A Stochastic Automata Model and its Algebraic Approach

Pedro R. D'Argenio[1]*, Joost-Pieter Katoen[2], and Ed Brinksma[1]

[1]*Dept. of Computer Science. University of Twente.*
P.O.Box 217. 7500 AE Enschede. The Netherlands.
{dargenio,brinksma}@cs.utwente.nl

[2]*Lehrstuhl für Informatik VII. University of Erlangen.*
Martensstrasse 3. D-91058 Erlangen. Germany.
katoen@informatik.uni-erlangen.de

August 8, 1997

Abstract

We discuss a new model for the analysis and simulation of stochastic systems which we call stochastic automata. Basically, they are a combination of the timed automata model and generalised semi-Markov processes (GSMPs for short). We discuss their behaviour and we compare them to the GSMPs model. In addition, we define a stochastic process algebra that supports general distribution (both continuous and discrete). Its semantics is given in terms of stochastic automata. We show that stochastic automata can be expressed in terms of the process algebra. We discuss a concrete example and we finish by discussing our current work on this topic and possible future directions.

1991 Mathematics Subject Classification: 68Q55, 68Q60, 68Q75, 93E03.

1991 CR Categories: C.4, D.3.1, F.3.1, F.3.2, I.6.2.

Keywords: stochastic automata, process algebra, stochastic systems, real-time, timed automata, performance analysis.

Note: This is a revised version of the article appeared in E. Brinksma and A. Nymeyer editors, *Proc. of 5th International Workshop on Process Algebras and Performance Modeling*, Enschede, The Netherlands, Technical Report CTIT 97-14. University of Twente, June 1997.

*Supported by the NWO/SION project 612-33-006.

1 Introduction

In the world of performance modelling, many models have been defined to analyse and simulate systems such as queuing networks, stochastic Petri-nets, or generalised semi-Markov processes. It has been argued many times that, in these kind of models, the difficulty of the design and modelling of a system whose performance is studied rapidly grows with the size and complexity of the system itself.

In the last few years, this phenomenon has drawn the attention of many researchers into extending process algebras with stochastic and real-time features [15, 9, 11, 3, 6, 16]. *Stochastic process algebras*, as they are usually called, considerably simplify the tractability of complex systems because, in this framework, systems do not need to be modelled as a whole, but as a composition of small subsystems. Another advantage is that stochastic process algebras not only allow to study the performance of a system, but also its functionality.

In this paper, we introduce a new model to study stochastic systems that we named *stochastic automata*. The stochastic automata model is an extension of the traditional automata model with ideas borrowed from timed automata [2, 12] and GSMPs [8, 25]. Basically, a stochastic automaton is an automaton with *clocks*. Clocks are set randomly according to an associated distribution function and their value decreases as time passes. The occurrence of a transition at a certain time is controlled by the clocks. Each transition has associated a set of clocks and it must take place as soon as all these clocks have expired, i.e., they have decreased further than the value zero.

We use stochastic automata as the underlying semantics of a stochastic process algebra. Actually, the stochastic automata model and the process algebra turn out to be equally expressive. In this way, the process algebra can be regarded as a language to describe stochastic automata. This result closely follows the methodology of [7] where a process calculus for timed automata was introduced.

Usually, the semantics of stochastic process algebras such as TIPP [9, 13], PEPA [15], and EMPA [3], is defined in terms of extended transition systems, which, basically, have each transition labelled not only with the action name, but also with a distribution function that determines the timing of such a transition. But the inherent interleaving characteristic of transition systems demands a careful treatment of the definition of parallel composition. In traditional interleaving process algebras like CCS [18], it holds that

$$a;P \parallel b;Q = a;(P\|b;Q) + b;(a;P\|Q)$$

Stochastic process algebras extend the prefixing into $a_F;P$ where F is a distribution function which determines the probability of the random delay after which the action a must happen. In this setting, it is *not* generally true that

$$a_F;P \parallel b_G;Q = a_F;(P\|b_G;Q) + b_G;(a_F;P\|Q) \tag{1}$$

since, in the left hand side process, the time of a and b starts to count from the same initial moment, while in the right hand side process, the delay of one action starts to count *after* the other action has already occurred. A possible first solution is to restrict the attention only to exponential distributions. Their memoryless property restores the expansion law, i.e., in this context, equation (1) holds [15, 13, 3].

An alternative solution was proposed in [6] by moving to true concurrency semantics, where the expansion law is not longer generally true. The only restriction of this approach

is that the set of chosen distribution functions should form a monoid. The drawback of this solution is that, even for very simple recursive processes, the underlying semantic object (namely, a stochastic variation of an event structure) is infinite.

With our stochastic process algebra, we propose a more elegant solution. We separate the stochastic information from the action name. (We should remark that a similar approach has been used in [11].) Instead of writing $a_F; P$, we write $\{\!|x_F|\!\}\ (\{x_F\}\mapsto a; P)$. The operator $\{\!|x_F|\!\}\ \ldots$ sets the clock x_F according to the distribution function F, and the operation $\{x_F\}\mapsto \ldots$ prevents the prefixing $a; P$ to happen until clock x_F has expired. In this setting we can have an alternative expansion law:

$$\{\!|x_F|\!\}\ (\{x_F\}\mapsto a; P) \mid\mid \{\!|y_G|\!\}\ (\{y_G\}\mapsto b; Q) \ = \\ \{\!|x_F, y_G|\!\}\ (\{x_F\}\mapsto a; (P \mid\mid \{y_G\}\mapsto b; Q) + \{y_G\}\mapsto b; (\{x_F\}\mapsto a; P \mid\mid Q))$$

In addition, this separation of concerns (setting of time according to a distribution function, expiration of such a time, and actual activity) introduces more expressive power. We observe that in principle *any* kind of (continuous or discrete) distribution function is allowed in this model, while we maintain a finite semantic object in a reasonable way (comparable to regular processes in CCS).

The aim of this article is to introduce and discuss the stochastic automata model and the general stochastic process algebra. Theoretical concerns will be discuss in deep in a forthcoming report.

The paper is organised as follows. In Section 2, we introduce the stochastic automata model. We discuss its behavioural properties and we informally discuss the underlying semantics. In Section 3, GSMPs are shown to be properly included in the stochastic automata model. The stochastic process algebra is introduced in Section 4. We discuss its intuitive behaviour and define its semantics in terms of stochastic automata. Moreover, we show that any stochastic automaton can be expressed by the process algebra. In Section 5, a non-trivial example is introduced. We describe the CSMA/CD protocol in terms of our stochastic process algebra. Finally, in Section 6, we conclude by discussing work in progress, future directions of our research, and related work.

2 The Stochastic Automata Model

In this section, we introduce a kind of automaton that allows us to represent processes with stochastic information. The basic idea is borrowed from timed automata [2, 12] but the idea of the behaviour and compositionality (see Section 4) is mainly based on the approach of [26] by combining it with ideas of discrete event systems, in particular GSMPs [8, 25].

First we enumerate all the ingredients of a stochastic automaton.

Definition 2.1 A *stochastic automaton* is a structure $(\mathcal{S}, s_0, \mathcal{C}, \mathbf{A}, \longrightarrow, \kappa, F)$ where:

- $\mathcal{S}$ is a set of *locations*,
- $s_0 \in \mathcal{S}$ is the *initial location*,
- $\mathcal{C}$ is a set of *clocks*,
- $\mathbf{A}$ is a set of *actions*,

- $\longrightarrow \subseteq \mathcal{S} \times (\mathbf{A} \times \wp(\mathcal{C})) \times \mathcal{S}$ is the set of *edges*. We usually denote $(s, a, C, s') \in \longrightarrow$ by $s \xrightarrow{a,C} s'$ and we say that C is its *trigger set*,
- $\kappa : \mathcal{S} \to \wp(\mathcal{C})$ is the *clock setting function*,
- $F : \mathcal{C} \to (\mathbb{R} \to [0,1])$ assigns to each clock a *distribution function* such that $F(x)(t) = 0$ for $t < 0$; we write F_x instead of $F(x)$.

Notice that each clock $x \in \mathcal{C}$ is a random variable with distribution F_x. □

As in [7], the information of which clock should be set is related to the locations instead of the edges. This will be helpful for compositionality when we use stochastic automata as the semantic interpretation of a process algebra (see Section 4). In stochastic automata, clocks are randomly set according to a certain associated distribution function and they count down. A clock expires if it has reached or decreased below the value 0. The occurrence of an action is controlled by the expiration of clocks. Thus, whenever $s \xrightarrow{a,C} s'$ and the system is in location s, *a must* happen as soon as all the clocks in the trigger set C have expired. Immediately afterwards all clocks in $\kappa(s')$ are randomly set according to their respective distributions. The idea of clocks that decrease in time is borrowed from GSMPs [8, 25] (see also Section 3).

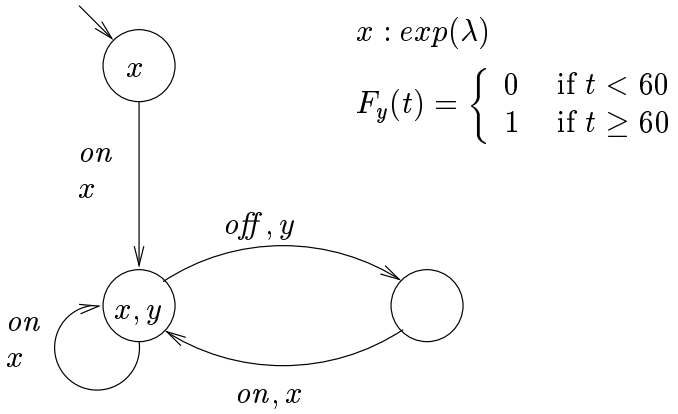
For the reader familiar with timed automata model [2, 12] we may remark that constraints are implicit in stochastic automata. On the one hand, guards on $s \xrightarrow{a,C} s'$ would be made explicit by $\bigwedge_{x \in C} x \leq 0$ (we recall that clocks are counting down). That is, all the clocks that trigger the edge $s \xrightarrow{a,C} s'$ should have expired. In this case, we say that $s \xrightarrow{a,C} s'$ is enabled. On the other hand, the invariant of s would be $\forall s \xrightarrow{a,C} . \bigvee_{x \in C} x > 0$, which says that it is allowed to idle in location s while no action is enabled. Notice that as soon as an edge becomes enabled, the invariant becomes false; thus, the system must leave this location by executing an enabled edge.

We give a simple example to understand the intuition of the model. Figure 1 represents a switch that controls a light in a corridor or stairway. In the picture, circles represent locations, variables enumerated in each location are the clocks that should be set according to the function κ, and edges are represented by the arrows. The initial state is represented by the small ingoing arrow. The distribution function of each clock is given beside the picture.

The switch may be turned on at any time according to an exponential distribution with arrival rate λ, even if the light is still on. It switches automatically off exactly 60 seconds after the most recent time the light was switched on. Since we considered that exact 60 seconds must past before the light is turned off, y is a random variable that takes value 60 with probability 1. Notice that we can easily change the system to consider that clock y is not precise and has a drift of ϵ units of time. We can assume that such a drift is uniformly distributed. Then, y would become a random variable with uniform distribution in $[60 - \epsilon, 60 + \epsilon]$.

The semantics of a stochastic automata is given in terms of a probabilistic transition system. We will not formally define the semantics of a stochastic automaton. Instead, we give a flavour of the underlying semantic model. We define a kind of probabilistic transition system in which we allow any kind of probability spaces, including thus continuous probabilities. We consider separately probabilistic transitions and non-deterministic transitions. In this way, our model is close to those of [10] and [11], although we do not distinguish between timed transitions and discrete transitions. Instead, we use the approach of time-stamped actions.

Figure 1: The switch



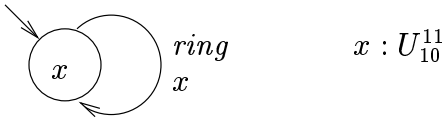
Definition 2.2 Let $Prob(H)$ denote the set of probability spaces $(\Omega, \mathcal{F}, P)$ such that $\Omega \subseteq H$. A *probabilistic transition system* is a structure $(\Sigma, \Sigma', \sigma_0, \mathbf{A} \times \mathbb{R}_{\geq 0}, T, \longrightarrow)$ where

- Σ and Σ' are two disjoint sets of *states*, with the *initial state* $\sigma_0 \in \Sigma$. States in Σ are called *probabilistic states* and states in Σ' are called *non-deterministic states*.
- $\mathbf{A}$ is a set of *actions* and $\mathbb{R}_{\geq 0}$ is the set of non-negative real numbers. We write $a(d)$ for (a, d) $(\in \mathbf{A} \times \mathbb{R}_{\geq 0})$.
- $T : \Sigma \to Prob(\Sigma')$ is the *probabilistic transition relation*. Since T is defined as a (total) function, there is exactly one probabilistic transition relation for each probabilistic state.
- $\longrightarrow \subseteq \Sigma' \times (\mathbf{A} \times \mathbb{R}_{\geq 0}) \times \Sigma$ is the *labelled (or non-deterministic) transition relation*. We write $\sigma' \stackrel{a(d)}{\longrightarrow} \sigma$ for $\langle \sigma', a(d), \sigma \rangle \in \longrightarrow$ and its intended meaning is that whenever the system is in the non-deterministic state σ', it can perform an action a at time d and move to the probabilistic state σ. □

The probabilistic transition system of a given stochastic automaton would be defined as follows. Suppose that the system arrives at a location s with the clocks having values according to a valuation v. We identify this situation with a probabilistic state (s, v). As soon as this location is reached, clocks in $\kappa(s)$ are randomly set according to a distribution function defining new valuations v' where the clocks which are not in $\kappa(s)$ preserve the same value of v. Hence, this procedure defines a probabilistic transition and the elements of the probability space defined by $T(s, v)$ are the non-deterministic states $(s, v')'$. (We take the convention that non-deterministic states are primed to distinguish them from probabilistic states.) Once the clocks are set, we calculate which is the first edge $s \xrightarrow{a,C} s'$ that becomes enabled. So there will be a smallest non negative real $d \in \mathbb{R}_{\geq 0}$ such that, for every clock $x \in C$, $v'(x) - d \leq 0$. This will induce a labelled transition $(s, v')' \stackrel{a(d)}{\longrightarrow} (s', v'')$, where v'' is obtained by decreasing all the values in v' by d time-units with the exception of the triggered clocks in C which are set to infinity. Notice that more than one edge can become enabled at the same time, in such a case non-determinism arises.

To understand the formal semantics, we consider a simple example. Figure 2 represents an alarm bell that rings randomly between 10 and 11 seconds according to a uniform distribution. We write $x : U_{10}^{11}$ to mean that x is a random variable with a uniform distribution function F_x in the interval $[10, 11]$.

Figure 2: The alarm bell



Its probabilistic transition system would be given by

$$\Sigma = \{(s_0, x := d) \mid d \in \mathbb{R}\} \qquad T(s_0, x := d) = (\Sigma', \mathcal{F}, P)$$

$$\Sigma' = \{(s_0, x := d)' \mid d \in \mathbb{R}\} \qquad (s_0, x := d)' \stackrel{ring(d)}{\longrightarrow} (s_0, x := d)$$

where $(\Sigma', \mathcal{F}, P)$ is the probability space in which $\mathcal{F}$ is some appropriate transformation of the Borel space in $\mathbb{R}$ (basically, a bijection from $\mathbb{R}$ to Σ') and P is the probabilistic measure for a uniform distribution in the interval $[10, 11]$.

3 Stochastic automata and GSMPs

The generalised semi-Markov process model (GSMP for short) [8, 25] is a general method to analyse and simulate discrete-event systems. It has been shown to be an effective tool to study complex and non-trivial systems.

Definition 3.1 A *generalised semi-Markovian process* (GSMP) is defined by the structure $(\mathcal{Z}, z_0, x_0, \mathcal{C}, \eta, \nu, F)$ where

- $\mathcal{Z}$ is the set of *(output) states*, with *initial state* $z_0 \in \mathcal{Z}$;
- $\mathcal{C}$ is the set of *clock events*, with the *initially triggered clock* $x_0 \in \mathcal{C}$;
- $\eta : \mathcal{Z} \to \wp(\mathcal{C})$, with $\eta(z_0) = \{x_0\}$, assigns a set of *active* clock events to each output state;
- $\nu : \mathcal{Z} \times \mathcal{C} \to \mathcal{Z}$ assigns the *next state* according to the current state and the clock that is triggered; and
- $F : \mathcal{C} \to (\mathbb{R} \to [0, 1])$ assigns to each clock a *continuous distribution function* such that $F(x)(0) = 0$; we write F_x instead of $F(x)$. □

We have restricted our attention to a subclass of GSMPs which is sufficient for our purposes. In fact, the only significant restriction is that the next state function is deterministic. That is, the next state is uniquely determined by the present state and the triggered clock event. In general GSMPs, this function is probabilistic, i.e., the next state is going to be

chosen with certain probability from a set of states. Two other minor restrictions are considered. First, the assignment of a distribution function to a clock event may depend on the history of the GSMP. In our framework, it only depends on the clock name. This is not a real restriction since we can introduce as many clock events as necessary to represent the more general GSMP. (Actually, each history of the general GSMP may become a clock event in the reduced GSMP.) Second, sometimes clocks are allowed to have different rates. This is not very usual, and moreover, under certain conditions, such "multirated" GSMPs can be represented by GSMPs where the clock rates are all 1, just like our case.

We remark also that, usually, the initial state of a GSMP is studied apart. For simplicity, we consider that the system has an initial state z_0 and that it was reached by triggering some clock x_0. We defined η such that, for the initial case, it is consistent with the original definition of GSMP.

A GSMP behaves as follows. Suppose that the system is in a certain state z. The active clocks in $\eta(z)$ will have some non-negative value and all other clock events (the inactive ones) have value ∞. The active clock with the smallest value is chosen to be triggered. Say x is that clock, and d its value. Notice that such a clock is unique with probability 1, since in GSMP all the clocks are continuous random variables. The next state is given by $\nu(z, x)$. The set of new clocks is given by $\eta(\nu(z,x))\backslash(\eta(z)\backslash\{x\})$, i.e., the clocks active in the new state which were not active before. All these new clocks are set according to their respective distribution function given by F. The old clocks which are still active take as a new value the previous value decreased d units of time (the value of x just before being triggered). Clocks which are not active are set to ∞. In this new state with the new valuation, the process is repeated.

As for stochastic automata, we have also defined an operational semantics for GSMPs in terms of probabilistic transition systems.

We show the relation between stochastic automata and GSMP by given a formal translation. This translation shows that the GSMP model is properly included in the stochastic automata model.

Definition 3.2 Let $\mathcal{G} = (\mathcal{Z}, z_0, x_0, \mathcal{C}, \eta, \nu, F)$ be a GSMP. The translation of $\mathcal{G}$ into a stochastic automaton, is defined by $\mathcal{M}(\mathcal{G}) \stackrel{\text{def}}{=} (\mathcal{Z} \times \wp(\mathcal{C}), (\nu(z_0, x_0), \emptyset), \mathcal{C}, \mathcal{C}, \longrightarrow, \kappa, F)$ where $\longrightarrow$ is defined by

$$\frac{x \in \eta(z)}{(z, C) \xrightarrow{x,\{x\}} (\nu(z, x), \eta(z)\backslash\{x\})}$$

and $\kappa(z, C) \stackrel{\text{def}}{=} \eta(z)\backslash C$. □

In the pair (z, C), C carries the information of which clocks were already active. Notice that there are much too many locations (z, C). In fact, the only "useful" (reachable) locations have the format $(\nu(z, x), \eta(z)\backslash\{x\})$ for some appropriate $z \in \mathcal{Z}$ and $x \in \eta(z)$. This can be notice by observing the source of the edge defined in the rule of Definition 3.2 and that the initial state is $(\nu(z_0, x_0), \emptyset) = (\nu(z_0, x_0), \eta(z_0)\backslash\{x_0\})$. Moreover, notice that $\kappa(\nu(z, x), \eta(z)\backslash\{x\}) = \eta(\nu(z, x))\backslash(\eta(z)\backslash\{x\})$, which is the set of new clock events in the output state $\nu(z, x)$. Besides, for each active clock in the output state z, there is an output edge from any location (z, C), that is, $\eta(z) = \bigcup\{x \mid (z, C) \xrightarrow{x,\{x\}}\}$.

It is clear that a translation is not possible in general in the other way around since the stochastic automata model not only allows a more general class of distribution function, but

also allows non-determinism. We have proven that the translation given in Definition 3.2 preserves probabilistic bisimulation equivalence.

4 A stochastic process algebra

In the following we introduce a stochastic process algebra. The methodology that we follow to define the syntax and the semantics is inspired by [7] where a process algebra for timed automata was introduced.

Let $\mathbf{A}$ be a set of *actions*. Let $\mathcal{CN}$ be a set of clock names and $\mathcal{DF}$ a set of distribution functions. We define $\mathcal{C} \stackrel{\text{def}}{=} \mathcal{CN} \times \mathcal{DF}$ to be the set of clocks. We denote x_G for $(x, G) \in \mathcal{C}$. We define the distribution assignment function $F : \mathcal{C} \to (\mathbb{R} \to [0,1])$ by $F(x_G) \stackrel{\text{def}}{=} G$.

Definition 4.1 Let $\mathbf{V}$ a set of *process variables*. The syntax of the process algebra $\mathcal{L}$ is defined according to the following grammar:

$$p ::= \mathbf{stop} \mid a;p \mid C \mapsto p \mid p + p \mid \{\!|C|\!\}\ p \mid p||_A p \mid p[f] \mid X$$

where $C \subseteq \mathcal{C}$, $a \in \mathbf{A}$, $A \subseteq \mathbf{A}$, $f : \mathbf{A} \to \mathbf{A}$, and $X \in \mathbf{V}$. A *recursive specification* E is a set of *recursive equations* having the form $X = p(\mathbf{V})$ for each $X \in \mathbf{V}$, where $p(\mathbf{V}) \in \mathcal{L}$. Every recursive specification has a distinguished process variable called *root*. □

Process **stop** represents inaction; it is the process that cannot perform any action. The intended meaning of $a;p$ (named *(action-)prefixing*) is that action a must be performed as soon as possible followed by the execution of p. $C \mapsto p$ is the *triggering condition*; process p is executed as soon as all the clocks in C have expired. $p + q$ is the *choice*; it executes the fastest of processes p and q, and if both of them become enabled at the same time, the choice is made non-deterministically. The *clock setting operation* $\{\!|C|\!\}\ p$ sets the clocks in C according to their respective distribution function. We choose a LOTOS-like parallel composition. Thus, $p||_A q$ executes p and q in parallel, and they are synchronised by actions in A. We should remark that synchronisation happens as soon as all the processes are ready to do it. This happens straightforwardly by considering the union of the triggering sets. Finally, the *renaming* operation $p[f]$ is a process that behaves like p except that actions are renamed by f. We will assume the following precedence among the operators: $+ < ||_A < \{\!|C|\!\} = C\mapsto = a; < [f]$.

As a simple example, we give the specification of the switch described in Section 2.

$$\begin{aligned}
Arrival &= \{\!|x_G|\!\}\ \{x_G\} \mapsto on; Arrival \\
Switch_{off} &= on; Switch_{on} \\
Switch_{on} &= on; Switch_{on} + \{\!|y_K|\!\}\ \{y_K\} \mapsto off; Switch_{off} \\
System &= Arrival||_{\{on\}} Switch_{off}
\end{aligned} \tag{2}$$

In this case G is an exponential distribution with rate λ and K gives probability 1 to the value 60. Process *Arrival* models the arrival of people which occurs with exponential distribution of rate λ. *Switch* models the switch itself which initially is *off*. Notice that the switch is always enabled to accept an "*on*" and hence no clock controls this activity on the switch part of the system. Process *System* describes the whole system, allowing people to turn on the switch, i.e., process *Arrival* and *Switch* should synchronise on the action *on*.

In the sequel, we need the notion of free and bound clock variables. Let $p \in \mathcal{L}$. A clock x is free in p if it has a subterm $C{\mapsto}q$ such that $x \in C$ which do not appear in a context $\{\!|C'|\!\}\ldots$ with $x \in C'$. A clock x is bound in p if it has a subterm $\{\!|C|\!\}\ q$ such that $x \in C$. We denote by $fv(p)$ and $bv(p)$ the sets of free and bound clock variables respectively.

To each term in the language we associate a stochastic automaton. In order to define the automaton associated to a parallel composition, we need to consider the additional operation $\overline{\mathsf{ck}}$. $\overline{\mathsf{ck}}(p)$ is a process that behaves like p except that no clock is set at the very beginning. We only allow occurrences of $\overline{\mathsf{ck}}$ within the scope of static operations, namely, parallel composition and renaming operation. The sets of free and bounded variables for $\overline{\mathsf{ck}}(p)$ are defined by $fv(\overline{\mathsf{ck}}(p)) = fv(p) \cup \kappa(p)$ and $bv(\overline{\mathsf{ck}}(p)) = bv(p)$, where κ is defined in Table 1.

To associate a stochastic automaton to a given term in the language, we need to define the different parts of the stochastic automaton. We start by defining predicates κ and $\longrightarrow$ as the least relations satisfying rules in Table 1. However, not all the processes can have a straightforward stochastic automaton as a semantic interpretation. To associate a stochastic automaton to a term, clocks names must be considered with care as we see as follows.

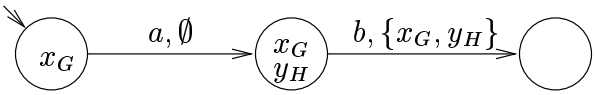
Table 1: Stochastic automata for $\mathcal{L}$ $(X = p \in E)$

$\kappa(\mathbf{stop}) = \emptyset$	$\kappa(\{\!|C|\!\}\ p) = C \cup \kappa(p)$	$\kappa(p+q) = \kappa(p) \cup \kappa(q)$
$\kappa(a;p) = \emptyset$	$\kappa(C{\mapsto}p) = \kappa(p)$	$\kappa(p\|\|_A q) = \kappa(p) \cup \kappa(q)$
$\kappa(X) = \kappa(p)$	$\kappa(p[f]) = \kappa(p)$	$\kappa(\overline{\mathsf{ck}}(p)) = \emptyset$
$a;p \xrightarrow{a,\emptyset} p$	$\dfrac{p \xrightarrow{a,C'} p'}{\{\!|C|\!\}\ p \xrightarrow{a,C'} p'}$	$\dfrac{p \xrightarrow{a,C} p'}{\begin{array}{l} p\|\|_A q \xrightarrow{a,C} p'\|\|_A \overline{\mathsf{ck}}(q) \\ q\|\|_A p \xrightarrow{a,C} \overline{\mathsf{ck}}(q)\|\|_A p' \end{array}}\ a \notin A$
$\dfrac{p \xrightarrow{a,C} p'}{\begin{array}{l} p+q \xrightarrow{a,C} p' \\ q+p \xrightarrow{a,C} p' \end{array}}$	$\dfrac{p \xrightarrow{a,C'} p'}{C{\mapsto}p \xrightarrow{a,C\cup C'} p'}$	$\dfrac{p \xrightarrow{a,C} p' \quad q \xrightarrow{a,C'} q'}{p\|\|_A q \xrightarrow{a,C\cup C'} p'\|\|_A q'}\ a \in A$
$\dfrac{p \xrightarrow{a,C} p'}{X \xrightarrow{a,C} p'}$	$\dfrac{p \xrightarrow{a,C} p'}{p[f] \xrightarrow{f(a),C} p'[f]}$	$\dfrac{p \xrightarrow{a,C} p'}{\overline{\mathsf{ck}}(p) \xrightarrow{a,C} p'}$

Consider the process

$$p_1 \equiv \{\!|x_G|\!\}\ (a; \{x_G\}{\mapsto}(\{\!|x_G, y_H|\!\}\ \{y_H\}{\mapsto}b; \mathbf{stop})) \tag{3}$$

The second occurrence of x_G is intended to be bound to the outermost clock setting as shown by the grey arrow. Using the rules in Table 1, the following stochastic automaton would be obtained



In this sense, x_G would be captured by the innermost clock setting as shown by the black arrow in formula (3). Therefore, we consider that clocks are different if they are set in different places, although they may have the same name.

Situations of clock capture also occur in contexts with summations and parallel composition. Consider the process $p_2 \equiv \{\!|x_G|\!\}\ \{x_G\}\!\mapsto\! a; \mathbf{stop} + \{\!|x_G|\!\}\ \{x_G\}\!\mapsto\! b; \mathbf{stop}$ where G is some continuous distribution function. p_2 should not be considered equivalent to $p_3 \equiv \{\!|x_G|\!\}\ (\{x_G\}\!\mapsto\! a; \mathbf{stop} + \{x_G\}\!\mapsto\! b; \mathbf{stop})$. Intuitively, in p_2, a and b are enabled at the same time with probability 0 (i.e. is improbable to have a non-deterministic choice between a and b), because the clocks responds to different settings. Instead, our intuition says that the process p_3 *always* enables a and b at the same time because they respond to exactly the same setting of the same clock. In a naive interpretation of p_2, its associated automaton would become isomorphic to that one of p_3 which contradicts our intuition.

Similarly, an attempt to define the associated stochastic automaton of process $p_4 \equiv \{\!|x_G|\!\}\ \{x_G\}\!\mapsto\! a; \mathbf{stop} ||_\emptyset \{\!|x_G|\!\}\ \{x_G\}\!\mapsto\! b; \mathbf{stop}$, would unify the two independent clocks x_G going again against our intuition.

In this sense, we need to characterise the set of processes that do not have *conflict of variables*. A process does not have conflict of variables if for every of its subterms p, the following conditions holds:

1. $p \equiv C\!\mapsto\! q$ implies $C \cap \kappa(q) = \emptyset$
2. $p \equiv q + q'$ implies $\kappa(q) \cap \kappa(q') = fv(q) \cap \kappa(q') = \kappa(q) \cap fv(q') = \emptyset$
3. $p \equiv q||_A q'$ implies $bv(q) \cap var(q') = var(q) \cap bv(q') = \emptyset$

Notice that all processes defined above have conflict of variable.

Definition 4.2 For every process p such that p does not have conflict of variable, the *stochastic automaton associated to* p is defined by $[\![p]\!]^S \stackrel{\text{def}}{=} (\mathcal{L}, p, \mathcal{C}, \mathbf{A}, \longrightarrow, \kappa, F)$. □

The reader is invited to check that processes for the switch system defined in (2) do not have conflict of variable, and that the stochastic automaton associated to the process *System* is the one depicted in Figure 1 up to the consideration that for every process p, $\overline{\mathsf{ck}}(\overline{\mathsf{ck}}(p))$ is isomorphic to $\overline{\mathsf{ck}}(p)$.

The restriction to processes which do not have conflict of variable is not an actual problem, since we can always α-convert properly any process into one that does not have conflict of variables. With "properly" we mean that the distribution function associated to the clock must be preserved. For instance, p_4 can be α-converted into $\{\!|x_G|\!\}\ \{x_G\}\!\mapsto\! a; \mathbf{stop} ||_\emptyset \{\!|y_G|\!\}\ \{y_G\}\!\mapsto\! b; \mathbf{stop}$. In this way, any process in the language has an associated stochastic automaton modulo α-conversion.

The process algebra introduced below has the property of expressing any (finitely branching) stochastic automaton. The proof of the following theorem follows closely the ideas of a similar theorem in [7].

Theorem 4.3 *For every finitely branching*[1] *stochastic automaton SA there is a guarded recursive specification*[2] *E with root X such that the reachable part of SA and the reachable part of* $[\![X]\!]^S$ *are isomorphic.*

Proof (Sketch). The proof consists of associating a process variable to each location s of SA and defining each one of them as the term that sets the clocks of $\kappa(s)$ over the summation of the outgoing edges represented by prefixings with its respective triggering sets as follows. Let $SA = (\mathcal{S}, \mathbf{A}, \mathcal{C}, s_0, \longrightarrow, \kappa, F)$. For each location $s \in \mathcal{S}$ define a different variable X_s. Define the recursive specification E with root X_{s_0} and recursive equations

$$X_s = \{\!|\kappa(s)|\!\} \left(\sum \{C \mapsto a; X_{s'} \mid s \xrightarrow{a,C} s'\} \right)$$

where $\sum\{p_i \mid i \in \{1, \ldots, n\}\} \stackrel{\text{def}}{=} p_1 + p_2 + \cdots + p_n$. In particular, $\sum \emptyset \stackrel{\text{def}}{=} \mathbf{stop}$.

Now, we restrict SA and $[\![X_{s_0}]\!]^S$ to their reachable parts, and the isomorphism is given by the function that maps every location in SA into its corresponding variable in $[\![X_{s_0}]\!]^S$. □

5 An example: the CSMA/CD protocol

In this section we use the models introduced above to describe the carrier sense multiple access protocol with collision detection (CSMA/CD). The CSMA/CD is widely used on LANs in the multiple access control (MAC) sublayer, in particular, in the IEEE 802.3 standard based on the well known Ethernet system (see [27] for more information). Formal verifications of this protocol have been studied in [20, 10, 19].

The informal description of the protocol is as follows. When a station has data to send, it first listens to the channel to check if there is some transmission at that moment. If the channel is busy, the station waits a random amount of time and retries to transmit. When the channel is idle, the station may transmit a frame. However, it could be the case that two stations check at the same time that the channel is idle and both of them start to transmit. This would cause a collision of both transmissions. In this case the stations detect the collision and abort their transmissions immediately. After waiting a random amount of time, the stations try to transmit again the same frame.

The propagation delay of messages along the channels plays an important role. Before, we said that "two stations can check at the same time that the channel is idle". Of course this situation is improbable. What actually happens is that one station starts to transmit and the other checks the channel before the message has reached it. So, the second station "believes" the channel is idle and transmits, which causes the collision.

We consider the propagation delay to be uniformly distributed between 0 and Δ, that is, Δ is the maximal propagation delay. When a collision occurs the caused noise burst has to travel back to the original station, thus, in the worst case, 2Δ units of time will be need to hear a collision. So, we model the waiting time to retry as a random variable distributed with a uniform distribution between 0 and 2Δ.

[1] A stochastic automaton is *finitely branching* if for every location s its set of outgoing arrows $\{s \xrightarrow{a,C} s' \mid a \in \mathbf{A}, C \in \mathcal{C}, s' \in \mathcal{S}\}$ is finite.

[2] A process variable is *guarded* if all its occurrences appear in a context of a prefix. A recursive specification E is *guarded* if $X = p \in E$ implies that all process variables in p are guarded.

The station behaves as follows. Messages to be sent arrive in intervals distributed exponentially[3] with rate Λ. We denote this distribution function by F_Λ. The station listens if the channel is busy during a time uniformly distributed between 0 and a small δ before it begins to transmit. If the channel is busy, the station waits randomly according to a uniform distribution over $[0, 2\Delta]$ before retry. Messages have a minimal redundancy that takes k units of time to be transmitted and the duration of the rest is exponentially distributed with rate λ. We denote F^{k}_λ such a distribution function. If a collision occurs before the transmission is completed, the station behaves just as if the channel were busy. In the following we model the station. We abbreviate $\{\!|z_F|\!\}\ \{z_F\}\!\mapsto\! a; P$ by $a(F); P$ provided $z_F \notin fv(P)$. This is a usual notation in stochastic process algebras. In addition, we write U^a_b to refer to a uniform distribution function in the interval $[a, b]$.

$$
\begin{aligned}
STATION &= send(F_\Lambda); START \\
START &= busy; RETRY + begin(U^\delta_0); TRANS \\
TRANS &= end(F^{\mathsf{k}}_\lambda); STATION + cd; RETRY \\
RETRY &= retry(U^{2\Delta}_0); START
\end{aligned}
$$

The channel behaves as follows. When it is idle, any station may begin. Once some station begins to transmit, the signal will propagate along the channel within Δ units of time. Afterwards, all stations are able to listen a busy channel. We consider such a time to be distributed according to a uniform distribution over $[0, \Delta]$. If some other station begins to transmit before the propagation of the first signal is complete, a collision occurs. All the stations will be able to listen such a collision within Δ units of time (distributed uniformly). Notice that some stations may interpret the collision just as a busy channel, since they may have not started any transmission. After exactly Δ units of time, every station has been able to detect the collision. Notice that this time is not probabilistic (or with probability 1), so we model it to be distributed according to the function

$$
K_\Delta(t) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } t < \Delta \\ 1 & \text{if } t \geq \Delta \end{cases}
$$

If the message is sent successfully, the end of the signal will take some time to be propagated, which will keep the channel busy for a while. Such a time is distributed according to a uniform in $[0, \Delta]$. The model of the channel is as follows. We assume i range between 1 and n, where n is the amount of stations.

$$
\begin{aligned}
CHAN &= \textstyle\sum_i begin_i; INUSE_i \\
INUSE_i &= safe(U^\Delta_0); BUSY_i \\
&\quad + \textstyle\sum_j begin_j; \{\!|x^1_{U^\Delta_0}, \ldots, x^n_{U^\Delta_0}, y_{K_\Delta}|\!\}\ CDETECT_\emptyset \\
BUSY_i &= \textstyle\sum_{j \neq i} busy_j; BUSY_i + end_i; \{\!|x_{U^\Delta_0}|\!\}\ REMANENCE
\end{aligned}
$$

[3] Actually, we are not going to model this part neatly as a queue on top of the station, but we sloppily include this information directly into the station specification.

$$
\begin{aligned}
REMANENCE &= \textstyle\sum_i busy_i; REMANENCE + \{x_{U_0^\Delta}\}\mapsto ok; CHAN \\
CDETECT_I &= \textstyle\sum_i begin_i; CDETECT_I \\
&\quad + \textstyle\sum_{i\notin I}\{x^i_{U_0^\Delta}\}\mapsto \left(cd_i; CDETECT_{I\cup\{i\}} + busy_i; CDETECT_{I\cup\{i\}}\right) \\
&\quad + \textstyle\sum_{i\in I} busy_i; CDETECT_I \\
&\quad + \{y_{K_\Delta}\}\mapsto ok; CHAN
\end{aligned}
$$

For all $i \in \{1, \ldots . n\}$, we define the renaming function f_i by $f_i(a) = a_i$ for all $a \in \mathbf{A}$. The complete system is modelled by

$$
SYSTEM = \left(STATION[f_1] \;||_\emptyset \cdots ||_\emptyset \; STATION[f_n]\right) \;||_{\{busy_i, cd_i, begin_i, end_i\}} \; CHAN
$$

In Figures 3 and 4, we depict the station and the channel, respectively. We have omitted the subindices of distribution functions. Instead we enumerate them beside the automata. In particular, in the channel we have draw some grey coloured edges. Although they actually appear in the channel and in the complete system, the reader may check that they cannot happen, or better, they happen with probability 0.

6 Further discussions

Conclusions and further work. We introduced a new model to represent stochastic systems. We have compared it to a useful model in performance analysis, namely GSMPs, and we showed that GSMPs are properly contained in our model. In addition, we defined a stochastic process algebra whose expressivity is richer than existing ones. We show that the process algebra and its underlying semantic model, the stochastic automata, are equally expressive.

The direction of our current work is two fold. On the one hand, we address the study of stochastic systems by using stochastic automata and, on the other, we study the compositional properties of the stochastic process algebra and its axiomatisation.

Regarding the first direction, as it was already said, we have defined a semantics of stochastic automata in terms of probabilistic transition systems. This semantics leads to an algorithm for discrete event simulation. We use the notion of adversaries or schedulers [29, 24] to resolve non-deterministic choices. Since parallel composition of stochastic automata can be easily defined (actually, it is defined just like for the process algebra, see Table 1), the simulation algorithm can compose the complete stochastic automaton on the fly, which reduces the state space explosion problem. Although (probabilistic) adversaries allow to obtain a complete probabilistic final model, the inclusion of them as a new ingredient is not that appealing since it would require an additional effort when modelling systems. Because of that, our current work is to characterise the set of stochastic automata in whose underlying transition system every choice is resolved probabilistically.

Although simulation is a powerful tool from the performance point of view, analytical methods are far more effective to study the correctness of a system. Usually errors are events with low probability, so, the use of simulation may not guarantee that they are not present or that their probability is low enough to be considered. Model checking has proven to be a powerful and simple tool to verify timed systems. Some early papers like [1] have shown the possibility of borrowing ideas from model checking on timed automata and applying them

Figure 3: The station in the CSMA/CD protocol

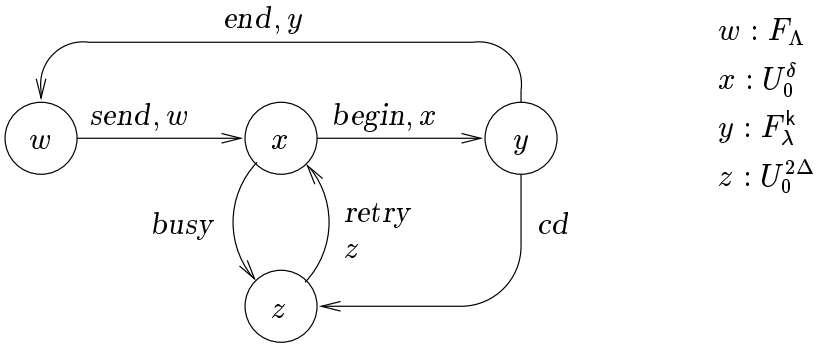
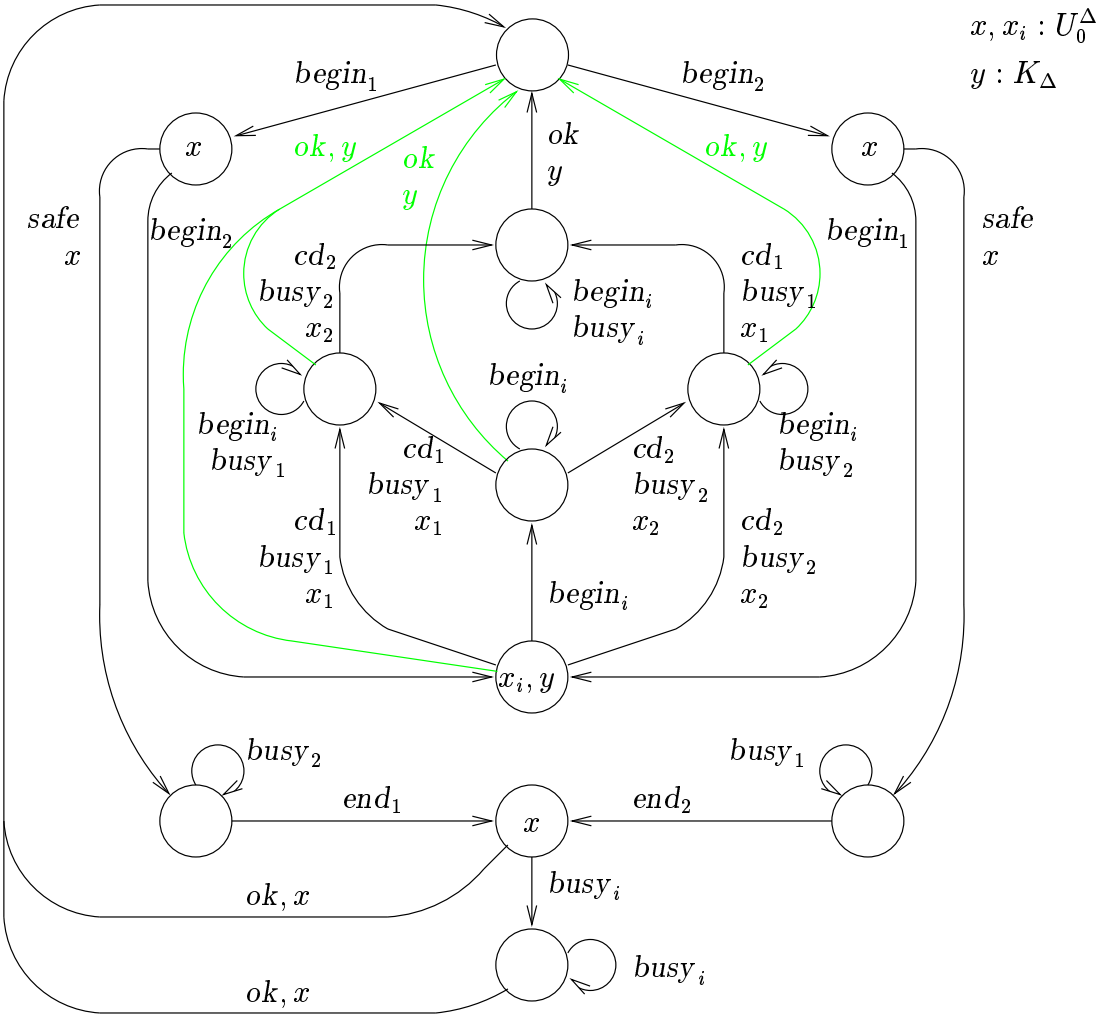


Figure 4: The channel in the CSMA/CD protocol



to stochastic systems. Our work will also address the use of model checking on stochastic automata.

Regarding the process algebra, we are studying congruence results. Probabilistic bisimulation turns out not to be a congruence. An easy example is as follows. Processes $p_1 \equiv a;\mathbf{stop} + \{\!|x_G|\!\}\ \{x_G\}\mapsto b;\mathbf{stop}$ and $p_2 \equiv a;\mathbf{stop} + \{\!|x_G|\!\}\ \{x_G\}\mapsto c;\mathbf{stop}$ $(b \neq c)$ are probabilistically bisimilar if $G(0) = 0$, since in both cases, only the action a at time 0 can be performed (with probability 1). However, $p_1||_{\{a\}}\mathbf{stop}$ and $p_2||_{\{a\}}\mathbf{stop}$ are not bisimilar. In this context, the execution of action a is preempted since there is no possible synchronisation, then b or c may happen (at a certain time greater than 0). We are now studying a strictly finer relation which takes into account potential activity of a given process. Besides, our current work also involves the study of possible axiomatisations. We are not aiming to be complete with respect to some equivalence. We expect to give a set of axioms that allows to reduce any process into some basic term using the basic operation, namely, prefixing, summation, triggering, and clock setting. As we have discussed in the introduction, we know it is possible to have an expansion law. In fact, to obtain a finite axiomatisation of the parallel composition, we are considering to include the additional operations, $\lfloor\!\lfloor_A$ (left-merge) and $|_A$ (communication merge). The axiomatisation resembles in many cases the work done in [7].

Related work on stochastic process algebra and discrete event simulation. In this paper we established a link between stochastic process algebras and discrete-event simulation, in particular the more formal model of GSMPs. To our knowledge the use of (a variation of) GSMPs as a semantic model for stochastic process algebras is novel. Katoen et. al. [17] indirectly used GSMPs as a semantic model: they map a non-Markovian stochastic process algebra onto event structures, and obtain for a subclass of event structures (the so-called stochastic deterministic ones) a GSMP. The intermediate model, event structures, has become obsolete in our approach. In addition, for recursive processes infinite event structures are obtained which makes the approach of [17] less suited for the use of efficient regenerative simulation techniques. The finite representations we obtain do not suffer from this problem.

The relation between process algebras and discrete-event simulation models has been studied from several perspectives. Harrison & Strulo [11] developed a stochastic process algebra to formally describe discrete-event simulation. They defined an operational semantics using action, timed, and probabilistic transition relations and developed notions like weak and strong bisimulation (plus axiomatisations). Although their work is closely related to ours, our model appears to be more intuitive and resembles more closely the conceptual ideas of realistic simulation languages. In particular, measure theory comes in in our case when defining the formal interpretation of stochastic automata, not directly in the semantics of the language. Pooley [21] investigated the mapping of a high-level language for describing discrete-event simulation models, baptised extended activity diagrams, onto the timed process algebra TCCS and the process-based simulation language DEMOS (Discrete-Event Modelling On Simula) [4]. Using this framework Pooley is able to check certain properties of a model a priori to simulation, by analysing the (T)CCS specification. In this extensive work distribution functions were neglected and the role of process algebra with respect to simulation is different from ours. Tofts and Birtwistle use process algebras, basically CCS and its synchronous variant SCCS, to provide a denotational semantics of DEMOS [5]. They focus on analysing properties like absence of deadlock and livelock and do not consider timing aspects.

We conclude by mentioning some other approaches in the field of stochastic process algebras dealing with non-exponential distributions. A partial-order semantics for such process

algebras was introduced by Brinksma et. al. [6]; this model was used in [17] to obtain GSMPs as mentioned above. Priami [22] extended his stochastic extension of the π-calculus with general distributions. He used operational semantics and basically decorated the transition relation in such a way that causality information can be easily obtained from that in a post-processing phase. To our knowledge no mapping onto a performance model has been given yet. Herzog [14] used stochastic task graphs, a performance model for which efficient numerical analysis methods exist, as a semantic model of a (deterministic) process algebra. Finally, we mention the recent work of Tofts [28] who uses a weighted synchronous version of CCS (WSCCS) to represent general (discrete) distributions and compositions thereof. When focusing on bounded probabilities he shows how to perform certain performance assessments in a compositional way.

References

[1] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for probabilistic real-time systems. In J. Leach Albert, B. Monien, and M. Rodríguez, editors, *Proceedings* 18^{th} *ICALP,* Madrid, *LNCS* 510, pages 113–126. Springer-Verlag, 1991.

[2] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] M. Bernardo and R. Gorrieri. Extended markovian process algebra. In U. Montanari and V. Sassone, editors, *Proceedings CONCUR 96,* Pisa, Italy, *LNCS* 1119 of pages 314–330. Springer-Verlag, 1996.

[4] G.M. Birtwistle. *Discrete Event Modelling on Simula.* MacMillan, 1979.

[5] G.M. Birtwistle and C. Tofts. Process semantics for simulation. Technical report, University of Swansea, 1996.

[6] E. Brinksma, J.-P. Katoen, R. Langerak, and D. Latella. A stochastic causality-based process algebra. *The Computer Journal*, 38(7):552–565, 1995.

[7] P.R. D'Argenio and E. Brinksma. A calculus for timed automata (Extended abstract). In B. Jonsson and J. Parrow, editors, *Proceedings of the 4th International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems,* Uppsala, Sweden, *LNCS* 1135, pages 110–129. Springer-Verlag, 1996.

[8] P.W. Glynn. A GSMP formalism for discrete event simulation. *Proceedings of the IEEE*, 77(1):14–23, 1989.

[9] N. Götz, U. Herzog, and M.Rettelbach. TIPP - Introduction and application to protocol performance analysis. In H. König, editor, *Formale Beschreibungstechniken für verteilte Systeme*, FOKUS series. Saur Publishers, 1993.

[10] H.A. Hansson. *Time and Probability in Formal Design of Distributed Systems*, volume 1 of *Real–Time Safety Critical Systems.* Elsevier, 1994.

[11] P. Harrison and B. Strulo. Stochastic process algebra for discrete event simulation. In F. Bacelli, A. Jean-Marie, and I. Mitrani, editors, *Quantitative Methods in Parallel Systems*, Esprit Basic Research Series, pages 18–37. Springer-Verlag, 1995.

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

[13] H. Hermanns and M.Rettelbach. Syntax, semantics, equivalences, and axioms for MTIPP. In U. Herzog and M. Rettelbach, editors, *Proc. of the 2nd Workshop on Process Algebras and Performance Modelling*, pages 71–87. University of Erlangen, July 1994.

[14] U. Herzog. A concept for graph-based stochastic process algebras, generally distributed activity times, and hierarchical modelling. In Ribaudo [23], pages 1–20.

[15] J. Hillston. *A Compositional Approach to Performance Modelling.* Distinguished Dissertation in Computer Science. Cambridge University Press, 1996.

[16] J.-P. Katoen. *Quantitative and Qualitative Extensions of Event Structures.* PhD thesis, Department of Computer Science, University of Twente, April 1996.

[17] J.-P. Katoen, E. Brinksma, D. Latella, and R. Langerak. Stochastic simulation of event structures. In Ribaudo [23], pages 21–40.

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

[19] X. Nicollin, J. Sifakis, and S. Yovine. Compiling real-time specifications into extended automata. *IEEE Transactions on Software Engineering*, 18(9):794–804, September 1992.

[20] J. Parrow. Verifying a CSMA/CD-protocol with CCS. In S. Aggarwal and K. Sabnani, editors, *Protocol Specification, Testing, and Verification, VIII,* Atlantic City, NJ, USA, pages 373–384. North-Holland, June 1988.

[21] R.J. Pooley. Integrating behavioural and simulation modelling. In H. Beilner and F. Bause, eds, *Quantitative Evaluation of Computing and Communication Systems*, LNCS 977, pages 102–116. Springer-Verlag, 1995.

[22] C. Priami. Stochastic π-calculus with general distributions. In Ribaudo [23], pages 41–57.

[23] M. Ribaudo, editor. *Proc. of the 4th Workshop on Process Algebras and Performance Modelling*, Torino, Italy, 1996. Università di Torino.

[24] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[25] G.S. Shedler. *Regenerative Stochastic Simulation.* Academic Press, 1993.

[26] J. Sifakis and S. Yovine. Compositional specification of timed systems. In *Proceedings of the 13th Annual Symp. on Theoretical Aspects of Computer Science, STACS'96*, *LNCS* 1046, pages 347–359, Grenoble, France, 1996. Springer-Verlag.

[27] A.S. Tanenbaum. *Computer Networks.* Prentice-Hall International, third edition, 1996.

[28] C. Tofts. Compositional performance evaluation. In E. Brinksma, editor, *Proceedings of the Third Workshop on Tools and Algorithms for the Construction and Analysis of Systems,* Enschede, The Netherlands, *LNCS* 1217, pages 290–305. Springer-Verlag, April 1997.

[29] M.Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In 26^{th} *Annual Symposium on Foundations of Computer Science, Portland, Oregon*, pages 327–338. IEEE Computer Society Press, 1985.