}$ (resp. $\mathcal{U}$), $\alpha \in \mathtt{Reg}(T_u, \mathcal{X})$, and $\alpha'' \in \mathtt{succ}(\alpha)$.

Note that the new states are built exactly as the ones of the region automaton as defined in [7].

This construction differs from the one for region automata because constraints and resets are maintained on the edges. These constraints are modified in order to force the corresponding edge to enter only one of the time successor clock regions (in the sense that for other regions the constraint is always false).

It is important to note that a timed automaton with urgent actions in

region form may not meet the requirement that at most one urgent transition can exit form the same state. This is not a problem because this automaton is intended as an intermediate state in the transformation.

Example 4.2 In Figure 2 it is shown the automaton of Figure 1 in region form, denoted by T_u^{1r}. Note that the constraints explicitly shows the time successor clock region to which they refer. Note that all the edges with a *false* constraint have been removed and, in the states, there is only the $[x = 0]$ region because both the original edges reset x.

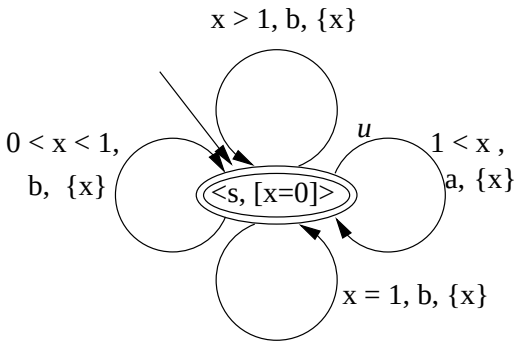

Fig. 2. Automaton T_u^{1r}

4.2 Making the urgent transitions ℓ consistent

The second step of the transformation will adapt the constraints of the urgent transitions of T_u^r making them consistent with the semantics we gave in Section 3. More precisely clock constraints are adapted according to the behavior expressed by the rule **(Urgent)**. In this step we consider only the urgent actions and neglect the other ones which remain unchanged. The third step will adapt these according to the semantics.

Let $\langle q, \alpha \rangle$ be a state of T_u^r such that in state q of T_u there was an outgoing urgent transition $e_q = (q, \psi, \gamma, \sigma, q')$. The latter became, in T_u^r, a set of transitions $E_\alpha^q = \{(\langle q, \alpha \rangle, \mathtt{min}(\psi) \wedge \bigwedge_{x \in \mathcal{X}} R_{T_u}(\alpha'', x), \gamma, \sigma, \langle q', [\gamma \to 0]\alpha'' \rangle) \in \mathcal{U}^r \mid \alpha'' \in \mathtt{succ}(\alpha)\}$. Note that α is the clock region when $\langle q, \alpha \rangle$ is entered by T_u^r and all the outgoing urgent transitions E_α^q are labeled by the same action. We adapt the clock constraints of these to handle the expiration time expressed by ℓ, i.e. to force the action to be executed within ℓ time units from the instant it becomes enabled.

There are three possible cases.

- First, $\mathtt{min}(\psi)$ is implied by α, that is, if $\nu \in \alpha$ then $\nu \models \psi$. In other words the urgent action is already enabled when the state is entered. Here we simply add to each transition in E_α^q a new constraint imposing that the time elapsed in the state be less than $\ell = 1$. To do this we add in T_u^r a new

clock variable for each state. Whenever a state is entered the correspondent clock is reset, so it can be used in the constraints of outgoing edges as a measure of the time elapsed in the state.

- Second, $\texttt{min}(\psi)$ is equivalent to false or it is consistent, but it will never be true letting the time to elapse from α. For the latter case consider, for instance, $\alpha = [x = 1 \wedge y = 2]$ and $\psi = x < 1 \wedge y > 2$. In this case we do nothing.
- Third, $\texttt{min}(\psi)$ is not implied by α and will be implied by a time successor of α. In this case we have to ensure that, starting from the instant in which $\texttt{min}(\psi)$ will become true, the transition will be taken within ℓ time units. This case requires some new notation and definitions. Using the total order $\leq_\alpha$ defined in the set $\texttt{succ}(\alpha)$ we can determine, as the time elapses, the first clock region in which $\texttt{min}(\psi)$ will be true. Let us denote this clock region by $\texttt{fst\_succ}(\alpha, \texttt{min}(\psi)) = min_{\alpha' \in \texttt{succ}(\alpha)}(\alpha' \Rightarrow \texttt{min}(\psi))$. Moreover we can establish the immediate predecessor, according to the total order, of a clock region α' in the set $\texttt{succ}(\alpha)$. Let use denote this by $\texttt{prec}(\alpha')$. Note that if α' is the minimum in $\texttt{succ}(\alpha)$, then its predecessor is α.

Definition 4.3 [Set of Crucial Clocks] The set $\texttt{cruc}(\alpha, \texttt{min}(\psi)) = \mathcal{X} - \{x \in \mathcal{X} \mid R_{T_u}(\texttt{prec}(\texttt{fst\_succ}(\alpha, \texttt{min}(\psi))), x) \Rightarrow \texttt{select}(\texttt{min}(\psi), x)\}$ contains the only clocks that determine the truth of the constraint $\texttt{min}(\psi)$ in the region $\texttt{fst\_succ}(\alpha, \texttt{min}(\psi))$.

Let us explain the concept of "crucial" by an example.

Example 4.4 Let $\texttt{min}(\psi)$ be $0 < x < 2 \wedge 1 < y < 3$. If α is $[x = 0 \wedge 0 < y < 1]$ then $\texttt{fst\_succ}(\alpha, \texttt{min}(\psi)) = [1 < y < 2 \wedge 0 < x < 1, \texttt{fract}(y) > \texttt{fract}(x)]$ and $\texttt{prec}(\texttt{fst\_succ}(\alpha, \texttt{min}(\psi))) = [y = 1 \wedge 0 < x < 1]$. In $\texttt{prec}(\texttt{fst\_succ}(\alpha, \texttt{min}(\psi)))$, the value of clock x implies the atomic constraint $\texttt{select}(\texttt{min}(\psi), x) = 0 < x < 2$, so x is not crucial for $\texttt{min}(\psi)$ becomes true. Thus, we have $\texttt{cruc}(\alpha, \texttt{min}(\psi)) = \{y\}$.

If α is $[y = 1 \wedge x = 0]$ then $\texttt{fst\_succ}(\alpha, \texttt{min}(\psi)) = [1 < y < 2 \wedge 0 < x < 1, \texttt{fract}(y) = \texttt{fract}(x)]$ and $\texttt{prec}(\texttt{fst\_succ}(\alpha, \texttt{min}(\psi)))$ is α itself. Here both x and y are crucial clocks.

Note that the set of crucial clocks always contains at least one element. If this were not true, the clock region $\texttt{prec}(\texttt{fst\_succ}(\alpha, \texttt{min}(\psi)))$ would implies $\texttt{min}(\psi)$. But, by definition, $\texttt{fst\_succ}(\alpha, \texttt{min}(\psi))$ is the minimum clock region that implies $\texttt{min}(\psi)$ and $\texttt{prec}(\texttt{fst\_succ}(\alpha, \texttt{min}(\psi)))$ is strictly less than it using the order defined in $\texttt{succ}(\alpha) \cup \{\alpha\}$. A contradiction.

The constraint $\texttt{select}(\texttt{min}(\psi), x)$, given any crucial clock x, can be used to determine a constraint that force the urgent action to be executed within $\ell = 1$ time units from it became enabled. To do this we add to any transition in E^q_α the additional constraint $\texttt{add}(\alpha, \texttt{min}(\psi)))$ constructed as follows. Given a crucial clock x (we can choose any one):

- $\mathtt{add}(\alpha, \mathtt{min}(\psi)))$ is $(x < c + 1)$ if $\mathtt{select}(\mathtt{min}(\psi), x)$ is either $(x = c)$ or $(c \leq x \# d)$ or $(x \geq c)$. Here $c < d$ and $\# \in \{\leq, <\}$.
- $\mathtt{add}(\alpha, \mathtt{min}(\psi)))$ is $(x \leq c + 1)$ if $\mathtt{select}(\mathtt{min}(\psi), x)$ is either $(c < x \# d)$ or $(c < x)$. Here $c < d$ and $\# \in \{\leq, <\}$.

Adding $\mathtt{add}(\alpha, \mathtt{min}(\psi))$ to all transitions in E^q_α we ensures that the urgent action will be executed according to its semantics.

Definition 4.5 Let $T^r_u = (Q^r, \Sigma, \mathcal{E}^r, \mathcal{U}^r, I^r, R^r, \mathcal{X})$ be a timed automaton with urgent transitions in region form. The *ℓ-consistent* version of it, ℓT^r_u, is the timed automaton $(Q^r, \Sigma, \mathcal{E}^r, \mathcal{U}^r_\ell, I^r, R^r, \mathcal{X}^r)$ where $\mathcal{X}^r = \mathcal{X} \cup \{x_q | q \in Q^r\}$ and $\mathcal{U}^r$ is constructed as follows:

- $(\langle q, \alpha\rangle, \psi \wedge x_{\langle q,\alpha\rangle} < \ell, \gamma \cup \{x_{\langle q',\alpha'\rangle}\}, \sigma, \langle q', \alpha'\rangle) \in \mathcal{E}'$ iff $(\langle q, \alpha\rangle, \psi, \gamma, \sigma, \langle q', \alpha'\rangle) \in \mathcal{U}^r$ and $(\alpha \Rightarrow \psi)$,
- $(\langle q, \alpha\rangle, \mathtt{min}(\mathtt{min}(\psi) \wedge \bigwedge_{x\in\mathcal{X}} R_{T_u}(\alpha'', x) \wedge \mathtt{add}(\alpha, \mathtt{min}(\psi))), \gamma \cup \{x_{\langle q',\alpha'\rangle}\}, \sigma, \langle q', \alpha'\rangle) \in \mathcal{E}'$ iff $(\langle q, \alpha\rangle, \mathtt{min}(\psi) \wedge \bigwedge_{x\in\mathcal{X}} R_{T_u}(\alpha'', x), \gamma, \sigma, \langle q', [\gamma \to 0]\alpha''\rangle) \in \mathcal{U}^r$, $\alpha \in \mathtt{Reg}(T_u, \mathcal{X})$, $\alpha'' \in \mathtt{succ}(\alpha)$ and $(\alpha \not\Rightarrow \psi)$

4.3 The quiet version of a timed automaton with urgent transitions

Now, in order to achieve the desired behavior, in each state of ℓT^r_u we have to turn off all the originally non-urgent outgoing transitions when at least one of the edges obtained by the originally urgent transition is enabled. This is the third transformation step.

Let $e = (\langle q, \alpha\rangle, \psi, \gamma, \sigma, \langle q', \alpha'\rangle)$ be a non-urgent outgoing transition from a state in ℓT^r_u. We map e in some transitions $e' = (\langle q, \alpha\rangle, \psi \wedge \theta, \gamma, \sigma, \langle q', \alpha'\rangle)$ of the new automaton where θ is the constraint that will become false when at least one of the outgoing urgent transitions of a state in ℓT^r_u becomes true.

Definition 4.6 Let ψ be a constraint without redundancies over a set of clocks $\mathcal{X}$. The *upper opening* $\mathcal{O}(\psi)$ of ψ is obtained by deleting from ψ all the constraints of the form $x \leq c$ and $x < c$, and by substituting all the constraints of the form $x = c$ by $x \geq c$, for all $x \in \mathcal{X}$. Clearly this definition requires constraints of the form $c \# x \# d$, $\# \in \{<, \leq\}$, to be considered as $c \# x \wedge x \# d$.

The upper opening of the disjunction of all urgent edges constraints (without redundancies), outgoing from a state in ℓT^r_u, describes a right-infinite time interval to the beginning of which a urgent transition must be taken. The negation of this disjunction must be added to all the constraints of non-urgent edges of T^r_u outgoing from the same state.

The negation of a complex formula can introduce disjunction of constraints. We denote by $\mathtt{DNF}^+$ an operation that, given a constraint which contains negations, push the negation operator inside, using the logical axioms for $\neg, \wedge, \vee$,

until it is applied to atomic constraints. Then it transforms the negations of these constraints to the correspondent positive ones ($x = c$ will be translated into $x < c \vee c < x$). Finally, it transforms the formula in disjunctive normal form. It returns the set containing all the conjunctive components of the formula.

Definition 4.7 Let $\ell T_u^r = (Q^r, \Sigma, \mathcal{E}^r, \mathcal{U}_\ell^r, I^r, R^r, \mathcal{X}^r)$ be the *ℓ-consistent* version of a timed automaton with urgent transitions in region form T_u^r. The *quiet* version of it, T^r, is the timed automaton $(Q^r, \Sigma, \mathcal{E} = \mathcal{U}_\ell^r \cup \mathcal{E}', I^r, R^r, \mathcal{X}^r)$ where $\mathcal{E}'$ is constructed as follows:

$(\langle q, \alpha\rangle, \psi \wedge \phi, \gamma \cup \{x_{\langle q', \alpha'\rangle}\}, \sigma, \langle q', \alpha'\rangle) \in \mathcal{E}'$ iff
$(\langle q, \alpha\rangle, \psi, \gamma, \sigma, \langle q', \alpha'\rangle) \in \mathcal{E}^r$, and
$\phi \in \mathtt{DNF}^+(\neg(\bigvee_u \mathcal{O}(\mathtt{min}(\psi_u))))$ for all $(\langle q, \alpha\rangle, \psi_u, \gamma_u, \sigma_u, \langle q'', \alpha''\rangle) \in \mathcal{U}_\ell^r$.

Example 4.8 Figure 3 shows the automaton T^{1r} which is the quieted version of the automaton T_u^1 of Figure 1. Note that the constraint $x > 1$ on the edge for b has been modified to $1 < x \wedge x \leq 1$ by the last transformation. Thus, being always false has been removed. In figure, the clock $x_{s,[x=0]}$ is omitted because it is useless in this case.

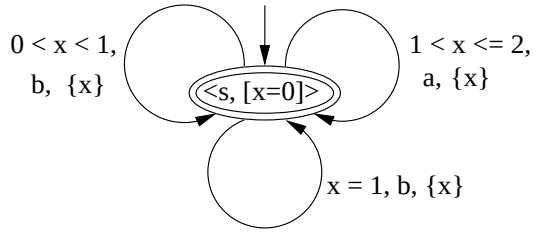

Fig. 3. Automaton T^{1r}

The transformation allows to state the following result.

Proposition 4.9 *Let T_u be a timed automaton with urgent transitions, T_u^r the corresponding timed automata in region form and T^r its quiet version. T_u and T^r accept the same timed language.*

5 Using timed automata with urgent transitions as a specification formalism

In the previous section we defined a new feature for the timed automata specification formalism. After that we showed how to compile a timed automaton with urgent transition into a standard timed automaton.

Indeed, a specification formalism needs a way to define systems as a composition of components. For timed automata this mechanism is the parallel

composition of Definition 2.5. The parallel composition of timed automata with urgent transitions is defined in the same way, but the following remarks:

- the urgency of a transition of a component with a synchronization action σ extends to the transition obtained using the rule (i) of Definition 2.5 even if the transition of the partner was not urgent
- the urgency of a transition of a component extends to the transition obtained by the rules (ii) and (iii) of Definition 2.5
- if the interleaving of actions leads to a reachable state with more than one outgoing urgent transition, then the composition is not possible.

The latter restriction follows from the limitation we gave in Definition 3.1. We plan to extend our definition to manage multiple outgoing urgent transitions. Most of the future work will concern the definition of a precise behavior in that case.

It is easy to see that the transformation defined in the previous section is not a congruence with respect to parallel composition. In other words if we have two timed automata, T_u^1 and T_u^2, with urgent transition the automaton $T_u^1 \parallel T_u^2$, when defined, is not equivalent, in general, to $T_1^r \parallel T_2^r$ (the standard parallel composition of the quiet version of them).

This fact makes the transformation not feasible for modular specifications, i.e. it must be applied to the whole system (the parallel composition of all components).

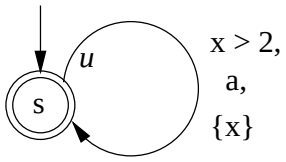

Fig. 4. Automaton $T_u'^1$

Consider the automaton $T_u'^1$ of Figure 4, the automaton T_u^1 of Figure 1 and the parallel composition $T_u^1 \parallel T_u'^1$. The action a, being a synchronization action, can be performed only when the value of the clock x is greater than 2. Thus b can be performed when the value of x is in $(0, 2]$. Using the quiet version of T_u^1 (Figure 3) in the parallel composition leads to a wrong behavior: $T^{1r} \parallel T_u'^1$ cannot perform the action a.

5.1 An example

In this last section we show a simple example. In Figure 5 is given the specification of a scheduler. The scheduler assigns resources to two processes, $P1$ and $P2$, alternatively. To each process the resources are assigned for 2 time units. If, during the elaboration of a process, an interruption occurs, it must be handled immediately. This is specified by considering the interruption handling a

urgent action. It is important to note that, because of urgent transitions, the scheduler behaves in the correct way independently from the environment in which will be introduced. The interruption handling will preempt all the non-urgent transitions of the environment thus preserving the intended behavior of the scheduler.

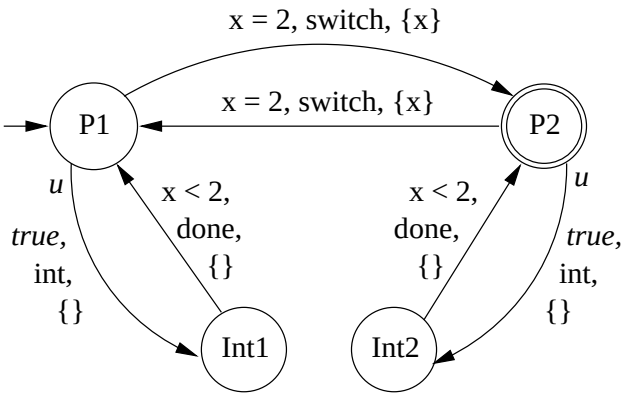

Fig. 5. A simple scheduler

6 Related works

The notion of urgency for timed formalisms has been studied in the past.

In [12] the urgency of actions has been investigated in the process algebra field with the concept of discrete time.

A closer approach to ours can be found in [13,14,15]. There the states of a timed automaton are associated with time progress conditions (TPC). TPC are state conditions which specify that the time can progress at a state by δ only if all the intermediate times δ', $0 \leq \delta' < \delta$, satisfy it.

TPC are computed from *deadlines*. Deadlines are clock constraints associated to transitions in addition to the usual constraints (which, in this setting, are called *guards*). The defined class of timed automata is called *Timed Automata with Deadlines* (TAD).

Given a state q, its TPC is intuitively computed as follows. Consider the set $I = \{i \mid t_i \text{ is a transition outgoing from } q\}$ of indexes of transitions from q. The TPC of q, c_q, is obtained as the negation of the disjunction of the deadlines, d_i, of all the transitions from q, $c_q = \neg \bigvee_{i \in I} d_i$. In a state of a run, (q, ν), the time can progress by δ, $(q, \nu) \xrightarrow{\delta} (q, \nu + \delta)$, if $\forall \delta' < \delta . \nu + \delta' \models c_q$.

Given a transition in a TAD, with guard ψ and deadline d, we can found in [14] the following remark.

"The relative position of d with respect to δ determines the urgency of the action. For a given δ, the corresponding d may take two extreme values: first, $d = \delta$, meaning that the action is eager and, second, $d = false$, meaning that

the action is lazy. A particularly interesting case is the one of a delayable action *where d is the falling edge of a right-closed guard δ (cannot be disabled without enforcing its execution).*

The condition $d \Rightarrow \delta$ guarantees that if time cannot progress at some state, then at least one action is enabled from this state. Restriction to right-open TPC guarantees that deadlines can be reached by continuous time trajectories and permits to avoid deadlock situations in the case of eager transitions. For instance, consider the case where $d = \delta = x > 2$, implying the TPC $x \leq 2$, which is not right-open. Then, if x is initially 2, time cannot progress by any delay δ, according to above definition. The guard ψ is not satisfied either, thus, the system is deadlocked."

This limitation is very intuitive: if the eager transition has a left-open guard, the time at which it can be fired is undefined. Using our concept of urgent transition we avoid this problem because the transition can be fired in the interval $[0, \ell)$. On the other hand to fire "as soon as possible" a transition with a left-closed guard, say $2 \leq x$, we have only to change it in $2 = x$.

Example 6.1 Let us use our notion of urgency to model a producer-consumer system that is considered in [15]. The partners are supposed to communicate with a zero-length buffer. Figure 6(a) and 6(b) show the producer and the consumer respectively.

The producer in state 1 is producing a new item. This process requires a time that is between l_p and u_p time units. When in state 3, the producer can communicate with a *handshake* in the time interval $[l'_p, u'_p]$ to send an item. Conversely the consumer in state 2 can communicate with a *handshake* in the time interval $[l'_c, u'_c]$ to receive an item. When in state 4 the consumer is consuming the item needing a time between l_c and u_c time units. Note that actions *produce* and *consume* represent the end of the correspondent processes. Also note that *handshake* is a urgent action for both the components. The parallel composition is shown in Figure 7. In [15] authors use both different notions of compositions of guards and deadlines to obtain different behaviors of the whole system. Here we study how these behaviors can be simulated in our model varying the constraint ψ for the *handshake* action in Figure 7 and the urgency parameter ℓ.

The first behavior to consider is the usual one coming from standard parallel composition. The constraint ψ is the conjunction of the components *handshake*-transition constraints. The parameter ℓ in this case is $\mathtt{max}((u'_p - l'_p), (u'_c - l'_c))$, i.e. the length of the largest interval expressed in the components behavior. The system can execute the *handshake* only if both clock x and y are in the intervals expressed in the specification of components. This behavior, as observed in [15], could be too restrictive since it may cause deadlock.

To relax this condition we can set the constraint ψ of Figure 7 to $x \geq l'_p \wedge y \geq l'_c$. Now the upper bounds are not considered and, starting from the instant in which the constraint becomes satisfied, the action has to be executed

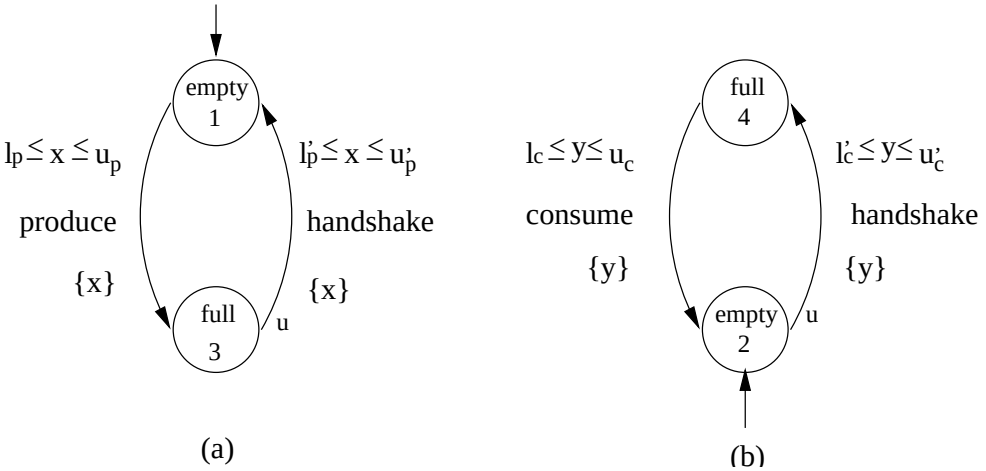

Fig. 6. A Producer-Consumer system: the components

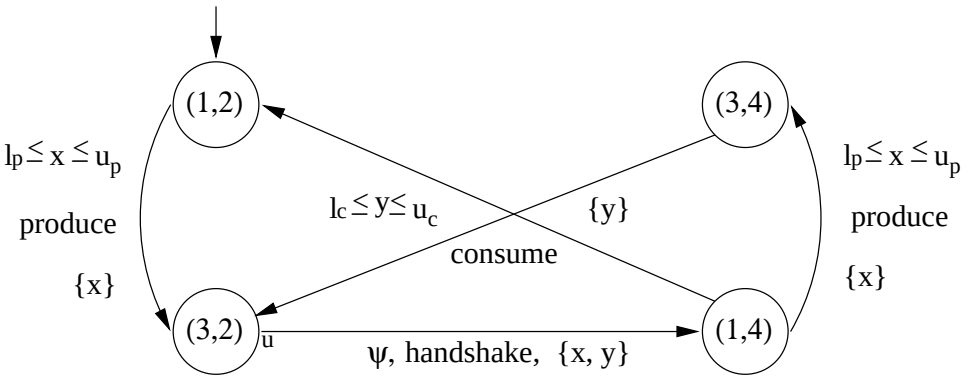

Fig. 7. A Producer-Consumer system: parallel composition

within ℓ time units. We set ℓ to $\mathtt{min}((u'_p - l'_p), (u'_c - l'_c))$. This ensures that at most one of the upper bounds can be violated. This behavior is similar to the one specified in [15] by a deadline $x = u'_p \wedge y \geq u'_c \ \vee \ y = u'_c \wedge x \geq u'_p$. There the *handshake* can be postponed as long as one of the intervals is not violated, but when the end of this interval is reached the action becomes urgent and must be executed. Clearly, since in the current approach we can specify only one outgoing action for each state, our model is a slight more restrictive. As a matter of fact, if the shortest interval is the one associated to clock y and the clock values are such that the first constraint satisfied is $y \geq l'_c$, then, when the constraint ψ becomes true (i.e. $x \geq l'_p$ becomes true), the action must be executed before the upper bound associated to y (actually it should wait until $x = u'_p$) because we definitely chose $\ell = (u'_c - l'_c)$. A similar remark can be done considering the choice of $\ell = (u'_p - l'_p)$.

There is another synchronization scheme for this example that authors in [15] call "best-effort", that is "either no upper bound is violated if possible, or the transition is executed as soon as possible". To express a similar scheme we use the constraint $\psi = x \geq l'_p \wedge y \geq l'_c$ as above and a constant ℓ as small as we want to precisely approximate the "as soon as possible" behavior. In

this example this is precisely definite since the constraint $\psi = x \geq l'_p \wedge y \geq l'_c$ uses the operator $\geq$. Indeed, the action should be executed when $x = l'_p$ or $y = l'_c$ (depending on which one becomes true after). In this version of the work we always have to consider the approximation, even for precisely definite behaviors (i.e. those with operators "$\geq$" and "$=$"). However, note that by using the approximation we can manage also the constraints in which is used the operator $>$ in place of $\geq$.

7 Conclusion

In this paper we introduce a notion of urgency for timed automata. We compare it with other approaches to urgency, in particular the one of [13,14,15].

In this version of the paper we introduce the limitation that no more than one urgent transition can exit from a state. We plan to remove this limitation in future works.

References

[1] Aceto, L., Bouyer, P., Burgueño, A. and Guldstrand Larsen, K. The Power of Reachability Testing for Timed Automata. Proc. Foundations Software Technology and Theoretical Computer Science, Springer LNCS 1530, 245–256, 1998.

[2] Aceto, L., Burgueño, A. and Guldstrand Larsen, K. Model Checking via Reachability Testing for Timed Automata. Proc. TACAS, Springer LNCS 1384, 263–280, 1998.

[3] Alur, R., Courcoubetis, C. and Dill, D.L. Model-Checking in Dense Real-time. *Information and Computation*, 104, 2–34, 1993.

[4] Alur, R., Courcoubetis, C., Halbwachs, N., Dill, D.L. and Wong-Toi, H. Minimization of Timed Transition Systems. Proc. CONCUR 1992, Springer LNCS 630, 340–354, 1992

[5] Alur, R., Courcoubetis, C. and Henzinger, T.A. The Observational Power of Clocks. Proc. CONCUR 1994, Springer LNCS 836, 162–177, 1994.

[6] Alur, R. and Dill, D.L. Automata for Modelin Real-time Systems. Proc. ICALP'90, Springer LNCS 443, 322–335, 1990.

[7] Alur, R. and Dill, D.L. A Theory of Timed Automata. *Theoretical Computer Science*, 126, 183–235, 1994.

[8] Alur, R., Fix, L. and Henzinger, T.A. Event-Clock Automata: A Determinizable Class of Timed Automata. *Theoretical Computer Science*, 211, 253-273 (1999).

[9] Alur, R. and Henzinger, T.A. Back to the Future: Towards a Theory of Timed Regular Languages. Proc. FOCS 1992, 177–186, 1992.

[10] Alur, R. and Henzinger, T.A. A Really Temporal Logic. *Journal of ACM*, 41, 181–204, (1994).

[11] Barbuti, R., De Francesco, N. and Tesei,L. Timed Automata with non-Instantaneous Actions To appear in *Fundamenta Informaticae*.

[12] Bolognesi, T. and Lucidi, F. Timed Process Algebras with Urgent Interactions and a Unique Powerful Binary Operator. REX Workshop 1991, Springer LNCS 600, 124-148, 1992.

[13] Bornot, S. and Sifakis, J. Relating Time Progress and Deadlines in Hybrid Systems. HART 1997, Springer LNCS 1201, 286–300, 1997.

[14] Bornot, S., Sifakis, J. and Tripakis, S. Modeling Urgency in Timed Systems. COMPOS 1997, Springer LNCS 1536, 103–129, 1998.

[15] Bornot, S. and Sifakis, J. Modeling Urgency in Timed Systems. To appear on *Information and Computation*.

[16] Choffrut ,C. and Goldwurm, M. Timed Automata with periodic Clock Constraint. Int. Report 225-98, 1998.

[17] Demichelis,F. and Zielonka, W. Controlled Timed Automata Proc. CONCUR 98, Springer LNCS 1566, 455–469, 1998.

[18] Gupta, V., Henzinger, T.A. and Jagadeesan, R. Robust Timed Automata. Proc. HART 97, Springer LNCS 1201, 331-345, 1997.

[19] Henzinger, T.A. and Kopke, P.W. Verification Methods for the Divergent Runs of Clock Systems. Proc. Formal Techniques in Real-Time and Fault-Tolerant Systems, Springer LNCS 863, 351–372, 1994.

[20] Henzinger, T.A., Nicollin, X., Sifakis, J. and Yovine, S. Symbolic Model Checking for Real-Time Systems. *Information and Computation*, 111, 193–244, 1994.

[21] Lanotte, R., Maggiolo-Schettini, A. and Peron, A. Timed Cooperating Automata *Fundamenta Informaticae*, 43, 96-107, (2000).

[22] Yovine, S. Model Checking Timed Automata. Lectures on Embedded Systems, Springer LNCS 1494, 114–152, 1996.

Petri Nets with Discrete Phase Type Timing: A Bridge Between Stochastic and Functional Analysis

Andrea Bobbio [1]

DISTA, Università del Piemonte Orientale, Alessandria, Italy

András Horváth [2]

Dipartimento di Informatica, Università di Torino, Torino, Italy

Abstract

The addition of timing specification in Petri Nets (PN) has followed two main lines: intervals for functional analysis or stochastic durations for performance and dependability analysis. The present paper proposes a novel technique to analyze time or stochastic PN models based on discretization. This technique can be seen as a bridge between the world of functional analysis and the world of stochastic analysis. The proposed discretization technique is based on the definition of a new construct called *Discrete Phase Type Timing - DPT* that can represent a discrete cumulative density function (cdf) over a finite support (or a deterministic cdf) as well as an interval with non-deterministic choice (or a deterministic duration). In both views, a preemption policy can be assigned and a strong (the transition must fire when the interval expires) or a weak (the transition can fire when the interval expires) firing semantics. The paper introduces the DPT construct and shows how the expanded state space can be built up resorting to a compositional approach based on Kronecker algebra. With this technique a functional model can be quantified by adding probability measures over the firing intervals without modifying the (compositional) structure of the PN model.

1 Introduction

The inclusion of timing specification in Petri net models has followed two main streams of research. In the first one time is assigned as a deterministic value (constant or interval) while in the second stream, the activities are assumed to have a random duration. We will refer to the first class of models as Time Petri Nets (TPN) and to the second class as Stochastic Petri Nets (SPN).

[1] Email: bobbio@di.unito.it

[2] Email: horvath@di.unito.it

This is a preliminary version. The final version is considered for publication in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

TPN are devoted to specify and verify properties of systems where timing is a critical parameter that may affect the behavior of the system. In this line of research [18], time is assigned as a constant value or as an interval defined by a min (*earliest firing time* - *EFT*) and a max (*latest firing time* - *LFT*) value. The firing semantics is interleaving and with non-determinism (no weight is assigned to the action of atomic firing inside the allowed interval or for resolving conflicts). Further developments along this line are documented in [4,9]. In [15] a modified firing semantics is introduced: time is assigned as intervals, and firing may be forced when the maximum time expires *(strong firing semantics)* or firing may be not mandatory when the maximum time expires *(weak firing semantics)*. Analysis of TPN models involves the search for reachable conditions through the exploration of *firing zones* [4,22].

Since the initial work in [20,19], SPN have found a sound theoretical base and consolidated applications when the firing time assigned to timed transitions is an exponentially distributed random variable, so that the evolution of the system throughout its reachability graph is mapped into a continuous time Markov chain (CTMC). A number of tools exploit this paradigm and the most extensive applications are in the area of performance and dependability modeling and analysis [2]. However, reality is not always exponential and attempts have been made to include in SPN generally distributed transitions [1,7]. Particular emphasis has been devoted to models in which deterministic times [3,17] are combined with exponential random variables. In order to completely specify the non-Markovian stochastic process underlying the behavior of a SPN with generally distributed transition times, each transition is assigned an *age variable*. The way in which the age variable accounts for the time in which the transition has been enabled is governed by three *memory policies* [7]. In the *preemptive repeat different (prd)* policy (also called *enabling memory*) the age variable is reset each time the transition is disabled or fires; in the *preemptive repeat identical (pri)* policy [6], when the transition is disabled its age variable is reset, but when the transition is enabled again an identical firing time must be completed. Finally, in the *preemptive resume (prs)* policy the age variable maintains its value when the transition is disabled and then re-enabled, and is reset only when the transition fires.

Under the restriction that the marking process arising from these SPN is a Markov regenerative process [11,7], an analytical solution can be envisaged. Otherwise, an approximate solution can be obtained through a "Markovianization", by assigning to each transition a continuous Phase type distributions [8].

More recently, a new class of SPN has been explored, namely the one obtained by assigning to each timed transition a Discrete phase type (DPH) distribution [12,23,21,16]. DPH distributions are distributions arising from the time to absorption in discrete-time Markov chains with absorbing states and have been extensively explored in [5], where a fitting algorithm has been also provided. The peculiarity of the class of DPH distributions, is that it contains

cdf with finite support like the deterministic or the (discrete) uniform. The use of DPH in Petri net models, allows to include cdf with finite support and any mixture of preemption policies. Moreover the transition matrix over the expanded state space may be expressed in a compositional way by means of Kronecker algebra [21], without the need of generating and storing the complete matrix. Hence, the cost of storing the model is of the same order as the cost of storing the reachability graph of the untimed PN and the solution may exploit efficient algorithms [10] for block matrices in Kronecker form.

This paper shows that the discretization technique, up to now adopted in SPN, can be seen as a bridge between the world of the functional analysis and the world of the stochastic analysis. To this end, we define an extended construct, called *Discrete Phase Type Timing (DPT)*, that encompasses the features of the DPH distributions and of the intervals (or constants) with non-determinism. By assigning to each timed transition of a PN a DPT we can build up both a functional model, in the line of those discussed in [18,4,15] and a stochastic model in the line of those discussed in [21,16]. One goal of this paper is to show that a functional model can be quantified by adding probability measures over the firing intervals without modifying the structure of the underlying PN model. Since the DPT class inherits the properties of DPH random variables and non-deterministic intervals, the DPT-PN model shares the same characteristics examined in [21,16] for DPH stochastic PN. In particular, the compositional structure of the expanded state space [21] can be exploited also for the functional model. Moreover, we can associate to any DPT a *preemption* policy, so that functional analysis can be carried out taking into account interruption and restart mechanisms that were not covered by previous models, and, furthermore, we can accommodate in the model both weak and strong firing semantics as defined in [15].

Discretization implies a state expansion and incurs in the state space explosion problems. The compositional approach via Kronecker algebra may alleviate this problem, and we can benefit from efficient storage techniques as those presented in [13]. Moreover, where DPT are used for functional analysis, the PN model may be interfaced with efficient discrete model-checking tools [14].

The paper is organized as follows. Section 2 introduces the Discrete Phase Type Timing structures that will be used to describe the local evolution of the transitions of the model. Section 3 describes the global evolution of the process. In Section 4 two demonstrative examples are given. Conclusions are drawn in Section 5.

2 Structures and matrices to describe local evolution of transitions

This section is organized as follows. Section 2.1 and Section 2.2 introduces the DPT structures and corresponding matrices for functional and stochastic

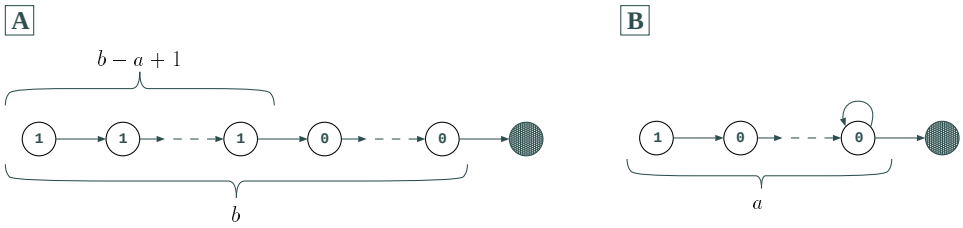

Fig. 1. **A:** Local evolution of a transition with interval firing $[a, b]$ in case of strong time semantics and prd or prs policy. **B:** Local evolution of a transition with interval firing $[a, \infty]$ in case of either weak or strong time semantics and prd or prs policy

analysis, respectively. These structures describe how the local descriptor of a transition evolves in a step if the transition is enabled. The applied structures depend on the adopted memory policy as well.

Throughout the paper we assume that minimal and maximal firing times are integer values and the minimal firing time is strictly positive. Note that a model in which all minimal and maximal firing times are integer multiples of a common time unit can be handled the same way. Zero minimal firing time can be handled as well by properly supplementing the model by immediate transitions.

2.1 Functional analysis

Transitions with prd or prs preemption policy

The structure used to represent the local evolution of an enabled transition with strong time semantics and firing interval $[a, b]$ is depicted in Figure 1A. When the initial phase of the structure is chosen the process may enter any of the phases signed with 1. The arrows represent the possible state-jumps; having more than one outgoing arc from a phase indicates a non-deterministic choice. The transition fires if a state-jump to the filled state occurs. In every step, if the transition is enabled, the process steps to the next phase. This structure ensures that the firing time of the transition will be in the interval $[a, b]$ and the transition fires for certain when it reaches the upper limit of its firing interval. The structure is represented by the row vector $\boldsymbol{t_0}$ that describes the possible initial phases, the square matrix $\boldsymbol{T}$ that describes the possible state-jumps and the column vector $\boldsymbol{t_f}$ that gives the phases of which firing may happen. These vectors and matrices, which will describe the local evolution of an enabled transition, are

$$\boldsymbol{t_0} = [\underbrace{1, \ldots, 1}_{b-a+1}, \underbrace{0, \ldots, 0}_{a-1}], \quad \boldsymbol{T} = \begin{bmatrix} 0 & 1 & 0 & \ldots & \\ 0 & 0 & 1 & 0 & \ldots \\ & & \ddots & & \\ 0 & \ldots & 0 & 1 & \\ 0 & \ldots & & 0 & \end{bmatrix}, \quad \boldsymbol{t_f} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}.$$

In case of firing interval $[a, \infty]$ (still with strong time semantics) the structure depicted in Figure 1B is applied. Having been enabled for a time units the transition may either fire or remain in the last phase. Observing the structure one can easily write its descriptors $\boldsymbol{t_0}, \boldsymbol{T}$ and $\boldsymbol{t_f}$.

Assuming weak time semantics, the local evolution of an enabled transition with firing interval $[a, b]$ is followed using the structure shown in Figure 2A. When the transition is enabled for b time units it either fires or the process steps to the phase signed by the circle with thicker line from which there is no outgoing arc. This structure guarantees that the firing time will be in the interval $[a, b]$ and makes it possible that the transition does not fire in the interval. If the firing interval is $[a, \infty]$, there is no difference between strong and weak time semantics. Hence the structure depicted in Figure 1B is used.

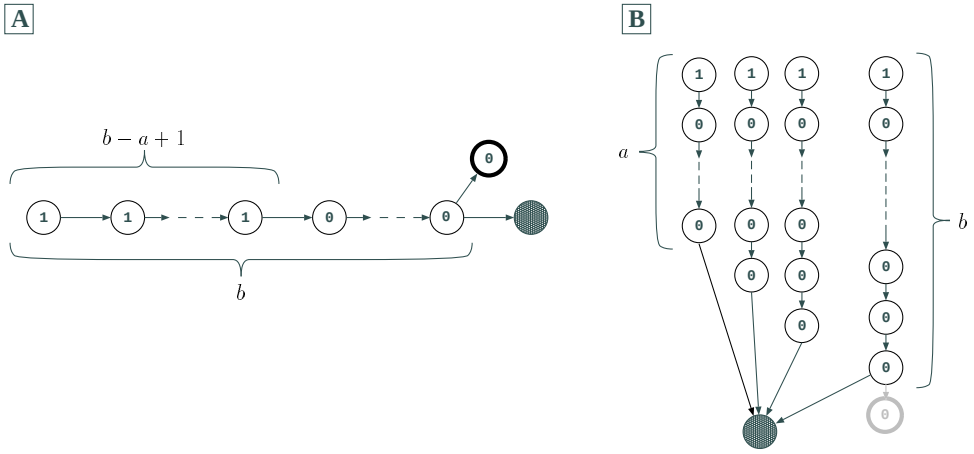

Fig. 2. **A**: Local evolution of a transition with interval firing $[a, b]$ in case of weak time semantics and prd or prs policy. **B:** Local evolution of a transition with interval firing $[a, b]$ in case of pri policy.

The above described structures are used in a different manner in case of prd than in case of prs transitions. When a prs transition is preempted the phase in which it was preempted is recorded; in case of re-enabling the process enters this state. Instead, when a prd transition is re-enabled, the initial state is chosen according to $\boldsymbol{t_0}$.

Transitions with pri policy

In case of pri transitions, the amount of time the transition spent enabled is lost and we have to ensure that the firing time of the transition is the same after it is re-enabled. This requirement can not be fulfilled with the structures presented above. Instead, the structure of Figure 2B is used. When the transition gets preempted the column in which the process was when the preemption happened is recorded; in case of re-enabling the process enters the first phase of this column. The realization of a transition with firing interval $[a, \infty]$ would require an infinite state structure, and hence it is not considered.

When we adopt weak time semantics the gray phase drawn with thicker

line is present as well. So that, it is possible that the transition does not fire in the interval $[a, b]$. When the process is in this phase when getting preempted, it returns to this phase in case of re-enabling.

2.2 Stochastic analysis

Transitions with prd or prs preemption policy

A discrete Phase type (DPH) distribution is the distribution of the time to absorption in a discrete-time Markov chain. As a counterpart of the weak time semantics in stochastic analysis, we introduce and make use of possibly defective discrete Phase type (PDDPH) distributions (see Appendix A for the definition of PDDPH distributions). A PDDPH distribution is associated to every timed transition of the Petri net. A Markov chain describing a PDDPH distribution may have two absorbing states. A jump to the absorbing phase referred to as *firing* absorbing phase (drawn with a filled circle) causes the transition to fire. A jump to the other absorbing phase, referred to as *local* absorbing phase and drawn with a thicker line, means that the transition can not fire anymore in its current sampling period (i.e. it can fire in the future only if it gets disabled and then it is re-sampled again). Without loss of generality we may assume that there are no other absorbing states or absorbing group of states in the chain.

A simple example for PDDPH distribution is depicted in Figure 3A. When the firing time of a transition is sampled the process enters one of the phases according to the initial probabilities; initial probabilities are written next to the phases. In Section 2.1 several outgoing arcs of a place represented a non-probabilistic choice. Instead, during stochastic analysis, outgoing arcs of a given phase represent a probabilistic choice among the arcs, i.e. a real value from the interval $[0, 1]$ is associated to each arc (these values are not depicted in Figure 3A) which gives the probability that a given arc will be chosen in the next step (the sum of the values associated to the outgoing arcs of a given place is 1). While in functional analysis weak time semantics implies a non-deterministic choice on whether the transition fires or not, using PDDPH distributions leads to a probabilistic choice. By computing the steady state probability of the two absorbing states one can determine how "defective" the transition is.

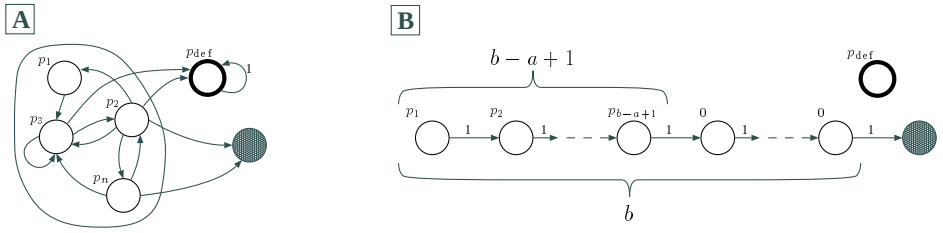

Fig. 3. **A:** Possibly defective DPH distribution. **B:** Possibly defective DPH distribution with finite support $[a, b]$ for prd or prs transitions.

A PDDPH distribution is described by its initial probability vector

$$\boldsymbol{t_0} = [p_1, p_2, \ldots, p_n, p_{\text{def}}],$$

its transition matrix $\boldsymbol{T}$ which governs the phase-jumps, and a column vector $\boldsymbol{t_f}$ which contains the probabilities of jumping to the firing absorbing state. We make two comments at this point. First, it is not necessary to have local absorbing state. Second, the row of $\boldsymbol{T}$ that corresponds to the local absorbing state contains a single non-zero value which is a 1 in the diagonal.

With PDDPH distributions one can realize a finite support distribution as shown in Figure 3B, it can be either defective or not, depending on the actual value of p_{def}. When not defective this structure is the stochastic counterpart of the structure shown in Figure 1A, when defective it is the stochastic counterpart of the structure shown in Figure 2A. In functional analysis, the only knowledge we have is that the transition fires in the interval $[a, b]$ in a non-deterministic manner. Instead, in stochastic analysis, it is possible to define the exact probability that the transition fires at a given time instant in the interval $[a, b]$.

Figure 4A depicts the possibility of having defective or non-defective DPH distributions with support $[a, \infty]$. The process enters either the local absorbing state or the state on the left side of the figure after which it has to take at least a steps before absorption.

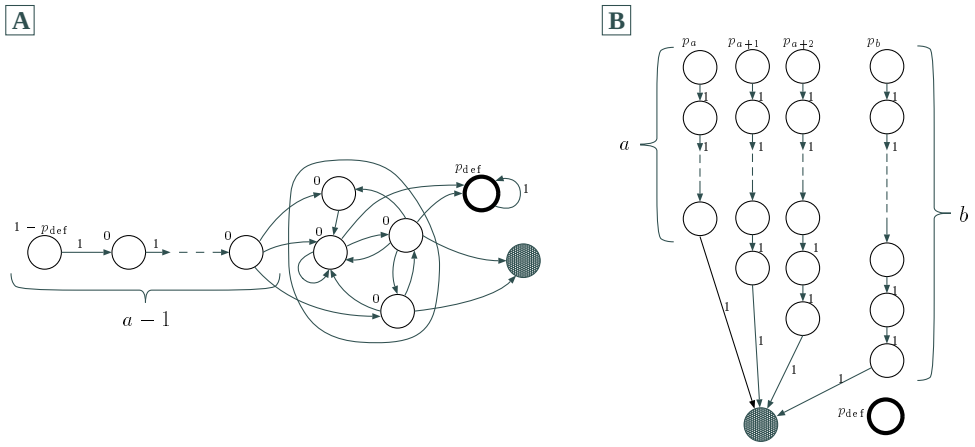

Fig. 4. **A:** Possibly defective DPH distribution with support $[a, \infty]$ for prd or prs transitions. **B:** Possibly defective DPH distribution with finite support $[a, b]$ for pri transitions.

The difference between handling prs and prd transitions in case of preemption is the same as when performing functional analysis.

Transitions with pri preemption policy

As for functional analysis a different structure is applied for transitions with pri policy. This structure, which is the probabilistic counterpart of the structure shown in Figure 2B, is depicted in Figure 4B.

3 Global process

For describing the global evolution of the process the method presented in [21] is followed with two important differences. First, we handle prs transitions in a manner that corresponds exactly to the definition of the prs preemption policy [3]. Second, we extend [21] with the possibility of having transitions with pri preemption policy in the model.

In order to describe the global evolution of the process we need extended knowledge on the reachable markings of the net. We call the graph we use extended reachability graph (*erg*). A node of the *erg* carries the following information:

- the number of tokens in the places of the net,
- the set of preempted transitions with memory (the phase in which these transitions were preempted has to be recorded),
- the set of prs transitions that were candidates for firing but did not fire; these transitions, because of the definition of prs memory policy, are candidates for firing immediately when they get enabled.

The last entry of the above list requires some further explanation. In the considered model (either functional or stochastic) it can happen that two or more transitions have the same firing time instant. These transitions are called candidates for firing. Having a set of candidates the resulting marking depends on the order of firings and it can happen that a transition firing prevents another candidate from firing. The set of possible orders (in case of functional analysis) or the probability of a given order (in case of probabilistic analysis) can be determined based on priority considerations; this issue is not in the scope of this paper. Not having additional priority information on the transitions, one can assume that all orders of the candidates are possible (functional analysis) or all orders have equal probability (probabilistic analysis) [21].

A simplest example of *erg* is shown in Figure 5. Even if the net has two reachable markings, still the *erg* has four nodes. In marking P_3 three situations have to be distinguished: the marking was reached by the firing of T_1, the marking was reached by the firing of T_2 and T_1 was not candidate for firing, the marking was reached by the firing of T_2 and T_1 was candidate for firing.

The following notations are used to describe the procedure. We assume that the number of nodes in the *erg* is finite and denoted by N. The ith node of the *erg* will be denoted by $\boldsymbol{m}_i$. The set of prs and pri transitions are denoted by $\mathcal{S}$ and $\mathcal{I}$, respectively. The firing interval of a pri transition T_i is denoted by $[a^{(i)}, b^{(i)}]$. The set of transitions that are enabled in $\boldsymbol{m}_i$ is denoted by $\mathcal{A}_i$. The set of transitions that are disabled but have memory in $\boldsymbol{m}_i$ is denoted by $\mathcal{B}_i$. The set of transitions that were candidates but did not fire when the process entered extended marking $\boldsymbol{m}_i$ is denoted by $\mathcal{C}_i$. The number

[3] A prs transition that was candidate for firing but did not fire has to be candidate for firing immediately when it gets enabled again. This fact was not considered in [21].

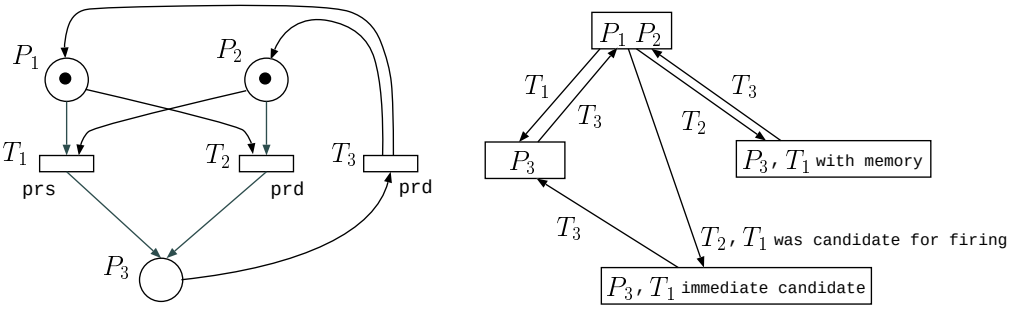

Fig. 5. In marking P_3 transition T_1 may either have or not have memory

of transitions in the net is M. The transitions are ordered and the ith one is denoted by T_i.

The local evolution of a transition is described by the vectors and matrix introduced in Section 2.1 for functional and in Section 2.2 for stochastic analysis. The descriptors of transition T_i, $1 \leq i \leq M$ are denoted by $\boldsymbol{t_0}^{(i)}$, $\boldsymbol{T}^{(i)}$ and $\boldsymbol{t_f}^{(i)}$.

3.1 Global descriptor

During the analysis, the transient descriptors of the system are stored in the vectors $\boldsymbol{p}_i$, $1 \leq i \leq N$. In a vector $\boldsymbol{p}_i$ every position corresponds to a combination of local descriptors of the transitions that are enabled or are disabled but have memory in the extended marking $\boldsymbol{m}_i$. In case of functional analysis the vector $\boldsymbol{p}_i$ contains 0s and 1s. An entry 1 in $\boldsymbol{p}_i$ means that it is possible that the process is in $\boldsymbol{m}_i$ with descriptors corresponding to the position of the entry. Instead, when performing stochastic analysis, $\boldsymbol{p}_i$ may contain any real value in $[0, 1]$. In this case, an entry of $\boldsymbol{p}_i$ gives the probability that the process is in $\boldsymbol{m}_i$ with descriptors corresponding to the position of the entry.

Let us denote the number of phases of the structure representing transition T_i by n_i and the local descriptor of T_i by l_i. The vector $[l_1, l_2, \ldots, l_M]$, which contains the descriptors of all the transitions, together with the index of extended marking defines a state of the process. If a transition T_1 is disabled and does not have memory its descriptor equals 1. When T_i is enabled in an extended marking we have $1 \leq l_i \leq n_i$. The phase in which a prs transition gets disabled has to be recorded; for a prs transition T_i that is disabled but has memory in $\boldsymbol{m}_j$ $1 \leq l_i \leq n_i$. In case of a pri transition T_i with firing interval $[a, b]$ that is disabled but has memory in $\boldsymbol{m}_j$ only the column in which it got disabled is recorded (Figure 2B and 4B), hence $1 \leq l_i \leq b - a + 1$.

In the following we describe how to find a given combination of the local descriptors in the vector $\boldsymbol{p}_i$. We use a so-called mixed-based numbering scheme which is closely related to the Kronecker product operator. Let us use the

notation

$$n_i^{(k)} = \prod_{j=k}^{M} s_i^{(j)}, \text{ where } s_i^{(j)} = \begin{cases} 1 & \text{if } T_j \notin \mathcal{A}_i \cup \mathcal{B}_i \\ n_j & \text{if } T_j \in \mathcal{A}_i, \\ n_j & \text{if } T_j \in \mathcal{B}_i \cap \mathcal{S}, \\ b^{(j)} - a^{(j)} + 1 & \text{if } T_j \in \mathcal{B}_i \cap \mathcal{I}, \end{cases}$$

i.e. $s_i^{(j)}$ is the number of different values the descriptor l_j may have in extended marking $\boldsymbol{m}_i$. Then, in the state space spanned by the local descriptors, the possibility or the probability that the process is in discrete marking $\boldsymbol{m}_i$ and the local descriptors are $[l_1, l_2, \ldots, l_M]$ is given by the mth entry of $\boldsymbol{p}_i$ with

$$m = (\ldots((l_1 - 1)s_i^{(2)} + (l_2 - 1))s_i^{(3)} \ldots)s_i^{(M)} + l_M - 1 = \sum_{k=1}^{M}(l_k - 1)n_i^{(k+1)},$$

where $n_i^{(M+1)} = 1$.

Let us assume that the initial extended marking is $\boldsymbol{m}_i$. Then, initially

$$\boldsymbol{p}_i = \bigotimes_{k, T_k \in \mathcal{A}_i} \boldsymbol{t_0}^{(k)},$$

where $\otimes$ denotes the Kronecker-product operator (see Appendix B for the definition of the operator).

Note that none of the disabled transitions can have memory in the initial marking. All entries of all the other transient vectors are set to 0 at the beginning of the analysis.

3.2 Global evolution

In the following we show how to build the matrix $\boldsymbol{P}$ that describes the global evolution of the process.

For functional analysis the resulting matrix is the incidence matrix of the model, i.e. a 1 or a 0 in position (i, j) means that macrostate j is reachable or not in one step from macrostate i, respectively. The evolution of the process is followed by successive multiplication of the transient vector $[\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_N]$ by $\boldsymbol{P}$. This way we are given the set of possible macrostates at time $1^+, 2^+, 3^+, \ldots$, i.e. right after integer multiples of the chosen time unit.

In case of stochastic analysis a probabilistic choice is defined in the intervals assigned to the transitions with a given common step-size δ. The matrix $\boldsymbol{P}$ is the transition matrix of the underlying discrete-time Markov chain of the process, i.e. the value in position (i, j) gives the one step probability from macrostate i to macrostate j. Once again, the evolution of the process is followed by successive multiplication of the transient vector $[\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_N]$ by $\boldsymbol{P}$. This way we are given the probability of macrostates at time $\delta, 2\delta, 3\delta, \ldots$.

The matrix $\boldsymbol{P}$ is built as an $N \times N$ block-matrix in which each block is expressed as the sum of Kronecker-products of suitable matrices. The diagonal

blocks $\boldsymbol{P}_{ii},\ 1 \leq i \leq N$ are square matrices describing the evolution of the process inside the macrostate corresponding to $\boldsymbol{m}_i$. The off-diagonal blocks $\boldsymbol{P}_{ij},\ 1 \leq i,j \leq N$ describe the jumps from any state of macrostate $\boldsymbol{m}_i$ to any state of macrostate $\boldsymbol{m}_j$. First we consider the case when none of the transitions fire.

Evolution when no firing happens

When none of the transitions fire in a step the process remains in the same macrostate. Hence, this case contributes to the diagonal entries $\boldsymbol{P}_{ii},\ 1 \leq i \leq N$ of $\boldsymbol{P}$. A diagonal block is expressed by the following Kronecker-product

$$\boldsymbol{P}_{ii} = \bigotimes_{j=1}^{M} \boldsymbol{Q}_j \tag{1}$$

where

- $\boldsymbol{Q}_j = \boldsymbol{T}^{(j)}$ if $T_j \in \mathcal{A}_i$, i.e. T_j is enabled in $\boldsymbol{m}_i$ and its evolution is described by $\boldsymbol{T}^{(j)}$;
- $\boldsymbol{Q}_j$ is an identity matrix, its size is $n_j \times n_j$ if $T_j \in \mathcal{B}_i \cap \mathcal{S}$ and $b^{(j)} - a^{(j)} + 1 \times b^{(j)} - a^{(j)} + 1$ if $T_j \in \mathcal{B}_i \cap \mathcal{I}$, i.e. the descriptor of a disabled transition having memory is kept;
- $\boldsymbol{Q}_j = 1$ if $T_j \notin \mathcal{A}_i \cup \mathcal{B}_i$, i.e. the transition does not contribute to the evolution of the macrostate and has no influence on the Kronecker-product.

Evolution in case of firings

Firing of a set of transitions is considered by the expression

$$\boldsymbol{P}_{ij} += \sum_{\mathcal{L} \in S(\mathcal{A}_i)} W_{ij}(\mathcal{L}) \bigotimes_{k=1}^{M} \boldsymbol{Q}_k(\mathcal{L}) \tag{2}$$

where the function $S(\bullet)$ gives the set of non-empty subsets of its argument, and $W_{ij}(\mathcal{L})$ has the following meaning:

- In case of functional analysis its value is 1 if from extended marking $\boldsymbol{m}_i$ having the transitions in $\mathcal{L}$ as candidates the next extended marking can be $\boldsymbol{m}_j$; it is 0 otherwise. $W_{ij}(\mathcal{L})$ can be determined by considering all possible orders of $\mathcal{L}$.
- In case of stochastic analysis the value of $W_{ij}(\mathcal{L})$ is the probability that being in $\boldsymbol{m}_i$, having the transitions in $\mathcal{L}$ as candidates the next extended marking is $\boldsymbol{m}_j$. The value of $W_{ij}(\mathcal{L})$ may be determined by assigning a probability to all possible orders of $\mathcal{L}$. The choice when all ordering has the same probability is discussed in [21].

During the calculation of $W_{ij}(\mathcal{L})$ the set $\mathcal{C}_i$ has to be considered as well.

In (2) instead of $=$ we use $+=$ because if a sequence of firings leads back to the same extended marking (it can easily happen in the presence of immediate transitions) the quantity of the right hand side is added to the quantity defined in (1). The term $\boldsymbol{Q}_k(\mathcal{L})$ is determined according to the following situations:

- If $T_k \in \mathcal{L}$, i.e. if T_k is one of the candidates the following cases has to distinguished:
  - if T_k is neither enabled nor it has memory in $\boldsymbol{m}_j$ (i.e. $T_k \notin \mathcal{A}_j \cup \mathcal{B}_j$), then $\boldsymbol{Q}_k = \boldsymbol{t_f}^{(k)}$,
  - if T_k is re-enabled in $\boldsymbol{m}_j$ (i.e. $T_k \in \mathcal{A}_j$), then $\boldsymbol{Q}_k = \boldsymbol{t_f}^{(k)}\boldsymbol{t_0}^{(k)}$,
  - if T_j is not enabled but has memory in $\boldsymbol{m}_j$ (i.e. $T_k \in \mathcal{B}_j$) which can happen as a result of a sequence of firing, then $\boldsymbol{Q}_k = \boldsymbol{t_f}^{(k)}\boldsymbol{t_0}^{(k)}$.
- If $T_k \in (\mathcal{A}_i/\mathcal{L}) \cap (\mathcal{A}_j \cup \mathcal{B}_j)$, i.e T_k is enabled in $\boldsymbol{m}_i$, it does not fire and is enabled or has memory in $\boldsymbol{m}_j$, then $\boldsymbol{Q}_k = \boldsymbol{T}^{(k)}$.
- If $T_k \in (\mathcal{A}_i/\mathcal{L})$ and $T_k \notin \mathcal{A}_j \cup \mathcal{B}_j$, i.e. T_k gets disabled and does not have memory in $\boldsymbol{m}_j$, then $\boldsymbol{Q}_k = \boldsymbol{e}^{(k)} - \boldsymbol{t_f}^{(k)}$ (where $\boldsymbol{e}^{(k)}$ is a vector of size n_k with all entries equal to one).
- If $T_k \in (\mathcal{B}_i \cap \mathcal{B}_j)$, i.e. it is not enabled but has memory in both $\boldsymbol{m}_i$ and $\boldsymbol{m}_j$, then $\boldsymbol{Q}_k$ is the identity matrix of proper size (its size is $n_k \times n_k$ if T_k is of prs size, while its size is $b^{(k)} - a^{(k)} + 1 \times b^{(k)} - a^{(k)} + 1$ if T_k is of pri type).
- If T_k is of prs type and $T_k \in (\mathcal{B}_i \cap \mathcal{A}_j)$, i.e. it is not enabled but has memory in $\boldsymbol{m}_i$ and gets enabled in $\boldsymbol{m}_j$, then $\boldsymbol{Q}_k$ is the identity matrix of size $n_k \times n_k$.

- If T_k is of pri type and $T_k \in (\mathcal{B}_i \cap \mathcal{A}_j)$, i.e. it is not enabled but has memory in $\boldsymbol{m}_i$ and gets enabled in $\boldsymbol{m}_j$, then $\boldsymbol{Q}_k$ is of size $b^{(k)} - a^{(k)} + 1 \times n_k$ and defined as

$$[\boldsymbol{Q}_k]_{ij} = \begin{cases} 1, \text{ if } j = (i-1)a^{(k)} + i, \\ 0, \text{ otherwise.} \end{cases}$$

  This matrix insures that being re-enabled the local descriptor of the pri transition corresponds to the first phase of the proper column.
- If $T_k \notin (\mathcal{A}_i \cap \mathcal{B}_i)$ and $T_k \in (\mathcal{A}_j \cap \mathcal{B}_j)$, i.e. it is neither enabled nor has memory in $\boldsymbol{m}_i$ and either enabled or has memory in $\boldsymbol{m}_j$, then $\boldsymbol{Q}_k = \boldsymbol{t_0}^{(k)}$.
- If T_k is neither enabled nor has memory in both in $\boldsymbol{m}_i$ and $\boldsymbol{m}_j$, then $\boldsymbol{Q}_k = 1$, and hence does not have any influence on the Kronecker-product.

3.3 Complexity

The complexity of the proposed procedure is different for functional than for stochastic analysis for two reasons.

- The structure that describes the local evolution of the process is usually larger in case of stochastic analysis. In case of functional analysis, a transition with firing interval $[1:2]$ is represented by a two phase structure. In case of probabilistic analysis a discrete probability mass function is placed in the interval. If it is done with step-size 0.2 (which means that the firing time may be 1,1.2,1.4,...,2.0) 10 phases are needed.
- In case of functional analysis, a state described by a global descriptor is possible or not in a given time instant. So that the transient vectors contain 0s and 1s which can be stored efficiently using binary decision diagrams. In

case of probabilistic analysis the transient vectors contain real values whose storage requires more memory.

The length of the transient vector describing an extended marking is the product of the number of different values that the local descriptors may have in that marking. This product can be very large if many transitions are enabled or have memory in an extended marking.

The matrix describing the global evolution of the process is stored in a memory-efficient manner by using Kronecker-expressions. Different algorithms can be implemented to perform Kronecker-products, a comparison of the complexity of these algorithm is found in [10].

4 Examples

Two examples are given in this section. The first, simple one is presented to show that a net may have different functional behavior for different preemption policies. On the second example both functional and stochastic analysis are performed.

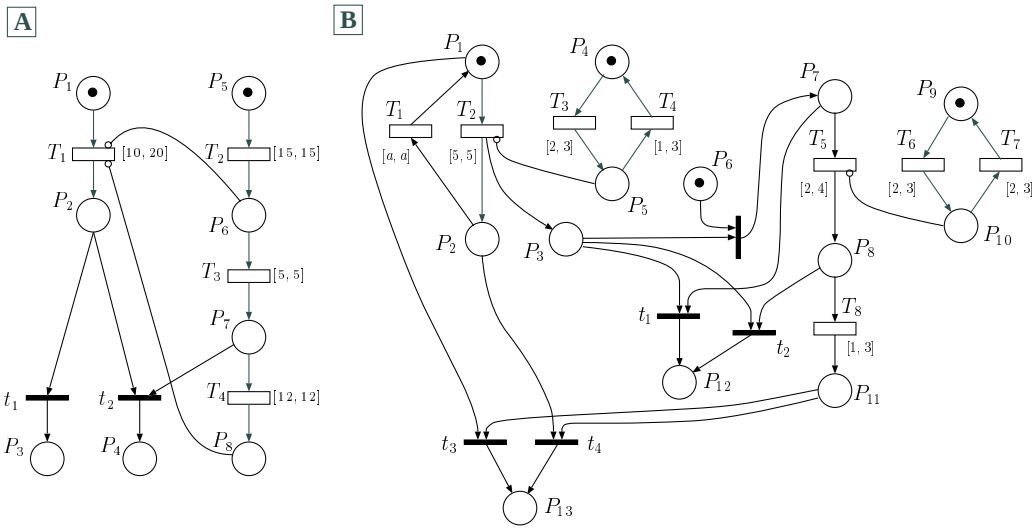

Fig. 6. **A:** Different reachable markings in case of adopting different preemption policies for T_1. **B:** Producer-consumer model.

The time Petri net depicted in Figure 6A has different reachable markings when adopting different preemption policies for transition T_1. We assume strong time semantics. Whichever policy is chosen, it is possible that T_1 fires before T_2 (if the firing time of T_1 is chosen lower than 15); in this case the net ends up in marking (P_3, P_8). If T_2 fires before T_1 the behavior depends on the memory policy of T_1:

- prd policy: T_1 either fires or does not fire in marking (P_1, P_7) (it fires if its rechosen firing time is less than 12),
- prs policy: T_1 fires in marking (P_1, P_7) since its "clock" starts from 15 entering this marking and its maximal firing time is 20,

- pri policy: T_1 does not fire in marking (P_1, P_7) since its firing time is not rechosen and it is greater than 15.

The second example, which is a simple producer-consumer model, is depicted in Figure 6B. Production, represented by deterministic transition T_2, may get preempted. The preemption is modeled by transitions T_3 and T_4, and places P_4 and P_5. Transition T_2 is of prs type, i.e. the work done is not lost in case of preemption. Production restarts after a time units (represented by transition T_1). Consumption consists of two steps represented by transitions T_5 and T_8. The first phase of consumption may get preempted which feature is modeled by the subnet P_9, P_{10}, T_6 and T_7. For transition T_5 prd or prs memory policy is considered. The aim of the analysis is to determine if the consumer finishes its two-phase job before the arrival of the next one. A token in place P_{12} indicates an error (i.e. another job arrived before the consumer finished the previous one), while a token in place P_{13} indicates that one cycle of production-consumption was successful. The model is evaluated with strong time semantics.

From the functional point of view possible questions are: "Is it possible that a token appears in place P_{12}?" or "What is the shortest, longest cycle-time?". From the stochastic point of view one could ask: "What is the probability that a token appears in place P_{12}?" or "What is the probability of successfully finishing a cycle before time t?". In case of performing stochastic analysis a discrete probability distribution is defined on the interval. For all non-deterministic transitions we assumed to have discrete uniform distribution with step-size 0.1 (for example the firing time of transition T_3 may be 2.0,2.1,2.2,...,2.9 or 3.0 with equal probability of 1/11).

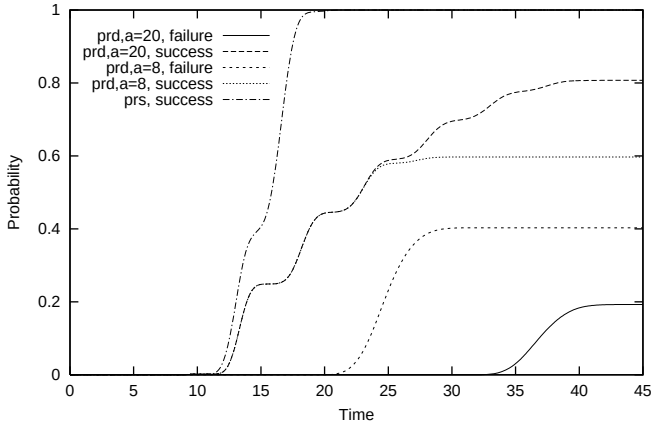

Fig. 7. Probability of failure or correctly finished cycle as a function of time

The example was evaluated, using a preliminary implementation of the presented Kronecker-based description, with $a = 8$, $a = 20$ and with either prd or prs memory policy for transition T_5. From the point of view of functional analysis the results are the following. For both values of a: if the adopted memory policy is prd consumption may either terminate in time or may not,

Analysis	prd, $a = 8$	prd, $a = 20$	prs, $a = 8$	prs, $a = 20$
Functional	2133	3645	3249	5625
Stochastic	14696100	24236100	16964100	46052100

Table 1
Size of the discretized state space

while for prs policy it terminates always before the next production. The earliest possible time to finish correctly the production and consumption is 9 time units for all the cases. As a probabilistic result, Figure 7 depicts the probability of having a correct or erroneous outcome as a function of time (in case of prs policy for transition T_5, erroneous outcome is not possible and for both values of a we have the same curve).

The size of the discretized state space for the different cases are given in Table 1. As one could observe functional analysis requires much smaller state space.

5 Conclusion

The paper has introduced a new construct called *Discrete Phase Type Timing - DPT* that can represent probabilistic or non-deterministic choice over an interval. For both cases it gives the possibility of assigning preemption policy to the transitions of the system. Both weak and strong time semantics can be handled (or even mixed in the same model).

A compositional approach, based on Kronecker algebra, was given to build the matrix that describes the evolution of the expanded state space. This description is similar to the one given in [21] with the differences that it follows the behavior of a prs transition in an exact manner and provides the possibility of having pri transitions in the model. It was shown as well that the same compositional description can be utilized for functional and for stochastic analysis.

Through a simple example the possibility of performing both functional and probabilistic analysis of the same model has been demonstrated.

References

[1] M. Ajmone Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani. The effect of execution policies on the semantics and analysis of stochastic Petri nets. *IEEE Transactions on Software Engineering*, 15:832–846, 1989.

[2] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* Wiley Series in Parallel Computing, 1995.

[3] M. Ajmone Marsan and G. Chiola. On Petri nets with deterministic and exponentially distributed firing times. In *Lecture Notes in Computer Science*, volume 266, pages 132–145. Springer Verlag, 1987.

[4] B. Berthomieu and M. Diaz. Modelling and verification of time dependent systems using Time Petri Nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, 1991.

[5] A. Bobbio, A. Horváth, M. Scarpa, and M. Telek. Acyclic discrete phase type distributions: Properties and a parameter estimation algorithm. Technical Report of Budapest University of Technology and Economics - Submitted for publication, 2000.

[6] A. Bobbio, V.G. Kulkarni, A. Puliafito, M. Telek, and K. Trivedi. Preemptive repeat identical transitions in Markov Regenerative Stochastic Petri Nets. In *Petri Nets and Performance Models '95*, pages 113–122. IEEE CS Press, 1995.

[7] A. Bobbio, A. Puliafito, and M. Telek. A modeling framework to implement combined preemption policies in MRSPNs. *IEEE Transactions on Software Engineering*, 26:36–54, 2000.

[8] A. Bobbio and M. Telek. Non-exponential stochastic Petri nets: an overview of methods and techniques. *Computer Systems: Science & Engineering*, 13(6):339–351, 1998.

[9] G. Bucci and E. Vicario. Compositional validation of time-critical systems using communicating time Petri nets. *IEEE Transactions on Software Engineering*, 21:969–992, 1995.

[10] P. Buchholz, G. Ciardo, P. Kemper, and S. Donatelli. Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models. *INFORMS Journal on Computing*, 13(3):203–222, 2000.

[11] Hoon Choi, V.G. Kulkarni, and K. Trivedi. Markov regenerative stochastic Petri nets. *Performance Evaluation*, 20:337–357, 1994.

[12] G. Ciardo. Discrete-time markovian stochastic petri nets. In W. J. Stewart, editor, *Computations with Markov Chains*, pages 339–358. Kluwer, 1995.

[13] G. Ciardo and A. Miner. A data structure for the efficient Kronecker solution of GSPNs. In *Proc. 8th Int. Workshop on Petri Nets and Performance Models (PNPM'99)*, pages 22–31. IEEE CS Press, 1999.

[14] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[15] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezzeè. A unified high level Petri net formalism for time-critical systems. *IEEE Transactions on Software Engineering*, 17:160–171, 1991.

[16] A. Horváth, A. Puliafito, M. Scarpa, and M. Telek. A discrete time approach to the analysis of non-Markovian stochastic Petri nets. In *Tools 2000*, volume 1786 of *Lecture Notes in Computer Science*, pages 171–187, Schaumburg, IL, USA, March 2000. Springer-Verlag.

[17] C. Lindemann. *Performance Modelling with Deterministic and Stochastic Petri Nets*. John Wiley, 1998.

[18] P. Merlin and D. J. Faber. Recoverability of communication protocols. *IEEE Transactions on Communication*, 24(9):1036–1043, 1976.

[19] M.K. Molloy. On the integration of delay and throughput measures in distributed processing models. Technical report, Phd Thesis, UCLA, 1981.

[20] S. Natkin. Les reseaux de Petri stochastiques et leur application a l'evaluation des systemes informatiques. Technical report, These de Docteur Ingegneur, CNAM, Paris, 1980.

[21] M. Scarpa and A. Bobbio. Kronecker representation of Stochastic Petri nets with discrete PH distributions. In *International Computer Performance and Dependability Symposium - IPDS98*, pages 52–61. IEEE CS Press, 1998.

[22] E. Vicario. Static analysis and dynamic steering of time dependent systems. *IEEE Transactions on Software Engineering (to be published)*, 2001.

[23] R. Zijal, G. Ciardo, and G. Hommel. Discrete deterministic and stochastic petri nets. In *Proc. Measurement, Modeling, and Valuation of Computer- and Communication-Systems (MMB)*, pages 103–117, Freiberg, Germany, 1997. VDE-Verlag.

Appendix A: Discrete phase type distributions

Possibly defective discrete phase type (PDDPH) distributions are defined in terms of discrete-time Markov chains (DTMC) with absorbing states. A discrete random variable X is PDDPH distributed if and only if there exists a DTMC $\{Z_i, i \geq 0\}$ with $n+1$ states of which the $(n+1)^{th}$ is absorbing (there can be other absorbing states as well) and $Pr\{X \leq i\} = Pr\{Z_i = n+1\}$, i.e. X is the time to reach state $n+1$. If state $n+1$ is not the only absorbing state the distribution can be defective. A PDDPH distribution is given by the initial probability vector of its DTMC ($\boldsymbol{t_0}$) and the one-step transition matrix ($\boldsymbol{T}$) that governs the evolution among the states excluding state $n+1$. Then the distribution of X is given by $Pr\{X \leq i\} = 1 - \boldsymbol{t_0 T}^i \boldsymbol{e}$, where $\boldsymbol{e}$ is a vector with all entries equal to 1.

Appendix B: Kronecker-product operator

The Kronecker-product $\boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{B}$ of matrix $\boldsymbol{A}$ of size $(a_r \times a_c)$ and matrix $\boldsymbol{B}$ of size $(b_r \times b_c)$ is of size $(a_r b_r \times a_c b_c)$ and is defined by

$$\boldsymbol{C}_{i,j} = \boldsymbol{A}_{i_2,j_2} \boldsymbol{B}_{i_1,j_1}, \text{ where } i = i_2 a_r + i_1, j = j_2 a_c + j_1.$$

Extending Timed Automata for Compositional Modeling Healthy Timed Systems

Víctor Braberman [1,2]

Computer Science Department, FCEyN,
Universidad de Buenos Aires,
Buenos Aires, Argentina

Alfredo Olivero [3,4]

Department of Information Technology, FIyCE,
Universidad Argentina de la Empresa,
Buenos Aires, Argentina

Abstract

We introduce the notion of Timed I/O Components as Timed Automata "à la" Alur & Dill where an "admissible" I/O interface is declared. That notion has, what we consider, a key modeling property: non-zeno preservation under syntactically-checkable "I/O compatibility" among interacting components. Also a reduced parallel composition is possible based on the ability of statically detect influence of behavior between components [8,10]. On the other hand, with some simple extra conditions, modular assume-guarantee style of reasoning like [13,16] is valid in our model.

1 Introduction: on Non-Zeno and Non-Blocking Models

Well-defined models of timed systems usually are required to be "non-zeno". Roughly speaking, non-zenoness means that any finite run can be extended to a time-divergent infinite run (i.e., no "black-alleys", time can always progress). On the one hand, zenoness is usually a symptom of ill-modeling, on the other hand non-zenoness is required to perform some verification procedures when semantics is restricted to divergent runs.

[1] Research supported by UBACyT grant X156 and TW72
[2] Email: vbraber@dc.uba.ar
[3] Research supported by UADE grant ING6-01
[4] Email: aolivero@uade.edu.ar

This is a preliminary version. The final version is considered for publication in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

Unfortunately, non-zenoness is not preserved by parallel composition. Non-Zeno systems may produce time-locks when connected.

I/O Timed Components (I/O TCs) are compositional models developed for expressing non-zeno timed behavior built on top of Timed Automata (TAs) [8]. They impose a modeling discipline for guaranteeing that parallel composition among "compatible" I/O TCs is a natural way to constrain the behavior of individual components without introducing zeno behavior. Let us pinpoint some interesting aspects of I/O TCs:

- I/O interfaces allow simple syntactic checks that ensure non-zeno preservation under parallel composition.
- By using I/O interfaces it is possible to calculate, in a quite precise way, the influence of a component on the behavior of another component (see [8,10]). As far as we know, this is a completely new goal for I/O interfaces.
- Since I/O TCs are built on top of a simple notion of TAs "à la" Alur-Dill -with a communication based on label sharing- they are immediately supported by several checking tools like Kronos [12], Uppaal [6],etc.
- I/O TCs are defined without resorting to "receptiveness games" like in live I/O Timed Automata [13], Reactive Modules [5], etc. Conditions for checking good I/O label division are easy to automate.
- We believe that they are suitable to compactly model high-level non-blocking abstractions (see discussion in [8] and Sect. 3).
- With some extra constraints on I/O TCs, it is possible to apply "assume-guarantee"-style rules (e.g.,[16]) for refinement checking. Like in [16] those constraints are not based on more general but complicated receptiveness games [13,5]). Those constraints are not basic properties for I/O TCs (which are mainly inspired in non-zeno preservation). We think that this separation has theoretical and practical interest.
- I/O notions are rather independent of the underlying timed (or untimed[5]) formalism used to describe the dynamics.

It is worth mentioning related work on preserving "reactivity and activity" of components. In [7], an algebraic framework based on the temporal properties of synchronization operation is presented (they aim at getting high level synchronization facilities). Our point of view is a functional classification of transitions. In that line of research, authors of [16] present non-blocking Timed Processes to get a family of automata where they can apply an assume/guarantee style of reasoning. Communication between components is based on signal change instead of label sharing and it is suited to circuit modeling. Differently from our approach, output changes are constrained to be

[5] We believe that this I/O model can be adapted to the untimed framework by changing timed divergence conditions with fairness constraints [11,13], an usual way to specify progress in the untimed framework.

non-transient and the update of inputs is independent from the update of outputs. Since that model is focused on breaking circularity of assume-guarantee rules, the underlying notion of non-zenoness does not need to rule out black-alleys; instead definitions rule out forcing infinitely many transitions within a finite interval.

Liveness and I/O interfaces have been considered in a general setting for simulation proof methods "à la" Lynch-Vaandrager [13] geared towards theorem provers. In that work, Live Timed I/O automata using a notion of "responsiveness" is defined based on games which embeds several proposals for fair I/O timed systems [17,21], etc. A closer model are the Reactive Modules of [5]. Unlike our notions, it is based on receptiveness games to define non-blocking and I/O variables to communicate modules.

In next section we recall Timed Automata. In Sect. 3, we formally present I/O Timed Components. Some applications are mentioned in Sect. 4. Conditions to get assume-guarantee rule are discussed in Sect. 5. Finally, we summarize the results and mention some future work.

2 Timed Automata

Timed Automata (TA) is one of the most widely used formalism to model and analyze timed systems and is supported by several tools (e.g., [12,6,15], etc.). This presentation partially follows [23]. Given a finite set of clocks (non-negative real variables) $X = \{x_1, x_2, \ldots, x_n\}$, a *valuation* is a total function $v : X \stackrel{tot}{\rightarrow} \mathbb{R}_{\geq 0}$ where $v(x_i)$ is the value associated with clock x_i. We define V_X as the set $[X \stackrel{tot}{\rightarrow} \mathbb{R}_{\geq 0}]$ of total functions mapping X to $\mathbb{R}_{\geq 0}$. $\mathbf{0} \in V_X$ denotes the function that evaluates to 0 all clocks. Given $v \in V_X$ and $t \in \mathbb{R}_{\geq 0}$, $v+t$ denotes the valuation that assigns to each clock $x \in X$ the value $v(x)+t$. Given a set of clocks X, a subset $\alpha \subseteq X$ and a valuation v we define $Reset_\alpha(v)$ as a valuation that assigns zero to clocks in α and keeps the same value than v for the remaining clocks. Given a set of clocks X we define the sets of clock constraints Ψ_X according to the grammar: $\Psi_X \ni \psi ::= x \sim c | x - x' \sim c | \psi \wedge \psi | \psi \vee \psi$, where $x, x' \in X, \sim \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$.

A valuation $v \in V_X$ satisfies $\psi \in \Psi_X$ ($v \models \psi$) iff the expression evaluates **true** when each clock is replaced with its current value specified in v.

Definition 2.1 [Timed Automata] A timed automaton (TA) is a tuple $A = \langle S, X, \Sigma, E, I, s_0 \rangle$ where S is a finite set of locations, X is a finite set of clocks, Σ is a set of labels, E is a finite set of edges, (each edge $e \in E$ is a tuple $\langle s, a, \psi, \alpha, s' \rangle$ where: $s \in S$ is the source location, $s' \in S$ is the target location, $a \in \Sigma$ is the label, $\psi \in \Psi_X$ is the guard, $\alpha \subseteq X$ is the subset of clocks reset at the edge), $I : S \stackrel{tot}{\rightarrow} \Psi_X$ is a total function associating with each location a clock constraint called location's Invariant, and $s_0 \in S$ is the initial location.

Given a TA $A = \langle S, X, \Sigma, E, I, s_0 \rangle$ we define $Locs(A) = S$, $Clocks(A) =$

X, $Labels(A) = \Sigma$, $Edges(A) = E$, $Inv(A) = I$, $Init(A) = s_0$, and given an edge $e = \langle s, a, \psi, \alpha, s' \rangle \in E$ we define $src(e) = s$, $Label(e) = a$, $Guard(e) = \psi$, $Rst(e) = \alpha$, $tgt(e) = s'$. The *State Space* Q_A of a TA A is the set of states $(s, v) \in S \times V_X$ for which $v \models I(s)$ and $q_0 = (Init(A), \mathbf{0})$ is its *initial state*. Given a state $q = (s, v)$ we denote: $q + t = (s, v + t)$, $q^{@} = s$, and $q(x_i) = v(x_i)$. The semantics of a TA A can be given in terms of the *Labeled Transition System* (LTS) of A, denoted $G_A = \langle Q_A, q_0, \mapsto, \Sigma \rangle$. The relation $\mapsto$ is the set of (time or discrete) transitions between states. Let $t \in \mathbb{R}_{\geq 0}$; the state (s, v) has a *time transition* to $(s, v + t)$ denoted $(s, v) \mapsto_t^{\lambda} (s, v + t)$ if for all $t' \leq t$, $v + t' \models I(s)$, where λ is a fictitius label. Let Σ^* denote $\Sigma \cup \{\lambda\}$. Let $e \in E$ be an edge; the state $(src(e), v)$ has a *discrete transition* to the state $(tgt(e), v')$ denoted $(src(e), v) \mapsto_0^{Label(e)} (tgt(e), v')$ if $v \models Guard(e)$ and $v' = Reset_{Rst(e)}(v)$.

We write $q \mapsto_0^l$ (the label $l \in \Sigma$ is enable at the state $q \in Q_A$) if $q \mapsto_0^l q'$ for some $q' \in Q_A$. Given a subset $\Sigma' \subseteq \Sigma$, we write $q \mapsto_0^{\Sigma'}$ (all labels $l \in \Sigma'$ are enable at the state $q \in Q_A$) if $q \mapsto_0^l$ for all $l \in \Sigma'$.

A *finite run* r of A starting at q is a finite sequence $q \mapsto_{t_0}^{a_0} q_1 \mapsto_{t_1}^{a_1} ... \mapsto_{t_{n-1}}^{a_{n-1}} q_n$ of states and transitions in G_A. The *time of occurrence* of the k^{th} $(k \leq n-1)$ transition is equal to $\sum_{i=o}^{k-1} t_i$ and is denoted as $\tau_r(k)$. The *time length* of the run (denoted as τ_r) is equal to $\tau_r(n)$. An infinite run is just an infinit sequence of states and transitions in G_A. The set of finite and infinite runs starting at q is denoted as $R_A(q)$. We call $Lab(r)$ the set of all labels in the run r.

A *divergent run* is an infinite run such that $\sum_{i=o}^{\infty} t_i = \infty$. The set of divergent runs of a TA A starting at state q is denoted $R_A^{\infty}(q)$. A TA is *non-zeno* when any finite run starting at the initial state can be extended to a divergent run, that is, the set of finite runs is equal to the set of finite prefixes of divergent runs. We say that the state q is *reachable* if there is a finite run starting at the initial state which ends at q; we denote the set of all reachable states in a TA A as $Reach(A)$.

Given a run $r = q \mapsto_{t_0}^{a_0} q_1 \mapsto_{t_1}^{a_1} ... \mapsto_{t_{n-1}}^{a_{n-1}} q_n ... \in R_A(q)$, the exhibited *timed-event sequence* of r, is a sequence $r_{\Sigma^*} = (a_0, \tau_r(0)), (a_1, \tau_r(1)), (a_2, \tau_r(2)), ...(a_{n-1}, \tau_r(n-1)), ...$ of pairs $(l, t) \in (\Sigma^*) \times \mathbb{R}_{\geq 0}$.

Given a run $r \in R_A(q)$ and a set of labels $L \subseteq \Sigma$, the exhibited *timed-event sequence over* L, denoted as r_L, is the maximum subsequence of r_{Σ^*} containing pairs (l, t) such that $l \in L$. (the sequence r_L shows the L-labeled transitions and their time stamps). Given a timed-event sequence over L named r_L, its length is denoted as $\#r_L$, its k-th pair (with $k < \#r_L$) is denoted as $r_L[k]$ and its prefix up to the k-th pair (with $k < \#r_L$) is denoted as $r_L[0...k]$. Given a pair $p = (l, t)$ in r_L, we define $lab(p) = l$ and $time(p) = t$.

Given two TAs A and A' and a set of labels $L \subseteq \Sigma \cap \Sigma'$, we say that $A \leq_L A'$ (A is a *refinement* of A' w.r.t L) iff for all finite run $r \in R_A(q_0)$ there exists a run $r' \in R_{A'}(q_0')$ such that $\tau_r = \tau_{r'}$ and $r_L = r'_L$.

The parallel composition of TAs is defined over classical synchronous prod-

uct of automata.

Definition 2.2 [Parallel composition] Given two TA $A_1 = \langle S_1, X_1, \Sigma_1, E_1, I_1, s_{0_1}\rangle$, and $A_2 = \langle S_2, X_2, \Sigma_2, E_2, I_2, s_{0_2}\rangle$ where $X_1 \cap X_2 = \emptyset$. Let E' be the set of edges defined over the $S_1 \times S_2$ as follows:

$\langle (s_1, s_2), a, \psi, \alpha, (s_1', s_2')\rangle \in E' \iff$

$\langle s_1, a, \psi, \alpha, s_1'\rangle \in E_1 \wedge a \notin \Sigma_{||} \wedge s_2 = s_2'$, or

$\langle s_2, a, \psi, \alpha, s_2'\rangle \in E_2 \wedge a \notin \Sigma_{||} \wedge s_1 = s_1'$, or

$\langle s_i, a, \psi_i, \alpha_i, s_i'\rangle \in E_i \wedge a \in \Sigma_{||} \wedge \psi = (\psi_1 \wedge \psi_2) \wedge \alpha = \alpha_1 \cup \alpha_2$

where $\Sigma_{||} = \Sigma_1 \cap \Sigma_2$.

The parallel composition $A_1||A_2$ is defined as: $A = \langle S, X_1 \cup X_2, \Sigma_1 \cup \Sigma_2, E, I, (s_{0_1}, s_{0_2})\rangle$ where $S \subseteq S_1 \times S_2$ is the set of locations reachable traversing the edges of E' from the initial location (s_{0_1}, s_{0_2}), $E \subseteq E'$ is the subset of edges with source and target in S, and for all $(s_1, s_2) \in E$, $I((s_1, s_2)) = I(s_1) \wedge I(s_2)$.

The $\|$ operator is commutative and associative. We will denote $\|_{i \in I}\ A_i$ the parallel composition of an indexed set of TA. If q is a state of that parallel composition $\Pi_i(q)$ will denote the local state of TA A_i (locations and local-clock values).

3 I/O Timed Components

In this section we define I/O concepts formally.

Given a TA A, we will divide $Labels(A)$ (its set of labels) into three sets: In_A (input-labels), Out_A (output-labels) and ϵ_A (internal-labels), such that $\{In_A, Out_A, \epsilon_A\} \in Part(Labels(A))$, where $Part(S)$ is the set of all partitions of the set S. We define the set Exp_A of *exported* labels (or interface labels) of A as $Exp_A = In_A \cup Out_A$.

A set of *input selections* of A is a set $I^A = \{I_1^A, I_2^A, \ldots, I_k^A\} \in Part(In_A)$, a set of *output selections* of A is a set $O^A = \{O_1^A, O_2^A, \ldots, O_h^A\} \in Part(Out_A)$. Note that $I^A \cup O^A \cup \{\epsilon_A\} \in Part(Labels(A))$.

Let us define what is a correct I/O (uncontrollable/controllable) interface labels for a TA.

Definition 3.1 [Admissible Input/Output interface for a TA] Given a non-zeno TA A, and the sets I^A, O^A of input and output selections of A, the pair (I^A, O^A) is an admissible input/output interface for A iff the following conditions hold:

For any state $q \in Reach(A)$

(i) for any input selection $I_n^A \in I^A$ there exists a label $i \in I_n^A$ such that $q \mapsto_0^i$. That is, given any input selection $I_n^A \in I^A$, the TA can always synchronize using some of the labels of I_n^A (there is always at least one alternative of every input selection enabled at each state).

(ii) there exists a run $r \in R_A^\infty(q)$ such that $Lab(r) \cap In_A = \emptyset$. Input is not mandatory and thus non-zenoness must be guaranteed without them [6].

(iii) for any output selection $O_m^A \in O^A$, if there exists a label $o \in O_m^A$ such that $q \mapsto_0^o$ then $q \mapsto_0^{O_m^A}$. All labels of an output selection are simultaneously enabled or disabled.

(iv) for any run $r \in R_A(q)$, if a label $o \in Out_A$ appears an infinite number of times in r, then necessarily $r \in R_A^\infty(q)$ (non-transientness of outputs [7]).

In the Appendix A.1 we show how to check I/O admissibility.

Definition 3.2 [I/O TCs] An I/O Timed Component (or I/O TC) is a tuple $(A, (I^A, O^A))$ where A is a non-zeno TA and (I^A, O^A) is an admissible I/O interface for A.

An output selection of size greater than one models alternative behaviors of the component according to the state of the component exporting those labels as input selection (similar to an external non-deterministic choice in Process Algebra-like notations, see example 3.4).

Given an I/O TC $C = (A, (I^A, O^A))$, C may also denote the underlying TA A when it can be deduced from the context. Thus, operations performed on I/O TCs should be understood as operations on its underlying TAs.

Definition 3.3 [Compatible Components] Given two I/O TCs $C_1 = (A_1, (I^{A_1}, O^{A_1}))$ and $C_2 = (A_2, (I^{A_2}, O^{A_2}))$, they are compatible components if and only if:

(i) $Labels(A_1) \cap Labels(A_2) \subseteq Exp_{A_1} \cap Exp_{A_2}$ (i.e., all common labels are exported by both A_1 and A_2),

(ii) for all $I_n^{A_1} \in I^{A_1}$ and $I_m^{A_2} \in I^{A_2}$ if $\#I_n^{A_1} > 1$ and $\#I_m^{A_2} > 1$ then $I_n^{A_1} \cap I_m^{A_2} = \emptyset$ (intersection of input selections of size greater than one must be empty).

(iii) $Out_{A_1} \cap Out_{A_2} = \emptyset$ (the components don't share output labels).

(iv) for all $I \in I^{A_1} \cup I^{A_2}$ and $O \in O^{A_1} \cup O^{A_2}$ then either $I \cap O = \emptyset$ or $I \subseteq O$ (output selection covers all input alternatives).

We refer to a set of pair-wise compatible components as a *compatible set of components*. I/O compatibility means that underlying TAs can not block each other and moreover, we will show that the composition of compatible components is itself a component and therefore a non-zeno automata.

[6] Note that this property is stronger than non-zenoness since it also requires time divergence avoiding input-labeled transitions. It is similar to progressiveness in [19] and feasibility in [21].

[7] This requirement together with the previous divergence property (item (ii) of Def. 3.1) and non-zenoness of the underlying TA are closely related to the notion of Strong I/O Feasibility of [21].

Example 3.4 CSMA/CD (Carrier Sense, Multiple Access with Collision Detection) is widely used protocol on LANs on the MAC sublayer. It solves the problem of sharing a single channel in a broadcast network (a multi-access channel). When a station has data to send it first listens to the channel to check whether it is idle or busy. If the bus seems idle it begins sending the message, else it waits a random amount of time and then repeats the sensing operation. When a collision occurs, the transmission is aborted simultaneously in all the stations that were transmitting and they wait a random time to start all over again. We formally model the timing aspects of the protocol using I/O timed components (see Fig.1) based on the model presented in [18]. Sender components share a bus component. We suppose that the bus is a 10Mbps Ethernet with worst case propagation delay σ of 26 ms. Messages have a fixed length of 1024 bytes, and so the time λ to send a complete messages, including the propagation delay, is 808 ms. The bus is error-free, no buffering of incoming messages is allowed. Note that $\{SendOK_i, SendBusy_i\}$ is an output selection of sender i and the selection depend on the input actually enabled in the bus state. In fact, $SendBusy_i$ is enabled when the head of a message has already propagated. It takes at most σ to propagate the collision signal to all the senders. The sender stays at most δ in the transmission location. Note also that the sender non-deterministically makes a new attempt to send before 2σ elapsed since the last attempt. In models like Timed Process [16], it would be necessary for the sender component to issue a signal standing for the sensing of the bus state, and then wait for the status answer of the bus component (which can not arrive at zero time due to a "non-immediate response" constraint in that model). That two phase modeling idiom, common in software models, can be reduced in our modeling framework using appropriate Input and Output selections.

In [8], the reader can found how several examples taken from the literature are modeled as I/O TCs.

3.1 I/O Components: Composition and Non-Zenoness

Let us state some results that help to prove that a TA-model is non-zeno. Firstly, we will see how an admissible interface can be derived for the parallel composition of two compatible I/O TCs. This is a rather strong result which implies the following fact: given two compatible I/O TCs C_1 and C_2 then the composition, which turns out to be non-zeno, is also a I/O TC (i.e., $A_1 \parallel A_2$ is non-zeno and moreover it can be given an admissible I/O interface). Briefly, the new input interface is constituted by the original input selections that do not loose "selectivity property" of item (i) of Def. 3.1. That property is preserved for any input selection whenever there is no matching output selection and it is not properly included into another input selection.

Something similar can be done to build the new output interface. Since output selections that intersect with input selections of size greater than one

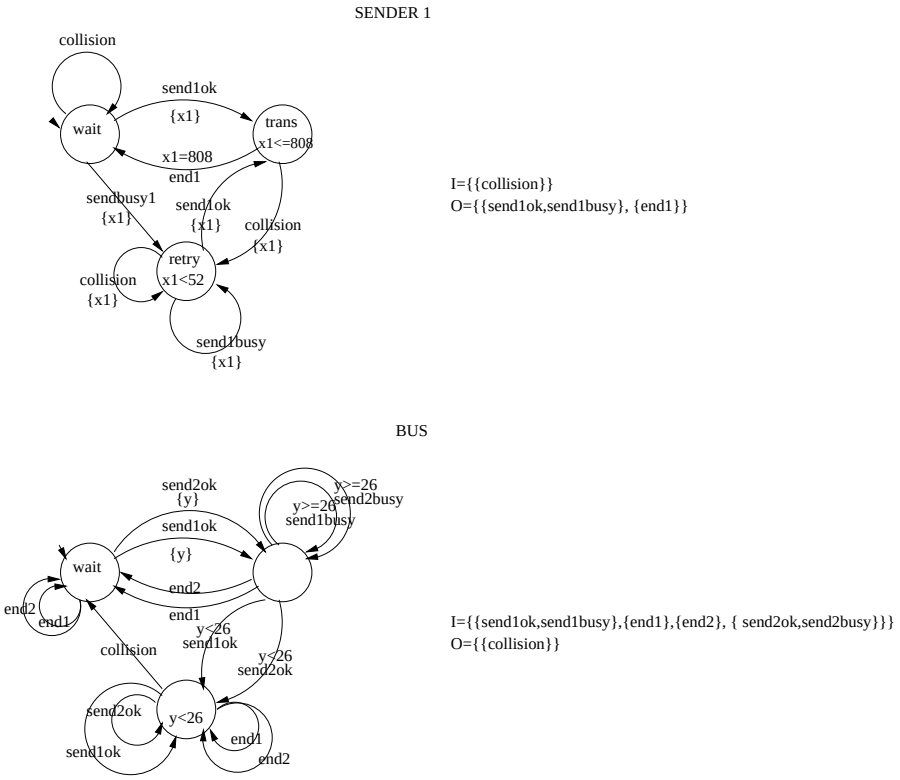

Fig. 1. I/O Components of the CSMA/CD Protocol

may loose the simultaneous availability property (item (iii) of Def. 3.1), they are not part of the new output selections. However, all the labels of those "lost" output selections can be safely added as output selections of size 1 (singletons trivially satisfy item (iii) of Def. 3.1). Thus, all exported labels of the components are exported in the composition. This fact is important to prove that this construction can be generalized to the parallel composition of n components:

Theorem 3.5 *Given an indexed set* $S = \{(A_i, (I^{A_i}, O^{A_i})\}_{1 \leq i \leq n}$ *of* n *I/O TCs such that they are pair-wise compatible, we define the sets* $I_A = \bigcup_{1 \leq i \leq n} I^{A_i}$, $I_{A_{>1}} = \bigcup_{1 \leq i \leq n} \{I \in I^{A_i} / \#I > 1\}$, $O_A = \bigcup_{1 \leq i \leq n} O^{A_i}$.

$(A, (I^A, O^A))$ *is a component where:*

$A = \|_{1 \leq i \leq n} A_i$

$I^A = \{I \in I_A / \ \forall I' \in I_A : I \not\subset I' \wedge \forall O' \in O_A : I \cap O' = \emptyset\}$ *and,*

$O^A = \{O \in O_A / \ \forall I' \in I_{A_{>1}} : I' \cap O = \emptyset\} \cup \{\{o\} / o \in O \in O_A \wedge \exists I' \in I_{A_{>1}} : I' \cap O \neq \emptyset\}$

Proof. See Appendix A. The basic idea is that, from the point of view of a component, its partners do not block its outputs: they just select them (items (i) and (iii) of Def. 3.1), also it does not require inputs to allow time elapse (item (ii) of Def. 3.1). On the other hand, a subset of I/O TCs can not engage themselves in an infinite activity in a finite interval of time since this

is ruled out by item (iv of Def. 3.1). □

In the example 3.4 the resulting interface of the parallel composition $A =^{def} SENDER_1 \parallel BUS$ is $I^A = \{\{Send2ok, Send2Busy\}, \{end2\}\}, O^A = \{\{end1\}, \{Send1ok\}, \{Send1Busy\}, \{collision\}\}$. Note that since simultaneous availability of output selection $\{Send1ok, Send1Busy\}$ is lost, they became singleton output-selections.

4 Applications of I/O TCs

Non Zeno Models:

Compatibility is a syntactical condition that ensures non-zenoness of the resulting parallel composition. As was already said, non-zenoness is a property required to perform some verification procedures. In [8,9] we model Real-Time System execution architectures by means of I/O TCs. We use I/O compatibility to ensure that I/O TCs modeling the connectors and the environment do not block the rest of the system (the tasks). As was already explained, I/O selections may be an useful mechanism to model in a single transition action/result on software entities.

Reduction:

Safety requirements are commonly modeled by means of virtual components (Observers) which are composed in parallel with the system under analysis (*SUA*) (e.g., [1,9]). In [8,10] we present a technique that, given the *SUA* and an observer, builds a smaller parallel composition equivalent to the original one up to the branching structure of the LTS. In a few words, we develop a technique that calculates the components that may be forgotten at each observer location since their future behavior do not influence the future evolution of the *SUA* up to the observer. Under some reasonable assumptions on the topology of the observers, those remaining sets (the relevant components) are proper subsets of the set of all components. The time needed for verification is drastically reduced in some cases. The core of that technique is a notion of potential "direct influence" of an automaton behavior over another automaton behavior. A naive solution would say that an automaton A potentially influences another automaton B iff they share a label. Unfortunately, this would lead to a rather large symmetrical overestimation. Then, by using the I/O interface attached to TAs, we are able to define an asymmetrical condition of behavioral influence that could be statically checked. That is, we provide a better overestimation of potential influence than simple label sharing. It is worth mentioning that the technique presented in [10] is based on a simpler notion of I/O interface than the one presented in this article. The details of that "relevance calculus" using the definitions of this paper can be found in [8].

5 On Breaking Circularity in Assume-Guarantee Rules

The authors of [16] present a simple modularity principle for abstraction relations in Timed Processes. Assume-guarantee rule has an apparent circularity: to prove that $A \parallel B$ is a refinement of $A' \parallel B'$ it suffices to prove that (1) A is a refinement of A' assuming that the environment behaves like B', and (2) B is a refinement of B' assuming that the environment behaves like A'. For this rule to be true in our setting, we have to add a couple of conditions. Firstly, let us define when an state is non urgent from the point of view of outputs.

Definition 5.1 [Non-Urgent state] Given a I/O TC $C = (A, (I^A, O^A))$, a state q is not output urgent (denoted as $NU(q)$) iff there exists a run $r \in R_A(q)$ such that $0 < \tau_r$ and $Lab(r) \subseteq \epsilon_A$.

Definition 5.2 [Non-Blocking Extra Conditions] We say that an I/O TC satisfies the Non-Blocking Extra Conditions if and only if:

(i) Guards and Invariants are closed predicates (i.e., its binary relations are only $\leq$, $=$ or $\geq$).

(ii) Inputs do not disable nor enable urgent outputs: given a state $q \in Reach(A)$ and a label $i \in In_A$, if $q \mapsto_0^i q'$ then $NU(q)$ iff $NU(q')$ [8] .

It is easy to see that those properties are preserved by parallel composition $A = \|_{j \in J} A_j$. Firstly note that guards and invariants of A are inherited from the components A_j. For the item (ii) of Def. 5.2, if $q \mapsto_0^i q'$ then i is an unmatched input of one component, namely k, and thus q and q' just differ in the local state of k. Also, $NU(q)$ if and only if for all $j \in J$ $NU(\Pi_j(q))$ (since the set of internal labels of the composition A is the union of internal labels of components A_j). Therefore, $NU(q)$ iff $NU(q')$ since $NU(\Pi_k(q))$ iff $NU(\Pi_k(q'))$ and the rest of the components remain the same.

Theorem 5.3 (Assume/Guarantee) *Given the I/O TCs A, B, A', B' satisfying the non-blocking extra conditions such that A and B are I/O compatible, and A' and B' have the same I/O interface that A and B resp. If $(A \parallel B') \leq_{Exp_A} A'$ and $(A' \parallel B) \leq_{Exp_B} B'$ imply that $(A \parallel B) \leq_{Exp_A \cup Exp_B} (A' \parallel B')$.*

Proof. See appendix. □

6 Conclusions and Future Work

We present I/O Timed Components, a simple compositional notion that extends Timed Automata "à la" Alur-Dill to get live non-zeno models [8], also providing some important methodological advantages like influence detection

[8] This property can be checked, for instance, using the verification engine of Kronos tool[12].

[10]. Assume-guarantee modular reasoning like [16] is obtained by adding a couple of constraints to I/O TCs without resorting to games. In our opinion, keeping non-zeno preservation conditions apart from the ones that break circularity in assume guarantee has practical and theoretical value.

We believe that admissible interfaces of a TA could be ordered according to the information it provides about availability of labels. That is, $(I_1, O_1) \leq (I_2, O_2)$ iff the admissibility of the interface (I_1, O_1) for a TA A implies the admisibility of (I_2, O_2) for A. We would like to study if this relationship between interfaces could be a declarative way to define the I/O interface of the composition.

Conditions for assume-guarantee could be weakened, for instance: it is sufficient for A and B to satisfy that inputs do not enable urgent outputs, and for A' and B' to satisfy that inputs do not disable urgent outputs.

References

[1] Alpern, B., and F. Schneider, *Verifying Temporal Properties without Temporal Logic*, ACM Trans. Programming Languages and Systems, **11 (1)** (1989), 147–167.

[2] Alur, R., "Techniques for Automatic Verification of Real-Time Systems," Ph.D. thesis, Stanford University, 1991.

[3] Alur, R., C. Courcoubetis, and D. Dill, *Model-Checking for Real-Time Systems* In Proceedings of Logic in Computer Science, IEEE Computer Society, Los Alamitos, Calif, 414-425, 1990. Also in Information and Computation, **104 (1)** (1993) 2–34.

[4] Alur, R. and D. Dill, *A Theory of Timed Automata*, Theoretical Computer Science, **126** (1994) 183–235.

[5] Alur, R., and T. Henzinger, *Modularity for Timed and Hybrid Systems,* In Proceedings of CONCUR'97, LNCS 1243, 1997.

[6] Bengtsson, J., K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi, *UPPAAL- A Tool Suite for the Automatic Verification of Real-Time Systems,* In Proceedings of Hybrid Systems III, LNCS 1066, Springer Verlag, 1996, 232–243.

[7] Bornot, S., and J. Sifakis. *An Algebraic Framework for Urgency* To appear in Information and Computation, Academic Press.

[8] Braberman, V. "Modeling and Checking Real-Time System Designs," Ph.D Thesis, Universidad de Buenos Aires, 2000. [Thesis]

[9] Braberman, V., and M. Felder, *Verification of Real-Time Designs: Combining Scheduling Theory with Automatic Formal Verification,* In Proceedings of 7th European Conf. on Software Eng./ 7th ACM SIGSOFT Symposium on the Foundations of Software Eng., (ESEC/FSE 99), LNCS 1687, Springer Verlag, Sept. 1999, 494–510.

[10] Braberman, V., D. Garbervetski, and A. Olivero, *Ignoring Components during the Verification of Timed Systems* Submitted to IEEE RTSS'01.

[11] Clarke, E., O. Grumberg and D. Peled, "Model Checking", MIT Press, January (2000), 330pp..

[12] Daws, C., A. Olivero, S. Tripakis, and S. Yovine, *The Tool KRONOS,* In Proceedings of Hybrid Systems III, LNCS 1066, Springer Verlag, 1996, 208–219.

[13] Gawlick, R., R. Segala, J. Sogaard-Andersen, N. Lynch *Liveness in Timed and Untimed Systems,* In Proceedings of ICALP , LNCS 820, Springer Verlag, 166-177, 1994. Also in Information and Computation (1998).

[14] Henzinger, T.A., X. Nicollin, J. Sifakis, and S. Yovine, *Symbolic model checking for real-time systems.* Information and Computation, **111(2)** (1994), 193–244.

[15] Larsen, K.G., F. Laroussinie, *CMC: A Tool for Compositional Model-Checking of Real-Time Systems,* In Proceedings of. FORTE-PSTV'98, 439-456, Kluwer Academic Publishers, 1998.

[16] Kurshan, R. P., S. Tasiran, R. Alur, and R. K. Brayton, *Verifying Abstractions of Timed Systems,* In Proceedings CONCUR 96, LNCS 1119, Springer Verlag, 1996.

[17] Merritt, M., F. Modugno, and M. Tuttle, *Time Constrained Automata,* In Proceedings of CONCUR'91, LNCS 527, Springer Verlag, 1991.

[18] Nicollin, X., J. Sifakis, and S. Yovine, *Compiling Real-Time Specification into Extended Automata,* IEEE Trans. on Soft. Eng.,Vol. **18 (9)** (1992), 794–804.

[19] Springintveld, J., F. Vaandrager, P. D'Argenio , *Testing Timed Automata*, To appear in Theoretical Computer Science, **254 (1-2)** (2001), 225–257.

[20] Tripakis, S. "L'Analyse Formelle des Systemès Temporisés en Practique", Phd. Thesis, Univesité Joseph Fourier, December 1998.

[21] Vaandrager, F., N. Lynch, *Action Transducers and Timed Automata,* In Proceedings of CONCUR'92, LNCS 630, 436-455, 1992.

[22] Yi, Wang, *Real-Time Behavior of Asynchronous Agents,* In Proceedings of CONCUR'90, LNCS 458, Springer Verlag, 1990.

[23] Yovine, S., *Model-Checking Timed Automata,* Embedded Systems, G. Rozemberg and F. Vaandrager eds., LNCS 1494, Springer Verlag, 1998.

Appendix

A On I/O Timed Components

Lemma A.1 *Given two I/O-compatible components* $C_1 = (A_1, (I^{A_1}, O^{A_1}))$ *and* $C_2 = (A_2, (I^{A_2}, O^{A_2}))$*, we define the sets* $I_A = I^{A_1} \cup I^{A_2}$*,* $I_{A_{>1}} = \{I \in I_A / \#I > 1\}$*,* $O_A = O^{A_1} \cup O^{A_2}$*.*

$C = (A, (I^A, O^A))$ *is a component where:*
$A = \|_{1 \leq i \leq n} A_i$
$I^A = \{I \in I_A / \ \forall I' \in I_A : I \not\subset I' \wedge \forall O' \in O_A : I \cap O' = \emptyset\}$ *and,*
$O^A = \{O \in O_A / \ \forall I' \in I_{A_{>1}} : I' \cap O = \emptyset\} \cup \{\{o\} / o \in O \in O_A \wedge \exists I' \in I_{A_{>1}} : I' \cap O \neq \emptyset\}$

Proof. The most difficult point is the proof that $A_1 \parallel A_2$ is indeed non-zeno regardless input transitions (item (ii), Def. 3.1). We will see that any state reachable *by a finite run* is not a timelock. Moreover, time can elapse avoiding input transitions. Let q be a reachable state by a finite run of $A_1 \parallel A_2$ then $q^1 = \Pi_{A_1}(q)$ and $q^2 = \Pi_{A_2}(q)$ are reachable states (by finite runs) of A_1 and A_2 resp. Let $k \in \mathbb{R}_{\geq 0}$ be a constant. From the definition of component, there must be runs r_1, r_2 starting in q^1 and q^2 resp. of time length equal to k such that r_1 does not contain any In_{A_1} transition and r_2 does not contain any In_{A_2} transition (thus they do not contain any label in I^A). Now, we show a procedure to obtain a run r of $A_1 \parallel A_2$ from r_1 and r_2. To obtain such a run, we would need to merge r_1 and r_2. If the discrete transitions of r_1 and r_2 are sorted according to the time of occurrence, it is easy to combine them obtaining r till the first output-labeled transition which shared by the other automaton is found. To outline the merge, lets $r_1 = q^1 \mapsto^{l_1}_{t_1} q^1_1 \mapsto^{l_2}_{t_2} ...q^1_n$, and $r_2 = q^2 \mapsto^{l'_1}_{t'_1} q^2_1 \mapsto^{l'_2}_{t'_2} ...q^2_{n'}$. Now, suppose that $t_1 \leq t'_1$ (the other case is symmetrical) and l_1 is not shared by A_2 (or it is λ). Then - thanks to the parallel composition interleaving semantics - the resulting run r can be build as follows: $r = q \mapsto^{l_1}_{t_1} (q^1_1, q^2 + t_1)$ concatenated with the run obtained using the same procedure from $(q^1_1, q^2 + t_1)$ with $r_1 = q^1_1 \mapsto^{l_2}_{t_2} ...$, and $r_2 = q^2 + t_1 \mapsto^{l'_1}_{t'_1 - t_1} q^2_1 \mapsto^{l'_2}_{t'_2} ...q'_{n'}$. Clearly this procedure can be iterated finitely till we reach the end of both runs (the variant is sum of the number of transitions of both runs), thus obtaining a run of A of time length k, or till a shared label is found.[9]

Without loss of generality, let us suppose that the earliest still non synchronized shared output-transition $q_j \mapsto^o_0 q_{j+1}$ belongs to r_1 and $o \in O \in O^{A_1}$. Let $I \in I^{A_2}$, $I \subseteq O$ be the corresponding matching input selection (i.e., $o \in I$) by compatibility (item (iv), Def. 3.3). By definition of input selection, there is a transition labeled $i' \in I$ enabled in A_2 at the time of occurrence of that j^{th} transition. By definition of output selection, at q_j there must be also a discrete transition $q_j \mapsto^{i'}_0 s$. By applying this procedure, we can fix up both runs to get a finite run starting at q such that either it has time length k or it ends with an output transition into an intermediate state q'. Therefore, since both TA are non-transient for output labeled transitions (item (iv) of I/O interface admissibility), by repeating the whole procedure from those in-

[9] Note that if one of the runs is empty then it just remains a set of discrete (0 time) transitions in the other run (both have originally the same time length) and therefore we can omit that suffix since we have already built a run of time length k.

termediate states (i.e., obtaining new r_1, r_2, etc.), a run of time length k is eventually built (if not, either the projection of that infinite run on A_1 or A_2 would show an infinite number of output-labeled transitions, and since there is a finite number of labels at least one output label would be repeated infinitely often thus violating item iv of I/O interface admissibility). The rest of the items of I/O interface are proven as follows:

- the new input and output labels are disjoint (input selections intersecting with an output selections are not part of the new interface).
- Input Selection Property (item (i)): given an state q of A and an input selection I of I^A, we know that I belongs either to I^{A_1} or to I^{A_2}. Without loose of generality, lets suppose that it belongs to I^{A_1}. Then, there exists $i \in I$ such that $\Pi_1(q) \mapsto_0^i r$. We also know that if $i \in Labels(A_2)$ then $\{i\} \in I^{A_2}$ (input selection of size 1) and thus there exists s such that $\Pi_2(q) \mapsto_0^i s$ and then $q \mapsto_0^i (r, s)$.
- Output Selection Property (item (iii)): Similar to the previous one.
- finally, a run containing an infinite number of internal or output-labeled transitions is necessarily time-divergent (item (iv)). Indeed, since any run of A can projected into a run of A_1 and a run of A_2 and one of those runs must exhibit an infinite number of outputs or internal transitions and therefore diverge.

□

Theorem 3.5

Given an indexed set $S = \{(A_i, (I^{A_i}, O^{A_i})\}_{1 \leq i \leq N}$ of N I/O TCs such that they are pair-wise compatible, we define the sets ${I_A}^n = \bigcup_{1 \leq i \leq n} I^{A_i}$, ${I_{A_{>1}}}^n = \bigcup_{1 \leq i \leq n} \{I \in I^{A_i} / \#I > 1\}$, ${O_A}^n = \bigcup_{1 \leq i \leq n} O^{A_i}$.

$C = (A, (I^A, O^A))$ is a component where:

$A = \|_{1 \leq i \leq N} A_i$

$I^A = \{I \in {I_A}^N / \ \forall I' \in {I_A}^N : I \ /\subset I' \wedge \forall O' \in {O_A}^N : I \cap O' = \emptyset\}$ and,

$O^A = \{O \in {O_A}^N / \ \forall I' \in {I_{A_{>1}}}^N : I' \cap O = \emptyset\} \cup \{\{o\} / o \in O \in {O_A}^N \wedge \exists I' \in {I_{A_{>1}}}^N : I' \cap O \neq \emptyset\}$

Proof. By induction. Base case is solved by the last lemma. Case n+1. By inductive hypothesis we know that

$C^n = (A^n, (I^n, O^n))$ is a component where:

$A^n = \|_{1 \leq i \leq n} A_i$

$I^n = \{I \in {I_A}^n / \ \forall I' \in {I_A}^n : I \not\subset I' \wedge \forall O' \in {O_A}^n : I \cap O' = \emptyset\}$ and,

$O^n = \{O \in {O_A}^n / \ \forall I' \in {I_{A_{>1}}}^n : I' \cap O = \emptyset\} \cup \{\{o\} / o \in O \in {O_A}^n \wedge \exists I' \in {I_{A_{>1}}}^n : I' \cap O \neq \emptyset\}$

We know that $C_{n+1} = (A_{n+1}, (I_{n+1}, O_{n+1}))$ is compatible with all $C_i = (A_i, (I^{A_i}, O^{A_i}))$ for $1 \leq i \leq n$. Let us show that is compatible with the

interface for the n components but firstly let pinpoint some facts about the interface (I^n, O^n) of C^n.

(i) An exported label of C_i $(1 \leq i \leq n)$ is also exported by C^n. This comes from the following facts: (a) Input labels remain as input labels in the biggest input selection containing it except in the case that the input selection matches with an output selection (in that case, $I \subseteq O$), and (b) Output labels remain in the interface.

(ii) Input selections of I^n are input selections of some of its constituent components (i.e., If $I \in I^n$ then there exists $k \in \mathbb{N} : 1 \leq k \leq n$ such that $I \in I^{A_k}$)

(iii) If O is an output selection of O^n, if there exist $k \in \mathbb{N} : 1 \leq k \leq n$ such that $O \in O^{A_k}$ and no Input selection of size greater than one intersects it or there exists $O' \in O^{A_k}$ and $O = \{a\} \subseteq O'$, and there exists $m : 1 \leq m \leq n$ such that $I' \in I^{A_m}$ and $I' \subseteq O'$.

Therefore, suppose that A_{n+1} has a common label with $\|_{1 \leq i \leq n} A_i$ then, for instance, that label belongs to a k^{th} automata and therefore belongs to the interface of the components C_k and C_{n+1}. If that label is an output label of the C_{n+1} component, that label is exported by C^n due to the first observation.

The compatibility (item (ii) of Def. 3.3) $I \cap I' \neq \emptyset$ then either $\#I = 1$ or $\#I' = 1$ is trivially true due to the observation that input selections of C^n are input selections of the original components and the pairwise compatibility. Similarly, if an output selection O of $O^{A_{n+1}}$ intersects with some input selection I of I^n then that input selection must be an input selection of some component and therefore that input selection must be included in the output selection (i.e., $I \subseteq O$). If an input selection I of $I^{A_{n+1}}$ intersects with some output selection of O^n namely O, then either it is an input selection of size one and it is trivially included in O, or, by the last observation, we know that there exists k such that $O \in O^{A_k}$ and thus $I \subseteq O$ (that is, due to pairwise compatibility, I must be the only input selection of size greater than 1 intersecting with O and then by the last observation O must belong to O^n).

Therefore, they are compatible components and by Lemma A.1:

such that (I'^{n+1}, O'^{n+1}) is a compatible interface, where:

$I'^{n+1} = \{I \in I^n \cup I^{A_{n+1}} / \ \forall I' \in I^n \cup I^{A_{n+1}} : I \not\subset I' \wedge \forall O' \in O^n \cup O^{A_{n+1}} : I \cap O' = \emptyset\}$ and,

$O'^{n+1} = \{O \in O^n \cup O^{A_{n+1}} / \ \forall I' \in I^n \cup I^{A_{n+1}} \wedge \#I > 1 : I' \cap O = \emptyset\} \cup \{\{o\} / o \in O \in O^n \cup O^{A_{n+1}} \wedge \exists I' \in I^n \cup I^{A_{n+1}} \wedge \#I > 1 : I' \cap O \neq \emptyset\}$

It is not difficult to see that this interface is equivalent to: (I^{n+1}, O^{n+1}) where

$I^{n+1} = \{I \in {I_A}^{n+1} / \ \forall I' \in {I_A}^{n+1} : I \not\subset I' \wedge \forall O' \in {O_A}^{n+1} : I \cap O' = \emptyset\}$ and,

$O^{n+1} = \{O \in {O_A}^{n+1} / \ \forall I' \in {I_{A_{>1}}}^{n+1} : I' \cap O = \emptyset\} \cup \{\{o\} / o \in O \in {O_A}^{n+1} \wedge \exists I' \in {I_{A_{>1}}}^{n+1} : I' \cap O \neq \emptyset\}$

In fact, if we write I^{n+1} in terms of I^n we need to add the input selections

of $I^{A_{n+1}}$ which are not strictly included in an Input Selection of other I^{A_k} and do not match with an output selection. On the other hand, we have to eliminate from I^n the input selections strictly included in an input selection of $I^{A_{n+1}}$ and the ones that match with an Output Selection of $O^{A_{n+1}}$. That is, $I^{n+1} = (I^n - \{I \in {I_A}^n / \exists I' \in I^{A_{n+1}} : I \subset I' \vee \exists O \in O^{A_{n+1}} : I \cap O \neq \emptyset\} \cup \{I' \in I^{A_{n+1}} / \forall I \in {I_A}^n : I' \not\subset I \wedge \forall O \in O^n_A : I' \cap O = \emptyset\}$

Let us show that the definition of I'^{n+1} specifies that manipulation: note that, though I^n may contain less Input Selections than the union of them ($\bigcup_{1 \leq i \leq n} I^{A_i}$), it is easy to see that (a) If an input selection of the union is not present in I^n then, either it is included on another input selection of I^n, or it intersects an output selection of O^n, and (b) $\exists O \in O^n : I \cap O \neq \emptyset$ iff $\exists k \leq n, O \in O^{A_k} : I \cap O \neq \emptyset$ (all output label remains). Therefore, the set $\{I' \in I^{A_{n+1}} / \forall I \in {I_A}^n : I' \not\subset I \wedge \forall O \in O^n_A : I' \cap O = \emptyset\}$ is equivalent to $\{I' \in I^{A_{n+1}} / \forall I \in I^n : I' \not\subset I \wedge \forall O \in O^n : I' \cap O = \emptyset\}$. This proves that in I'^{n+1}, the same input selections of $I^{A_{n+1}}$ filtered by the I^{n+1} are present. Finally, $I^n - \{I \in {I_A}^n / \exists I' \in I^{A_{n+1}} : I \subset I' \vee \exists O \in O^{A_{n+1}} : I \cap O \neq \emptyset\}$

is equivalent to $\{I \in I^n / \forall I' \in I^{A_{n+1}} : I \not\subset I' \wedge \forall O \in O^{A_{n+1}} : I \cap O = \emptyset\}$ and we can conclude that $I'^{n+1} = I^{n+1}$.

On the other hand, to write O^{n+1} in terms of O^n, the output selections of $O^{A_{n+1}}$ that do not match with input selections of size greater than one must be added as well as the singletons for the ones that match. Besides, the output selections of O^n must be checked against the input selections of I_{n+1} to eliminate and convert into singleton output selections the ones that match with input selections of size greater than one. Again, this is specified by the definition of O'^{n+1}. □

A.1 Guaranteeing I/O Admissibility

For the sake of self containment we provide sufficient syntactic constraints and checking-algorithms to guarantee that $(A, (I_A, O_A))$ is indeed a component.

For example, to satisfy the property of input being non-blocking, we can resort to the following syntactic property: $\forall I' \in I_A : \forall l \in Locs(A) : Inv(l) = \bigvee_{\{e : Label(e) \in I' \wedge src(e) = l\}} Guard(e)$. That is, while the invariant is valid at least one I'-labeled transition is enable.

To check that any output selection is simultaneously enabled one of the possible syntactic property is the following: $\forall l \in Locs(A), \forall o, o' \in O \in O^A$:

$$\bigvee_{\{e \in Edges(A) : src(l) = e \wedge Label(e) = o\}} Guard(e) = \bigvee_{\{e' \in Edges(A) : src(l) = e' \wedge Label(e') = o'\}} Guard(e')$$

To check non-zenoness we use an observer automaton with three locations: location 1 is entered non-deterministically from initial location 0 and it is left to go to a trap location 2 whenever input occurs. Then, we ask whether $A \parallel Observer$ satisfies the following TCTL [14] formula : $\forall\Box(@ = 1 \rightarrow$

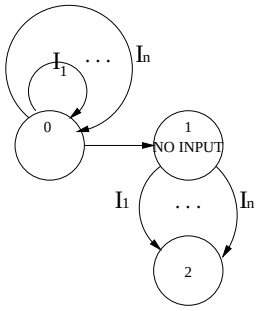

Fig. A.1. Observer for Checking Non-Zeno Regardless Input

$\exists \diamond_{\geq 1} @ = 1$), i.e., whether time can elapse without traversing an input edge (See Fig. A.1).

For non-transientness of outputs, it suffices to require that no pair of outputs or internal events can occur closer than one time unit. This can be done by resorting to an observer TA or, alternatively, adding and checking some syntactic constraints on output edges, for instance, having a minimum delay guard on a clock reset in the potential previous events. Another alternative is checking strong non-zenoness [20] for sequences containing an infinite number of output labels.

B Assume Guarantee

Lemma B.1 (Extending event sequences) *Given a TA A with closed predicates (item (i), Def. 5.2) and a set L of labels, if $r \in R_A(q_0)$ then there exists $r' \in R_A(q_0)$ such that,*

(i) $r_L = r'_L$,

(ii) $\tau_r \leq \tau_{r'}$, *and*

(iii) $\exists k : 0 \leq k < \#r_L : \tau_{r'} - time(r'_L[k]) \in \mathbb{N}$.

Proof. This can be done by following a procedure on r that, step by step, shifts forward not visible transitions (i.e., not L-labeled transitions) to be at integer distance of a visible transition. □

Theorem 5.3

Given the I/O TCs A, B, A', B' satisfying the non-blocking extra conditions such that A and B are I/O compatible, and A' and B' have the same I/O interface that A and B resp. If $(A \parallel B') \leq_{Exp_A} A'$ and $(A' \parallel B) \leq_{Exp_B} B'$ imply that $(A \parallel B) \leq_{Exp_A \cup Exp_B} (A' \parallel B')$.

Proof. This is the sketch of the proof. Let $L = Exp_A \cup Exp_B$. Let $r \in R_{A\|B}(q_0)$ be a finite run such that there is no run in $A' \parallel B'$ of the same time length exhibiting the sequence of timed events r_L. First note there exists a maximum $k < \#r_L$ such that there exits $r' \in R_{A'\|B'}(q'_0)$ with $r_L[0...k] = r'_L$. There are two cases:

(i) There exists $r' \in R_{A'\|B'}(q'_0)$ such that $r_L[0...k] = r'_L$, and $time(r'_L[k+1]) \geq time(r_L[k+1])$.

(ii) For all $r' \in R_{A'\|B'}(q'_0)$, $\tau_{r'} \geq time(r_L[k+1])$ and $r_L[0...k] = r'_L$ implies $time(r'_L[k+1]) < time(r_L[k+1])$ (something urgent must happen before).

Before treating the cases, let r be a run in $A \parallel B$, and r' a run in $A' \parallel B'$ such that $r_L[0...k] = r'_L$. It is easy to see that we can project the run r into a run of A and a run of B. On the other hand, run r' can be projected into a run of A' and a run of B'. Due to the hypothesis on exported labels and labeling, we can safely recombine those runs to get a run $r_{AB'}$ of $A \parallel B'$ and a run $r_{A'B}$ of $A' \parallel B$ with the same time length of r'. Let q_A be the last state of $r_{AB'}$ projected into A, q_B the last state of $r_{A'B}$ projected into B, $q_{A'}$ the last state of $r_{A'B}$ projected into A', and $q_{B'}$ the last state of $r_{AB'}$ projected into B'.

Case i:

Suppose that $lab(r_L[k+1]) = o \in O^A_m$ (the other case is analogous). Then, at the last state of $r_{AB'}$ it is possible to execute some $o' \in O^A_m$ (at q_A A can perform any output in O^A_m and B' is receptive to at least one of them). From the fact that $(A \parallel B') \leq_{Exp_A} A'$, we can show the existence of a run $r_{A'}$ of A' such that $r_{A'Exp_{A'}}$ is equal to $r'_L[0...k+1]$ projected into $Exp_{A'}$ $(= Exp_A)$. Due to the fact that $O^A_m = O^{A'}_s$ is simultaneously enabled, there is a run $r'_{A'}$ of A' exhibiting $r_L[0...k+1]$ projected into $Exp_{A'}$. On the other hand, r shows that o is enabled at q_B. We can replace the original projection of $r_{A'B}$ on A' by the run $r'_{A'}$. Then, from the fact that $(A' \parallel B) \leq_{Exp_B} B'$ we can conclude that there is a run $r_{B'}$ in B' exhibiting $r_L[0...k+1]$ projected into $Exp_{B'}$. Combining the new runs $r'_{A'}$ and $r_{B'}$ for A' and B' resp., we conclude that there is a run r^* in $R_{A'\|B'}(q'_0)$ such that its exhibited sequence over L $r^*_L[0...k+1]$ is equal to $r_L[0...k+1]$, a contradiction.

Case ii:

Let $r' \in R_{A'\|B'}(q'_0)$ such that $\tau_{r'} \geq time(r_L[k+1])$ and $r'_L[0...k] = r_L[0...k]$, let r'' be the prefix run of r' such that $r''_L = r_L[0...k]$. By the previous lemma and the assumptions of this case, there exists another run ρ in $A' \parallel B'$ such that $\rho_L = r''_L = r_L[0...k]$, $\tau_{r''} \leq \tau_\rho < time(r_L[k+1])$, and $\exists s : 0 \leq s \leq k : \tau_\rho - time(r_L[s]) \in \mathbb{N}$. This means that, when runs show that in $A' \parallel B'$ something urgent must happen before time $time(r_L[k+1])$, there must exist a maximum value for its ocurrence. As shown, this follows from the fact that, for any r'' such that $r''_L = r_L[0...k]$ there exists a longer ρ such that $\rho_L = r''_L = r_L[0...k]$ and there are a finite number of those ρ (ρ ends at integer time distance of some event and before $time(r_L[k+1])$). We will show that any such ρ ends at a state where time can ellapse arriving to an absurd.

We know that $NU(q_A)$ and $NU(q_B)$. From the fact that $q \mapsto^i p$ implies $NU(q) \Rightarrow NU(p)$ (item (ii), Def. 5.2), A can wait, and any finite number

of inputs of A (outputs of B') can not change this fact (an infinite number of outputs of B' would also imply time-divergence). Then, there exists $w \in R_{A\|B'}((q_A, q_{B'}))$ such that $\tau_w > 0$ and $Lab(w) \cap Out_A = \emptyset$. Since $\rho' = r_{AB'} \circ w$ ($r_{AB'}$ prolonged with w) is a run of $A \parallel B'$ there exists a run $r_{A'}$ of A' such that $\tau_{r_{A'}} = \tau_{\rho'}$ and $r_{A' Exp_{A'}} = \rho'_{Exp_{A'}}$. Thus, $r_{A'}$ can be split as $r'_{A'} \circ r''_{A'}$ such that $\tau_{r''_{A'}} = \tau_w$ and $r''_{A' Exp_{A'}} = w_{Exp_{A'}}$. Then from the last state of $r'_{A'}$ (denoted $q'_{A'}$) there exists a non-transient run ($r''_{A'}$) such that it exhibits no Output label. Then, there exists a state s in the run $r''_{A'}$ such that $NU(s)$. Using Def. 5.1 and item (ii) of Def. 5.2 ($q \mapsto^i p$ implies $NU(p) \Rightarrow NU(q)$) we can conclude $NU(q'_{A'})$. Analogously, $NU(q'_{B'})$. Therefore, the combination of those runs shows the possibility of $A' \parallel B'$ to exhibit σ' plus a positive time increment (the minimum possible increment between A' and B'). Thus, we arrive to an absurd.

□

Non-determinism in Probabilistic Timed Systems with General Distributions

Mario Bravetti [1] and Alessandro Aldini [2]

Dipartimento di Scienze dell'Informazione,
University of Bologna, Mura Anteo Zamboni 7, 40127 Bologna, Italy

Abstract

In this paper we address the problem of adequately handling non-deterministic choices in Generalised Semi-Markov Processes (GSMPs), i.e. probabilistic timed systems where durations of delays are expressed by means of random variables with a general probability distribution. In particular we want the probabilistic duration of a delay not to be decided all at once when the delay starts, but step by step in each system state (in the theory of GSMPs this corresponds to recording spent lifetimes instead of residual lifetimes of delays). In this way an adversary cannot take decisions a priori, based on the knowledge he may get about the future behavior of the system. In order to accomplish this, we consider Interactive Generalised Semi-Markov Processes (IGSMPs). We start by formalizing the class of well-named IGSMP models and the class of Interactive Stochastic Timed Transition Systems (ISTTSs) which are both closed under CSP parallel composition and hiding. Then, we introduce a semantics for IGSMPs which maps well-named IGSMP models onto ISTTSs by recording spent lifetimes of delays. Finally, we show that two weakly bisimilar IGSMPs give rise to two weakly bisimilar semantic models and that our semantic mapping is compositional with respect to both CSP parallel composition and hiding.

1 Introduction

The importance of modeling the behavior of concurrent systems with respect to time has been widely recognized [18,17,3,10,5]. Moreover due to the fact that either systems are frequently described at a high level of abstraction, or the temporal behavior of some system component is inherently probabilistic (e.g. transmission time of a message through a network), it is important

[1] Email: bravetti@cs.unibo.it
[2] Email: aldini@cs.unibo.it

This is a preliminary version. The final version is considered for publication in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

that the modeling paradigm employed allows for the specification of probabilistic time. To this aim several modeling techniques have been defined to specify systems including activities with probabilistic exponentially distributed duration [18,17,3,10,5]. These approaches have the advantage to be easily tractable from both the modeling and analysis viewpoint since the underlying performance models are just simple Markov Chains. The price to pay is a strong limitation in the expressiveness of these modeling paradigms, since even fixed (non probabilistic) durations cannot be represented. Some previous efforts have been made in order to develop models for general distributions [16,2,9,24,6,14]. In particular in [8] we have recognized that a formalism expressing systems with generally distributed delays should originate from probabilistic models which are well-founded from the viewpoint of probability theory. More precisely, we have considered Generalised Semi-Markov Processes (GSMPs), i.e. probabilistic timed systems where durations of delays are expressed by means of random variables with a general probability distribution. A GSMP describes the temporal behavior of a system by using *elements*, which act similarly as clocks of a Timed Automata (see e.g. [23]). In particular the temporal delays in the evolution of a system are represented by clocks (elements) whose duration is determined by an associated generally distributed random variable. In this way the temporal behavior of the system is "guided" by the events of start and termination of clocks (elements). Following the idea of [17], where the same problem is attacked for exponential distributions, we have introduced in [6] the possibility of specifying systems as the parallel composition of subsystems described by GSMPs, by developing the calculus of Interactive Generalized Semi-Markov Processes (IGSMPs).

An IGSMP represents the behavior of a component by employing both *standard action transitions*, representing the interactive behavior of the component, and *clock start transitions* and *clock termination transitions*, representing the timed probabilistic behavior of the component. Action transitions are just standard CCS/CSP transitions: when several action transitions are enabled in an IGSMP state, the choice among them is just performed nondeterministically and when IGSMPs are composed in parallel they synchronize following the CSP [19] approach, where the actions belonging to a given set S are required to synchronize. Clock start transitions are labeled with a clock name and a weight and represent the event of start of a temporal delay whose probabilistic duration is given by the distribution associated with the clock. When several clock start transitions are enabled in an IGSMP state, the choice among them is performed probabilistically according to the weights of the transitions. Clock termination transitions are labeled with a clock name and represent the event of termination of the corresponding temporal delay. A system stays in a state enabling several termination transitions until one of the temporal delays currently in execution terminates and the corresponding transition is performed.

Introducing non-determinism in probabilistic systems with general distri-

butions causes new problems to arise with respect to the classical theory of GSMPs. Such problems arise when we consider the interplay of non-deterministic choices and the probabilistic behavior of clocks when IGSMPs are actually executed. Following the classical approach of discrete event simulation (see e.g. [12]), in the instant a clock starts, the clock is set to a temporal value sampled from its duration distribution. As time passes clock counts down and it terminates when it reaches value zero. From a technical viewpoint this means that, while the GSMP proceeds from state to state, we keep track of the quantity of time that clocks must still spend in execution (the *residual lifetimes* of the clocks). This approach to the execution of an IGSMP, which has been previously applied in [14] to systems including non-determinism and generally distributed time, has the drawback that an adversary can base its decisions (concerning non-deterministic choices) on the knowledge obtained a priori about the future behavior of the system, e.g. the information about the quantity of time that a delay will spend in execution.

In this paper we consider a new alternative approach to the execution of systems including non-determinism and generally distributed time which adequately handles non-deterministic choices. The idea is that we want the probabilistic duration of a generally distributed delay not to be decided all at once when the delay starts, but step by step in each system state. More precisely, this is realized by keeping track of the quantity of time spent by clocks in execution (*spent lifetimes* of clocks), and by evaluating, when a new IGSMP state is entered, the distribution of the residual duration of the clock from (i) the duration distribution associated with the clock, and (ii) the time it has already spent in execution. Such an approach, which is based on recording spent lifetimes instead of residual lifetimes, is adherent to the classical behavior of Timed Automata [23] where clocks are increased (and not decreased) while time passes. Besides it indeed solves the problem of executing a system with non-deterministic choices because, since the residual duration of clocks is sampled in every state traversed by the IGSMP, the adversary cannot gain a priori knowledge on the system behavior. Finally, considering spent lifetimes instead of residual lifetimes is correct also from a probabilistic viewpoint, because in probability theory the two approaches are both valid alternative ways to interpret a GSMP [13]. It is worth noting that the choice of adopting this alternative approach for representing the execution of an IGSMP is conceptual and not at all related with the technical differences between the formalism considered in [14] and IGSMPs. We could apply the technique used in [14] to IGSMPs as well.

Similarly as in [14], based on our approach to the execution of an IGSMP, we produce a semantics for IGSMPs which maps an IGSMP onto a transition system where: (i) the passage of time is explicitely represented by transitions labeled with numeric time delays and (ii) duration probability distributions are turned into infinitely branching probabilistic choices which lead to states performing numeric time delays with different durations. Differently from [14]

we express semantic models of IGSMPs by means of "interactive" probabilistic timed transition systems which can be themselves composed and we develop a semantic mapping which is compositional with respect to parallel composition and hiding.

More precisely, we start (in Sect. 2) by formalising the model of IGSMPs and the model of well-named IGSMPs (a canonical form for IGSMPs which makes it simple to establish weak bisimulation over IGSMPs). With respect to [6], where well-named IGSMPs are defined as the class of semantic models obtained from the terms of a process algebra, in this paper we characterize well-named IGSMPs directly as a class of transition systems. Moreover we show that the class of well-named IGSMPs is closed with respect to CSP parallel composition and hiding and we introduce a notion of weak bisimulation over well-named IGSMPs. Then, (in Sect. 3) we introduce the model of Interactive Stochastic Timed Transition Systems (ISTTSs) which include: *standard action transitions*, representing the interactive behavior of a system component, *numeric time transitions* representing a fixed temporal delay, and *probabilistic transitions* (expressed by means of probability spaces) representing (infinitely branching) probabilistic choices. Moreover we show that the class of ISTTSs is closed with respect to CSP parallel composition and hiding and we introduce a notion of weak bisimulation over ISTTSs. Moreover, (in Sect. 4) we present the semantics for IGSMPs which maps IGSMPs onto ISTTSs and we show that weakly bisimilar IGSMPs give rise to weakly bisimilar semantic models and that the semantic mapping is compositional with respect to both CSP parallel composition and hiding. Finally, (in Sect. 5) we report some concluding remarks. The proofs of the results in this paper can be found in [4].

2 Interactive Generalized Semi-Markov Process

In this section we will present the model of Interactive Generalized Semi-Markov Processes (IGSMPs) and of well-named Interactive Generalized Semi-Markov Processes: a canonical form for IGSMPs which introduces some constraints on clock names and makes it simple to establish equivalence over IGSMPs. We first briefly introduce some basic notions about GSMPs, and then we discuss in depth the model of IGSMPs and well-named IGSMPs.

2.1 The GSMP Model

The class of generalized semi-Markov processes (GSMPs) has been introduced by Matthes (1962) in [21] and represents the temporal behavior of a system through elements (or clocks) each with an associated duration probability distribution (element lifetime). What characterizes GSMPs (e.g. with respect to Semi-Markov Processes) is the possibility of having multiple *active elements* in a state, so that when an active element terminates (*it dies*) a state change

occurs and the other elements continue their life in the following state, thus carrying over their residual duration.

Definition 2.1 A *generalized semi-Markov process* (GSMP) is a stochastic process defined on a set of states $\{s \mid s \in \mathcal{S}\}$ as follows. In each state s there is a set of active elements e taken from a set El. The set El is partitioned into two sets El' ed El^* with $El = El' \cup El^*$. If $e \in El'$ the element e has an exponentially distributed lifetime, if instead $e \in El^*$ it has an arbitrarily distributed lifetime. Whenever in a state s an active element e dies, the process moves to the state $s' \in \mathcal{S}$ with a given probability $P(s, e, s')$.

In GSMPs the following restrictions are considered [21]:

- When the process moves from a state to another, no more than one element of El^* can be born or die contemporaneously.
- The active elements of El^* that do not die in a state keep their residual duration.

A GSMP can be analyzed through simulative techniques and/or mathematical techniques (based on phase-type approximation or insensitivity [21]) in order to obtain performance measures of a system.

2.2 The IGSMP Model

The model of Interactive Generalized Semi-Markov Processes extends that of Generalized Semi-Markov Processes by expressing in addition to GSMP clocks (or elements) execution, also the execution of standard actions which can synchronize and have a zero duration. Such an approach, which is inspired from [17], is also quite usual in real-time process algebras [23] where transitions representing temporal delays are distinguished from standard action transitions which are performed in zero time. As far as probabilistic delays are concerned, they are modeled as in GSMPs by means of clocks C whose duration is expressed through general probability distributions. In the following we will distinguish different clocks used in an IGSMP through "names", where C_n denotes the clock with name n. In an IGSMP the execution of a clock C_n is represented by means of two events: the event of clock start C_n^+ followed by the relative event of clock termination C_n^-. Therefore in an IGSMP we have three types of transitions: standard action transitions representing action execution, clock start transitions representing events C_n^+ and clock termination transitions representing events C_n^-. When a transition C_n^+ is performed by the IGSMP the clock C_n starts and continues its execution in every state traversed by the IGSMP. Whenever the clock C_n terminates, then the IGSMP executes the corresponding termination transition C_n^-. In particular, since, as in GSMPs, each started clock C_n which has not terminated yet must continue its execution in each state traversed by the IGSMP, all such states must have an outgoing transition C_n^-. Obviously clocks which can be simultaneously under execution in an IGSMP state must have different names (even if they

have the same duration distribution), so that the event of termination of a clock C_n^- is always uniquely related to the corresponding event of start of the same clock C_n^+. Similarly as GSMPs, IGSMPs can also express probabilistic choices. This is obtained by associating with each start transition C_n^+ a weight $w \in \mathbb{R}^+$. In this way when a state of the IGSMP enables several clock start transitions $\langle C_n^+, w \rangle$, the choice of the clock to be started is performed probabilistically according to the weights w of the transitions. For instance, a state enabling two transitions labeled with $\langle C_n^+, w \rangle$ and $\langle C_{n'}^+, w' \rangle$ respectively starts clock C_n with probability $w/(w + w')$ and starts clock $C_{n'}$ with probability $w'/(w + w')$. On the other hand, similarly as in [17], IGSMPs also have, in addition to GSMPs, the capability of expressing non-deterministic choices. This because, as in standard labeled transition systems deriving from CCS/CSP terms, in the states of an IGSMP action transitions are just non-deterministically chosen. In particular alternative transitions labeled with invisible τ actions represent internal non-deterministic choices which are performed in zero time and can never be "resolved" through synchronization with other system components. On the contrary, visible actions a in an IGSMP are seen as incomplete actions which wait for a synchronization with other system components (they represent potential interaction with the environment). Therefore the choice of such actions in any IGSMP state is governed by an external form of non-determinism, as their execution completely depends on the environment. Note that since we adopt a CSP synchronization policy for IGSMPs which produces visible actions from the synchronization of visible actions (thus allowing for multiway synchronization) the only way to turn an incomplete action in a complete one is by means of a hiding operator, which turns visible actions into τ actions. Similarly as in [17] an IGSMP represents a *complete system* only when it does not include any transition labeled by a visible action. This approach differs from that of the stochastic automaton model of [14], where two different kinds of semantics have to be defined in order to describe the *actual behavior* of closed systems and the *potential behavior* of open systems. In our approach both the potential and the actual behavior of the system are represented within the same model and complete systems are obtained by hiding all the actions of the model. Note that IGSMPs representing complete systems may still include non-determinism due to multiple internal τ transitions enabled in the same state (internal non-determinism). Therefore, adversaries (or schedulers) play an important role in the performance analysis of IGSMPs in that they allow internal non-determinism to be removed from an IGSMP thus turning it into a GSMP.

More precisely, in an IGSMP we have four different kinds of state:

- *silent states*, enabling invisible action transitions τ and (possibly) visible action transitions a only. In such states the IGSMP just performs a non-deterministic choice among the τ transitions in zero time and may potentially interact with the environment through one of the visible actions (see e.g. Fig. 1.a).

- *probabilistic states*, enabling $\langle C_n^+, w\rangle$ transitions and (possibly) visible action transitions a only. In such states the IGSMP just performs a probabilistic choice among the clock start transitions in zero time and may potentially interact with the environment through one of the visible actions (see e.g. Fig. 1.b).
- *timed states*, enabling C_n^- transitions and (possibly) visible action transitions a only. In such states the IGSMP executes all the clocks labeling the outgoing termination transitions according to their residual duration distribution. The clock that terminates first determines the transition to be performed. Note that since, as in GSMPs, we assume that clocks cannot terminate at the same instant, we always have a unique clock terminating before the other ones (see e.g. Fig. 1.c). While the IGSMP sojourns in the state, it may (at any time) potentially interact with the environment through one of the outgoing visible action transitions.
- *waiting states*, enabling standard visible actions only or no transition at all. In such states the IGSMP sojourns indefinitely. It may, at any time, potentially interact with the environment through one of the outgoing visible action transitions (see e.g. Fig. 1.d).

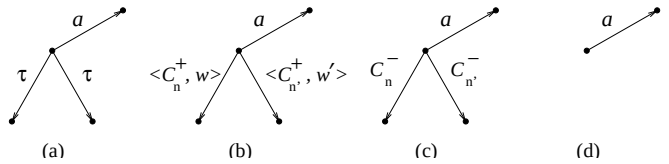

Fig. 1. Some examples of possible states of an IGSMP

In the following we present the formal definition of Interactive Generalized Semi-Markovian Transition System (IGSMTS), then we will define Interactive Generalized Semi-Markov Process as IGSMTSs possessing an initial state. Formally, we denote with PDF the set of probability distribution functions over $\mathbb{R}$ ranged over by $f, g, \ldots$ and with PDF^+ the set of probability distribution functions over $\mathbb{R}^+ \cup \{0\}$ such that $f(x) = 0$ for $x < 0$ (representing duration distributions). Weights, belonging to $\mathbb{R}^+$, are ranged over by $w, w', \ldots$. Moreover, we denote the set of standard action types used in a IGSMTS by Act, ranged over by $\alpha, \alpha', \ldots$. As usual Act includes the special type τ denoting internal actions. The set $Act - \{\tau\}$ is ranged over by $a, b, \ldots$. The set of clocks of an IGSMTS is denoted by $\mathcal{C} = \{C_n \mid n \in \mathcal{C}Names\}$, where $\mathcal{C}Names$ is a set of clock names. Given a set $\mathcal{C}$, we denote with $\mathcal{C}^+ = \{\langle C_n^+, w\rangle \,|\, C_n \in \mathcal{C}, w \in \mathbb{R}^+\}$ the set of events denoting the starting of a clock and $\mathcal{C}^- = \{C_n^- \,|\, C_n \in \mathcal{C}\}$ the set of events denoting the termination of a clock. Let $\mathcal{C}^+ \cup \mathcal{C}^-$ be ranged over by $\theta, \theta', \ldots$. The set of states of an IGSMTS is denoted by Σ, ranged over by $s, s', \ldots$. We assume the following abbreviations that will make the definition of IGSMTSs easier. Let us suppose that $T \subseteq (\Sigma \times Labels \times \Sigma)$ is a transition relation, where $Labels$ is a set of transition labels, ranged over by l. We use $s \xrightarrow{\;l\;} s'$ to stand for

$(s,l,s') \in T$, $s \xrightarrow{l}$ to stand for $\exists s' : s \xrightarrow{l} s'$, and $s \not\xrightarrow{l}$ to stand for $\nexists s' : s \xrightarrow{l} s'$.

Definition 2.2 An Interactive Generalized Semi-Markovian Transition System (IGSMTS) is a tuple $\mathcal{G} = (\Sigma, \mathcal{C}, D, Act, T_+, T_-, T_a)$ with

- Σ a set of states,
- $\mathcal{C}$ a set of clocks,
- $D : \mathcal{C} \longrightarrow PDF^+$ a function that assigns a duration probability distribution function to each clock,
- Act a set of standard actions,
- $T_+ \subseteq (\Sigma \times \mathcal{C}^+ \times \Sigma)$, $T_- \subseteq (\Sigma \times \mathcal{C}^- \times \Sigma)$, and $T_a \subseteq (\Sigma \times Act \times \Sigma)$ three transition relations representing clock start and termination events and action execution, respectively, such that: [3]

1 $\forall s \in \Sigma.$
$$s \xrightarrow{\tau} \implies \nexists \theta . s \xrightarrow{\theta}$$
2 $\forall s \in \Sigma.$
$$\exists C_n, w. s \xrightarrow{\langle C_n^+, w\rangle} \implies \nexists C_{n'} . s \xrightarrow{C_{n'}^-}$$
3 $\exists \mathcal{S} : \Sigma \longrightarrow \mathcal{P}(\mathcal{C})$ the *active clock function*, such that $\forall s \in \Sigma.$

a) - $s \xrightarrow{\alpha} s' \implies \mathcal{S}(s') = \mathcal{S}(s)$

- $s \xrightarrow{\langle C_n^+, w\rangle} s' \implies \mathcal{S}(s') = \mathcal{S}(s) \cup \{C_n\}$
- $s \xrightarrow{C_n^-} s' \implies C_n \in \mathcal{S}(s) \wedge \mathcal{S}(s') = \mathcal{S}(s) - \{C_n\}$

b) $\exists C_n, w. s \xrightarrow{\langle C_n^+, w\rangle} \implies C_n \notin \mathcal{S}(s)$

c) $C_n \in \mathcal{S}(s) \wedge s \not\xrightarrow{\tau} \wedge \nexists C_{n'}, w. s \xrightarrow{\langle C_{n'}^+, w\rangle} \implies s \xrightarrow{C_n^-}$

4 $\forall s \in \Sigma.$
$$s \xrightarrow{\langle C_n^+, w\rangle} s' \implies act(s') \subseteq act(s)$$
where the *enabled action function* $act : \Sigma \longrightarrow \mathcal{P}(Act)$ is defined by $act(s) = \{\alpha \mid s \xrightarrow{\alpha}\}$.

Definition 2.3 An Interactive Generalized Semi-Markov Process (IGSMP) is a tuple $\mathcal{G} = (\Sigma, \mathcal{C}, D, Act, T_+, T_-, T_a, s_0)$, where $s_0 \in \Sigma$ is the initial state of the IGSMP and $(\Sigma, \mathcal{C}, D, Act, T_+, T_-, T_a)$ is an IGSMTS such that function $\mathcal{S}$ in item 3 of Definition 2.2 also satisfies $\mathcal{S}(s_0) = \emptyset$.

The constraints over transition relations T_+, T_- and T_a guarantee that each state of the IGSMP belongs to one of the four kind of states above. In particular, the first requirement says that if a state can perform internal τ ac-

[3] For the sake of readability here and in the rest of the paper we assume the following operator precedence when writing constraints for transition relations: existential quantifier > "and" operator > implication.

tions then it cannot perform events of clock starts or clock terminations. Such a property derives from the assumption of *maximal progress*: the possibility of performing internal actions prevents the execution of delays. The second requirement says that if a state can perform clock start events then it cannot perform clock termination events. Such a property derives from the assumption of *urgency of delays*: clock start events cannot be delayed but must be performed immediately, hence they prevent the execution of clock termination transitions. The third requirement checks that clock starting and termination transitions are consistent with the set of clocks that should be in execution in each state of the IGSMP. This is done by defining a function $\mathcal{S}$ which maps each state onto the expected set of clocks in execution, i.e. the set of clocks which have started but not terminated yet. In particular, in the initial state s_0 such a set is empty. The constraint a) defines the construction rule of the active clock set for each state reachable from s_0. In the case of a transition from a state s to a state s' labeled with an standard action, the active clocks of s' stem from the active clocks of s, as no clock can be terminated given that a standard action has been performed. If a transition from s to s' is labeled with a clock start event $\langle C_n^+, w\rangle$, then s' inherits the active clock set of s and adds to this set the started clock C_n. Finally, in the case of a transition from s to s' labeled with a clock termination event C_n^-, s' inherits the active clock set of s without such a terminated clock C_n. Constraints b) and c) concern the legality of the outgoing transitions of a state. In particular, the former says that a the name of a clock labeling a starting transition must be fresh (i.e. no clock with such a name must be currently in execution). The latter says that each state without τ and $\langle C_n^+, w\rangle$ outgoing transitions must have a $C_{n'}^-$ outgoing transition for each active clock $C_{n'}$. This definition preserves both the maximal progress and the urgency of delays assumptions and, in each state where it is possible, guarantees the possibility of terminating each delay that is still active. The fourth requirement of Definition 2.2 implements the following constraint over the structure of IGSMPs which makes their theory simpler. The unique role of clock start transitions in an IGSMP must be to lead to a timed state where the started clocks are actually executed, hence the execution of such transitions cannot cause new behaviors to be performable by the IGSMP. Such a constraint is satisfied by the semantic models of terms of the calculus of IGSMPs introduced in [6]. Formally, we require that the set of action transitions enabled after a clock start transition is a subset of (or equal to) the set of action transitions enabled before such a transition. This guarantees that no new behaviors can be introduced by clock start transitions because: (i) no new behavior beginning with a τ transition can be executable after a clock start transition (states enabling clock start transitions cannot enable τ transitions), and (ii) every potential behavior beginning with a transition a executable after a clock start transition can never be actually executed by hiding a, because before the clock start transition there is a potential behavior beginning with the same action a, which, when hidden, preempts the

clock start (see the following Sect. 2.5 about the hiding of IGSMPs).

2.3 The Well-Named IGSMP Model

The model of well-named IGSMPs represents a canonical form for IGSMPs which introduces some constraints on clock names and makes it simple to develop an equivalence notion over IGSMPs which matches clocks with the same duration distribution. Well-named IGSMPs are exactly the class of semantic models obtained from the terms of the process algebra introduced in [6]. Here we will characterize well-named IGSMPs directly as a class of transition systems.

The constraint on the use of clock names in an IGSMP that we consider concerns the names k which are used for clocks when they start. As we already explained the name used for a starting clock must be fresh, i.e. no clock with such a name must be currently in execution. The requirement that we now add is that the new clock name which is used must depend from the duration distribution f associated with the starting clock and from the names of the clocks (with the same distribution f) already in execution, according to a fixed rule. In particular, we take the set of clock names to be defined by $\mathcal{C}Names = (PDF^+ \times \mathbb{N}^+)$, where "$f, i$" is a name for a clock with associated distribution f. The name "f, i" which is used for a starting clock must be such that i is the least $i \in \mathbb{N}^+$ which is not used in the name of any clock with the same distribution f already in execution. Note that, similarly as for standard ST models [1,7,11,15], using just duration distributions as clock names is not sufficient because indexes $i \in \mathbb{N}^+$ are needed in order to uniquely relate the clock termination event to the corresponding clock start event, even in the situation where several clocks with the same duration distribution are simultaneously executed (this is also observed in [8]).

Since in a well-named IGSMP names for clocks cannot be chosen arbitrarily and the clock names which are considered make it clear by themselves which is the duration distribution associated with a clock, with respect to IGSMTSs (Definition 2.2), in the definition of well-named IGSMTSs we omit set $\mathcal{C}$ and function D.

Definition 2.4 A well-named Interactive Generalized Semi-Markovian Transition System is a tuple $\mathcal{G} = (\Sigma, Act, T_+, T_-, T_a)$ where Σ and Act are defined as in Definition 2.2, while the definition of the transition relations T_+, T_- and T_a is obtained from that given in Definition 2.2 by substituting the constraint b) of item 3 with:

$b)$ $\quad \exists C_{f,i}, w.\, s \xrightarrow{\langle C^+_{f,i}, w\rangle} \implies i = \min\{j \mid j \in \mathbb{N}^+, C_{f,j} \notin \mathcal{S}(s)\}$

Note that the new version of constraint b) guarantees that the name used for a starting clock is always fresh as required by the old version of constraint b) (see Definition 2.2).

Definition 2.5 A well-named Interactive Generalized Semi-Markov Process is a tuple $\mathcal{G} = (\Sigma, Act, T_+, T_-, T_a, s_0)$, where $s_0 \in \Sigma$ is the initial state of the well-named IGSMP and $(\Sigma, Act, T_+, T_-, T_a)$ is a well-named IGSMTS such that function $\mathcal{S}$ in item 3 of Definition 2.2 also satisfies $\mathcal{S}(s_0) = \emptyset$.

As an important remark, we would like to point out that, since the rule expressed by constraint b) of Definition 2.5 reuses the indexes i of terminated clocks, each IGSMP with a finite set of states can be transformed into a well-named IGSMP with a finite set of states, by renaming clocks.

2.4 Parallel of Well-Named IGSMPs

Now, we address the problem of defining parallel composition à la CSP [19] of well-named IGSMPs, where the standard actions of a given set S are required to synchronize and the synchronization of two actions of type a is again an action of type a.

Intuitively, it should be clear that when composing in parallel two IGSMPs, a suitable renaming of the clocks is necessary in order to obtain a IGSMP, i.e. preserve the requirements on transition relations of Definition 2.3. Indeed composing in parallel two IGSMPs could lead to some conflict concerning the identification of the clocks of the composed model through names. More precisely, we have to cope with a name conflict whenever two clocks with the same name "f, i" are simultaneously in execution in both IGSMPs. In such a case the same name identifies two different clocks by compromising the relationship between the start and termination events of the two clocks. When considering well-named IGSMPs instead of just IGSMPs we have in addition the problem of preserving the rule for the name of starting clocks expressed by constraint b) of Definition 2.5.

The solution that we adopt consists in using l and r (left and right) as references to the two well-named IGSMPs $\mathcal{G}', \mathcal{G}''$ which are composed in parallel with $\mathcal{G}' \parallel_S \mathcal{G}''$ and relating each clock name locally used in $\mathcal{G}'$ (or $\mathcal{G}''$) to the well-named IGSMP $\mathcal{G}'$ (or $\mathcal{G}''$) through the reference l (or r). In this way C_{f,l_i} (C_{f,r_i}) denotes the clock $C_{f,i}$ executed by $\mathcal{G}'$ ($\mathcal{G}''$). In order to obtain a well-named IGSMP, when building the composed model, such "extended" names are renamed so that the rule for the name of starting clocks expressed by constraint b) of Definition 2.5 is satisfied. For instance, let us suppose that both $\mathcal{G}'$ and $\mathcal{G}''$ execute a clock with the same duration distribution f. For both well-named IGSMPs in isolation we represent such an event by activating the clock $C_{f,1}$. Somehow in the composed model we have to distinguish such clocks through names because they can be simultaneously in execution. Let us suppose that in $\mathcal{G}' \parallel_S \mathcal{G}''$ the first delay with distribution f that starts is the one executed by $\mathcal{G}'$. According to the well-naming rule in the composed model such a clock must get name "$f, 1$". Hence we map $C_{f,1}$ to the "extended" name of the clock $C_{f,1}$ executed by $\mathcal{G}'$, thus creating the following mapping:

$$C_{f,1} \longrightarrow C_{f,l_1}$$

denoting that the first clock with distribution f of the composed model $C_{f,1}$ corresponds to the first clock with distribution f of the lefthand well-named IGSMP. Then, if the second clock to be executed is the clock $C_{f,1}$ belonging to the righthand well-named IGSMP, in the composed model we create the fresh name "$f, 2$" (according to the well-naming rule) and have in addition the following mapping:

$$C_{f,2} \longrightarrow C_{f,r_1}$$

In Table 1 we present an example of execution of a composed model $\mathcal{G}' \parallel_S \mathcal{G}''$ by showing how the mapping function (between the clock names of the composed model $\mathcal{G}' \parallel_S \mathcal{G}''$ and the corresponding clock names locally used in $\mathcal{G}'$ and $\mathcal{G}''$) for clocks with distribution f evolves.

Well-named IGSMPs	*Composed Model*	*Mapping Function*
$\mathcal{G}'$ *starts* $C_{f,1}$	$\mathcal{G}' \parallel_S \mathcal{G}''$ *starts* $C_{f,1}$	$C_{f,1} \longrightarrow C_{f,l_1}$
$\mathcal{G}''$ *starts* $C_{f,1}$	$\mathcal{G}' \parallel_S \mathcal{G}''$ *starts* $C_{f,2}$	$C_{f,1} \longrightarrow C_{f,l_1}$ $C_{f,2} \longrightarrow C_{f,r_1}$
$\mathcal{G}''$ *starts* $C_{f,2}$	$\mathcal{G}' \parallel_S \mathcal{G}''$ *starts* $C_{f,3}$	$C_{f,1} \longrightarrow C_{f,l_1}$ $C_{f,2} \longrightarrow C_{f,r_1}$ $C_{f,3} \longrightarrow C_{f,r_2}$
$\mathcal{G}''$ *ends* $C_{f,1}$	$\mathcal{G}' \parallel_S \mathcal{G}''$ *ends* $C_{f,2}$	$C_{f,1} \longrightarrow C_{f,l_1}$ $C_{f,3} \longrightarrow C_{f,r_2}$
$\mathcal{G}'$ *starts* $C_{f,2}$	$\mathcal{G}' \parallel_S \mathcal{G}''$ *starts* $C_{f,2}$	$C_{f,1} \longrightarrow C_{f,l_1}$ $C_{f,2} \longrightarrow C_{f,l_2}$ $C_{f,3} \longrightarrow C_{f,r_2}$

Table 1
Renaming of the clocks in $\mathcal{G}' \parallel_S \mathcal{G}''$

By following such a procedure, we build the composed model by dynamically storing all current mappings between the clock names of the composed model and the local clock names of the two well-named IGSMPs by employing a table (mapping function) for each distribution f. In general, when a clock C_{f_i} with distribution f is started by one of the two composed well-named IGSMPs, we do the following: (i) we choose the first index j for the distribution f which is unused in the composed model (by checking the table related to the duration probability distribution f), and we use the name "f, j" for the

clock in the composed model; (ii) we add to the table related to distribution f the mapping $C_{f,j} \longrightarrow C_{f,l_i}$ if the clock is executed by the lefthand well-named IGSMP or $C_{f,j} \longrightarrow C_{f,r_i}$ if the clock is executed by the righthand well-named IGSMP. When a clock C_{f_i} with distribution f is terminated by one of the two composed well-named IGSMPs, we do the following: (i) we establish the name "f, j" associated with the terminating clock in the composed model by checking the table related to distribution f (it must include $C_{f,j} \longrightarrow C_{f,l_i}$ if the clock is executed by the lefthand well-named IGSMP or $C_{f,j} \longrightarrow C_{f,r_i}$ if the clock is executed by the righthand well-named IGSMP); (ii) we remove from the table related to the duration probability distribution f the mapping for the name "f, j" of the composed model.

Now we formally define the parallel composition $\mathcal{G}_1 \parallel_S \mathcal{G}_2$ of two well-named IGSMPs $\mathcal{G}_1$ and $\mathcal{G}_2$, where the synchronization set S is a subset of $Act - \{\tau\}$.

We denote with $Loc = \{l, r\}$, ranged over by loc the set of locations, where l stands for left and r for right. We denote a mapping function, whose elements are pairs (j, loc_i), with $mapf$ which ranges over the set $MapF$ of partial bijections from $\mathbb{N}^+$ to $Loc \times \mathbb{N}^+$. Moreover, a global mapping M is a relation from PDF^+ to $\mathbb{N}^+ \times (Loc \times \mathbb{N}^+)$ such that $\forall f \in PDF^+ . M_f \in MapF$ [4], i.e. M is a global mapping including a mapping function for each different duration distribution. We denote the set of global mappings M by $\mathcal{M}$. In the following we use the shorthand $f : (j, loc_i)$ for $(f, (j, loc_i)) \in M$. Finally we make use of the auxiliary function $n : MapF \longrightarrow \mathbb{N}^+$ that computes the new index to be used for a clock name according to the well-naming rule, by choosing the minimum index not used by the other clocks with the same distribution already in execution, i.e. $n(mapf) = min\{k \mid k \notin dom(mapf)\}$.

Definition 2.6 The parallel composition $\mathcal{G}_1 \parallel_S \mathcal{G}_2$ of two well-named IGSMPs $\mathcal{G}_1 = (\Sigma_1, Act, T_{+,1}, T_{-,1}, T_{a,1}, s_{0,1})$ and $\mathcal{G}_2 = (\Sigma_2, Act, T_{+,2}, T_{-,2}, T_{a,2}, s_{0,2})$, with S being the synchronization set, is the tuple $(\Sigma, Act, T_+, T_-, T_a, (s_{0,1}, s_{0,2}, \emptyset))$ with

- $\Sigma = \Sigma_1 \times \Sigma_2 \times \mathcal{M}$ the set of states,
- $T_+ \subseteq (\Sigma \times \mathcal{C}^+ \times \Sigma)$, $T_- \subseteq (\Sigma \times \mathcal{C}^- \times \Sigma)$, and $T_a \subseteq (\Sigma \times Act \times \Sigma)$ are the least transition relations, such that $\forall (s_1, s_2, M) \in \Sigma$.

1$_{\mathbf{l}}$ $s_1 \xrightarrow{\alpha} s_1',\ \alpha \notin S \implies (s_1, s_2, M) \xrightarrow{\alpha} (s_1', s_2, M)$

2 $s_1 \xrightarrow{a} s_1' \wedge s_2 \xrightarrow{a} s_2',\ a \in S \implies$
$(s_1, s_2, M) \xrightarrow{a} (s_1', s_2', M)$

3$_{\mathbf{l}}$ $s_1 \xrightarrow{\langle C^+_{f,i}, w\rangle} s_1' \wedge s_2 \xrightarrow{\tau} \not\longrightarrow \implies$
$(s_1, s_2, M) \xrightarrow{\langle C^+_{f,n(M_f)}, w\rangle} (s_1', s_2, M \cup \{f : (n(M_f), l_i)\})$

4$_{\mathbf{l}}$ $s_1 \xrightarrow{C^-_{f,i}} s_1' \wedge s_2 \xrightarrow{\tau} \not\longrightarrow \wedge \nexists C_{g,h}, w.\, s_2 \xrightarrow{\langle C^+_{g,h}, w\rangle} \wedge f{:}(j, l_i) \in M \implies$

[4] Given a relation M from A to B, we denote with M_a the set $\{b \in B \mid (a, b) \in M\}$.

$(s_1, s_2, M) \xrightarrow{C^-_{f,j}} (s'_1, s_2, M - \{f:(j, l_i)\})$
and also the symmetric rules $\mathbf{1_r}, \mathbf{3_r}, \mathbf{4_r}$ referring to the local transitions of $\mathcal{G}_2$, which are obtained from the rules $\mathbf{1_l}, \mathbf{3_l}, \mathbf{4_l}$ by exchanging the roles of states s_1 (s'_1) and s_2 (s'_2) and by replacing l_i with r_i, hold true.

- $(s_{0,1}, s_{0,2}, \emptyset) \in \Sigma$ the initial state

Each state $s \in \Sigma$ of the composed model is represented by a triple including a pair of states ($s_1 \in \Sigma_1$ and $s_2 \in \Sigma_2$) and an auxiliary memory M containing all the mappings currently active in such a state. Rules **1** (**2**) describe the behavior of the composed model in the case of a standard action α performed by one (or both, via a synchronization) well-named IGSMPs, when $\alpha \notin S$ ($\alpha \in S$). Rules **3** and **4** define the behavior of the composed model in the case of delays locally performed by components. When in $\mathcal{G}_1$ ($\mathcal{G}_2$) occurs a transition labeled with $\langle C^+_{f,i}, w\rangle$, denoting the beginning of a delay with duration distribution f, then the new index $n(M_f)$ is determined for identifying the action at the level of the composed model, and the new mapping $f : (n(M_f), l_i)$ ($f : (n(M_f), r_i)$) is added to M. Conversely, when in $\mathcal{G}_1$ ($\mathcal{G}_2$) occurs a transition labeled with $C^-_{f,i}$, denoting the termination of a clock with duration distribution f, the particular clock with index j associated to l_i (r_i) in M_f terminates at the level of the composed model, and the index j becomes available. Note that the negative clauses in the premises enforce the maximal progress and the urgency of delays assumptions.

Theorem 2.7 *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two well-named IGSMPs. Then for each $S \subseteq Act - \{\tau\}$, $\mathcal{G}_1 \|_S \mathcal{G}_2$ is a well-named IGSMP.*

2.5 Hiding of Well-Named IGSMPs

Now, we address the problem of defining hiding of well-named IGSMPs, where the standard actions of a given set L are turned into invisible τ actions.

As we already explained, the capability of hiding actions make it possible to turn visible "incomplete" actions into invisible "complete" ones, thus giving the possibility of building a complete system from several system components. In particular while a visible action transition (as long as it is enabled) can delay indefinitely before being performed, when such an action is turned into an invisible action it must be executed in zero time.

Now we formally define the hiding $\mathcal{G}/L$ of a well-named IGSMP $\mathcal{G}$, where the set L of the visible actions to be hidden is a subset of $Act - \{\tau\}$.

Definition 2.8 The hiding $\mathcal{G}/L$ of a well-named IGSMP $\mathcal{G} = (\Sigma, Act, T_{+,1}, T_{-,1}, T_{a,1}, s_0)$ with L being the set of visible actions to be hidden is the tuple $(\Sigma, Act, T_+, T_-, T_a, s_0)$ where $T_+ \subseteq (\Sigma \times \mathcal{C}^+ \times \Sigma)$, $T_- \subseteq (\Sigma \times \mathcal{C}^- \times \Sigma)$, and $T_a \subseteq (\Sigma \times Act \times \Sigma)$ are the least set of transitions, such that $\forall s \in \Sigma$. [5]

[5] In order to distinguish transition of $T_{+,1}$, $T_{-,1}$ and $T_{a,1}$ from transitions of T_+, T_- and

$$\mathbf{1}\ s \xrightarrow{\alpha}_1 s',\ \alpha \notin L \implies s \xrightarrow{\alpha} s'$$

$$\mathbf{2}\ s \xrightarrow{a}_1 s',\ a \in L \implies s \xrightarrow{\tau} s'$$

$$\mathbf{3}\ s \xrightarrow{\theta}_1 s' \wedge \nexists a \in L.\, s \xrightarrow{a}_1 \implies s \xrightarrow{\theta} s'$$

Rules **1** and **2** are standard. Rule **3** says that the effect of the hiding operator over states of $\mathcal{G}$ which enable standard actions in L is to preempt all clock related transitions according to the maximal progress assumption.

Theorem 2.9 *Let $\mathcal{G}$ be a well-named IGSMP. Then for each $L \subseteq Act - \{\tau\}$, $\mathcal{G}/L$ is a well-named IGSMP.*

2.6 Equivalence of Well-Named IGSMPs

Now we will recall the notion of weak probabilistic bisimulation over IGSMPs which has been introduced in [6]. In particular weak bisimulation matches the execution of clocks with the same duration distribution similarly as in the dynamic approach of [7], deals with probabilistic choices similarly as in [20], and abstracts from standard τ actions similarly as in [22]. Such an equivalence is shown in [6] to be a congruence with respect to both parallel composition and hiding.

In our context we express cumulative probabilities by aggregating weights.

Definition 2.10 Let $\mathcal{G} = (\Sigma, Act, T_+, T_-, T_a)$ be a well-named IGSMTS.The function $TW : \Sigma \times PDF^+ \times \mathcal{P}(\Sigma) \longrightarrow \mathbb{R}^+ \cup \{0\}$, which computes the aggregated weight that a state $s \in \Sigma$ reaches a set of states $I \in \mathcal{P}(\Sigma)$ by starting a delay with duration distribution $f \in PDF^+$ is defined as: [6]

$$TW(s, f, I) = \sum \{\!|\, w \mid \exists i \in \mathbb{N}^+, s' \in I.\ s \xrightarrow{\langle C^+_{f,i}, w \rangle} s' \,|\!\}$$

Let $NPAct = Act \cup \mathcal{C}^-$, the set of non-probabilistic actions, be ranged over by σ. Let $\stackrel{\sigma}{\Longrightarrow}$ denote $(\xrightarrow{\tau})^* \xrightarrow{\sigma} (\xrightarrow{\tau})^*$, i.e. a sequence of transitions including a single σ transition and any number of τ transitions. Moreover, we define $\stackrel{\hat{\sigma}}{\Longrightarrow} = \stackrel{\sigma}{\Longrightarrow}$ if $\sigma \neq \tau$ and $\stackrel{\hat{\tau}}{\Longrightarrow} = (\xrightarrow{\tau})^*$, i.e. a possibly empty sequence of τ transitions.

Definition 2.11 Let $\mathcal{G} = (\Sigma, Act, T_+, T_-, T_a)$ be a well-named IGSMTS. An equivalence relation β on Σ is a *weak bisimulation* iff $s_1\ \beta\ s_2$ implies

- for every $\sigma \in NPAct$ and $s_1' \in \Sigma$,
$$s_1 \xrightarrow{\sigma} s_1' \text{ implies } s_2 \stackrel{\hat{\sigma}}{\Longrightarrow} s_2' \text{ for some } s_2' \text{ with } s_1'\ \beta\ s_2',$$

T_a we denote the former with "$\xrightarrow{}_1$" and the latter simply with "$\xrightarrow{}$".

[6] We use $\{\!|$ and $|\!\}$ to denote multiset parentheses. The summation of an empty multiset is assumed to yield 0. Since the method for computing the new index of a delay f that starts in a state P is fixed, we have that several transitions f^+ leaving P have all the same index i.

- $s_2 \stackrel{\hat{\tau}}{\Longrightarrow} s_2'$ for some s_2' such that, for every $f \in PDF^+$ and equivalence class I of β,
$$TW(s_1, f, I) = TW(s_2', f, I)$$

Two states s_1 and s_2 are weakly bisimilar, denoted by $s_1 \approx s_2$, iff (s_1, s_2) is included in some weak bisimulation. Two well-named IGSMPs $(\mathcal{G}_1, s_{0,1})$ and $(\mathcal{G}_2, s_{0,2})$ are weakly bisimilar, if their initial states $s_{0,1}$ and $s_{0,2}$ are weakly bisimilar in the well-named IGSMTS obtained with the disjoint union of $\mathcal{G}_1$ and $\mathcal{G}_2$.

3 Interactive Stochastic Timed Transition Systems

In this section we introduce Interactive Stochastic Timed Transition Systems (ISTTSs) that will be used in the next section to define a semantics for IGSMPs. We first briefly introduce some basic notions about probability spaces.

3.1 Probability Spaces

In this section we recall some basic notions related to measure theory and we introduce some notation that will be used in the rest of the paper.

Definition 3.1 A *σ-algebra* on a set Ω, denoted by $\mathcal{F}$, is a family of subsets of Ω that contains Ω and is closed under complementation and countable union. The elements of a σ-algebra $\mathcal{F}$ are called *measurable sets*. The pair $(\Omega, \mathcal{F})$ is called a *measurable space*.

Definition 3.2 The *Borel σ-algebra* on a set Ω with a topology, denoted by $\mathcal{B}(\Omega)$, is defined to be the *σ-algebra* generated by the open subsets (or equivalently, by the closed subsets) of Ω.

Definition 3.3 A *finite measure* μ on a measurable space $(\Omega, \mathcal{F})$ is a function that assigns a non-negative real value to each element of $\mathcal{F}$, such that $\mu(\emptyset) = 0$ and, supposed $\{C_i\}_{i \in I}$, with $I \subseteq \mathbb{N}$, to be a family of disjoint elements of $\mathcal{F}$, $\mu(\cup_{i \in I} C_i) = \sum_{i \in I} \mu(C_i)$. The triple $(\Omega, \mathcal{F}, \mu)$ is called a *measure space*. If we have in addition that $\mu(\Omega) = 1$, then the triple $(\Omega, \mathcal{F}, \mu)$ is also called a *probability space*.

We now define some operations over measure spaces and probability spaces.

Definition 3.4 Let $(\Omega, \mathcal{F}, \mu)$ be a measure space and let f be a function defined on Ω, then $f(\Omega, \mathcal{F}, \mu)$ denotes the triple $(f(\Omega), \{C \subseteq f(\Omega) \mid f^{-1}(C) \in \mathcal{F}\}, \mu'\})$, where $\forall C \subseteq f(\Omega).\ \mu'(C) = \mu(f^{-1}(C))$.

It is easy to verify that, since $(\Omega, \mathcal{F}, \mu)$ is a measure space, $f(\Omega, \mathcal{F}, \mu)$ is a measure space as well. Such a measure space is called the measure space *induced* by f from $(\Omega, \mathcal{F}, \mu)$. Moreover if $(\Omega, \mathcal{F}, \mu)$ is a probability space, then $f(\Omega, \mathcal{F}, \mu)$ is a probability space as well.

Definition 3.5 Let $(\Omega, \mathcal{F}, \mu)$ be a measure space and let p be a positive real number, then $p \cdot (\Omega, \mathcal{F}, \mu)$ denotes the triple $(\Omega, \mathcal{F}, \mu')$, where $\forall C \subseteq \Omega.\ \mu'(C) = p \cdot \mu(C)$.

It is easy to verify that, since $(\Omega, \mathcal{F}, \mu)$ is a measure space, $p \cdot (\Omega, \mathcal{F}, \mu)$ is a measure space as well.

Definition 3.6 Let $(\Omega', \mathcal{F}', \mu')$ and $(\Omega'', \mathcal{F}'', \mu'')$ be two measure spaces, then $(\Omega', \mathcal{F}', \mu') + (\Omega'', \mathcal{F}'', \mu'')$ denotes the triple $(\Omega' \cup \Omega'', \{C \subseteq \Omega' \cup \Omega'' \mid C \cap \Omega' \in \mathcal{F}' \wedge C \cap \Omega'' \in \mathcal{F}''\}, \mu)$, where $\forall C \subseteq \Omega' \cup \Omega''.\ \mu(C) = \mu'(C \cap \Omega') + \mu''(C \cap \Omega'')$.

It is easy to verify that, since $(\Omega', \mathcal{F}', \mu')$ and $(\Omega'', \mathcal{F}'', \mu'')$ are measure spaces, $(\Omega', \mathcal{F}', \mu') + (\Omega'', \mathcal{F}'', \mu'')$ is a measure space as well. Moreover, considered a set of probability spaces $\{(\Omega_i, \mathcal{F}_i, \mu_i)\}_{i \in I}$, with $I \subseteq \mathbb{N}$ finite index set, and a set $\{p_i\}_{i \in I}$ of positive real numbers such that $\sum_{i \in I} p_i = 1$, we have that $\sum_{i \in I} p_i \cdot (\Omega_i, \mathcal{F}_i, \mu_i)$ is a probability space.

Definition 3.7 Let $(\Omega', \mathcal{F}', \mu')$ and $(\Omega'', \mathcal{F}'', \mu'')$ be two measure spaces, then $(\Omega', \mathcal{F}', \mu') \cdot (\Omega'', \mathcal{F}'', \mu'')$ denotes the triple $(\Omega' \times \Omega'', \{C' \times C'' \mid C' \in \mathcal{F}' \wedge C'' \in \mathcal{F}''\}, \mu')$, where $\forall C' \subseteq \Omega', C'' \subseteq \Omega''.\ \mu'(C' \times C'') = \mu'(C') \cdot \mu''(C'')$.

It is easy to verify that, since $(\Omega', \mathcal{F}', \mu')$ and $(\Omega'', \mathcal{F}'', \mu'')$ are measure spaces, $(\Omega', \mathcal{F}', \mu') \cdot (\Omega'', \mathcal{F}'', \mu'')$ is a measure space as well. Moreover, if $(\Omega, \mathcal{F}, \mu)$ is a probability space, then $(\Omega', \mathcal{F}', \mu') \cdot (\Omega'', \mathcal{F}'', \mu'')$ is a probability space as well.

We now show that a probability distribution function on real numbers defines a unique probability space over the Borel σ-algebra of real numbers.

Theorem 3.8 *Let $F \in PDF$ be a probability distribution function on $\mathbb{R}$. There is a unique probability measure P on $\mathcal{B}(\mathbb{R})$ such that $\forall a, b \in \mathbb{R}, a < b.\ P(\,(a, b]\,) = F(b) - F(a)$.*

Finally, we present a notion of equivalence over measure spaces which relates measure spaces with different domains by assuming that they assign measure 0 to the every set of elements not included in their domain.

Definition 3.9 Let $(\Omega', \mathcal{F}', \mu')$ and $(\Omega'', \mathcal{F}'', \mu'')$ be measure spaces. We say that $(\Omega', \mathcal{F}', \mu')$ is equivalent to $(\Omega'', \mathcal{F}'', \mu'')$, written $(\Omega', \mathcal{F}', \mu') \approx (\Omega'', \mathcal{F}'', \mu'')$, if $\forall C' \in \mathcal{F}'.\ C' \cap \Omega'' \in \mathcal{F}'' \wedge \mu''(C' \cap \Omega'') = \mu'(C')$ and $\forall C'' \in \mathcal{F}''.\ C'' \cap \Omega' \in \mathcal{F}' \wedge \mu'(C'' \cap \Omega') = \mu''(C'')$.

Note that the definition above implies that if $(\Omega', \mathcal{F}', \mu') \approx (\Omega'', \mathcal{F}'', \mu'')$ then both $\Omega' - \Omega'' \in \mathcal{F}'$ with $\mu'(\Omega' - \Omega'') = 0$ and $\Omega'' - \Omega' \in \mathcal{F}''$ with $\mu''(\Omega'' - \Omega') = 0$.

3.2 The ISTTS Model

In this section we formally introduce Interactive Stochastic Timed Transition Systems (ISTTS) which include three type of transitions: *standard ac-*

tion transitions, representing the interactive behavior of a system component, *probabilistic transitions* (expressed by means of probability spaces) representing (infinitely branching) probabilistic choices and *numeric time transitions* representing a fixed temporal delay.

As far as standard actions are concerned they have exactly the same behavior as in IGSMPs. In ISTTS non-deterministic choices can arise not only from transitions labeled with standard visible actions (like in IGSMPs), but also from transitions representing the passage of time. As usual in the real time literature (see e.g. [23]), several timed transition leaving a state offer the possibility to the observer to choose the amount of time after which he wants to observe the status of the system.

In ISTTS we have two different kinds of state:

- *silent states* which are exactly like in IGSMPs.
- *probabilistic states* enabling probabilistic transitions, expressed by a probability space PS, and (possibly) visible action transitions a only. In such states the ISTTS just chooses a new state in zero time according to the probability space and may potentially interact with the environment through one of its visible actions (see e.g. Fig. 2.a).
- *timed states* enabling numeric timed transitions t and (possibly) visible action transitions a only. In such states the ISTTS just performs a non-deterministic choice among the numeric timed transitions (which cause the amount of time labeling the transition to pass) and may potentially interact with the environment through one of its visible actions (see e.g. Fig. 2.b).

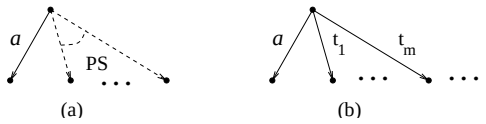

Fig. 2. Some examples of possible states of an ISTTS

In the following we present the formal definition of Interactive Stochastic Timed Transition System (ISTTS), then we will define Rooted Interactive Stochastic Timed Transition Systems as ISTTSs possessing an initial state. Formally, given a time domain $TD \subseteq \mathbb{R}^+ \cup \{0\}$, we use $t, t', \ldots$, representing time values, to range over TD.

Definition 3.10 An Interactive Stochastic Timed Transition System (ISTTS) is a tuple $\mathcal{D} = (\Sigma, TD, Act, P, T_t, T_a)$ with

- Σ a set of possibly infinite states,
- TD a time domain, i.e. the set of possible values over which the labels of the numeric timed transitions range,
- Act a set of standard actions,
- $P : \Sigma' \to PS(\Sigma - \Sigma')$, where $\Sigma' \subset \Sigma$ and $PS(\Sigma'')$ denotes the family of probability spaces $(\Sigma''', \mathcal{F}, \mu)$ over sets of states $\Sigma''' \subseteq \Sigma''$, the probabilistic

transition relation which associates a probability space with some of the states of the ISTTS; and $T_t \subseteq (\Sigma \times TD \times \Sigma)$ and $T_a \subseteq (\Sigma \times Act \times \Sigma)$ two transition relations representing time passage and action execution, respectively. P, T_t and T_a must be such that $\forall s \in \Sigma$.

- $s \xrightarrow{\tau} \implies s \notin dom(P) \wedge \nexists t.s \xrightarrow{t}$
- $s \in dom(P) \implies \nexists t.s \xrightarrow{t}$
- $s \xrightarrow{\tau} \vee \exists t.s \xrightarrow{t} \vee s \in dom(P)$

Definition 3.11 A Rooted Interactive Stochastic Timed Transition System (RISTTS) is a tuple $\mathcal{D} = (\Sigma, TD, Act, P, T_t, T_a, s_0)$, where $s_0 \in \Sigma$ is the initial state and $(\Sigma, TD, Act, P, T_t, T_a)$ is an ISTTS.

The meaning the constraints over transition relations is the following. The first requirement says that (similarly as in IGSMPs) if a state that can perform internal τ actions then it cannot perform neither probabilistic transitions nor timed transitions (*maximal progress* assumption). The second requirement says that (similarly as in IGSMPs) if a state that can perform probabilistic transitions then it cannot perform timed transitions (*urgency of choices* assumption). The third requirement says that (similarly as in IGSMPs) we cannot have states where time is not allowed to pass (time deadlocks).

3.3 Parallel of Rooted ISTTSs

Now we define, similarly as for IGSMPs, the parallel composition à la CSP of RISTTSs.

In such a parallel composition the discrete timed transitions of the composed RISTTSs are constrained to synchronize, so that the same amount of time passes for both systems, i.e. when time advances for one RISTTS it must also advance for the other RISTTS.

Definition 3.12 The parallel composition $\mathcal{D}_1 \|_S \mathcal{D}_2$ of two RISTTSs $\mathcal{D}_1 = (\Sigma_1, TD, Act, P_1, T_{t,1}, T_{a,1}, s_{0,1})$ and $\mathcal{D}_2 = (\Sigma_2, TD, Act, P_2, T_{t,2}, T_{a,2}, s_{0,2})$, with $S \subset Act - \{\tau\}$ being the synchronization set, is the tuple $(\Sigma, TD, Act, P, T_t, T_a, (s_{0,1}, s_{0,2}))$ with:

- $\Sigma = \Sigma_1 \times \Sigma_2$ the set of states
- P the partial function defined over $\Sigma_1 \times \Sigma_2$ obtained from P_1 and P_2 as follows: $\forall s_1 \in \Sigma_1, s_2 \in \Sigma_2$.

$$P(s_1, s_2) = Id^1_{s_2}(P_1(s_1)) \quad \text{if } s_1 \in dom(P_1) \wedge s_2 \xrightarrow{t}$$

$$P(s_1, s_2) = Id^2_{s_1}(P_2(s_2)) \quad \text{if } s_2 \in dom(P_2) \wedge s_1 \xrightarrow{t}$$

$$P(s_1, s_2) = P(s_1) \cdot P(s_2) \quad \text{if } s_1 \in dom(P_1) \wedge s_2 \in dom(P_2)$$

$$P(s_1, s_2) \text{ is not defined} \quad \text{otherwise}$$

with $Id^1_{s_2} : \Sigma_1 \longrightarrow (\Sigma_1 \times \{s_2\})$ defined by $\forall s \in \Sigma_1. Id_{s_2}(s) = (s, s_2)$ and $Id^2_{s_1} : \Sigma_2 \longrightarrow (\{s_1\} \times \Sigma_2)$ defined by $\forall s \in \Sigma_2. Id_{s_1}(s) = (s_1, s)$.

- $T_t \subseteq (\Sigma \times TD \times \Sigma)$ and $T_a \subseteq (\Sigma \times Act \times \Sigma)$ the least transition relations, such that

  1$_\mathbf{l}$ $s_1 \xrightarrow{\alpha} s_1',\ \alpha \notin S \implies (s_1, s_2) \xrightarrow{\alpha} (s_1', s_2)$

  1$_\mathbf{r}$ $s_2 \xrightarrow{\alpha} s_2',\ \alpha \notin S \implies (s_1, s_2) \xrightarrow{\alpha} (s_1, s_2')$

  2 $s_1 \xrightarrow{a} s_1' \wedge s_2 \xrightarrow{a} s_2',\ a \in S \implies (s_1, s_2) \xrightarrow{a} (s_1', s_2')$

  3 $s_1 \xrightarrow{t} s_1' \wedge s_2 \xrightarrow{t} s_2' \implies (s_1, s_2) \xrightarrow{t} (s_1', s_2')$

- $(s_{0,1}, s_{0,2}) \in \Sigma$ the initial state.

When evaluating the probability spaces associated by function P to the states of the composed model we make use of induced probability spaces (see Definition 3.4) and we enforce the maximal progress assumption. Moreover we produce a single "global" probability space whenever both RISTTSs engage in probabilistic choices (we assume that choices are performed independently). When evaluating action transitions we just make use of standard rules. Finally we require timed transitions to synchronize.

Theorem 3.13 *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two RISTTSs. Then for each $S \subseteq Act - \{\tau\}$, $\mathcal{D}_1 \parallel_S \mathcal{D}_2$ is a RISTTS.*

3.4 Hiding of Rooted ISTTSs

Now we define, similarly as for IGSMPs, the hiding of RISTTSs.

Definition 3.14 The hiding $\mathcal{D}/L$ of a RISTTS $\mathcal{D}_1 = (\Sigma, TD, Act, P_1, T_{t1}, T_{a1}, s_0)$, with $L \subset Act - \{\tau\}$ being the set of visible actions to be hidden, is the tuple $(\Sigma, TD, Act, P, T_t, T_a, s_0)$, with:

- P the partial function obtained from P_1 by removing from its domain those states (and the associated probability spaces) which enable at least one transition labeled with an action in L
- $T_t \subseteq (\Sigma \times TD \times \Sigma)$ and $T_a \subseteq (\Sigma \times Act \times \Sigma)$ the least transition relations, such that $\forall s \in \Sigma$. [7]

  1 $s \xrightarrow{\alpha}_1 s',\ \alpha \notin L \implies s \xrightarrow{\alpha} s'$

  2 $s \xrightarrow{a}_1 s',\ a \in L \implies s \xrightarrow{\tau} s'$

  3 $s \xrightarrow{t}_1 \wedge\ \nexists a \in L.\, s \xrightarrow{a}_1 \implies s \xrightarrow{t}$

Similarly as for IGSMPs, in the definition of the hiding operator in addition to standard rules we make use of rules which enforce the maximal progress assumption.

Theorem 3.15 *Let $\mathcal{D}$ be a RISTTS. Then for each $L \subseteq Act - \{\tau\}$, $\mathcal{D}/L$ is a RISTTS.*

[7] In order to distinguish transition of $T_{w,1}$, $T_{t,1}$ and $T_{a,1}$ from transitions of T_w, T_t and T_a we denote the former with "$\xrightarrow{}_1$" and the latter simply with "$\xrightarrow{}$".

3.5 Equivalence of Rooted ISTTSs

Now we introduce a notion of weak bisimulation for RISTTSs which constitutes an extension of the approach of [20] to probability spaces and abstracts from standard τ actions similarly as in [22].

Given an equivalence relation β on a set Σ and a set $I \subseteq \Sigma$, we first define the function $EC_{I,\beta} : I \to \Sigma/\beta$ which maps each state $s \in I$ into the corresponding equivalence class $[s]_\beta$ in Σ.

Definition 3.16 Let $\mathcal{D} = (\Sigma, TD, Act, P, T_t, T_a)$ be an ISTTS. An equivalence relation β on Σ is a *weak bisimulation* iff $s_1 \ \beta \ s_2$ implies

- for every $\alpha \in Act$,
$$s_1 \xrightarrow{\alpha} s_1' \text{ implies } s_2 \stackrel{\hat{\alpha}}{\Longrightarrow} s_2' \text{ for some } s_2' \text{ with } s_1' \ \beta \ s_2',$$
- for every $t \in TD$,
$$s_1 \xrightarrow{t} s_1' \text{ implies } s_2 \xrightarrow{t} s_2' \text{ for some } s_2' \text{ with } s_1' \ \beta \ s_2',$$
- $s_2 \stackrel{\hat{\tau}}{\Longrightarrow} s_2'$ for some s_2' such that, denoted $P(s_1) = (\Sigma_1, \mathcal{F}_1, \mu_1)$ and $P(s_2') = (\Sigma_2, \mathcal{F}_2, \mu_2)$, we have that $EC_{\Sigma_1,\beta}(P(s_1)) \approx EC_{\Sigma_2,\beta}(P(s_2'))$

Two states s_1 and s_2 are weakly bisimilar, denoted by $s_1 \approx s_2$, iff (s_1, s_2) is included in some weak bisimulation. Two RISTTSs $(\mathcal{D}_1, s_{0,1})$ and $(\mathcal{D}_2, s_{0,2})$ are weakly bisimilar, if their initial states $s_{0,1}$ and $s_{0,2}$ are weakly bisimilar in the ISTTS obtained with the disjoint union of $\mathcal{D}_1$ and $\mathcal{D}_2$.

In the last item we exploit induced probability spaces (see Definition 3.4) and equivalence between probability spaces (see Definition 3.9) to check that states s_1 and s_2' have the same aggregated probability to reach the same equivalence classes.

4 A Semantics for Interactive Generalized Semi-Markov Processes

In this section we present a semantics for well-named Interactive Generalized Semi-Markov Processes which maps them onto Interactive Stochastic Timed Transition Systems. Such a semantics explicitely represents the passage of time by means of transitions labeled with numeric time delays and turns probability distributions of durations into infinitely branching probabilistic choices which lead to states performing numeric time delays with a different duration. In particular, differently from [14] where a technique based on *residual lifetimes of clocks* is used, the states of the semantics of an Interactive Generalized Semi-Markov Process encode the *spent lifetimes of clocks*. This means that, in a timed state of the IGSMP where several clocks $C_{n_1} \ldots C_{n_k}$ are in execution, the time delay originated by a clock C_{n_i} is determined according to its residual distribution of duration which is evaluated from (i) its associated duration distribution and (ii) its spent lifetime. Once we have

sampled a time value t_i from the residual duration distribution of each clock C_{n_i}, we just take the minimum $t_{\min}$ of the sampled values and we consider the clock $C_{n_{\min}}$ which sampled such a time value. Such a "winning clock" is the clock that terminates in the timed state of the IGSMP. After this event the other clocks (which are still in execution) carry over their spent lifetimes, which now is given by $t'_i = t_i + t_{\min}$. Since, according to this approach, the residual duration of a clock is re-sampled in each IGSMP state until it terminates, an adversary (or scheduler) which resolves non-deterministic choices in an IGSMP cannot gain information about the future behavior of the system on which to base its decisions.

Example 4.1 Let us consider the IGSMP depicted in Fig. 3, where three temporal delays are started by activating three clocks C_{n_1}, C_{n_2}, and C_{n_3}. In particular, we concentrate on the case in which C_{n_2} is the first clock to terminate.

In Fig. 4 we show the semantics of the IGSMP of Fig. 3 obtained by following an approach similar to that of [14], which encodes in each state the *residual lifetimes of clocks*. Each state is enriched with the set of active clocks together with their residual lifetimes. In state $\langle s_0, \emptyset \rangle$ (where no clock is active) three numeric time delays t_1, t_2, and t_3 are sampled and associated with the lifetime of the clocks C_{n_1}, C_{n_2}, and C_{n_3}, respectively. Depending on which is the clock $C_{n_{\min}}$ sampling the minimum time value t_{min} in state $\langle s_0, \emptyset \rangle$, we move to one of three different classes of states, one for each possible winning clock. Afterwards, a temporal transition labeled with a numeric time value t between 0 and t_{min} is taken, and each residual duration is accordingly modified by subtracting t_{min} from the residual lifetime of each clock. For the sake of readability in Fig. 4 we just depict one trace leading from s_0 to a state s_1 which belongs to the class of states for which C_{n_2} is the winning clock (i.e. t_2 is t_{min}), and then from s_1 to the state s_2 via the transition labeled with the time value t_2, so that in s_2 the clock C_{n_2} is terminated. In state s_2 the residual lifetimes of the remaining active clocks C_{n_1} and C_{n_3} are $t_1 - t_{min}$ and $t_3 - t_{min}$ respectively. By exploiting this information an adversary may already know which clock between C_{n_1} and C_{n_3} will terminate first and consequently guide the nondeterministic choice in state s_2.

In Fig. 5 we show the semantics of the IGSMP of Fig. 3 obtained by following the approach that we adopt in this paper, which is based on the *spent lifetimes of clocks*. Each state is enriched with: (*i*) the set of active clocks together with their spent lifetimes, and (*ii*) a pair $C_n : t$ containing the time value sampled by the winning clock in a timed state of the IGSMP and the clock name. The latter field is set to "$-$" whenever the IGSMP is not in a timed state. The sampling executed in state $\langle s_0, \emptyset, - \rangle$ leads to a state where the three starting clocks are associated with the spent lifetime 0 (because the corresponding transition does not represent a passage of time but simply the result of the sampling), and the winning clock C_n and its sampled value are reported too. As in the case of Fig. 4, in Fig. 5 we just report one trace leading

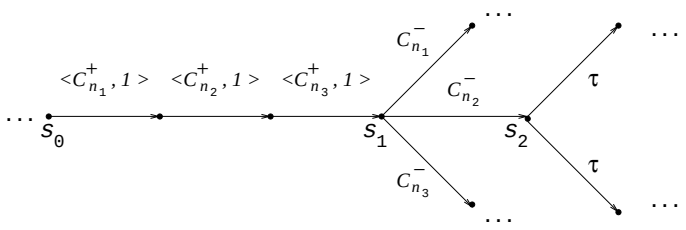

Fig. 3. Example of an IGSMP

from s_0 to a state s_1 which belongs to the class of states for which C_{n_2} is the winning clock (i.e. C_{n_2} is $C_{n_{min}}$ and t_2 is its sampled value), and then from s_1 to the state s_2 via the transition labeled with the value t_2, so that in s_2 the clock C_{n_2} is terminated. In state s_2 the spent lifetimes of the remaining active clocks C_{n_1} and C_{n_3} are both equal to t_2, and their residual durations depend on both such a value and the duration distribution associated with the clocks. Since, according to this approach, the time to termination of clocks C_{n_1} and C_{n_3} is re-sampled, an adversary cannot gain in advance any information about the future behavior of the system and he cannot exploit this information when resolving the nondeterministic choice in state s_2.

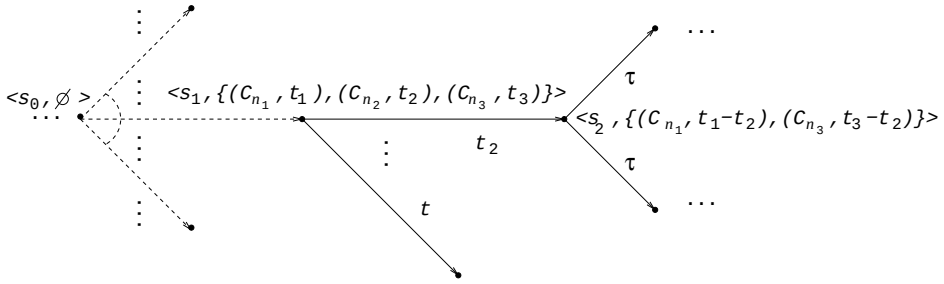

Fig. 4. Example of semantics based on residual lifetimes

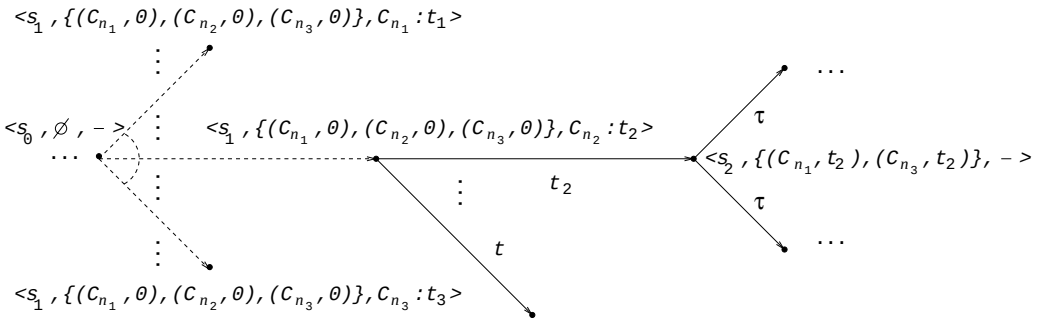

Fig. 5. Example of semantics based on spent lifetimes

In the following we introduce some preliminary definitions which are needed to define the semantics of IGSMPs.

Definition 4.2 Given a duration probability distribution $f \in PDF^+$ and a time value $t \in \mathbb{R}^+ \cup \{0\}$, we denote by $[f \mid t]$ the residual duration distribution of a clock C_n with duration distribution f which, after t time units from

when it started, has not terminated yet (t is its spent lifetime). More formally, if T is a random variable with distribution f, i.e. $\forall t' \in \mathbb{R}^+ \cup \{0\}.\ f(t') = P(T \leq t')$, then $[f \mid t]$ is the probability distribution defined as follows. For all $t' \in \mathbb{R}^+ \cup \{0\}$ we have that:

$$[f \mid t](t') = P(T \leq t' + t \mid T > t)$$

Theorem 4.3 *Given $f \in PDF^+$ and $t \in \mathbb{R}^+ \cup \{0\}$, we have that for all $t' \in \mathbb{R}^+ \cup \{0\}$:*

$$[f \mid t](t') = \frac{f(t+t') - f(t)}{1 - f(t)}$$

Consider a family of probability distribution functions $f_1, \ldots, f_k \in PDF$. We denote by $\mathcal{R}(f_1, \ldots, f_k)$ the probability space $\prod_{i=1\ldots k}(\mathbb{R}, \mathcal{B}(\mathbb{R}), P_i)$, where P_i is the unique probability measure on $\mathcal{B}(\mathbb{R})$ obtained from f_i (see Theorem 3.8).

Definition 4.4 Let the residual duration distribution of the set of clocks $C_{n_1}, \ldots, C_{n_k}$ in execution in an IGSMP state be $f_1, \ldots, f_k$, i.e. the probability that a certain tuple of residual durations $(t_1, \ldots, t_k)$ is sampled from the clocks is described by the probability space $\mathcal{R}(f_1, \ldots, f_k)$. For each $I \subseteq \{1, \ldots, k\}$ such that $|I| \geq 2$, the event $Term(I)$ of contemporaneous termination of the clocks $\{C_{n_i} \mid i \in I\}$ in execution is the following measurable subset of the sample space $\mathbb{R}^k$:

$$Term(I) = \{(t_1, \ldots, t_k) \mid \exists t.\ (\forall i \in I.\ t_i = t) \wedge (\forall i \notin I.\ t_i > t\}$$

Since in an IGSMP clocks in execution in a state cannot terminate at the same time instant (see Sect. 2.1) we have that each event $Term(I)$ of contemporaneous termination of a subset $\{C_{n_i} \mid i \in I\}$ of the clocks in execution $C_{n_1}, \ldots, C_{n_k}$ occurs with probability 0. More formally, we have that in each state of an IGSMP, if $(\mathbb{R}^k, \mathcal{F}, P)$ is the probability space $\mathcal{R}(f_1, \ldots, f_k)$ expressing the residual duration of the clocks $C_{n_1}, \ldots, C_{n_k}$ in execution in the state, for each $I \subseteq \{1, \ldots, k\}$ such that $|I| \geq 2$, we have $P(Term(I)) = 0$. We exploit this fact in order to reduce the domain of the probability space for a set of active clocks. In particular instead of considering the entire $\mathcal{R}(f_1, \ldots, f_k)$ we can just restrict to consider $\breve{\mathcal{R}}(f_1, \ldots, f_k)$ defined as follows.

Definition 4.5 $\breve{\mathcal{R}}(f_1, \ldots, f_k)$ is the triple $(\breve{\mathbb{R}}^k, \breve{\mathcal{F}}, \breve{P})$ defined as follows. Let $(\mathbb{R}^k, \mathcal{F}, P)$ be the probability space $\mathcal{R}(f_1, \ldots, f_k)$, then we have:

- $\breve{\mathbb{R}}^k = \mathbb{R}^k - \bigcup_{I \subseteq \{1,\ldots,k\}, |I| \geq 2} Term(I)$
- $\breve{\mathcal{F}} = \{E \subseteq \mathcal{F} \mid E \subseteq \breve{\mathbb{R}}^k\}$
- $\breve{P} = \{(E, p) \mid E \in \breve{\mathcal{F}} \wedge (E, p) \in P\}$

Theorem 4.6 *Let $(\mathbb{R}^k, \mathcal{F}, P)$ be the probability space $\mathcal{R}(f_1, \ldots, f_k)$. If $\forall I \subseteq \{1, \ldots, k\}$. $|I| \geq 2 \Rightarrow P(Term(I)) = 0$, then $\breve{\mathcal{R}}(f_1, \ldots, f_k)$ is a probability space.*

$$(P1) \quad \frac{(\exists C_n.\ s \xrightarrow{C_n^-}) \ \wedge \ \{C_{n_1}, \ldots, C_{n_k}\} = dom(v)}{P(\langle s, v, - \rangle) = Sample_{s,v}^{\{\mathbf{n}_i\}}(\ \breve{\mathcal{R}}([D(C_{n_1}) \mid v(C_{n_1})], \ldots, [D(C_{n_k}) \mid v(C_{n_k})])\ )}$$

$$(P2) \quad \frac{(\exists C_n, w.\ s \xrightarrow{<C_n^+, w>}) \ \wedge \ Pr = \{\ (\ <C_n, s'>,\ w/\,TW(s)\ )\ \mid\ s \xrightarrow{<C_n^+, w>} s'\ \}}{P(\langle s, v, - \rangle) = \sum_{<C_n, s'> \in dom(Pr)} Pr(<C_n, s'>) \cdot P(\langle s', v \cup \{(C_n, 0)\}, - \rangle)}$$

$$(T1) \quad \langle s, v, C_n : t \rangle \xrightarrow{t'} \langle s, v + t', - \rangle \qquad 0 \leq t' < t$$

$$(T2) \quad \frac{s \xrightarrow{C_n^-} s'}{\langle s, v, C_n : t \rangle \xrightarrow{t} \langle s', (v - C_n) + t, - \rangle}$$

$$(T3) \quad \frac{(\nexists \theta.\ s \xrightarrow{\theta}) \ \wedge \ s \xrightarrow{\tau} \!\!\!\!/}{\langle s, v, - \rangle \xrightarrow{t} \langle s, v, - \rangle} \qquad t \geq 0$$

$$(A1) \quad \frac{s \xrightarrow{\alpha} s'}{\langle s, v, - \rangle \xrightarrow{\alpha} \langle s', v, - \rangle} \qquad (A2) \quad \frac{s \xrightarrow{a} s'}{\langle s, v, C_n : t \rangle \xrightarrow{a} \langle s', v, - \rangle}$$

$$TW(s) = \sum \{\!|\ w \mid \exists C_n.\ s \xrightarrow{<C_n^+, w>}\ |\!\}$$

$$Sample_{s,v}^{\{\mathbf{n}_i\}}(t_1, \ldots, t_k) = \langle s, v, C_{n_{min}} : t_{min} \rangle$$

where *min* is the only index i such that: $\mathrm{t_i} = \min_{\mathrm{j} \in \{1, \ldots, \mathrm{k}\}} \mathrm{t_j}$

Table 2
Semantic rules for IGSMPs

We are now in a position to formally define the semantics of an IGSMP.

Definition 4.7 The semantics of an IGSMP $\mathcal{G} = (\Sigma, \mathcal{C}, D, Act, T_+, T_-, T_a, s_0)$ is the RISTTS $[\![\mathcal{G}]\!] = (\Sigma', \mathbb{R}^+ \cup \{0\}, Act, P, T_t, T_a, s_0')$ where:

- $\Sigma' = (\Sigma \times Spent \times Sample)$ is the set of states of the RISTTS, where *Spent*, ranged over by v, is the set of partial functions from $\mathcal{C}$ to $\mathbb{R}^+ \cup \{0\}$, expressing the time already spent in execution by the clocks currently

in execution in the IGSMP (clocks in the domain of *Spent*), and *Sample*, ranged over by *sample*, is the set $(\mathcal{C} \times (\mathbb{R}^+ \cup \{0\}))) \cup \{-\}$, where a pair (C_n, t), also written $C_n : t$, denotes that the IGSMP is currently executing a set of clocks and that clock C_n has sampled the minimum residual time delay with t being the value of such a delay; while "$-$" denotes that started clocks are not under execution (e.g. the IGSMP is in a choice state or in a silent state).

- $\mathbb{R}^+ \cup \{0\}$ is the time domain: we consider continuous time.
- Act is the set of standard actions considered in the IGSMP.
- P, which associates a probability space (expressing next state probability) to some of the states in Σ', is defined to be the least partial function on Σ' satisfying the operational rules in the first part of Table 2.
- T_t is the set of timed transitions which are defined as the least relation over $\Sigma' \times (\mathbb{R}^+ \cup \{0\}) \times \Sigma'$ satisfying the operational rules in the second part of Table 2.
- T_a is the set of action transitions which are defined as the least relation over $\Sigma' \times Act \times \Sigma'$ satisfying the operational rule in the third part of Table 2.
- $s'_0 = \langle s_0, \emptyset, - \rangle$ is the initial state of the RISTTS, where the IGSMP is in the initial state and no clock is in execution.

In Table 2 we make use of the following notation. Given $v \in$ *Spent*, we define $v - C_n$ to be the partial function obtained from v by removing C_n (and the associated value) from its domain. We define $v + t$, with $t \in \mathbb{R}^+ \cup 0$, to be the partial function obtained from v by adding t to the time value associated with each clock in the domain of v. We use the notation $\{\mathbf{n}_i\}$ to stand for $\{\mathbf{n}_i\}_{i=1...k}$, representing the sequence of names $n_1, \ldots n_k$ (in Table 2 the length k of the sequence is always clarified by the context in which $\{\mathbf{n}_i\}$ is used). Finally in the forth part of Table 2 we define two auxiliary functions. The function $TW : \Sigma \longrightarrow \mathbb{R}^+ \cup \{0\}$ computes the overall weight of the clock start transitions leaving a state of an IGSMP. Moreover, given a sate of the IGSMP $s \in \Sigma$, a partial function mapping active clock into their spent lifetimes $v \in$ *Spent*, and a sequence $\{n_1, \ldots, n_k\}$ of clock indexes, the function $Sample_{s,v}^{\{\mathbf{n}_i\}}$ maps a tuple $(t_1, \ldots, t_k)$ of time values sampled by clocks into the corresponding state $\langle s, v, C_{n_{\min}} : t_{\min} \rangle$ reached in the RISTTS, where *min* is the index of the clock which sampled the least time value. Note that function $Sample_{s,v}^{\{\mathbf{n}_i\}}$ is used in Table 2 for deriving (via induction, see Definition 3.4) a probability space over the states of the RISTTS from the probability space $\breve{\mathcal{R}}([D(C_{n_1}) \mid v(C_{n_1})], \ldots, [D(C_{n_k}) \mid v(C_{n_k})])$ over residual durations sampled by active clocks in a state of the IGSMP.

Theorem 4.8 *Let $\mathcal{G}'$, $\mathcal{G}''$ be two well-named IGSMPs. If $\mathcal{G}' \approx \mathcal{G}''$ then $[\![\mathcal{G}']\!] \approx [\![\mathcal{G}'']\!]$.*

The following theorems show that the semantics of well-named IGSMPs is

indeed compositional.

Theorem 4.9 *Let $\mathcal{G}'$, $\mathcal{G}''$ be two well-named IGSMPs. For each $S \subseteq Act - \{\tau\}$ we have $[\![\mathcal{G}']\!] \parallel_S [\![\mathcal{G}'']\!] \approx [\![\mathcal{G}' \parallel_S \mathcal{G}'']\!]$.*

Theorem 4.10 *Let $\mathcal{G}$ be a well-named IGSMP. For each $L \subseteq Act - \{\tau\}$ we have $[\![\mathcal{G}]\!]/L \approx [\![\mathcal{G}/L]\!]$.*

5 Conclusion

Dealing with non-determinism in probabilistic systems with general distributions raises a series of problems, including the correct management of the residual durations of generally distributed delays in system states and the interplay of non-deterministic choices and probabilistic behaviors of temporal delays. The former problem can be solved by representing the temporal behavior of a system by using clocks (as in Timed Automata [23]) or elements (as in GSMPs [21]) whose durations are associated with generally distributed random variables. In particular we can correctly manage the residual durations of active clocks (elements) in system states with two different approaches which are borrowed from the theory of GSMPs: one based on spent clock lifetimes and one based on residual clock lifetimes. In this paper we have shown how to apply the former approach to the specification and analysis of concurrent systems including generally distributed delays, instead of using the latter approach as previously done in the literature [14]. In the former approach (similarly as in Timed Automata [23]) states are enriched with spent lifetimes of clocks and for each timed state where a clock C_n is active, the residual duration of C_n is sampled depending on both its spent lifetime and its associated duration distribution. In the latter approach (similarly as in classical Discrete Event Simulation [12]), the lifetime of each clock is sampled all at once at the clock start event, and states are enriched with residual lifetimes of clocks directly determining their residual duration. As we have shown, the drawback of the approach based on residual lifetimes, with respect to the one we propose, is that an adversary which is in charge of solving non-determinism may get information about the future system behavior since the duration of delays is decided a priori.

Our approach has been formalized by starting from the theory of Interactive GSMPs [6]. We first have characterized IGSMPs as a class of transition systems and then we have introduced the class of the Interactive Stochastic Timed Transition Systems (ISTTSs), which are both closed with respect to CSP parallel and hiding operators. Then we have used ISTTSs to define a compositional semantics for IGSMPs which realizes the approach based on spent lifetimes mentioned above.

References

[1] L. Aceto, M. Hennessy, *"Adding Action Refinement to a Finite Process Algebra"*, in Information and Computation 115:179-247, 1994

[2] M. Ajmone Marsan, A. Bianco, L. Ciminiera, R. Sisto, A. Valenzano, *"A LOTOS Extension for the Performance Analysis of Distributed Systems"*, in IEEE/ACM Trans. on Networking 2:151-164, 1994

[3] M. Bernardo, *"Theory and Application of Extended Markovian Process Algebra"*, Ph.D. Thesis, University of Bologna (Italy), 1999

[4] M. Bravetti, A. Aldini, *"Non-determinism in Probabilistic Timed Systems with General Distributions"*, Technical Report UBLCS-2001-08, University of Bologna (Italy), July 2001

[5] M. Bravetti, M. Bernardo, *"Compositional Asymmetric Cooperations for Process Algebras with Probabilities, Priorities, and Time"*, in Proc. of the *1st Int. Workshop on Models for Time-Critical Systems (MTCS 2000)*, ENTCS 39(3), State College (PA), 2000

[6] M. Bravetti, R. Gorrieri, *"The Theory of Interactive Generalized Semi-Markov Processes"*, to appear in Theoretical Computer Science

[7] M. Bravetti, R. Gorrieri, *"Deciding and Axiomatizing Weak ST Bisimulation for a Process Algebra with Recursion and Action Refinement"*, to appear in ACM Transactions on Computational Logic

[8] M. Bravetti, M. Bernardo, R. Gorrieri, *"Towards Performance Evaluation with General Distributions in Process Algebras"*, in Proc. of the *9th Int. Conf. on Concurrency Theory (CONCUR '98)*, LNCS 1466:405-422, 1998

[9] E. Brinksma, J.P. Katoen, R. Langerak, D. Latella, *"A Stochastic Causality-Based Process Algebra"*, in Computer Journal 38:553-565, 1995

[10] P. Buchholz, *"Markovian Process Algebra: Composition and Equivalence"*, in Proc. of the *2nd Int. Workshop on Process Algebra and Performance Modelling (PAPM '94)*, pp. 11-30, Erlangen (Germany), 1994

[11] N. Busi, R.J. van Glabbeek, R. Gorrieri, *"Axiomatizing ST Bisimulation Equivalence"*, in Proc. of the *IFIP Working Conf. on Programming Concepts, Methods and Calculi (PROCOMET '94)*, pp. 169-188, 1994

[12] C.G. Cassandras, *"Discrete Event Systems. Modeling and Performance Analysis"*, Aksen Associates, Irwin, 1993

[13] D.R. Cox, *"The Analysis of non-Markovian Stochastic Processes by the Inclusion of Supplementary Variables"*, in Proc. of the Cambridge Philosophical Society 51:433-440, 1955

[14] P.R. D'Argenio, *"Algebras and Automata for Timed and Stochastic Systems"*, Ph.D. Thesis, Univ. Twente, 1997

[15] R.J. van Glabbeek, F.W. Vaandrager, *"Petri Net Models for Algebraic Theories of Concurrency"*, in Proc. of the *Conf. on Parallel Architectures and Languages Europe (PARLE '87)*, LNCS 259:224-242, 1987

[16] N. Götz, U. Herzog, M. Rettelbach, *"TIPP - A Stochastic Process Algebra"*, in Proc. of the *1st Workshop on Process Algebras and Performance Modelling (PAPM '93)*, pp. 31-36, Edinburgh (UK), 1993

[17] H. Hermanns, *"Interactive Markov Chains"*, Ph.D. Thesis, Univ. Erlangen-Nürnberg, 1998

[18] J. Hillston, *"A Compositional Approach to Performance Modelling"*, Cambridge University Press, 1996

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

[20] K.G. Larsen, A. Skou, *"Bisimulation through Probabilistic Testing"*, in Information and Computation 94:1-28, 1991

[21] K. Matthes, *"Zur Theorie der Bedienungsprozesse"*, in Trans. of the *3rd Prague Conf. on Information Theory, Stat. Dec. Fns. and Random Processes*, pp. 513-528, 1962

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

[23] X. Nicollin, J. Sifakis, S. Yovine, *"Compiling Real-Time Specifications into Extended Automata"*, in IEEE Trans. on Software Engineering, 18(9):794-804, 1992

[24] C. Priami, *"Stochastic π-Calculus with General Distributions"*, in *Proc. of the 4th Workshop on Process Algebras and Performance Modelling (PAPM '96)*, CLUT, pp. 41-57, Torino (Italy), 1996

Towards a Process Algebra for Shared Processors

Mikael Buchholtz, Jacob Andersen, and Hans Henrik Løvengreen

Informatics and Mathematical Modelling
Technical University of Denmark
DK–2800 Lyngby, Denmark

Abstract

We present work-in-progress on a timed process algebra that models sharing of processor resources allowing preemption at arbitrary points of time. This enables us to model both the functional and the timely behaviour of concurrent processes executed on a single processor. Applications of the model for program semantics and kernel development are outlined.

1 Introduction

To argue about correctness of a real-time system both the functional and timely behaviour of the system have to be taken into account. One abstraction of a system is to view it as a number of concurrent processes. In many real systems this abstraction will be realised by executing the processes on a shared processor where execution is switched among the processes. For such a system, the fact that processes have to share the processor cannot be ignored, since it may influence both the timely and the functional behaviour of the system.

Systems of processes sharing a common processor is well described in the theory of scheduling, see e.g. [7]. However, scheduling theory ignores what the processes actually do, so it is not sufficient to describe the functional behaviour of a system.

As an alternative approach, many process algebras have been developed that provide an elegant way to describe both the functional and timely behaviour of a system. However, these process algebras typically assume maximal parallelism, which may be interpreted as if each process had its own dedicated processor. Therefore, they cannot adequately model real-time systems where processes share a processor.

Under the assumption of maximal parallelism, you may say that the processor will always be available to a process. This means that the passing of

This is a preliminary version. The final version is considered for publication in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

time may be described without considering whether the processor is available or not. In many process algebras this leads to modelling time as transitions like $P \xrightarrow{t} P'$, stating that the process P changes into P' while t units of time pass.

To remedy the problem that this model of time cannot be used as a model of systems with a shared processor, we propose a new model. Our simple idea is to say a process may let time pass in two different ways, which gives rise to transitions of the following two types

- $P \xrightarrow{\delta(t)} P'$ stating that t units of time pass, *without* the process using the processor.
- $P \xrightarrow{\rho(t)} P'$ stating that t units of time pass, while the process *does use* the processor.

In this model we are able to express that one process preempts another process currently using the processor. Thus, we are able to describe scheduling of processes in accordance with standard scheduling theory.

In this paper we present work-in-progress on a process algebra, which uses the above model as the underlying model of time. To keep the process algebra simple, we have restricted ourselves to focusing on a simple, but typical case where a number of sequential processes composed in parallel share a single processor. Extending this setup to a more general setting is left for future work.

This paper is organised as follows: First, we present the process algebra, giving its syntax and semantics. Second, we give some examples of applications in which the process algebra has been used. Afterwards, we briefly comment on related works. As this paper presents work-in-progress, we conclude by outlining future work.

2 Syntax and Semantics

In this section we present the process algebra. First, we give the syntax and a short informal semantics. Second, we give a formal operational semantics of the algebra.

The process algebra defines three syntactic categories

Prog of programs executed on a mono processor with T ranging over it.

Par of parallel compositions with A ranging over it.

Proc of process expressions with P ranging over it.

In general, when we define a variable to range over some set we also let any primed or indexed versions of the variables range over the same set. The

syntax of the process algebra is given by the grammar

$$T ::= \langle\langle A \rangle\rangle$$
$$A ::= A_1 \parallel A_2 \mid P$$
$$P ::= e.P \mid P_1 \,[]\, P_2 \mid \text{rec } X : P \mid X \mid \text{STOP}$$

An entire program is placed in a separate syntactic category and is also characterised syntactically by double angled brackets. This allows us to argue about an entire program separately. $A_1 \parallel A_2$ is parallel composition of the two constituents which will share a common processor.

$e.P$ is prefix of an event e to the process P. Events belong to the set $E = AE \cup TE \cup CE$ – a set of actions, a set of timed events, and a set of communication events. Actions are events that take no time and which a process may engage in without interaction from other processes. We let α range over AE. In this paper, the semantics of actions is not further defined.

We use the positive real numbers, $\mathbb{R}_+$, to model time and let t range over this set. The timed events belong to the set $TE = \{\delta(t), \rho(t) \mid t \in \mathbb{R}_+\}$. The timed event $\delta(t)$ means that the process *delays* for t time units, without requesting processor time. $\rho(t)$ means that the process *requests* t units of processor time in order to continue.

Intuitively, communication between processes takes place on channels from the set $Chan$ with c ranging over it. The set of communication events is $CE = \{c!, c? \mid c \in Chan\}$. Two processes may synchronise by performing the events $c!$ and $c?$, respectively, on the same channel c.

$P_1 \,[]\, P_2$ is non-deterministic choice between P_1 and P_2. Standard recursion is written rec $X : P$ where X is a process variable from some set $PVar$. We require that a process variable may only appear inside a recursive process, where it has been defined. STOP means that the process has terminated.

We give the process algebra an operational semantics that defines a labelled transition system $\langle S, L, \longrightarrow \rangle$. A labelled transition system is a three-tuple consisting of a set of states S, a set of labels L with l ranging over it, and a transition relation $\longrightarrow \subseteq S \times L \times S$, which describes possible transitions between states.

We give the semantics for all three syntactic categories of the process algebra by one common labelled transition system. Hence, the states of this labelled transition system will be a subset of $Prog \cup Par \cup Proc$.

The labels are given as $L = \{\tau\} \cup TL \cup CL$. All transitions describing actions will be labelled with the label τ. The set TL contains the timed labels, which reuse the names of the timed events, i.e. $TL = \{\delta(t), \rho(t) \mid t \in \mathbb{R}_+\}$. Transitions labelled $\delta(t)$ means that t time units passes without using the processor while a label $\rho(t)$ means that the processor is in use. Communication labels are taken from the set $CL = \{c!, c?, c \mid c \in Chan\}$ with cl ranging over it. The labels $c!$ and $c?$ are used for requests for communication while the label c is used when the communication is performed.

The transition relation is defined by the smallest relation satisfying the inference rules given in the following.

Prefix:

$$\alpha.P \xrightarrow{\tau} P$$

$$\delta(t).P \xrightarrow{\delta(t)} P \qquad \delta(t).P \xrightarrow{\delta(t')} \delta(t-t').P \quad \textbf{if } 0 < t' < t$$

$$\rho(t).P \xrightarrow{\rho(t)} P \qquad \rho(t).P \xrightarrow{\rho(t')} \rho(t-t').P \quad \textbf{if } 0 < t' < t$$

$$\rho(t).P \xrightarrow{\delta(t')} \rho(t).P$$

$$c!.P \xrightarrow{c!} P \qquad c!.P \xrightarrow{\delta(t)} c!.P$$

$$c?.P \xrightarrow{c?} P \qquad c?.P \xrightarrow{\delta(t)} c?.P$$

Actions α take no time. They are internal to the process and may be seen as abstractions of what the process actually does. A delay event $\delta(t)$ may be performed in one step, where all the time passes at once, or it may be divided into several transitions. The request for processor time has the same properties. Furthermore, an extra inference rule is added to describe what happens if the process is preempted. In this case the process will let time pass, but its request for processor time is unchanged. Note that there is a difference between the intuitive understanding of the event $\rho(t)$ and the label $\rho(t)$. The event means that the process requests t units of processor time, while a transition labelled $\rho(t)$ means that the process actually gets t units of processor time.

Communication can either be performed instantaneously or postponed without using the processor. When processes are joined in a program, as we will see shortly, communication is made urgent.

Choice:

$$\frac{P_1 \xrightarrow{\iota} P_1'}{P_1 \,[]\, P_2 \xrightarrow{\iota} P_1'} \qquad \frac{P_2 \xrightarrow{\iota} P_2'}{P_1 \,[]\, P_2 \xrightarrow{\iota} P_2'}$$

$$\frac{P_1 \xrightarrow{\delta(t)} P_1' \quad P_2 \xrightarrow{\delta(t)} P_2'}{P_1 \,[]\, P_2 \xrightarrow{\delta(t)} P_1' \,[]\, P_2'}$$

In the rules for choice we use a set of initial labels $\{\tau\} \cup CL \cup \{\rho(t) \mid t \in \mathbb{R}_+\}$, ranged over by ι. If either of the processes in the choice can perform an initial event, the whole construct proceeds as that process. Otherwise both processes must be able to delay and the choice will be postponed.

Recursion:

$$\frac{P[\text{rec } X : P/X] \xrightarrow{l} P'}{\text{rec } X : P \xrightarrow{l} P'}$$

Recursion is done in the standard way, where $P[P_1/X]$ is the process P with

any free occurrences of the process variable X substituted with the process P_1.

Stop:

$$\mathrm{STOP} \xrightarrow{\delta(t)} \mathrm{STOP}$$

Even a terminated process cannot prevent time from passing.

Parallel composition:

$$\frac{A_1 \xrightarrow{\tau} A_1'}{A_1 \parallel A_2 \xrightarrow{\tau} A_1' \parallel A_2} \qquad \frac{A_2 \xrightarrow{\tau} A_2'}{A_1 \parallel A_2 \xrightarrow{\tau} A_1 \parallel A_2'}$$

$$\frac{A_1 \xrightarrow{\rho(t)} A_1' \quad A_2 \xrightarrow{\delta(t)} A_2'}{A_1 \parallel A_2 \xrightarrow{\rho(t)} A_1' \parallel A_2'} \qquad \frac{A_1 \xrightarrow{\delta(t)} A_1' \quad A_2 \xrightarrow{\rho(t)} A_2'}{A_1 \parallel A_2 \xrightarrow{\rho(t)} A_1' \parallel A_2'}$$

$$\frac{A_1 \xrightarrow{\delta(t)} A_1' \quad A_2 \xrightarrow{\delta(t)} A_2'}{A_1 \parallel A_2 \xrightarrow{\delta(t)} A_1' \parallel A_2'}$$

$$\frac{A_1 \xrightarrow{c!} A_1' \quad A_2 \xrightarrow{c?} A_2'}{A_1 \parallel A_2 \xrightarrow{c} A_1' \parallel A_2'} \qquad \frac{A_1 \xrightarrow{c?} A_1' \quad A_2 \xrightarrow{c!} A_2'}{A_1 \parallel A_2 \xrightarrow{c} A_1' \parallel A_2'}$$

$$\frac{A_1 \xrightarrow{cl} A_1'}{A_1 \parallel A_2 \xrightarrow{cl} A_1' \parallel A_2} \qquad \frac{A_2 \xrightarrow{cl} A_2'}{A_1 \parallel A_2 \xrightarrow{cl} A_1 \parallel A_2'}$$

For time to pass in the parallel composition it is required that both of its constituents are able to let it do so. Furthermore, the rules for parallel composition ensure that at most one process is allowed to use the processor.

Program:

$$\frac{A \xrightarrow{\tau} A'}{\langle\!\langle A \rangle\!\rangle \xrightarrow{\tau} \langle\!\langle A' \rangle\!\rangle} \qquad \frac{A \xrightarrow{c} A'}{\langle\!\langle A \rangle\!\rangle \xrightarrow{c} \langle\!\langle A' \rangle\!\rangle} \qquad \frac{A \xrightarrow{\rho(t)} A' \quad \forall c \bullet A \overset{c}{\not\longrightarrow}}{\langle\!\langle A \rangle\!\rangle \xrightarrow{\rho(t)} \langle\!\langle A' \rangle\!\rangle}$$

$$\frac{A \xrightarrow{\delta(t)} A' \quad \forall c \bullet A \overset{c}{\not\longrightarrow} \quad \forall t' \bullet A \overset{\rho(t')}{\not\longrightarrow}}{\langle\!\langle A \rangle\!\rangle \xrightarrow{\delta(t)} \langle\!\langle A' \rangle\!\rangle}$$

At the program level urgency of communication is ensured. That is, if a communication is possible, it will be performed before time can pass. Furthermore, the construct enforces the scheduling decision that the processor may not be idle if any process is able to run.

3 Applications

The process algebra has been developed while working on formal development of a real-time kernel for a shared processor [1]. In this section we give examples of a number of applications for which our time model and the process algebra have been used during this work.

3.1 Scheduling

The process algebra, as presented in section 2, allows any process to preempt another process. The process algebra specifies that preemption is possible at any given time i.e. may happen with an arbitrary fine granularity. However, such a behaviour will not occur in a real system. Instead, processes will be scheduled according to some scheduling strategy. In the process algebra, we will enforce scheduling strategies by giving restrictions on how a process is allowed to preempt another process.

We aim for the scheduling decisions to be made only at one specific place, thus making it easier to implement different strategies. This will be done at the outermost level, i.e. in the semantics of the program construct $\langle\langle A \rangle\rangle$. In order to make scheduling decisions in this construct it will be necessary to know, which processes that are currently requesting the processor. We add this information in the timed labels $\rho(t)$, by including an identification of the process, which uses the processor. Each sequential process will be given a name from a set of process names, *PName*, with p ranging over it. Accordingly, we change the labels for use of processor time to $\rho_p(t)$, stating that the process named p gets t units of processor time. Consequently, we must change the inference rules of the semantics, which involve transitions with the label $\rho(t)$.

If a process named p includes a prefix of the timed event $\rho(t)$ we now use the rules

$$\rho(t).P \xrightarrow{\rho_p(t)} P \qquad \rho(t).P \xrightarrow{\rho_p(t')} \rho(t-t').P \quad \textbf{if } 0 < t' < t$$
$$\rho(t).P \xrightarrow{\delta(t')} \rho(t).P$$

To include process names in the labels in parallel composition, we only need to change two rules

$$\frac{A_1 \xrightarrow{\rho_p(t)} A_1' \quad A_2 \xrightarrow{\delta(t)} A_2'}{A_1 \parallel A_2 \xrightarrow{\rho_p(t)} A_1' \parallel A_2'} \qquad \frac{A_1 \xrightarrow{\delta(t)} A_1' \quad A_2 \xrightarrow{\rho_p(t)} A_2'}{A_1 \parallel A_2 \xrightarrow{\rho_p(t)} A_1' \parallel A_2'}$$

As a first example of how to enforce scheduling in the semantics of the program construct, we regard strict priority based scheduling – meaning that a process may only run if no process with a higher priority wants to run. Each process is assigned a static priority (a natural number) by the function $prio : PName \to \mathbb{N}$. The rules for a program, which is executed using priority

based scheduling may then be given as

$$\frac{A \xrightarrow{\tau} A'}{\langle\langle A\rangle\rangle \xrightarrow{\tau} \langle\langle A'\rangle\rangle} \qquad \frac{A \xrightarrow{c} A'}{\langle\langle A\rangle\rangle \xrightarrow{c} \langle\langle A'\rangle\rangle}$$

$$\frac{A \xrightarrow{\delta(t)} A' \quad \forall c \bullet A \overset{c}{\not\rightarrow} \quad \forall p', t' \bullet A \overset{\rho_{p'}(t')}{\not\longrightarrow}}{\langle\langle A\rangle\rangle \xrightarrow{\delta(t)} \langle\langle A'\rangle\rangle}$$

$$\frac{A \xrightarrow{\rho_p(t)} A' \quad \forall c \bullet A \overset{c}{\not\rightarrow} \quad \forall p', t' \bullet prio(p') > prio(p) \Rightarrow A \overset{\rho_{p'}(t')}{\not\longrightarrow}}{\langle\langle A\rangle\rangle \xrightarrow{\rho_p(t)} \langle\langle A'\rangle\rangle}$$

We only pose restriction on which processes are allowed to use the processor, letting action and communication be performed by any process at any time.

In the same way as the rules for priority based scheduling we may give different sets of rules to enforce other scheduling strategies. In many scheduling strategies, however, it is necessary to add a scheduler state. We may incorporate such a state by adding it to the transition system at the program level. That is, if Sch is the type of scheduler state, then the type of states in the transition system will become $(Prog \times Sch) \cup Par \cup Proc$.

As an example we look at round robin scheduling where processes are scheduled in some fixed order. Each process will get at most a fixed quantum Q of processor time before it is preempted by the process next in line.

In the specification of round robin scheduling the scheduler state contains information on which process is the current process and on the time left of the current quantum i.e. we set $Sch = PName \times \mathbb{R}$.

$$\frac{A \xrightarrow{\tau} A'}{\langle\langle A\rangle\rangle, \langle p, q\rangle\rangle \xrightarrow{\tau} \langle\langle A'\rangle\rangle, \langle p, q\rangle\rangle} \qquad \frac{A \xrightarrow{c} A'}{\langle\langle A\rangle\rangle, \langle p, q\rangle\rangle \xrightarrow{c} \langle\langle A'\rangle\rangle, \langle p, q\rangle\rangle}$$

$$\frac{A \xrightarrow{\delta(t)} A' \quad \forall c \bullet A \overset{c}{\not\rightarrow} \quad \forall p', t' \bullet A \overset{\rho_{p'}(t')}{\not\longrightarrow}}{\langle\langle A\rangle\rangle, \langle p, q\rangle\rangle \xrightarrow{\delta(t)} \langle\langle A'\rangle\rangle, \langle p, q\rangle\rangle}$$

$$\frac{A \xrightarrow{\rho_p(t)} A' \quad \forall c \bullet A \overset{c}{\not\rightarrow}}{\langle\langle A\rangle\rangle, \langle p, q\rangle\rangle \xrightarrow{\rho_p(t)} \langle\langle A'\rangle\rangle, \langle p, q - t\rangle\rangle} \quad \textbf{if } t \leq q$$

$$\langle\langle A\rangle\rangle, \langle p, 0\rangle\rangle \xrightarrow{\tau} \langle\langle A\rangle\rangle, \langle next(p), Q\rangle\rangle$$

$$\frac{\forall t \bullet A \overset{\rho_p(t)}{\not\longrightarrow} \quad A \overset{\tau}{\not\rightarrow} \quad \forall c \bullet A \overset{c}{\not\rightarrow} \quad A \xrightarrow{\rho_{p'}(t')} A'}{\langle\langle A\rangle\rangle, \langle p, q\rangle\rangle \xrightarrow{\tau} \langle\langle A\rangle\rangle, \langle next(p), Q\rangle\rangle} \quad \textbf{if } p' \neq p$$

Only the current process is allowed to use the processor. We assume that the order in which the processes must be scheduled is given by function $next : PName \to PName$, such that $next(p)$ gives the process which must be granted the processor after the process p. We switch current process in one of two cases, which are specified in the last two rules. In the first case the current processes has used its entire quantum, while in the second case the current process does not request the processor. In the later case we additionally require that some process actually requests processor time. In doing so, we prevent the occurrence of an infinite sequence of (pointless) process switching steps.

3.2 Semantics of a Programming Language

We have used the process algebra to give semantics to a small programming language, which resembles Hoare's CSP [5] or Occam [3]. Programs of the language are from the syntactic category *RProg* with R ranging over it and consist of a fixed number of sequential processes in parallel. Each process contains one statement S from the syntactic category *Stm*. Programs are built from the grammar

$$R ::= R_1 \parallel R_2 \mid S$$

$$S ::= \texttt{delay}\ t \mid c\texttt{?} \mid c\texttt{!} \mid S_1\,;S_2 \mid \begin{array}{l}\texttt{[}\ \ c_1\texttt{?} \to S_1 \\ \texttt{[]}\ c_2\texttt{?} \to S_2\ \texttt{]}\end{array} \mid \begin{array}{l}\texttt{[}\ \ c\texttt{?} \to S_1 \\ \texttt{[]}\ \texttt{delay}\ t \to S_2\ \texttt{]}\end{array}$$

Statements are (possibly a sequential composition of) delay, synchronous communication, guarded alternative, or communication with time-out.

The language is a "real" programming language, so it does not explicitly describe how processor time is required to execute different language constructs. In order to give semantics to the language we give a translation from programs of the language into programs of the process algebra. This translation may be done on the structure of the language, translating each language construct separately.

To translate an entire program we use the function $\mathcal{T} : RProg \to Prog$ while the function $\mathcal{TR} : RProg \to Par$ translates parallel composition. These two translations are straight forward mapping directly between corresponding constructs. More interesting is the translation of statements where we use the continuation style function $\mathcal{TS} : Stm \to Proc \to Proc$. For the translation of statements below STOP is passed as continuation of an entire statement to describe that the statement terminates at the end.

$$\begin{aligned}\mathcal{T}[\![R]\!] &\stackrel{def}{=} \langle\!\langle \mathcal{TR}[\![R]\!] \rangle\!\rangle \\ \mathcal{TR}[\![R_1 \parallel R_2]\!] &\stackrel{def}{=} \mathcal{TR}[\![R_1]\!] \parallel \mathcal{TR}[\![R_2]\!] \\ \mathcal{TR}[\![S]\!] &\stackrel{def}{=} \mathcal{TS}[\![S]\!]\mathrm{STOP}\end{aligned}$$

As the first example of the translation of statements, regard the construct $\texttt{delay}\ t$, which delays for at least t time units. In the definition of the translation function the argument P is the process algebra expression, which describes

what happens after the execution of `delay` t.

$$\mathcal{TS}[\![\texttt{delay}\ t]\!]P \stackrel{def}{=} \rho(T_{delay}).\delta(t - T_{delay}).P$$

The translation gives that the construct first uses T_{delay} units of processor time to execute the statement and figure out for how long it has to delay. Afterwards the statement delays for $t - T_{delay}$ time units.

Communication in the language is synchronous just as it is in the process algebra. Thus, the translation becomes quite simple.

$$\mathcal{TS}[\![c?]\!]P \stackrel{def}{=} \rho(T_{rcom}).c?.P$$
$$\mathcal{TS}[\![c!]\!]P \stackrel{def}{=} \rho(T_{rcom}).c!.P$$

First, T_{rcom} units of processor time is used to get ready to perform communication. Second, communication may itself be performed in the same manner as for the process algebra.

Sequential composition is done in the standard way for continuation style semantics

$$\mathcal{TS}[\![S_1\,;S_2]\!]P \stackrel{def}{=} \mathcal{TS}[\![S_1]\!]\mathcal{TS}[\![S_2]\!]P$$

The (binary) alternative statement using communication guards should proceed as the statement following the guard if communication is possible.

$$\mathcal{TS}\left[\!\!\left[\begin{array}{l}\texttt{[}\ c_1\texttt{?} \rightarrow S_1 \\ \texttt{[]}\ c_2\texttt{?} \rightarrow S_2\ \texttt{]}\end{array}\right]\!\!\right] P \stackrel{def}{=} \rho(2 \cdot T_{rcom}).\left(\begin{array}{c}c_1?.\mathcal{TS}[\![S_1]\!]P \\ [\,]\ c_2?.\mathcal{TS}[\![S_2]\!]P\end{array}\right)$$

First, the processor is used for $2 \cdot T_{rcom}$ units of time. This processor time is used to decide between which channels the choice will be performed. Afterwards the actual choice can take place. The translation becomes relatively simple, since the choice operator of the process algebra has exactly the semantics we desire for the selection between communication guards.

As a final example, we look at communication with time-out. The statement proceeds as S_1 if communication is possible within t time units. Otherwise, a time-out occurs and the statement proceeds as S_2.

$$\mathcal{TS}\left[\!\!\left[\begin{array}{l}\texttt{[}\ c\texttt{?} \rightarrow S_1 \\ \texttt{[]}\ \texttt{delay}\ t \rightarrow S_2\ \texttt{]}\end{array}\right]\!\!\right] P \stackrel{def}{=} \rho(T_{rcom} + T_{rdelay}).\left(\begin{array}{c}c?.\mathcal{TS}[\![S_1]\!]P \\ [\,]\ \delta(t).\tau.\mathcal{TS}[\![S_2]\!]P\end{array}\right)$$

Note that if communication on the channel c is not ready, the choice may be postponed for up to t time units before a special time-out action τ can occur. Since actions are urgent, the choice will be made after at most t time units.

We see from the examples of the translation that it provides a clear overview of timings of the different statements. For example, use of processor time may easily be added as the timed event $\rho(t)$. In [1] we have used the technique described here for a larger language, which contains amongst other things recursion and alternative with multiple branches. The translations required for these new language construct follow the ideas described above closely and introduces no substantial novelties.

3.3 Development of a Real-Time Kernel

In [1] we have used the ideas for kernel development from the ProCoS project [8] to develop a real-time kernel for use in small embedded mono processor systems. In this, we have used the model of time presented in this paper to describe how processes share the processor.

As depicted in Figure 1, the development of the kernel comprises three development levels, each of which describes the embedded system at a certain level of abstraction. At the topmost level, the Programming Language Level, the system is regarded as a number of concurrent processes described by a program in a CSP-like language. At the middle level, the Machine Language Level, the system is still regarded as a number of concurrent processes. However, each process is given in an assembler-like language with instructions consisting of an op-code and arguments. Each processes is executed at its own virtual machine so management of e.g. a program counter, stack, and store can be described in detail. At the lowest level, the Kernel Level, the system is described as one virtual machine which executes all the processes, explicitly switching execution among them. In this description there is a clear distinction of which parts of the virtual machine that describe process behaviour and which parts that are not relate directly to a process i.e. the parts that must be handled by the kernel.

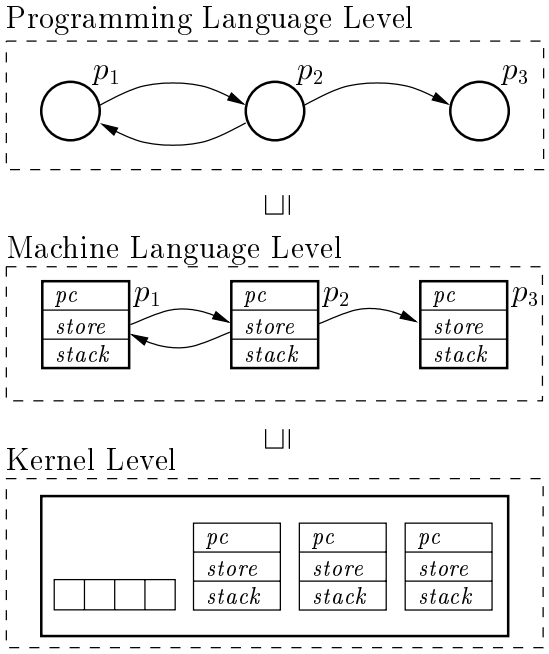

Fig. 1. Overview of kernel development levels for a system with three processes.

The behaviour of the system at each level is described by a labelled transition system. For the topmost level this transition system is given by an operational semantics of the CSP-like language according to the ideas described in section 3.2. At the other levels the transition systems are also defined by

a number of inference rules. At all levels, we use the same model of time, thus having transition systems with actions and timed labels as described for the process algebra. In the development of the kernel we have dealt with at great number of aspects, which are outside the scope of this paper. These include: assignment of variables with evaluation of expressions, communication with value passing, upper and lower bounds on execution times, an external interface using shared memory, kernel overhead, and timer interrupts.

Correctness of the system is based on a binary implementation relation, $\sqsubseteq$, between the transition systems describing two consecutive development levels. The implementation relation allows for a lower level to remove any nondeterminism of an upper level. Except for this difference between their behaviours, the lower level *must* behave in the same way as the upper level.

4 Related Work

Related work is to be found in concurrency models that simultaneously address the issues of time, resource sharing, and priorities. Within process algebra, a number of approaches deal with time and priorities (see e.g. [4]). However, the only process algebraic approach that addresses all three issues seems to be [2]. In this algebra, the usage of a set A of resources for a period of t time units is denoted by a timed event A^t and its behaviour given by the rule

$$A^t.P \xrightarrow{A^{t'}} A^{t-t'}.P \quad \textbf{if } 0 < t' \leq t$$

similar to one of our $\rho(t)$ rules. However, they do not provide any means for allowing a partially completed resource usage to be temporarily preempted by another process. Their model, therefore, seems adequate only for *non-preemptive* scheduling strategies, cf. the classical job-shop scheduling problems. The novelty of our approach thus basically lies in the rule

$$\rho(t).P \xrightarrow{\delta(t')} \rho(t).P$$

allowing for the arbitrary preemption found in real-time operating systems.

Our work has been inspired by the approach of [9] that distinguishes between *time passing* and (processor) *time consumption*. Their underlying behaviour model, however, does not have a notion of instantaneous events and they use logical characterisation of models rather than taking a constructive approach. The wealth of results and techniques developed for models based on transition systems, therefore, cannot immediately be related to this work.

5 Future Work

In section 3.1 we have described how a specific scheduling strategy may be expressed by modification of the transition rules. A more general approach would be to specify the scheduling at the level of the algebra as a kind of plug-in component. However, we have encountered problems in giving a unified

representation that will work for a broad range of schedulers. For example, we should cater for a scheduler, which itself uses processor time. We are currently looking at different ways of integrating scheduling into the process algebra.

Also, the model should be generalised to handle multi-processors and perhaps other preemptable resources.

A large task lies in describing the theoretical implications of the process algebra. Some of this ground has already been covered in our kernel development. However, the traditional bisimulation equivalence has shown to be too strong for this application. Rather, we have focused on implementation relations that allow an implementation to behave more deterministically than its specification. These relations also need further studies.

For instance, in the process algebra presented here, timed events may only be used to give precise timings. For real systems, however, exact timings of all operations are not always known. Furthermore, different knowledge of timings may be available at the different levels in the development. In our kernel work, we have solved these problem by introducing upper and lower bounds on timed events. This gives rise to events such as $\rho(t_l, t_u)$, stating that a process requires a non-deterministic amount of processor time between t_l and t_u time units. The implementation relation should then allow for an implementation to remove or reduce this non-determinism of timing.

A major concern of the implementation relation is that it should be *compositional*, i.e. distribute over parallel composition. However, ensuring this has proven to be a non-trivial task due to the non-monotonic nature of resource sharing and urgency.

One of the goals of our work is to link resource-aware models like the one presented here, with standard timed models (e.g. timed automata) making them amenable for analysis by tools such as Uppaal [6] and others. One way to accomplish this would be to derive a timed automaton from a processor constrained transition system expressed in our model.

References

[1] Andersen, J. and M. Buchholtz, "Formal Development of a Real-Time Kernel," Master's thesis, Department of Information Technology, Technical University of Denmark (2001).
URL `http://www.imm.dtu.dk/people/mib/masters/`

[2] Brémond-Grégoire, P. and I. Lee, *A process algebra of communicating shared resources with dense time and priorities*, Theoretical Computer Science (1997), pp. 179–219.

[3] Burns, A., "Programming in Occam-2," Addison-Wesley, 1988.

[4] Cleaveland, R., G. Löttgen and V. Natarajan, *Priority in process algebra*, in:

J. Bergstra, A. Ponse and S. Smolka, editors, *Handbook of Process Algebra*, North-Holland, 2001 pp. 711–765.

[5] Hoare, C. A. R., *Communicating sequential processes*, Communications of the ACM **21** (1978), pp. 666–677.

[6] Larsen, K. G., P. Pettersson and W. Yi, Uppaal *in a Nutshell*, Int. Journal on Software Tools for Technology Transfer **1** (1997), pp. 134–152.

[7] Liu, J. W. S., "Real-Time Systems," Prentice Hall, 2000.

[8] Søgaard-Andersen, J. F., C. Ø. Rump and H. H. Løvengreen, *A systematic kernel development*, SIGSOFT Software Engineering Notes **16** (1991), pp. 55–65.

[9] Zhou Chaochen, M. R. Hansen, A. P. Ravn and H. Rischel, *Duration specifications for shared processors*, in: *Formal Techniques in Real-Time and Fault-Tolerant Systems, Second International Symposium Proceedings*, 1991, pp. 21–32.

Privacy in Real-Time Systems

Ruggero Lanotte [1,2]

Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy

Andrea Maggiolo-Schettini [1,3]

Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy

Simone Tini [1,4]

Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy

Abstract

We study the problem of privacy in the framework of Timed Automata. By distinguishing between secret and observable actions we formulate a property of no-privacy in terms of a property of the language accepted by a Timed Automaton, and we give an algorithm checking such property.

1 Introduction

One of the main requirements of mobile code is that it must guarantee some kind of security to clients executing it. One of the security requirements is the client's *privacy*, namely that executing mobile code does not imply leaking of private information.

Several papers (see, among the others, [3,4,5,6,7]) dealing with privacy, consider *two-level* systems, where the *high level* (or *secret*) behavior is distinguished from the *low level* (or *observable*) one. In the mentioned papers, systems respect the property of privacy if there is no information flow from

[1] Research partially supported by MURST Progetto Cofinanziato TOSCA.

[2] Email: lanotte@di.unipi.it

[3] Email: maggiolo@di.unipi.it

[4] Email: tini@di.unipi.it

This is a preliminary version. The final version is considered for publication in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

the high level to the low level. This means that the secret behavior cannot influence the observable one, or, equivalently, no information on the observable behavior permits to infer information on the secret one.

Our aim is to study the problem of privacy in real-time systems in the framework of Timed Automata [1]. When using this formalism, the possible behaviors of a system are described by a set of infinite *timed words*, namely infinite sequences of pairs (action performed, time of firing). In describing two-level systems, we distinguish between high-level and low-level actions. We formulate a *no-privacy property* as follows: if, whenever one can observe a given timed sequence of observable actions, one is sure that the system performs a certain secret action, then the system is insecure. The reason is that one can infer information on the secret behavior from the observation of the observable one.

We give an algorithm that exploits the *region graph* obtained from a Timed Automaton and checks the no-privacy property for a given sequence of observable actions and a given secret action.

2 HL Timed Automata

In this section we introduce the formalism of HL Timed Automata, as an extension of Alur and Dill's Timed Automata.

2.1 Security alphabet and timed words

A *security alphabet* is a pair consisting of two disjoint finite sets of *actions* (L, H). The set L contains the *low actions*, which can be performed by the system and can be observed by the external environment, and the set H contains the *high actions*, which can be performed by the system and are visible only inside the system.

Given any *time domain* T (non-negative rational numbers, or non-negative real numbers, as examples), a *timed word* ω on (L, H) and T is a pair of functions (ω_1, ω_2) such that $\omega_1 : \mathbb{N} \to (L \cup H)$ and $\omega_2 : \mathbb{N} \to T$. Intuitively, ω describes the behavior of a system that performs action $\omega_1(i)$ at time $\sum_{h=0}^{i} \omega_2(h)$. A timed word must satisfy the *time progress property*, namely for each time value $t \in T$, there is some index i such that $\sum_{h=0}^{i} \omega_2(h) > t$.

Given a timed word $\omega = (\omega_1, \omega_2)$, let us denote with ω_L the projection of ω on L, namely the (possibly finite) sequence $(\omega_1(i_1), \omega_2(i_1)), (\omega_1(i_2), \omega_2(i_2)), \ldots$ such that, for each index i_j, $\omega_1(i_j) \in L$ and, for each $i_j < k < i_{j+1}$, $\omega_1(k) \in H$. The sequence ω_L describes the part of ω that can be observed by the external environment.
Let us denote with F_ω the function that gives the index in ω of the low action in position j in ω_L, namely $F_\omega(j) = i_j$.

2.2 Clock valuations and clock constraints

We assume a set X of variables measuring time, called *clocks*. Intuitively, clocks increase uniformly with time when an automaton is in whatsoever state.

A *clock valuation* over a set of clocks X is a mapping $v : X \to T$ assigning time values to clocks. For a clock valuation v and a time value t, let $v + t$ denote the clock valuation such that $(v+t)(x) = v(x)+t$. For a clock valuation v and a subset of clocks $Y \subseteq X$, let $v[Y]$ denote the clock valuation such that $v[Y](x) = 0$, if $x \in Y$, and $v[Y](x) = v(x)$, otherwise.

Given a set of clocks X, we consider the set of *clock constraints* over X, denoted $\Phi(X)$, which is defined by the following grammar, where ϕ ranges over $\Phi(X)$, $x \in X$, $c \in T$ and $\# \in \{<, \leq, =, \neq, >, \geq\}$:

$$\phi ::= x \# c \mid \phi \wedge \phi \mid \neg\phi \mid \phi \vee \phi \mid true\,.$$

We write $v \models \phi$ when *the clock valuation v satisfies the clock constraint ϕ*. More precisely, $v \models x \# c$ iff $v(x) \# c$, $v \models \phi_1 \wedge \phi_2$ iff both $v \models \phi_1$ and $v \models \phi_2$, $v \models \phi_1 \vee \phi_2$ iff either $v \models \phi_1$ or $v \models \phi_2$, $v \models \neg\phi$ iff $v \not\models \phi$, and $v \models true$.

2.3 The formalism

Definition 2.1 Given a security alphabet (H, L), a *HL Timed Automaton* (TA_{HL}) is a tuple $\mathcal{A} = ((L, H), A_1, \ldots, A_m)$, where, for each $1 \leq i \leq m$, $A_i = (Q_i, q_i^0, \delta_i, X_i)$ is a *sequential automaton*, with:

- a finite set of *states* Q_i
- an *initial state* $q_i^0 \in Q_i$
- a set of *clocks* X_i
- a set of *transitions* $\delta_i \subseteq Q_i \times \Phi(X_i) \times (L \cup H) \times 2^{X_i} \times Q_i$.

The sets of clocks $X_1, \ldots, X_m$ are pairwise disjoint.

Intuitively, a transition (q, ϕ, a, Y, q') of an automaton A_i fires in correspondence with the performance of action a when state q is active and the clock valuation of A_i satisfies the clock constraint ϕ. In such a case, state q' is entered and the clocks in Y are reset.

Let us describe now the behavior of a TA_{HL} $\mathcal{A} = ((L, H), A_1, \ldots, A_m)$.

A *configuration* of $\mathcal{A}$ is a tuple $s = ((q_1, v_1), \ldots, (q_m, v_m))$ such that, for each $1 \leq i \leq m$, q_i is a state in Q_i and v_i is a clock valuation over the set of clocks X_i.
The *initial configuration* s_0 is the tuple $((q_1^0, v_1^0), \ldots, (q_m^0, v_m^0))$, with q_i^0 the initial state of A_i and with v_i^0 the clock valuation such that $v_i(x)^0 = 0$ for each clock $x \in X_i$.

There is a *step* from configuration $s = ((q_1, v_1), \ldots, (q_m, v_m))$ to configuration $s' = ((q_1', v_1'), \ldots, (q_m', v_m'))$ through action a at time t, written $s \to_t^a s'$, if and only if, for each $1 \leq i \leq m$, there is a transition $(q_i, \phi_i, a, Y_i, q_i') \in \delta_i$ such

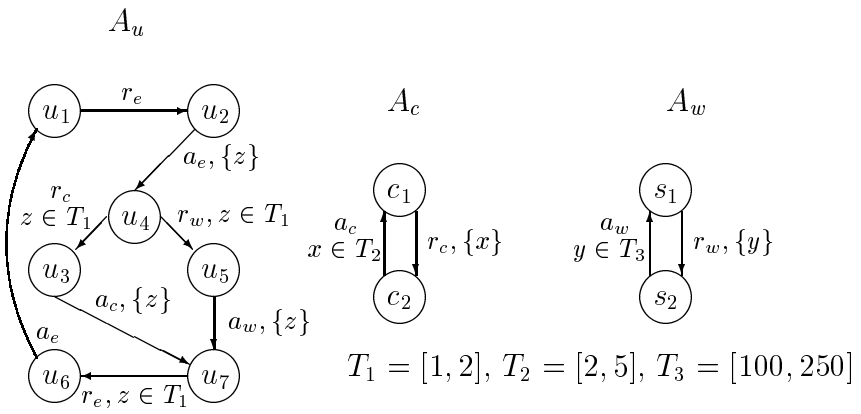

Fig. 1. The web system.

that $(v_i + t) \models \phi_i$ and $v'_i = (v_i + t)[Y_i]$. Intuitively, each clock constraint ϕ_i is satisfied by the clock valuation $v_i + t$ and all clocks in Y_i are reset.

A timed word (ω_1, ω_2) is *accepted* by $\mathcal{A}$ if there exists a infinite sequence of steps $s_0 \rightarrow^{\omega_1(0)}_{\omega_2(0)} s_1 \rightarrow^{\omega_1(1)}_{\omega_2(1)} \ldots$ from the initial configuration s_0.

The *language* accepted by $\mathcal{A}$ (denoted by $\mathcal{L}(\mathcal{A})$) is the set of timed words accepted by $\mathcal{A}$.

By application of a cartesian product construction, any TA_{HL} can be transformed into an equivalent (namely, accepting the same language) TA_{HL} consisting of only one sequential component, namely into an Alur and Dill's Timed Automaton.

Proposition 2.2 *For any TA_{HL} $\mathcal{A}$ there exists a TA_{HL} $\mathcal{A}'$ composed by one only sequential automaton such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

3 The No-privacy Property

Given sequences d and d', let $d \leq_P d'$ denote the fact that d is a prefix of d'.

Let $a \in H$, d be a finite sequence $(a_1, t_1), \ldots, (a_h, t_h)$ with $a_1, \ldots, a_h \in L$ and $t_1, \ldots, t_h \in T$, and i be an index $1 \leq i < h$. We define the $no - privacy$ property $NPr(d, i, a)$ for a TA_{HL} $\mathcal{A}$ as follows:

for each $\omega \in \mathcal{L}(\mathcal{A}), d \leq_P \omega_L$ implies $a \in \{\omega_1(F_\omega(i)+1), \ldots, \omega_1(F_\omega(i+1)-1)\}$.

Intuitively, $NPr(d, i, a)$ expresses that, whenever the sequence d of low symbols is read, the high symbol a is read between the low level actions a_i and a_{i+1}, and, therefore, there is an information flow from high level to low level, namely information on the secret behavior can be inferred from information on the observable behavior.

Example 3.1 We model the time attack on web privacy described in [2]. The attack compromises the privacy of user's web-browsing histories by allowing

a malicious web site to determine whether or not the user has recently visited some other, unrelated, web page w. A Java applet is embedded in the malicious web site and is run by the user's browser. The applet first performs a request to a file of w, and then performs a new request to the malicious site. So, the malicious site can measure the time elapsed between the two requests which it receives from the user, and, if such a time is under a certain bound, it infers that w was in the cache of the browser of the user, thus implying that w has been recently visited by the user.

In Fig. 1 we model this problem. Automaton A_c represents the cache. The time elapsed between a request r_c and an answer a_c is in the interval $[2, 5]$. Automaton A_w represents the site w. The time elapsed between a request r_w and an answer a_w is in the interval $[100, 250]$. The automaton A_u represents the requests by the user that downloads the page of the malicious site. First of all, it performs a request r_e to the malicious site. Then, when it receives the answer a_e, it performs a communication either with the cache or with the site w in a time belonging to the interval $[1, 2]$. Finally, it performs another request r_e and it waits for an answer from the malicious site. We assume that there are transitions from state c_1 to state c_1 labeled with symbols r_e, a_e, r_w and a_w. Analogously there are transitions from state s_1 to state s_1 labeled with symbols r_e, a_e, r_c and a_c.

The only visible actions for the malicious site are r_e and a_e, so the alphabet (L, H) is $(\{r_e, a_e\}, \{r_c, a_c, r_w, a_w\})$.

Now, if we consider $d = (r_e, 10)(a_e, 20)(r_e, 200)$, then we have the no privacy property $NPr(d, 2, a_w)$.

3.1 Region graph

Our aim is to show that the property $NPr(d, i, a)$ is decidable.

To this purpose, let us recall first the notion of *region graph* of a timed automaton, as given in [1]. By proposition 2.2 it suffices to consider automata with only one sequential component.

As in [1], without loss of generality we assume clock constraints permitting only comparison with integer constants. In fact, given any automaton $\mathcal{A}$, there is a constant t such that, for each constant c appearing in a clock constraint in $\mathcal{A}$, $c \cdot t$ is an integer. Let $\mathcal{A} \cdot t$ be the automaton obtained by replacing each c appearing in a clock constraint in $\mathcal{A}$ by $c \cdot t$. In [1] it is proved that a word $(a_1, t_1) \ldots (a_n, t_n) \ldots$ is in the language $\mathcal{L}(\mathcal{A})$ if and only if the word $(a_1, t_1 \cdot t) \ldots (a_n, t_n \cdot t) \ldots$ is in the language $\mathcal{L}(\mathcal{A} \cdot t)$. As a consequence, $NPr(d, i, a)$ holds for $\mathcal{A}$ if and only if $NPr(d \cdot t, i, a)$ holds for $\mathcal{A} \cdot t$.

Let us consider the equivalence relation $\sim$ over clock valuations that contains each pair of clock valuations v and v' such that:

- for each clock x, either $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$, or both $v(x)$ and $v'(x)$ are greater than c_x, with c_x the largest integer appearing in clock constraints over x.
- for each pair of clocks x and y with $v(x) \leq c_x$ and $v(y) \leq c_y$, $fract(v(x)) \leq$

$fract(v(y))$ if and only if $fract(v'(x)) \leq fract(v'(y))$ ($fract(z)$ indicates the fractional part of z).

- for each clock x with $v(x) \leq c_x$, $fract(v(x)) = 0$ if and only if $fract(v'(x)) = 0$.

Note that for each pair of valuations v and v', and for each clock constraint ϕ in $\mathcal{A}$, it holds that:

$$\text{if } v \sim v' \text{ then } v \models \phi \text{ iff } v' \models \phi.$$

A *clock region* is an equivalence class of clock valuations induced by $\sim$. We denote by $[v]$ the equivalence class of $\sim$ containing v. Note that the set of clock regions is finite.

Clock regions can be expressed with conditions of the form $x = c$, $c < x < c + 1$ and $x > c_x$.

A *region* is a pair $(q, [v])$, with q a state and $[v]$ a clock region. The *initial region* is the pair $(q^0, [v^0])$ with q^0 the initial state and v^0 the valuation such that $v^0(x) = 0$, for each clock x.

The *region graph* $R(\mathcal{A})$ is a graph having the regions of $\mathcal{A}$ as set of nodes and having an edge $\langle (q, \alpha), a, I, (q', \alpha') \rangle$, with I an interval of the form (c, c'),$[c, c]$ or (c_m, ∞) (where $c < c_m$, $c \in \mathbb{N}$ and c_m is the largest constant that appears in the constraints of $\mathcal{A}$) if and only if, for each pair of valuations $v \in \alpha$ and $v' \in \alpha'$, $(q, v) \rightarrow_t^a (q', v')$ for some time $t \in I$.

Note that, differently from [1], we label edges of the region graph also with an interval.

Without loss of generality, we can consider only the nodes that can be reached from the initial region without cycles of edges labeled with interval $[0, 0]$ which would violate the assumption of progress of time.

3.2 Checking no-privacy

Our algorithm checking no-privacy constructs a set of intervals by applying operations, which are defined below, to intervals of the region graph.

Given two intervals I and I', let us denote with $I \pm I'$ the interval

$$I \pm I' = \{t \pm t' \mid t \in I \text{ and } t' \in I'\}.$$

Given a step leading to a region (q, α) in a time in the interval I, and a step from (q, α) to another region (q', α') in the interval I', a time in $I + I'$ is needed to reach (q', α') through (q, α), provided that the waiting time in q does not depend on the time consumed to reach q.

Given two intervals I and I', let us denote with $I \oplus I'$ the interval such that $inf(I \oplus I') = inf(I) + inf(I') + 1$, $sup(I \oplus I') = sup(I) + sup(I') - 1$. Moreover, if $inf(I \oplus I') = sup(I \oplus I')$ then we assume that $(I \oplus I')$ is left-closed and right-closed. Otherwise, $I \oplus I'$ is left-closed if and only if both I and I' are, and $I \oplus I'$ is right-closed if and only if both I and I' are.

Given a step leading to a region (q, α) in a time in the interval I, and a step from (q, α) to another region (q', α') in the interval I', a time in $I \oplus I'$ is needed to reach (q', α') through (q, α), provided that the waiting time in q depends on the time consumed to reach q, in the sense that the longer is the time consumed before entering q, the shorter is the waiting time in q. This happens if there is a constraint $c < x < c + 1$ in α, for some clock x.

Finally, let us denote with $(I)_t$ the interval:

$$(I)_t = \begin{cases} I & \text{if } sup(I) \leq t \\ I \cup [sup(I), \infty) & \text{if } inf(I) \leq t < sup(I) \\ (t, \infty) & \text{otherwise.} \end{cases}$$

Note that $t \in I$ if and only if $t \in (I)_t$. We have introduced notation $(I)_t$ since the set of the intervals $(I)_t$, such that I is a sum (obtained by means of either $+$ or $\oplus$) of intervals of a region graph, is finite.

Let us define now the algorithm *Ch-path-symb*. Given regions p and q, a high symbol a, a constant t and a low symbol a', *Ch-path-symb* checks whether there exists a sequence of steps labeled with symbols in $H \setminus \{a\}$ followed by a step labeled with a' and taking from p to q at time t.

Algorithm 1

Ch-path-symb*(p,q:region, a:high-symbol, t:T, a': low symbol): boolean*

(i) *tovisit:=*$\{(p, [0,0], false)\}$*;*
(ii) *visited:=*$\emptyset$*;*
(iii) *while true do*
(iv) *if empty(tovisit) then return false*
(v) *else*
(vi) (r, I, tt)*:=extract(tovisit);*
(vii) *add(*(r, I, tt)*,visited);*
(viii) *if (tt=true and* $(r = q)$ *and* $t \in I$*) then return true;*
(ix) *if tt=false then*
(x) *for each edge* $\langle r, a'', I', r' \rangle \in R(\mathcal{A})$ *with* $a'' \in H \setminus \{a\} \cup \{a'\}$
(xi) *if* $x = t'$ *is a constraint in r then c:=*$(r', (I + I')_t, (a'' = a'))$
(xii) *else if* $I' = [0,0]$ *then c:=*$(r', I, (a'' = a'))$
(xiii) *else c:=*$(r', (I \oplus I')_t, (a'' = a'))$
(xiv) *if* $c \notin$ *visited* $\wedge ((I + I')_t \neq (t, \infty))$ *then Add(c,tovisit).*

A tuple $(r, I, false)$ means that the region r can be reached from p in a time in the interval I by reading symbols in $H \setminus \{a\}$. A tuple $(r, I, true)$ means that the region r can be reached from p in a time in the interval I by reading

symbols in $H \setminus \{a\}$ and, subsequently, symbol a'.

So the algorithm considers firstly the pair $(p, [0,0], false)$. Given a pair $(r, I, false)$ and an edge $\langle r, a'', I', r' \rangle$ in $R(\mathcal{A})$ for some symbol $a'' \in H \setminus \{a\} \cup \{a'\}$, if the clock region in r satisfies $x = t'$ for some clock x and constant t', then the algorithm considers either the pair $(r', (I + I')_t, false)$, if $a'' \neq a'$, or the pair $(r', (I + I')_t, true)$, if $a'' = a'$. The condition $x = t'$ in r ensures that a time in I' must be elapsed after r is entered. We use interval $(I + I')_t$ instead of $(I + I')$ to guarantee that *Ch-path-symb* generates finite intervals.

If, on the contrary, the clock region in r does not satisfy $x = t'$ for any clock x and constant t', then there are two cases. If $\langle r, a'', I', r' \rangle$ is such that $I' = [0,0]$, then the clock regions of r and r' coincide, and, therefore, we consider the tuple $(r', I, (a'' = a'))$. Otherwise, the tuple $(r, (I \oplus I')_t, (a'' = a'))$ is considered. The interval $(I \oplus I')_t$ takes into account that the minimal (resp. maximal) waiting time in r follows the maximal (resp. minimal) waiting time needed to reach r. Also in this case we use interval $(I \oplus I')_t$ instead of $(I \oplus I')$ to guarantee that *Ch-path-symb* generates finite intervals.

Finally, if a tuple (r, I, tt) with $r = q$, $t \in I$ and $tt = true$ is generated, then *Ch-path-symb* terminates successfully.

The following lemmata state the correctness of the algorithm.

Lemma 3.2 *For any pair of regions p and q, high symbol a, time t and low symbol a', Ch-path-symb(p,q,a,t,a') terminates.*

Proof. Both the regions of the graph and the intervals that can been generated by the algorithm are finite and, as a consequence, also the tuples (r, I, tt) that are generated are finite. □

Lemma 3.3 *If* $(\omega_1, \omega_2) \in \mathcal{L}(\mathcal{A})$ *with* $\{\omega_1(i), \omega_1(i+1), \ldots, \omega_1(j-1), \omega_1(j)\} \subseteq H \setminus \{a\}$ *and* $\omega_1(j+1)$ *the low symbol* a'*, then there exists an infinite sequence of steps* $(q_0, v_0) \rightarrow^{\omega_1(0)}_{\omega_2(0)} (q_1, v_1) \rightarrow^{\omega_1(1)}_{\omega_2(1)} \ldots$ *if and only if Ch-path-symb*$((q_i, [v_i])$,$(q_{j+2}, [v_{j+2}])$,a, $\sum_{h=i}^{j+1} \omega_2(h)$,$a')$.

Lemma 3.3 is a direct consequence of the properties of the region graph proved in [1].

We define also the algorithm *Ch-path* that checks whether there exists a sequence of steps labeled with high symbols and followed by a step labeled with the low symbol a' taking from a given region p to a given region q in a given time t. We obtain it from *Ch-path-symb* by replacing the condition $a'' \in H \setminus \{a\} \cup \{a'\}$ in line (x) with the condition $a'' \in H \cup \{a'\}$.

The following results are the analogous of Lemma 3.2 and Lemma 3.3.

Lemma 3.4 *For any pair of regions p and q, time t and low symbol a', Ch-path(p,q,t,a') terminates.*

Lemma 3.5 *Let* $(\omega_1, \omega_2) \in \mathcal{L}(\mathcal{A})$ *with* $\{\omega_1(i), \omega_1(i+1), \ldots, \omega_1(-1j), \omega_1(j)\} \subseteq H$ *and* $\omega_1(j+1)$ *the low symbol* a'*, then there exists an infinite sequence of steps* $(q_0, v_0) \rightarrow^{\omega_1(0)}_{\omega_2(0)} (q_1, v_1) \rightarrow^{\omega_1(1)}_{\omega_2(1)} \ldots$ *if and only if Ch-path* $((q_i, [v_i]),(q_{j+2}, [v_{j+2}]), \sum_{h=i}^{j+1} \omega_2(h),\ a')$.

The following algorithm *Ch-NPriv* checks whether there exists a sequence of steps from the initial region $(q^0, [v^0])$ due to a low sequence d that does not perform the high symbol a between the low symbols in $d(i)$ and $d(i+1)$. At iteration k, the set A contains the regions that can be reached from $(q_0, [v_0])$ by reading $d(0), \ldots, d(k)$, by reading symbols in $H \setminus \{a\}$ between $d(i)$ and $d(i+1)$, and by reading symbols in H between $d(j)$ and $d(j+1)$, for each $j \neq i$. So, if A is empty then the sequence of steps we were looking for does not exist, and $NPr(d, i, a)$ holds.

Algorithm 2

Ch-NPriv*(d:L − sequence, i:ℕ, a:high − symbol): boolean*

(i) *k:=0;*
(ii) *A:=*$\{(q^0, [v^0])\}$*;*
(iii) *while* $k \leq length(d)$ *do*
(iv) (a', t)*:=d(k);*
(v) *B:=*$\emptyset$*;*
(vi) *while* $A \neq \emptyset$ *do*
(vii) $(q, [v])$*:=extract(A);*
(viii) *for each region* $(q', [v']) \in R(\mathcal{A})$
(ix) *if (*$k = i$ *and* Ch-path-symb*(*$(q, [v])$,$(q', [v'])$,a,t,a'*)) OR*
(x) *(*$k \neq i$ *and* Ch-path*(*$(q, [v])$, $(q', [v'])$,t,a'*))*
(xi) *then Add(*$(q', [v'])$,*B);*
(xii) *A:=B;*
(xiii) *k:=k+1;*
(xiv) *return* $A = \emptyset$.

The following results state the correctness of the algorithm *Ch-NPriv*.

Lemma 3.6 *For any finite sequence* d*, index* i *and high symbol* a*, the algorithm Ch-NPriv*(d,i,a) *terminates.*

Theorem 3.7 *For any finite sequence* d*, index* i *and high symbol* a*, it holds that* $NPr(d, i, a)$ *if and only if Ch-NPriv*(d,i,a).

Note that *Ch-NPriv*(d,i,a) performs at most k times the body of the external cycle. The internal cycle calls either the algorithm *Ch-path-symb* or the algorithm *Ch-path* at most $|\mathcal{R}(\mathcal{A})|$ times, with $|\mathcal{R}(\mathcal{A})|$ the number of regions of $\mathcal{R}(\mathcal{A})$. Finally, both *Ch-path-symb* and *Ch-path* construct at most

$O(|\mathcal{R}(\mathcal{A})| \times t^2)$ tuples.

Corollary 3.8 *It is decidable in polynomial time whether* $NPr(d, i, a)$.

4 Further Work

In this paper we have introduced the "no-privacy" property, which corresponds to the ability, by an attacker, to infer information on the private behavior of a system from the observable behavior.

To model a real time system that respects privacy, we can consider properties derived from the property $NPr(d, i, a)$ considered in the paper.

The first step is to consider properties such as $\exists a \in H.NPr(d, i, a)$, meaning that the performance of some secret action follows a sequence of observable actions, and $\exists i \in [1, length(d)-1].NPr(d, i, a)$, meaning that the performance of some secret action is implied by a sequence of observable actions. Both properties are decidable, since $NPr(d, i, a)$ is decidable and it is sufficient to enumerate the cases.

An interesting property is $\exists d.NPr(d, i, a)$. This property holds if there is a sequence of observable actions implying a secret action. In this case we cannot enumerate the cases. Moreover, even if such a property would be decidable, we cannot enumerate to prove the property $\exists d.\exists i \in [1, length(d)-1].NPr(d, i, a)$.

So, one may consider weaker properties. As an example, the property $\exists d.\ length(d) \leq n$ and $NPr(d, i, a)$ considers only finite-length sequences of observable actions. (It is usually sufficient to observe a finite number of observable actions to describe time attacks on protocols). Note that the sequences d such that $length(d) \leq n$ are not finite because times are infinitely many.

We might also consider sequences d in $(\Sigma \times Interval)^*$ instead of $(\Sigma \times Time)^*$. This kind of sequences permit to consider more general behaviors. As an example, a possible sequence is $(a, [0, \infty))(b, [3, 3])(c, [2, 5))$, meaning that a is performed in the interval $[0, \infty)$, b is performed 3 units of time after a, and c is performed when a time in $[2, 5)$ after b is elapsed.

Our aim is to study the decidability of such properties.

References

[1] Alur, R., and D.L. Dill: *A theory of timed automata.* Theoretical Computer Science **126** (1994), 183–235.

[2] Felten, E.W., and M.A. Schneider: Timing attacks on Web privacy. Proc. 7^{th} ACM Conference on Computer and Communications Security, 25–32, 2000.

[3] Focardi, R., and R. Gorrieri: Automatic compositional verification of some security properties. Proc. Second International Workshop on Tools and

Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science 1055, Springer, Berlin, 1996, 167-186.

[4] Focardi, R., and R. Gorrieri: *A classification of security properties for process algebras.* Journal of Computer Security **3** (1995), 5–33.

[5] Focardi, R., R. Gorrieri, and F. Martinelli: Information flow analysis in a discrete-time process algebra. Proc. 13^{th} Computer Security Foundation Workshop, IEEE Computer Society Press, 2000.

[6] Volpano, D., and G. Smith: *Confinement properties for programming languages.* SIGACT News **29** (1998), 33–42.

[7] Smith, G., and D. Volpano: Secure information flow in a multi-threaded imperative language. Proc. ACM Symposium on Principles of Programming Languages, 1998, 355–364.

Characterizing Non-Zenoness on Real-Time Processes

Jitka Stříbrná [1,2] and Insup Lee [2]

Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
Email: {jitkas, lee}@saul.cis.upenn.edu

Abstract

In this paper we examine an important property of correct system design that is called *non-Zenoness* or *time progress*. We present a method that allows to check non-Zenoness on a restricted subclass of real-time processes. The processes that we examine are constructed by nondeterministic sum and parallel composition with synchronization. The method is based on the construction of a finite representation of the potentially infinite state space of a process, that preserves time progress.

1 Introduction

There has been a number of proposed formalisms to describe real-time systems, in which system designs could be formally specified, analyzed and tested before implementation. The existing formalisms include real-time process algebras, namely the Algebra of Communicating Shared Resources (ACSR) [4], [6], Timed Automata [1], and Timed Systems (TS) [3], among others. The process algebraic and timed systems frameworks slightly differ in their use of time. In ACSR, there is a distinction between two types of actions: *timed actions* and *instantaneous events*. Timed actions represent resource consumption of a fixed time duration, whereas instantaneous events are used mainly for process synchronization. In Timed Systems, (and similarly in Timed Automata), events represent state change and have no duration, however it is possible to let time pass in any state of the system, while certain temporal conditions are satisfied.

[1] The author was partially supported by GA ČR grant no. 201/00/1023
[2] This research was supported in part by NSF CCR-9988409, NSF CCR-0086147, NSF CISE-9703220, ARO DAAG55-98-1-0393, ARO DAAG55-98-1-0466, ARO DAAD19-01-1-0473, DARPA ITO MOBIES F33615-00-C-1707, and ONR N00014-97-1-0505.

This is a preliminary version. The final version is considered for publication in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

There are many properties that are required from a correctly designed process. The deadlock and livelock-freedom properties are among the most important. A feature that pertains specifically to timed systems, is called *non-Zenoness* or *time progress*. Informally speaking, this characteristic ensures that a system cannot perform an infinite number of transitions within a finite amount of time. Since this feature is usually defined as a quality of all infinite runs of a system, it is in general undecidable. This property has been studied for Timed Systems in [3], where the authors proposed a notion of *structural non-Zenoness* as a condition on a finite graph underlying a TS. The presence of this condition implies that in every infinite execution of the TS, time will always progress.

For real-time process algebras, we need to distinguish between *discrete* and *dense* time domain. When we consider discrete time domain where all timed actions take exactly one time unit to execute, time progress can be expressed as the quality that every infinite execution sequence contains infinitely many timed actions. Testing this property on any ACSR process may still involve searching infinite runs of a process. It would be therefore helpful to be able to define a similar condition as structural non-Zenoness in the context of process algebras. Since process algebras are in general more expressive than TS (TS are finite-state systems, whereas even rather simple ACSR process may be infinite-state), we will limit our attention to a core class of (possibly infinite-state) processes for which we will be able to construct a finite representation that will preserve the non-Zeno property.

The class of processes studied in this paper is basically determined by the operators of action and event prefix, nondeterministic choice and parallel composition where we allow event synchronization. For any process constructed out of these operators we will show how to define a finite labeled graph whose edges correspond in a precise way to the labeled transitions of the process. Additionally, any cycle in this finite graph that will only consist of edges labeled by events will be related to a Zeno execution run of the initial process. Although we have not been able to prove it yet, we also believe that the other direction might work, i.e. any Zeno run of a process will manifest itself as a aforementioned cycle. Later on in this paper we will add the restriction operator and modify the construction so that an analogous condition holds for this enriched class.

2 Background

The class of processes we will examine in this paper is a fragment of ACSR, a timed process algebra that includes features for representing synchronization, time, resource requirements, and priorities. The time domain of ACSR can be either discrete [4] or dense [6], however in this paper we will concentrate exclusively on discrete time. The actions of processes in this algebra are of two kinds: *timed actions* and *instantaneous events*. A timed action takes one

time unit to execute and is represented as a list of resources and associated fixed priorities, e.g. $\{(data, 2), (cpu, 1)\}$. A distinguished action $\emptyset$ stands for one time unit of idling. An instantaneous event takes no time to execute and is represented as a pair of a label and a fixed priority, e.g. $(chan, 3)$. A distinguished event label τ is used to denote the synchronization of events.

For the purpose of simplicity, we will not consider the internal structure of timed actions or events. We assume that capital letters $A, B, \ldots$ range over timed actions, and letters $e, f, \ldots$ range over instantaneous events. For every event e, there is a *complementary* event $\bar{e}$, such that $\bar{\bar{e}} = e$. The special synchronization event τ arises when two complementary events are executed in parallel. The Greek letter α will be used to denote either a timed action or an event.

We will consider processes defined by the following syntax:

$$P ::= NIL \mid X \mid A : P \mid e.P \mid P + P \mid P \| P \mid recX.P$$

NIL is a process that executes no action. There are two Prefix operators, $A : P$ and $e.P$, corresponding to the two types of actions. The operator $P + Q$ represents nondeterministic Choice. $P \| Q$ is the Parallel composition of P and Q, that can synchronize or interleave on events, whereas for timed actions, we only consider interleaving. The Recursion operator $recX.P$ allows to specify infinite behavior.

The operational semantics of the process terms is represented by labeled transition systems (LTS). These are determined by structured operational semantics rules presented in Figure 1. We use $P[recX.\ P/X]$ as the standard notation for substitution of $recX.\ P$ for all free occurrences of X in P.

$$\text{(ActT)}\quad \frac{-}{A : P \xrightarrow{A} P} \qquad \text{(ActI)}\quad \frac{-}{e.P \xrightarrow{e} P}$$

$$\text{(SumL)}\quad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \text{(SumR)}\quad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\text{(ParL)}\quad \frac{P \xrightarrow{\alpha} P'}{P \| Q \xrightarrow{\alpha} P' \| Q} \qquad \text{(ParR)}\quad \frac{Q \xrightarrow{\alpha} Q'}{P \| Q \xrightarrow{\alpha} P \| Q'}$$

$$\text{(ParS)}\quad \frac{P \xrightarrow{e} P', Q \xrightarrow{\bar{e}} Q'}{P \| Q \xrightarrow{\tau} P' \| Q'} \qquad \text{(Rec)}\quad \frac{P[recX.\ P/X] \xrightarrow{\alpha} P'}{recX.\ P \xrightarrow{\alpha} P'}$$

Fig. 1. SOS rules

We assume commutativity and associativity of the parallel composition operator (the notion of equivalence on labeled transition systems being strong bisimulation). The n-th power of a process P, denoted P^n, stands as an abbreviation for the term $\underbrace{P \| \ldots \| P}_{n}$. In this work we allow for interleaving of timed

actions in the context of parallel composition. The reason for that will be explained in Section 4, where the main construction of the paper is presented.

The full discrete version of ACSR (see [6]) contains these additional operators: Scope $P\Delta_t^e(Q,R,S)$, Restriction $P\backslash F$, and Close $[P]_I$. In the Scope operator, process P executes for a maximum of t time units. If P successfully terminates within t units by executing an event labeled with $\bar{e}$ then control proceeds to Q. If P fails to do so then control proceeds to R. Lastly, at any time while P is executing, it may be interrupted by S.

The Restriction operator $P\backslash F$ limits the behavior of P in the sense that no events with labels in F are permitted to execute. Finally, the Close operator $[P]_I$ produces a process that monopolizes the resources in I.

3 Non-Zenoness on Restricted ACSR

The notion of non-Zenoness appears among other desirable properties in the validation of process design correctness. Intuitively, the purpose of instantaneous events is process synchronization, whereas the actual performance is represented by timed actions. Therefore we do not want a process to engage in an infinite sequence of synchronizations where only finitely many timed actions would be carried out. This motivates the following definition, taken from [2].

Definition 3.1 An infinite execution sequence (trace) of actions is called *Zeno* if it consists solely of instantaneous events. An ACSR process is *non-Zeno* if its LTS does not contain any infinite Zeno trace.

As we mentioned before, searching a potentially infinite LTS for a Zeno trace may result in a non-terminating algorithm. However, if we closely examine all processes that may appear in an infinite run, we will see that they are built out of only a finite number of basic process terms by the operator of parallel composition (for the processes described by our fixed syntax). As an example, consider the process $P \stackrel{\text{df}}{=} recX.\ \alpha.X^2$. There is only one infinite run of this process, described below:

$$recX.\ \alpha.X^2 \xrightarrow{\alpha} [recX.\ \alpha.X^2]^2 \xrightarrow{\alpha} \ldots [recX.\ \alpha.X^2]^i \xrightarrow{\alpha} \ldots$$

The reachable states of this process are all powers of *one* term, and that is the process itself. It is (usually) only one component (or two, in case of synchronization) in a parallel composition of more terms that can perform a transition, and so if we construct all these basic processes and the transitions that they can carry out, we have all the information necessary to detect what types of actions are taken in any execution sequence of the original process.

For this purpose, we will introduce the notion of a *prime w.r.t. parallel composition*, called here only *prime* for brevity. A prime of a process P is any process reachable from P that cannot be expressed as parallel composition of two or more processes.

Definition 3.2 The *set of primes* Pr of a process is defined recursively, following the syntactical structure of the process:

(i) $Pr(NIL) = \{NIL\}$

(ii) $Pr(X) = \{X\}$

(iii) $Pr(A : P) = \{A : P\} \cup Pr(P)$

(iv) $Pr(e.P) = \{e.P\} \cup Pr(P)$

(v) $Pr(P + Q) = \{P + Q\} \cup Pr(P) \cup Pr(Q)$

(vi) $Pr(P \| Q) = Pr(P) \cup Pr(Q)$

(vii) $Pr(recX.P) = \{recX.P\} \cup \{Q[recX.P/X] \mid Q \in Pr(P)\}$

The purpose of the set of primes is to describe all reachable processes in a compact way. It will be used to reduce a potentially infinite semantic description by means of labeled transition system into a finite graph, that will preserve the properties that we are interested in. The significance of this notion is stated in the lemma that follows.

Lemma 3.3 *Every reachable state Q of a process P can be expressed as parallel composition of primes from $Pr(P)$.*

We will prove Lemma 3.3 indirectly, by combining the following propositions.

Proposition 3.4 *Every process P can be expressed as parallel composition of primes from $Pr(P)$, i.e. $P = P_1 \| P_2 \| \ldots \| P_k$, where $P_1, \ldots, P_k \in Pr(P)$.*

Proposition 3.5 *For a process P, a prime $P_i \in Pr(P)$, and a process R reachable from P_i in one transition step, R can be expressed as parallel composition of primes of P.*

The correctness of Proposition 3.4 can be easily verified directly from the definition of primes. Throughout the paper, we will refer to this form as *maximal decomposition* of P. The proof of Proposition 3.5 follows here:

Proof. We will prove the claim by induction on the structure of P.

(i) Case NIL - obvious.

(ii) Case X - obvious.

(iii) Case $A : P$

The primes of $A : P$ are $A : P$ and the primes of P, and we assume as induction hypothesis that the statement holds for $Pr(P)$. The only reachable state of $A : P$ is P, which can clearly be expressed in the required form.

(iv) Case $e.P$ - proved analogously to Case (iii).

(v) Case $P + Q$

The primes of $P + Q$ are $P + Q$ itself, together with $Pr(P)$ and $Pr(Q)$. Induction hypothesis covers primes from $Pr(P)$ and $Pr(Q)$, and so only

the case of $P + Q$ remains to be checked. All processes reachable from $P+Q$ are derivatives of either P, or Q, to which the induction hypothesis applies.

(vi) Case $P\|Q$

The primes of $P\|Q$ are the union of $Pr(P)$ and $Pr(Q)$, and so the verity of the statement can be concluded from induction hypothesis for the two sets.

(vii) Case $recX.\ Q$

If $P = rec\ X.Q$ then we will assume that for every $Q_i \in Pr(Q)$, for every $Q_i \xrightarrow{\alpha} R$, R can be expressed as composition of primes of Q. What we need to verify is that for every prime P_i of P and every process R reachable from P_i in one step, R can be expressed as composition of primes of P.

By definition, $Pr(P) = \{rec\ X.Q\} \cup \{Q_i[P/X] \mid Q_i \in Pr(Q)\}$ Therefore we will consider two cases:

(a) $P_i = rec\ X.Q$

(b) $P_i = Q_i[P/X]$, for some $Q_i \in Pr(Q)$

Regarding (a), we need to use the semantic rule for recursively defined processes, which says that

$$rec\ X.Q \xrightarrow{\alpha} R \iff Q[rec\ X.Q/X] \xrightarrow{\alpha} R.$$

From that, we can deduce that

$$Q[rec\ X.Q/X] \xrightarrow{\alpha} R \iff \exists R'.\ Q \xrightarrow{\alpha} R' \wedge R = R'[rec\ X.Q/X].$$

The induction hypothesis holds that $R' = Q_1\| \ldots \|Q_k$, where these Q_i are primes of Q, and so $R = (Q_1\| \ldots \|Q_k)[rec\ X.Q/X]$. Clearly, this expression is equivalent to $Q_1[rec\ X.Q/X]\| \ldots \|Q_k[rec\ X.Q/X]$, which then is the desired composition of primes from $Pr(P)$.

To validate the latter, we assume that we have $P_i \xrightarrow{\alpha} R$, and $P_i = Q_i[rec\ X.Q/X]$, for some $Q_i \in Pr(Q)$. For a particular prime Q_i, we again have that,

$$Q_i[rec\ X.Q/X] \xrightarrow{\alpha} R \iff \exists R'.\ Q_i \xrightarrow{\alpha} R' \wedge R = R'[rec\ X.Q/X],$$

where $R' = Q_1\| \ldots \|Q_k$. As above, R can be expressed as $Q_1[rec\ X.Q/X]\| \ldots \|Q_k[rec\ X.Q/X]$, which is the form we sought.

□

Now we can combine Propositions 3.4 and 3.5 to justify Lemma 3.3. For a fixed process P, P itself can be expressed as parallel composition of its primes. Every other process R reachable from P is derived after m transition steps, for some m. If $m = 1$ then Proposition 3.5 guarantees that R is of the required shape. Every R derived in $m+1$ steps will be an immediate successor of some Q derived in m steps, which can be written as $P_1\| \ldots \|P_k$, $P_i \in Pr(P)$, and R will be obtained by taking one (or two, in case of synchronization) prime P_j and replacing it by one-step expansion, i.e. $R = P_1\| \ldots \|P'\| \ldots \|P_k$, where $P_j \xrightarrow{\alpha} P'$. By Proposition 3.5, P' is itself a composition of primes, and so the

resulting derivative R will be parallel composition of primes from $Pr(P)$.

Example 3.6 An example of primes construction is demonstrated on a simple process $Ex \stackrel{\text{df}}{=} recX.\ e.(X\|X)$ that will be used throughout the paper.

(i) $Pr(recX.\ e.(X\|X)) = \{recX.\ e.(X\|X)\} \cup \{Q[recX.\ e.(X\|X)/X] \mid Q \in Pr(e.(X\|X))\}$, following rule (vii);

(ii) $Pr(e.(X\|X)) = \{e.(X\|X)\} \cup Pr((X\|X))$, by rule (iv);

(iii) $Pr(X\|X) = Pr(X) \cup Pr(X)$, by rule (v);

(iv) $Pr(X) = \{X\}$, by rule (ii).

By applying the recursive definition we obtain

(iv) $Pr(X) = \{X\}$;

(iii) $Pr(X\|X) = \{X\}$;

(ii) $Pr(e.(X\|X)) = \{e.(X\|X), X\}$;

(i) $Pr(recX.\ e.(X\|X)) = \{recX.\ e.(X\|X), e.[recX.\ e.(X\|X)]^2\}$.

3.1 Finite representation graph

Now we can define the *finite representation graph* G_P of a process P to be a labeled directed graph (V_P, E_P), where

- the set of vertices V_P is the set of reachable primes of P, i.e. $V_P = \{P' \in Pr(P) \mid \exists Q.\ P \xrightarrow{w} P'\|Q\}$;
- the set of edges E_P is defined as $E_P = \{P_1 \xrightarrow{\alpha} P_2 \mid \exists Q.\ P_1 \xrightarrow{\alpha}_P P_2\|Q\}$, where $\xrightarrow{\alpha}_P$ is the transition relation generated by P.

We have defined the graph so that all its vertices can be reached from P in the context of parallel composition, and every edge in the graph can be generated by SOS rules from the description of P. In Section 4, we will provide an algorithm to construct the graph G.

Definition 3.7 A (directed) cycle in a graph G is called *Zeno* if all edges alongside the cycle are labeled by events.

We will demonstrate this idea on process Ex. The corresponding LTS and the (desired) finite graph are shown in Figure 2. We can see that process Ex produces an infinite LTS, with a Zeno trace $(\xrightarrow{e})^\omega$. That trace is captured by the $\xrightarrow{e}$ Zeno loop in the respective finite graph.

Note that the set of primes may contain an expression not reachable from the original process. In this case, $e.[recX.\ e.(X\|X)]^2$ cannot be reached from $recX.\ e.(X\|X)$, and so it is eliminated in the graph definition.

3.2 Main result

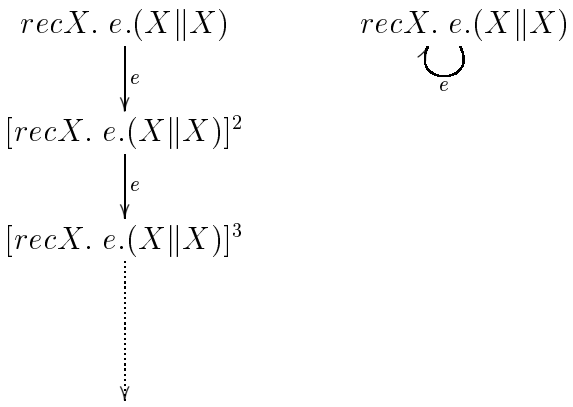

Fig. 2. Labeled transition system and finite graph of process Ex

Theorem 3.8 *Let P be a process defined by the syntax above. If there is a Zeno cycle in the graph G_P, then P is Zeno.*

Proof. We assume that in the graph G_P, there is a Zeno cycle $P_1 \xrightarrow{e_1} P_2 \xrightarrow{e_2} \ldots P_k \xrightarrow{e_k} P_1$. The vertices of G_P only contain primes that are reachable from P, therefore there must be an action sequence w and a process Q such that $P \xrightarrow{w} P_1 \| Q$. Now we can construct a Zeno trace of the form $w(e_1 e_2 \ldots e_k)^\omega$ in this way:

$$P \xrightarrow{w} P_1\|Q \xrightarrow{e_1} P_2\|R_1\|Q \xrightarrow{e_2} P_3\|R_2\|R_1\|Q \ldots \xrightarrow{e_k} P_1\|R_k \ldots \|R_1\|Q \xrightarrow{e_1} \ldots$$

where $P_i \xrightarrow{e_i} P_{i+1}\|R_i$ is some transition that gave rise to the directed edge $P_i \xrightarrow{e_i} P_{i+1}$.

□

4 Testing for Zeno Property Algorithmically

An algorithm that will detect a Zeno behavior of a given process consists of two parts. First part constructs a finite graph according to the definition in Subsection 3.1, and the second half searches that graph for Zeno cycles.

4.1 Graph Construction

The graph G (set of vertices and set of directed edges) is created recursively, following the definition of primes of a process. The construction is sketched in the procedure below, where the expression $P_i \xrightarrow{a}_G P_j$ denotes a directed edge from P_i to P_j, labeled by α, in the graph G.

(i) Case NIL
The set of vertices is $V_G = \{NIL\}$, and there are no edges in the graph induced by this vertex, i.e. $E_G = \emptyset$.

(ii) Case X
The set of vertices is $V_G = \{X\}$, and there are no edges in the graph induced by this vertex, i.e. $E_G = \emptyset$.

(iii) Case $A : P$

Assuming that we already have the complete graph G_P for P, we will add to it the vertex $A : P$ with edges $A : P \xrightarrow{A}_G P_i$, where $P_1 \| \ldots \| P_k$ is the maximal decomposition of P.

(iv) Case $e.P$

Analogously to (iii), we will add to the graph defined by P the vertex $e.P$ with edges $e.P \xrightarrow{e}_G P_i$, where $P_1 \| \ldots \| P_k$ is the maximal decomposition of P.

(v) Case $P + Q$

Assuming that G_P, resp. G_Q, are the graphs determined by P, resp. Q, the graph G for $P + Q$ is created as the union of G_P and G_Q, where a new vertex $P + Q$ is added. The vertices P and Q are removed together with all outgoing edges (under the condition that there are no incoming edges to P or Q). A new edge $P + Q \alpha_G P_i$ will be added to G, if and only if there is a corresponding edge $P \xrightarrow{\alpha} P_i$ in G_P, or $Q \xrightarrow{\alpha} P_i$ in G_Q.

(vi) Case $P \| Q$

The graph G for the process $P \| Q$ is taken to be the union of G_P and G_Q, the respective graphs for P and Q. No new vertices or edges are added.

(vii) Case $recX.\ P$

We assume that $G_P = (V_P, E_P)$ is the graph for P. We will construct G for the process $recX.\ P$ in two steps:

(a) we construct G_1 by taking the set $\{Q[recX.\ P/X] \mid Q \in V_P\}$ as vertices, with an edge $P_i[recX.\ P/X] \xrightarrow{\alpha} P_j[recX.P/X]$, if there was an edge $P_i \xrightarrow{\alpha} P_j$ in G_P;

(b) we construct the final graph $G_{recX.\ P}$ by adding an edge $recX.\ P \xrightarrow{\alpha} P_i$ whenever there was an edge $P[recX.\ P/X] \xrightarrow{\alpha} P_i$ in G_1. After adding all such edges, we remove the vertex $P[recX.\ P/X]$.

We will illustrate this algorithm on process Ex, by following the steps (iv) to (i) from Example 3.6, that describe its recursive decomposition:

(iv) Subprocess X

$V_X = \{X\}$, $E_X = \emptyset$.

(iii) Subprocess $X \| X$

$V_{X\|X} = \{X\}$, $E_{X\|X} = \emptyset$.

(ii) Subprocess $e.(X \| X)$

$V_{e.(X\|X)} = \{e.(X \| X), X\}$, $E_{e.(X\|X)} = \{e.(X \| X) \xrightarrow{e} X\}$.

(i) The original process $Ex = recX.\ e.(X \| X)$

(a) $V_1 = \{recX.\ e.(X \| X), e.[recX.\ e.(X \| X)]^2\}$, $E_1 = \{e.[recX.\ e.(X \| X)]^2 \xrightarrow{e} recX.\ e.(X \| X)\}$;

(b) we add the edge $recX.\ e.(X \| X) \xrightarrow{e} recX.\ e.(X \| X)$, and remove vertex $e.[recX.\ e.(X \| X)]^2$; the end result is $V_{recX.\ e.(X\|X)} = \{recX.\ e.(X \| X)\}$,

$E_{recX.\ e.(X\|X)} = \{recX.\ e.(X\|X) \xrightarrow{e} recX.\ e.(X\|X)\}$. In this way, we obtain a one-vertex graph with a Zeno loop, as pictured in Figure 2.

4.2 Algorithm Correctness

The correctness of the procedure follows from the two lemmas below, which state that the construction outlined above complies with the definition in Subsection 3.1.

Lemma 4.1 *For a process P, all vertices in G_P are reachable primes of P, i.e. for any $P_i \in V_P$, $P_i \in Pr(P)$ and there exists Q and w so that $P \xrightarrow{w} P_i\|Q$.*

Proof. By induction on structure of P, we will show that all vertices in G_P are reachable primes of P. From the construction it is clear that V_P only contains primes of P, and therefore, it only remains to be checked that these primes are reachable from P.

(i) Case NIL - obviously true.

(ii) Case X - obviously true.

(iii) Case $A : P$

If Q is a vertex in G_P then by induction hypothesis (IH), $P \xrightarrow{w} Q$ for some word w. Clearly, $A : P \xrightarrow{A} P$, therefore also $A : P \xrightarrow{A} P \xrightarrow{w} Q$.

(iv) Case $e.P$ is proved as Case (iii).

(v) Case $P + Q$

P_i is a vertex in G_{P+Q} if it is equal to $P + Q$, or it is a vertex from G_P (other than P), or it is a vertex in G_Q (other than Q); if $P_i = P + Q$ then it is trivially reachable from itself, and so we need to examine the other two possibilities.

If P_i belongs to G_P, then either $P_i \neq P$, and then by IH, $P \xrightarrow{w} P_i$ which implies that $P + Q \xrightarrow{w} P_i$, or $P_i = P$ and there is an incoming edge to P in the graph G_P. Since all vertices in G_P are reachable from P, it means that also P is reachable from itself, and so $P \xrightarrow{w} P = P_i$ and therefore $P + Q \xrightarrow{w} P_i$. We would use a similar argument for Q.

(vi) Case $P\|Q$

P_i is a vertex in $G_{P\|Q}$ if and only if P_i belongs to G_P or G_Q. By IH it follows that either $P \xrightarrow{w} P_i$ or $Q \xrightarrow{w} P_i$, hence either $P\|Q \xrightarrow{w} P_i\|Q$, or $P\|Q \xrightarrow{w} P\|P_i$.

(vii) Case $recX.\ P$

P_i is a vertex in $G_{recX.\ P}$ if there exists a $Q \in Pr(P)$, $Q \neq P$, so that $P_i = Q[recX.\ P/X]$. By IH, such Q would be reachable from P, i.e. $P \xrightarrow{w} Q$, which induces derivation $P[recX.\ P/X] \xrightarrow{w} Q[recX.\ P/X]$. From the second step in the construction of $G_{recX.\ P}$, we may conclude that $recX.\ P \xrightarrow{w} Q[recX.\ P/X] = P_i$, and so P_i is reachable from the original process $recX.\ P$.

□

Lemma 4.2 *There is an edge $P_i \xrightarrow{\alpha} P_j$ in G_P if and only if $P_i \xrightarrow{\alpha}_P P_j \| Q$, for some process Q.*

Proof. This statement is again proved by induction on the structure.

(i) Case NIL is straightforward as there are no edges in the respective graph.

(ii) Case X is straightforward as there are no edges in the respective graph.

(iii) Case $A : P$

New edges in the graph $G_{A:P}$ are of the form $A : P \xrightarrow{A} P_i$, where $P_1 \| \ldots \| P_k$ is the maximal decomposition of P into primes. The semantic rule for action prefix dictates that $A : P \xrightarrow{A}$, and so clearly, $A : P \xrightarrow{A} P_i \| Q$, with $Q = P_1 \| \ldots \| P_{i-1} \| P_{i+1} \| \ldots \| P_k$.

(iv) Case $e.P$ is proved analogously to Case (iii).

(v) Case $P + Q$

Any new edge in the graph G_{P+Q} is $P + Q \xrightarrow{\alpha} P_i$, which is induced either by $P \xrightarrow{\alpha} P_i$ in G_P, or $Q \xrightarrow{\alpha} P_i$ in G_Q. Since induction hypothesis holds for both, we can assume existence of a transition $P \xrightarrow{\alpha} P_i \| R$, or analogously starting from Q. By the semantic rule for Choice, we can conclude that in either case, $P + Q \xrightarrow{\alpha} P_i \| R$.

(vi) Case $P \| Q$ is straightforward as there are no new edges in the graph $G_{P\|Q}$.

(vii) Case $recX.\ P$

We will start from the assumption that $P \xrightarrow{\alpha} Q$ in G_P if and only if there exists a transition $P \xrightarrow{\alpha} Q \| R$ generated by definition of $recX.\ P$. Now, an edge $recX.\ P \xrightarrow{\alpha} Q[recX.\ P/X]$ in $G_{recX.\ P}$ relates back to the edge $P \xrightarrow{\alpha} Q$ in G_P, and an auxiliary edge $P[recX.\ P/X] \xrightarrow{\alpha} Q[recX.\ P/X]$ in the intermediate graph. From the induction hypothesis, we have that $P[recX.\ P/X] \xrightarrow{\alpha} Q[recX.\ P/X] \| R[recX.\ P/X]$ is a transition generated by $recX.\ P$, and from the operational rule for recursion we can conclude that $recX.\ P \xrightarrow{\alpha} Q[recX.\ P/X] \| R[recX.\ P/X]$.

□

Now we have validated that a Zeno cycle in a graph constructed in the above way represents a Zeno trace that can be derived from the original process (see Theorem 3.8).

4.3 Algorithm Complexity

The overall complexity of the algorithm depends on its two parts: graph construction and Zeno-cycle search. We will express it as a function of process size, which will be determined by the sum of the number of all operator and variable occurrences.

For a process of size n, the number of vertices in the completed graph is at most n: in each recursive step we add at most one vertex, and the number of steps corresponds to the number of operators and variable occurrences. Clearly, we can then bound the number of edges by n^2, even though a more subtle analysis would probably yield a considerably smaller result. Even if we consider the worst case, i.e. the complete graph on n vertices, the construction will be quadratic in n.

In order to search a graph G_P of process P for Zeno cycles, we will employ the *depth-first-search* technique ([5]). Several searches of the graph will be carried out, with initial vertices being all primes P_i in the decomposition of P. The running time of depth-first search is linear in size of the graph, expressed as the sum of number of vertices and edges, which is $O(n^2)$ here. From these observations we can conclude that the total complexity of the proposed algorithm, comprising both graph construction and Zeno-cycle search, is $O(n^2)$.

5 Adding Restriction

In this section, we will add the restriction operator $P\backslash F$, where F is a finite set of events, and modify the graph construction so that we can identify some Zeno behavior as previously. The transition rule for restriction is described below:

$$(\mathsf{Res}) \qquad \frac{P \xrightarrow{e} P', e, \bar{e} \notin F}{P\backslash F \xrightarrow{e} P'\backslash F}$$

If we want to be able to decompose the initial process into primes as before, we need to find a way to handle the restriction operator. The approach that we have chosen is to keep restrictions as extra information, and work with pairs $(Q,\ F)$, where Q is a prime of some initial process P that occurs in the definition of P within restriction context F. The graph construction is modified to reflect this fact, and works as follows:

(i) Case (NIL, F)
The set of vertices is $V_G = \{(NIL, F)\}$, and there are no edges in the graph induced by this vertex, i.e. $E_G = \emptyset$.

(ii) Case (X, F)
The set of vertices is $V_G = \{(X, F)\}$, and there are no edges in the graph induced by this vertex, i.e. $E_G = \emptyset$.

(iii) Case $(A : P, F)$
Assuming that we already have computed the graph for (P, F), we will add to it the vertex $(A : P, F)$ with edges $(A : P, F) \xrightarrow{A} (P_i, F)$, where $P_1 \| \ldots \| P_k$ is the maximal decomposition of P into subprocesses w.r.t. parallel composition.

(iv) Case $(e.P, F)$

Analogously to (iii), we will add to the graph defined by (P, F) the vertex $(e.P, F)$ with edges $(e.P, F) \xrightarrow{e}_G (P_i, F)$, if $e, \bar{e} \notin F$ and $P_1 \| \ldots \| P_k$ is the maximal decomposition of P into subprocesses w.r.t. parallel composition.

(v) Case $(P + Q, F)$

Assuming that G_P, resp. G_Q, are the graphs determined by (P, F), resp. (Q, F), the graph G for $(P + Q, F)$ is created as a union of G_P and G_Q, where a new vertex $(P + Q, F)$ is added, and vertices (P, F) and (Q, F) removed together with all outgoing edges (under the condition that there are no incoming edges to (P, F) or (Q, F)). There will be a new edge $(P + Q, F) \xrightarrow{\alpha} (P_i, E)$ in G, if and only if there is an edge $(P, F) \xrightarrow{\alpha} (P_i, E)$ in G_P, or $(Q, F) \xrightarrow{\alpha} (P_i, E)$ in G_Q.

(vi) Case $(P \| Q, F)$

The graph G for the process $(P \| Q, F)$ is taken to be a union of G_P and G_Q, the respective graphs for (P, F) and (Q, F); no new vertices or edges are added.

(vii) Case $(recX.\ P, F)$

We assume that $G_P = (V_P, E_P)$ is the graph for (P, F); we will construct G for the pair $(recX.\ P, F)$ in two steps:

(a) construct G_1 by taking the set $\{(Q[recX.\ P/X], F') \mid (Q, F') \in V_P\}$ as vertices, with an edge $(P_i[recX.\ P/X], F_i) \xrightarrow{\alpha} (P_j[recX.P/X], P_j)$, if there was an edge $(P_i, F_i) \xrightarrow{\alpha} (P_j, F_j)$ in G_P;

(b) construct the final graph $G_{(recX.\ P,F)}$ by adding an edge $(recX.\ P, F) \xrightarrow{\alpha} (P_i, F_i)$ whenever there was an edge $(P[recX.\ P/X], F) \xrightarrow{\alpha} (P_i, F_i)$ in G_1, and after adding all such edges, remove the vertex $(P[recX.\ P/X], F)$;

(viii) Case $(P \backslash E, F)$

The graph for $(P \backslash E, F)$ is taken to be the graph for $(P, E \cup F)$.

We will always initialize the construction with $F = \emptyset$. We will explain the algorithm on a process P defined as $P \stackrel{\text{df}}{=} recX.\ (e.X \| f.X) \backslash \{e\}$. The corresponding LTS and the constructed graph representation are presented in Figure 3. For conciseness we will write Q for $(e.X \| f.X) \backslash \{e\}$. The algorithm is initialized with the pair $(recX.\ (e.X \| f.X) \backslash \{e\}, \emptyset)$. Then we decompose the process according to the current top-level operator, and finally, after having reached the base case, we go back up, constructing the graph along the way.

(i) We call the algorithm with $((e.X \| f.X) \backslash \{e\}, \emptyset)$.

(ii) We make a recursive call to $(e.X \| f.X, \{e\})$.

(iii) We make two procedure calls for $(e.X, \{e\})$ and $(f.X, \{e\})$.

(iv) From $(e.X, \{e\})$ we call $(X, \{e\})$; likewise for $(f.X, \{e\})$.

(v) The base case $(X, \{e\})$ returns the graph with vertex set $V = \{(X, \{e\})\}$

and edge set $E = \emptyset$, to both calls.

(iv) For $(e.X, \{e\})$, the call returns graph with $V_1 = \{(e.X, \{e\}), (X, \{e\})\}$ and $E_1 = \emptyset$; for $(f.X, \{e\})$, we get $V_2 = \{(f.X, \{e\}), (X, \{e\})\}$ and $E_2 = \{(f.X, \{e\}) \xrightarrow{f} (X, \{e\})\}$.

(iii) The graph of $(e.X \| f.X, \{e\})$ is a union of the already constructed graphs: $V = \{(e.X, \{e\}), (X, \{e\}), (f.X, \{e\}), (X, \{e\})\}$ and $E = \{(f.X, \{e\}) \xrightarrow{f} (X, \{e\})\}$.

(ii) We keep the graph and pass it to the higher level.

(i) We perform substitution first, and obtain these vertices $(recX.\ Q, \{e\})$, $(e.recX.\ Q, \{e\})$, $(f.recX.\ Q, \{e\})$; then we add the edge $(recX.\ Q, \{e\}) \xrightarrow{f} (recX.\ Q, \{e\})$.

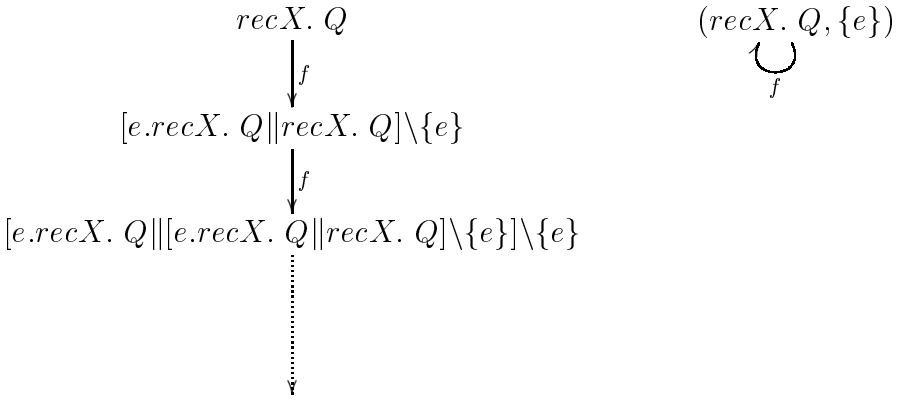

Fig. 3. Labeled transition system and finite graph of process P

The resulting graph contains an f loop from and to the initial state, therefore it will be discovered by DFS. It can be easily observed that this loop indeed corresponds to an infinite Zeno trace of the process P.

5.1 Algorithm Correctness and Complexity

For a given process P, we perform a depth-first search on the graph G constructed for $(P, \emptyset)$. The initial vertices for Zeno cycle search will be all pairs (P_i, F_i) satisfying the condition that P_i is contained in the maximal decomposition P into subprocesses w.r.t. parallel composition, and there is no vertex (P_i, E_i) in G with $F_i \subset E_i$.

We will utilize a couple of lemmas analogous to the previous section that will validate the correctness of our procedure. Firstly, we will spell out an analog of Lemma 4.2.

Lemma 5.1 *For a process P and restriction set F, every edge $(P_i, F_i) \xrightarrow{\alpha} (P_j, F_j)$ in $G_{(P,F)}$ corresponds to some transition $P_i \xrightarrow{\alpha}_P P_j \| Q$, where $\alpha \notin F_i$ and $F_i \subseteq F_j$.*

Proof. The notation (Q, F) captures the fact that Q occurs in P within restriction context F, and therefore $Q \xrightarrow{e}_P R$, only if e and $\bar{e}$ do not belong to F. We will prove the claim by structural induction.

(i) Case (NIL, F) is trivial as there is no transition in the corresponding graph.

(ii) Case (X, F) is trivial as there is no transition in the corresponding graph.

(iii) Case $(A : P, F)$
We assume that the statement is valid for (P, F). A new edge must be one of $(A : P, F) \xrightarrow{A} (P_i, F)$, with P_i contained in the maximal decomposition P into subprocesses w.r.t. $\|$. Timed actions are not effected by restriction, therefore clearly $A : P \xrightarrow{A} P_i \| Q$, where Q is the remaining part of P. Additionally, $A \notin F$ and $F \subseteq F$.

(iv) Case $(e.P, F)$
We assume that the statement is valid for (P, F). A new edge must be one of $(e.P, F) \xrightarrow{e} (P_i, F)$, with P_i being of the proper form, and $e, \bar{e} \notin F$. This implies that the transition is possible also in the context of P, i.e. $e.P \xrightarrow{e} P_i \| Q$, and again, $F \subseteq F$.

(v) Case $(P + Q, F)$
The semantic rule for Choice dictates that any transition of $P + Q$ is induced by P or Q. The restriction set remains the same, which concludes this case.

(vi) Case $(P \| Q, F)$
There are no new edges added for this operator, therefore this case is valid by application of induction hypothesis to (P, F) and (Q, F).

(vii) Case $(recX.\ P, F)$
Since the the restriction set remains the same, this case would be proved analogously to Case (vii) of Lemma 4.2.

(viii) Case $(P \backslash E, F)$
An edge $(P_i, F_i) \xrightarrow{\alpha} (P_j, F_j)$ in the graph for $(P \backslash E, F)$ corresponds to an edge $(P_i, F_i') \xrightarrow{\alpha} (P_j, F_j')$ in the graph for $(P, E \cup F)$. Therefore, by IH, $\alpha \notin F_i'$ and $F_i' \subseteq F_j'$. Additionally, $F_i' = E \cup F_i$ and $F_j' = E \cup F_j$ which in turn implies that $\alpha \notin F_i$ and $F_i \subseteq F_j$.

□

In the graph construction, we may now obtain vertices that are not reachable from the initial process. However, our algorithm is still correct because the depth-first search only visits those graph vertices that correspond to reachable subterms of the initial process. That is summed up in the lemma below:

Lemma 5.2 *If (P', F') is a vertex visited in the DFS of graph G for $(P, \emptyset)$ then there exist Q, E and w so that $P \xrightarrow{w} (P' \| Q) \backslash E$, where $E \subseteq F'$.*

This lemma is basically a corollary of the previous Lemma 5.1. Now we can

put these two claims together to obtain the final correctness theorem.

Theorem 5.3 *Let P be a process defined by the syntax above. If the depth-first search algorithm of the graph $G_{(P,\emptyset)}$ finds a Zeno cycle, then P is Zeno.*

Proof. A Zeno cycle $(P_1, F_1) \xrightarrow{e_1} (P_2, F_2) \xrightarrow{e_2} \ldots (P_k, F_k) \xrightarrow{e_k} (P_1, F_1)$ has the property that $F_1 \subseteq F_2 \subseteq \ldots F_k \subseteq F_1$, i.e. all these sets are equal. We can relate the graph cycle to a Zeno trace using Lemma 5.2 and 5.1:

(i) there exists a sequence w and Q such that $P \xrightarrow{w} (P_1 \| Q) \backslash E$, where $E \subseteq F_1$;

(ii) each edge $(P_i, F_i) \xrightarrow{e_i} (P_{i+1}, F_{i+1})$ in the Zeno cycle translates back to a transition $P_i \xrightarrow{e_i}_P P_{i+1} \| Q_i$, where $e_i \notin F_i$.

The initial segment of the Zeno trace that we thus obtain is specified below:

$$P \xrightarrow{w} (P_1 \| Q) \backslash E \xrightarrow{e_1} (P_2 \| Q_1 \| Q) \backslash E \xrightarrow{e_2} (P_3 \| Q_2 \| Q_1 \| Q) \backslash E \ldots \xrightarrow{e_k} (P_1 \| Q_k \ldots \| Q_1 \| Q) \backslash E \xrightarrow{e_1} \ldots$$

□

The overall complexity of this procedure can be estimated as in Subsection 4.3. If we now include the size of restriction sets in the size of an input process P, then we can again limit the time complexity of graph construction by n, the size of P, as each recursive step adds at most one vertex and there may only be n steps.

If we use the worst case scenario and assume our graph to be complete, then as before, the Zeno-cycle search will be performed in $O(n^2)$, which limits the total complexity.

6 Conclusion

The method presented in this paper enables us to identify some cases of Zeno processes, by constructing a finite representation for a limited subclass of ACSR processes that preserves Zeno property. The algorithm creates a finite graph out of a syntactic definition of a process and then searches for Zeno cycles within the graph. If such a cycle is found then the given process can engage in Zeno behavior.

The condition so far only works one way. We would like to extend it to an *iff* characterization by showing that every Zeno trace a process can perform manifest itself in the respective graph. We believe that this will certainly hold for the class of processes specified in Section 3. However, it seems that when synchronization together with restriction (hiding) on events is present, it may no longer be possible to obtain an iff condition.

Another issue is synchronization of timed actions, that is one of the main features of ACSR. It seems that a straightforward application of the graph construction is not feasible, the problem being that whenever timed action

synchronization is performed, *all* processes within a parallel composition must carry out a timed action. However, we may have derivations with ever growing number of parallel components, and clearly this may not be captured within the finite graph. Since timed actions are not of much interest in the Zeno property, we have chosen to ignore the synchronization mechanism, however this will be a topic of future interest. Other possible themes of future work might be to examine if a similar method may work for other enriched classes of processes, such as parametric processes or process algebras with dense time.

References

[1] Alur R. and Dill D.L.: A theory of timed automata. In TCS, 126:183–235, 1994.

[2] Ben-Abdallah H., Choi J.-Y., Clarke D., Kim Y., Lee I., and Xie H.-L.: A Process Algebraic Approach to the Schedulability Analysis of Real-Time Systems. In Real-time Systems 15, Kluwer Academic Publishers, 189–219, 1998.

[3] Bornot S., Gössler G., and Sifakis J.: On the Construction of Live Timed Systems. In Proceedings of TACAS'00, LNCS 1785, Springer-Verlag, 2000.

[4] Brémond-Grégoire P.: **A Process Algebra of Communicating Shared Resources with Dense Times and Priorities**. PhD Thesis, Department of Computer and Information Science, University of Pennsylvania. Tech. Report MS-CIS-94-24, 1994.

[5] Cormen T.H., Leiserson C.E., and Rivest R.L.: **Introduction to Algorithms**. The MIT Press, McGraw-Hill Book Company, 1990.

[6] Lee I., Brémond-Grégoire P., and Gerber R.: A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems. In Proceedings of the IEEE, 82(1):158–171, 1994.

Recent BRICS Notes Series Publications

NS-01-5 **Flavio Corradini and Walter Vogler, editors.** ***Preliminary Proceedings of the 2nd International Workshop on Models for Time-Critical Systems, MTCS '01,*** **(Aalborg, Denmark, August 25, 2001), August 2001. vi+ 127pp.**

NS-01-4 **Ed Brinksma and Jan Tretmans, editors.** ***Proceedings of the Workshop on Formal Approaches to Testing of Software, FATES '01,*** **(Aalborg, Denmark, August 25, 2001), August 2001. viii+156 pp.**

NS-01-3 **Martin Hofmann, editor.** ***Proceedings of the 3rd International Workshop on Implicit Computational Complexity, ICC '01,*** **(Aarhus, Denmark, May 20–21, 2001), May 2001. vi+144 pp.**

NS-01-2 **Stephen Brookes and Michael Mislove, editors.** ***Preliminary Proceedings of the 17th Annual Conference on Mathematical Foundations of Programming Semantics, MFPS '01,*** **(Aarhus, Denmark, May 24–27, 2001), May 2001. viii+279 pp.**

NS-01-1 **Nils Klarlund and Anders Møller.** ***MONA Version 1.4 — User Manual*****. January 2001. 83 pp.**

NS-00-8 **Anders Møller and Michael I. Schwartzbach.** ***The XML Revolution*****. December 2000. 149 pp.**

NS-00-7 **Nils Klarlund, Anders Møller, and Michael I. Schwartzbach.** ***Document Structure Description 1.0*****. December 2000. 40 pp.**

NS-00-6 **Peter D. Mosses and Hermano Perrelli de Moura, editors.** ***Proceedings of the Third International Workshop on Action Semantics, AS 2000,*** **(Recife, Brazil, May 15–16, 2000), August 2000. viii+148 pp.**

NS-00-5 **Claus Brabrand.** ***<bigwig> Version 1.3 — Tutorial*****. September 2000. ii+92 pp.**

NS-00-4 **Claus Brabrand.** ***<bigwig> Version 1.3 — Reference Manual*****. September 2000. ii+56 pp.**

NS-00-3 **Patrick Cousot, Eric Goubault, Jeremy Gunawardena, Maurice Herlihy, Martin Raussen, and Vladimiro Sassone, editors.** ***Preliminary Proceedings of the Workshop on Geometry and Topology in Concurrency Theory, GETCO '00,*** **(State College, USA, August 21, 2000), August 2000. vi+116 pp.**