# General Distributions in Process Algebra

Joost-Pieter Katoen and Pedro R. D'Argenio

Formal Methods and Tools Group, Dept. of Computer Science
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

**Abstract.** This paper is an informal tutorial on stochastic process algebras, i.e., process calculi where action occurrences may be subject to a delay that is governed by a (mostly continuous) random variable. Whereas most stochastic process algebras consider delays determined by negative exponential distributions, this tutorial is concerned with the integration of *general, non-exponential* distributions into a process algebraic setting. We discuss the issue of incorporating such distributions in an interleaving semantics, and present some existing solutions to this problem. In particular, we present a process algebra for the specification of stochastic discrete-event systems modeled as generalized semi-Markov chains (GSMCs). Using this language stochastic discrete-event systems can be described in an abstract and modular way. The operational semantics of this process algebra is given in terms of stochastic automata, a novel mixture of timed automata and GSMCs. We show that GSMCs are a proper subset of stochastic automata, discuss various notions of equivalence, present congruence results, treat equational reasoning, and argue how an expansion law in the process algebra can be obtained. As a case study, we specify the root contention phase within the standardized IEEE 1394 serial bus protocol and study the delay until root contention resolution. An overview of related work on general distributions in process algebra and a discussion of trends and future work complete this tutorial.

## 1 Introduction

The design and analysis of systems, like embedded systems, communication protocols or multi-media systems, requires insight in not only the functional, but also in the real-time and performance aspects of applications involved. Researchers in formal methods (i.e., concurrency theory) have recognized the need for the additional support of quantitative aspects, and various initiatives have been taken to accomplish such support. A prominent example is the treatment of real-time constraints, where specification formalisms like timed automata [2] have emerged, and impressive progress has been made in the development of efficient verification algorithms [15,72]. This has resulted in a number of tools (model checkers) that provide interesting experimental platforms for industrial case studies.

*Hard and soft real-time constraints.* Constraints that one typically considers in this real-time setting are 'hard', for instance,

"the system must always do a certain activity before time $t$"

E. Brinksma, H. Hermanns, and J.-P. Katoen (Eds.): FMPA 2000, LNCS 2090, pp. 375–429, 2001.

For many applications, though, real-time constraints are typically less stringent. Rather than requiring that certain activities *must always* occur before time $t$, in practice one is usually interested in more 'soft' real-time constraints, where a system is required to perform the activity *mostly* before $t$. The soft real-time requirements of systems typically address their performance characteristics, and are often also referred to as their quality-of-service (QoS) parameters. They are usually related to stochastic aspects of various forms of time delay, such as, for example, mean and variance of message transfer delay, service waiting times, failure rates, utilization, etc. In a soft real-time system one typically considers constraints like:

"the system should perform an activity before time $t$ in 92% of the cases"

In soft real-time systems, state changes take place in a discrete fashion and the time of occurrence of activities is controlled by random variables. These systems are also known as *stochastic discrete-event simulation* (DES) models. In contrast to most formalisms that are restricted to a particular set of probability distributions, like negative exponential or discrete distributions, the objective is to support *general* distributions, discrete or continuous. This makes the formalism more expressive and more interesting from a practical point of view.

*The need for a single framework.* Traditionally, there has been a clear separation between the functional and performance aspects of systems, and as a result different communities have constructed and analyzed their own, largely unrelated models for the aspects under their responsibility. This has resulted in what has been recognized as "the insularity problem of performance evaluation in the system design process" [47]. In modern systems, though, the difference between functional and performance features has become blurred, and both features are becoming of comparable interest. Thus, it would be beneficial to be able to check how changes in functionality affect performance issues, and vice versa. In addition, one would like to have a better relationship between the models that are used for qualitative and quantitative analysis, and avoid the use of different models that are mutually incompatible. A single framework where both aspects could be defined would be highly desirable [24,37]. This tutorial is focused on an integrated approach using *process algebra*.

### 1.1 Organization of the Paper

Section 2 contains an introduction on stochastic process algebra (to what extent do they differ from traditional process algebra?), justifies the usage of general distributions (why do we need them?), sketches the complications of incorporating general distributions in process algebra (why are things not so straightforward as for exponential distributions?), and provides an overview of possible solutions that have been suggested so far. Section 3 introduces generalized semi-Markov processes (GSMPs), a model for general distributions. Section 4 presents a couple of small examples that serve as a justification for providing a process algebraic framework for this model. Section 5 presents stochastic automata, an extension

of labelled transition systems that we use as a semantic model of our process algebra with general distributions, called SPADES (Stochastic Process Algebra for Discrete-Event Simulation) and symbolized as  . Section 7 gives an account of evaluation techniques for  specifications, including quantitative methods – discrete-event simulation – and qualitative methods – checking of (timed) safety properties. Section 9 discusses related work on process algebras with general distributions. Section 10 provides a summary of the tutorial and presents some topics for further research.

## 2 Fitting General Distributions in Process Algebra

In traditional process algebras, like ACP [39], CCS [77], CSP [58] and LOTOS [64], a (possibly concurrent) system is syntactically represented using powerful composition operators which facilitate the development of modular and well-structured specifications. The formal meaning of a process algebra term is defined in a mathematical model. By defining an appropriate equivalence relation on this model one is able to formally compare and transform (e.g., simplify or reduce) specifications. If, in addition, this relation is a congruence[1], then such transformation can be carried out component by component. This compositional nature reduces the complexity of the transformation significantly. Finally, due to the algebraic nature of the formalism it is possible to define equational rules on the syntax that allow to perform step-wise design and minimization at a purely syntactic level, without any reasoning in semantic terms.

### 2.1 Stochastic Process Algebra

Traditionally, process algebras have concentrated on the functional aspects of systems such as their observable behavior, control flow and synchronization as properties in relative time. In the late eighties, the interest grew in extending process algebras with quantitative information like time and (discrete) probabilities. These extensions are known as timed and probabilistic process algebras, respectively.

*Timed process algebra.* Timed extensions of ACP [5], CCS [78,99], CSP [89] and LOTOS [13,73] have been defined. The basic idea underlying timed process algebras is to change the role of action-prefix, denoted by $a;p$ for action $a$ and process $p$. Originally, the expression $a;p$ simply means that first an action $a$ is offered, and after the appearance of $a$ the process behaves like $p$. No statement is made about when action $a$ occurs. In timed process algebra there are basically two schools:

- replace $a;p$ by $(a,t);p$ denoting that action $a$ is offered after a delay of $t$ time units, or

[1] An equivalence relation is a congruence if two equivalent terms behave indistinguishable in any context.

– extend the language with a timed prefix like $t \mapsto p$ denoting that process $p$ is reached after a delay of $t$ time units; $t \mapsto a; p$ means that action $a$ is offered after $t$ time units.

(There are several finer points that we ignore here; see [80] for an overview.) The last distinction leads to a behavior where two distinct phases are separated. Phases, during which one or more actions occur together with their corresponding state changes, but where no time elapses, are distinguished from phases where time passes, but during which no actions happen. With some appropriate modifications, timed process algebras can be considered as high-level specification formalisms for timed automata [29].

*Probabilistic process algebra.* Probabilistic extensions of ACP [6], CCS [44], CSP [75] and LOTOS [76] have been studied. A recent overview of probabilistic process algebras can be found in [65]. The basic idea of these calculi is to incorporate a probabilistic choice operator that allows terms like $p \oplus_\pi q$ (with $\pi \in (0,1)$) where process $p$ can be selected with probability $\pi$ and process $q$ with $1-\pi$. Different semantic models have been used for probabilistic process calculi, depending on whether non-determinism is allowed or not. In the deterministic case, these languages represent discrete-time Markov chains (DTMC) [68]; in presence of non-determinism, models similar to Markov decision processes [34] are obtained.

*Markovian process algebra.* In stochastic process algebras, time and probability are integrated by considering delays of a continuous probabilistic nature. In languages like EMPA [9], PEPA [56] or TIPP [43], a non-negative real-valued rate is associated to actions that determines probabilistically the delay prior to an action. For rate $\lambda$, the term $(a, \lambda); p$ denotes that action $a$ is offered after a delay determined by a negative exponential distribution. More precisely, $(a, \lambda); p$ offers action $a$ within $t$ time units with probability $1 - e^{-\lambda t}$ and then evolves into process $p$. The mean duration until action $a$ is offered is thus $1/\lambda$. As a semantic model, transition systems are used where transitions are labelled with pairs of actions and rates. By omitting the action labels — but keeping the rate information — one obtains a (time-homogeneous) continuous-time Markov chain (CTMC) for which steady-state and transient performance metrics can be obtained using traditional techniques [95]. These process algebras provide a high-level specification formalism for CTMCs. Due to this property they are also called *Markovian* process algebras; recent overviews can be found in [19,50,57].

*Separating delays and actions.* The major distinction between Markovian process algebras is the treatment of time consumption in case of interaction. Technically, this amounts to the computation of the resulting rate in case two actions like $(a, \lambda)$ and $(a, \mu)$ synchronize. To our opinion, the most natural interpretation is to require both delays to have completed before the synchronization (on $a$) can take place — the so-called *patient communication* [55]. The thus resulting random variable equals the maximum of the random variables that are exponentially distributed with rates $\lambda$ and $\mu$, respectively. This random variable is, however, not exponentially distributed. To overcome this technical problem, several so-

lutions have been suggested that either lack a clear stochastic interpretation or have a somewhat restricted applicability.

The stochastic interpretation (i.e., maximum of random variables) can be obtained in a rather natural way by explicitly separating the advance of time and the occurrence of actions. In this way, synchronization only takes place via immediate actions. Thus, the usual prefix $a;p$ remains unchanged, but is complemented with a delay prefix $\lambda \mapsto p$ which evolves into $p$ after an exponential delay with mean $1/\lambda$. This separation of discrete and continuous phases is similar to that in some timed process algebras (see before) and has been proposed in the context of Markovian process algebras in [49,51,52].

### 2.2 Beyond Exponential Distributions

Integration of general, non-exponential distributions in a process algebraic setting has received scant attention. Instead, most work has been focussed on exponential distributions. Although exponential distributions yield analytically tractable models (i.e., CTMCs), and are useful for many applications, they are not realistic for modeling many phenomena in an adequate way. For example:

- in performance modeling, it is often convenient to incorporate empirical distributions into the model that have been obtained by measuring a realization of the system. These measurements may e.g., indicate the traffic intensity of a communication network at a working day, or indicate the length of communicated web pages during peak hours. These distributions are mostly not exponential.
- if a distribution function $G$ is only partially known, it is preferably approximated by a probability distribution $F$ with a "maximal indeterminacy" in the sense that it is impossible to recognize from $F$ any preference of one event over another. Thus, $F$ assumes the least about the structure of the distribution $G$, or, stated otherwise, it has the highest degree of randomness. Technically speaking, $F$ maximizes the *entropy* [93]:

$$-\int_{-\infty}^{\infty} f(x)\ln(f(x))\,dx$$

  where $f$ is the probability density function of $F$. Depending on which partial information about $G$ is available, different appropriate choices for $F$ remain. If only the mean and variance are known, the normal distribution is the most indeterminate; in cases where only the minimum and maximum are known, the uniform distribution on the interval between these bounds is the most indeterminate. The exponential distribution is the most indeterminate approximation only in cases where only the mean is known (of a positive random variable).
- empirical studies have shown that many system parameters, such as sizes of data files stored on web servers and transferred through the Internet, job service times in general-purpose computing environments, and node degrees

of certain graph structures (such as hyper links of web pages), exhibit so-called *heavy-tail* distributions, i.e., distributions with a very high variance. A distribution $F$ is heavy tailed [26] if for positive constant $c$:

$$F(x) \text{ "approximates" } 1 - c \cdot x^{-\alpha} \text{ for } 0 < \alpha < 2$$

$F$ has an infinite variance, and for $\alpha < 1$ it has an infinite mean. An important heavy-tail distribution is the Pareto distribution. For a heavy-tail distribution (with $\alpha$=1), about 60% of the probability mass is contained in just 1% of the observations; for an exponential distribution this dependency is roughly linear. If one observes heavy-tailed inter-arrivals, then the longer one has waited, the longer we should expect to wait — the expectation paradox. Instead, for exponential distributions the waiting time does not play any role, due to the memoryless property (see next Section).

- deterministic delays are prevalent and important in computer, communication and manufacturing systems. Typical examples of deterministically distributed parameters are: timeouts in communication protocols, hard deadlines in real-time systems, transmission delays of fixed-length packets, and cycle times of work-flow management systems.
- it has been argued that several phenomena in modern communication systems, in particular several aspects of multi-media communication systems, can be most adequately modeled by non-exponential distributions. For instance, the variability of the delay of sound and video frames (so-called jitter) is mostly assumed to be controlled by a normal distribution [11,38].

### 2.3 Interleaving + General Distributions = Non-trivial

Given the need for non-exponential distributions, the question is whether we cannot simply replace the exponential distributions in Markovian process algebra by general distributions. This turns out not to be straightforward. We illustrate this by discussing (a simplified version of) an important axiom in process algebra, known as the expansion law.

*Expansion law.* A popular mathematical model for providing semantics to traditional process algebras is labelled transition systems, (possibly infinite-state) automata where transitions that are labelled with actions describe how the system can evolve from one state to another. In mapping process algebra terms to this model, the independent parallel composition of two actions is treated as a choice between the two possible sequential orderings. Thus, in a parallel composition, actions of one component are interleaved with actions of the other — hence the term *interleaving* semantics. As a result, independent parallel composition (denoted $||$) can be reduced in terms of choice (denoted $+$) and prefix as expressed by, for example:

$$a; p \,||\, b; q = a; (p \,||\, b; q) + b; (a; p \,||\, q) \tag{1}$$

for actions $a, b$ and processes $p, q$ (cf. Fig. 1 for $p$ and $q$ being the process $\mathbf{0}$ that cannot perform any action). This principle, in its full generality known as the

*expansion law* [77], is widely accepted and has proven to be of crucial importance for process algebraic verification purposes [4].

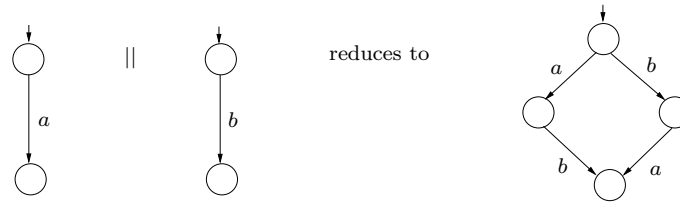



**Fig. 1.** Interleaving of the processes $a;\mathbf{0}$ and $b;\mathbf{0}$

*Exponential distributions and interleaving semantics fit well.* The semantics of Markovian process algebras are commonly defined using an extension of labelled transition systems. The structure of these transition systems closely resembles that of CTMCs. The elegant memoryless property of exponential distributions enables a smooth integration in an interleaving setting, since in analogy of (1) we have:

$$\lambda \mapsto p \,||\, \mu \mapsto q = \lambda \mapsto (p \,||\, \mu \mapsto q) + \mu \mapsto (\lambda \mapsto p \,||\, q) \tag{2}$$

To justify this law, consider the term $\lambda \mapsto p \,||\, \mu \mapsto q$. Let $U$ be the random variable modeling the delay before process $q$ can start — $U$ is thus exponentially distributed with rate $\mu$ — and suppose that the delay of the left process "finishes first" (with rate $\lambda$), say, after $y$ time units. The probability that the start of process $q$ has to delay for at most an additional $x$ time units is

$$\Pr[U \leqslant y{+}x \mid U > y]$$

Due to the memoryless property of exponential distributions, it holds that

$$\Pr[U \leqslant y{+}x \mid U > y] = \Pr[U \leqslant x] \tag{3}$$

Thus, the remaining duration until the initial delay of process $q$ finishes is (again) determined by an exponential distribution with rate $\mu$. Stated differently, the delay of the left process does not have any impact on the distribution of the remaining delay in the other process — the advance of time governed by memoryless distributions is independent. By symmetry, an analogous reasoning applies when the right-hand process finishes first.

*General distributions and expansion law.* If we allow actions to be delayed by general distributions $F$ and $G$, though, it turns out that the analogon of (2) is invalid:

$$F \mapsto p \,||\, G \mapsto q \neq F \mapsto (p \,||\, G \mapsto q) + G \mapsto (F \mapsto p \,||\, q) \tag{4}$$

The reason for this inequality is the absence of the memoryless property for general distributions. For instance, after the delay imposed by $F$ in the left-hand process, the residual delay of the right-hand process $G \mapsto q$ has to be

taken into account in order to correctly determine the remaining delay before process $q$ becomes enabled.

### 2.4 Some Solutions to the Problem

If the incorporation of general distributions into process algebra is not trivial, what are possible strategies to overcome this problem? Here, we summarize the main schools of thought.

*Still use Markovian process algebras.* In this category we find two kinds of solutions: approximation and exploiting insensitivity.

1. A possible solution is to *approximate* general distributions by appropriate probability distributions that can be described as series/parallel combinations of exponential distributions, possibly with feedback, thus residing in the class of Markovian process algebras. An interesting class of distributions for this purpose is the class of phase-type (PH) distributions [79]. They are defined as distributions of absorption times in finite CTMCs (but that may contain loops) with a single absorbing state, i.e., a state without any outgoing transition. PH-distributions can approximate any distribution on $[0, \infty)$ arbitrarily close; algorithms to fit a PH-distribution to empirical distributions do exist [3]. The encoding of PH-distributions in a Markovian process algebra has been considered in [49,51,52] where — due to the aforementioned separation between time and actions — any CTMC with a trivial initial distribution (and thus any such PH-distribution) can be specified. A similar approach can be taken by using a subset of PH-distributions such as Cox [25] or Erlang mixture distributions, see [52] and [23], respectively.
2. An alternative solution is to allow general distributions in a controlled way such that the stochastic property of *insensitivity* can be exploited. A stochastic process is insensitive if its steady-state distribution depends on the distribution of one or more of the random variables governing state residence times only through their mean. The theory of insensitivity has been applied to high-level specification formalisms such as stochastic Petri nets [48]. In [22,23] a syntactic construction is presented that guarantees the insensitivity of the stochastic process underlying a stochastic process algebra specification. By means of this construction it is guaranteed that for studying the steady-state behavior of the process under consideration, it is sufficient to consider that process in which each general distribution is replaced by, for instance, an exponential distribution with the same mean. Hence, the steady-state behavior of such processes can be analyzed using ordinary CTMC techniques.

The main benefit of both approaches is that existing frameworks can be used without any modification. The main disadvantage of approximation (approach 1) is that it leads to a state-space explosion since any general distribution is represented by a (possibly complex) CTMC. Besides, choice expressions like $F \mapsto p + G \mapsto q$ are difficult to treat as the choice between the approximations of $F$ and $G$ is determined by the first phase of their PH-distribution, and not

by their entire PH-distribution. The insensitivity approach (2) is applicable to a small class of processes and is restricted to steady-state properties.

*Drop the expansion law.* As we have seen before, obtaining an expansion law (1) is a serious problem when considering general distributions. A feasible option is to define a semantics for process algebra that does not obey this law. The thus obtained semantics is known as *true concurrency* or *partial-order* semantics. In these models, system runs are no longer represented as totally ordered sequences, but rather as partial orders where the occurrences of causally independent actions are unrelated. Important partial-order models include Petri nets [36], and event structures [81]. The idea of using true concurrency semantics for stochastic process algebra has been brought up in [18] where a stochastic variant of event structures was used as semantic model. As actions that occur concurrently are unordered, there is no need to keep track of the residual delay of random variables that "run in parallel". Analysis techniques that have considered for these event structures are discrete-event simulation [67], and a decomposition-based analysis method [12]. Similar approaches have been pursued in [83] where causal dependencies between delays are derived from the transition labels (so-called proved transitions) and in [53] where stochastic task graphs are used, a model that is quite similar to event structures.

The main advantage of this approach is the potential compact representation of the state space; the main disadvantage is that it leads to infinite-state semantic objects even for simple recursive terms. Recent investigations indicate, however, that finite objects can be obtained for the event structure approach (for finite-state process algebra terms) from which stochastic task graph models are generated that can be analyzed numerically [87].

*Keep track of residual lifetimes.* As a third solution, we refine the earlier discussed separation of time and actions a bit further and distinguish between:

- the start of a probabilistic delay,
- the completion of a probabilistic delay, and
- the occurrence of immediate actions.

To keep track of delays labelled transition systems are extended with *clocks*. A clock is initialized by sampling a probability distribution function, and starts counting down once initialized. All clocks count down at the same pace. Transitions are labelled with an action and a set of clocks; this transition is enabled when all clocks in the set have expired, i.e., have reached the value 0. Similarly, we extend traditional process algebra with two new constructs: for $C$ a finite set of clocks, $\mathsf{when}\ C \mapsto p$ denotes the process that after expiration of all clocks in $C$ behaves like $p$, and $\mathsf{set}\ C\ \mathsf{in}\ p$ denotes the process that behaves like $p$ after any clock $x$ in $C$ is initialized according to its distribution. The delay prefix $\lambda \mapsto p$ that we encountered before is now written as $\mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto p)$ with $x$ a clock controlled by an exponential distribution of rate $\lambda$. Note that in the exponential case the distinction between start and finish of a delay is not needed since

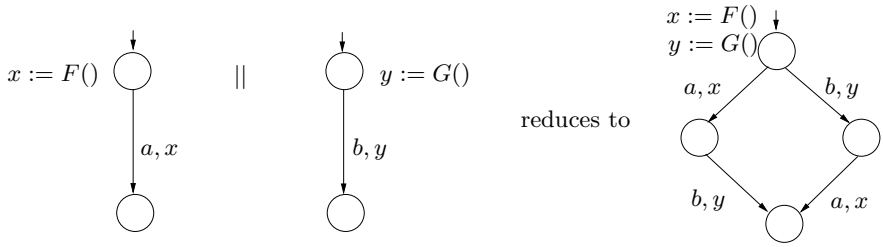



**Fig. 2.** Interleaving processes set $x$ in (when $x \mapsto a; \mathbf{0}$) and set $y$ in (when $y \mapsto b; \mathbf{0}$)

$$\begin{aligned} &\mathsf{set}\ x, y\ \mathsf{in}\ (\mathsf{when}\ x \mapsto a; \mathsf{when}\ y \mapsto p) \\ =\ &\mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto a); \mathsf{set}\ y\ \mathsf{in}\ (\mathsf{when}\ y \mapsto p) \end{aligned}$$

When mapping process algebra terms onto the extended labelled transition systems, the principle of interleaving is applied (cf. Fig. 2). In the resulting automaton, initially both clocks $x$ and $y$ are initialized and start counting down. If $x$ expires first, action $a$ happens, and a state is reached in which clock $y$ records the remaining time until action $b$ is enabled. A symmetric scenario happens when clock $y$ expires first. Accordingly, an expansion law can be obtained. For instance, for $p' = \mathsf{when}\ x \mapsto a; p$ and $q' = \mathsf{when}\ y \mapsto b; q$ we have:

$$\mathsf{set}\ x\ \mathsf{in}\ p' \,||\, \mathsf{set}\ y\ \mathsf{in}\ q' = \mathsf{set}\ x, y\ \mathsf{in}\ \ (\mathsf{when}\ x \mapsto a; (p \,||\, q') + \mathsf{when}\ y \mapsto b; (p' \,||\, q))$$

This idea has first been brought up in [30], and has been extended and refined later on [27,31,32,33]. In this tutorial we will follow this direction. A similar approach has been taken in [17] where a "start-termination" (ST) semantics — a model that has been originally developed to study refinement in process algebra [40] — is used for generally distributed delays.

The labelled transition systems extended with clocks closely resemble *generalized semi-Markov processes* (GSMPs), a model used for the study of stochastic discrete-event systems. Accordingly, the process algebra that allows for general distributions can be considered as a high-level specification formalism for GSMPs.

## 3 Generalised Semi-Markov Chains

GSMPs have been introduced by Glynn [41] and Whitt [100]; for an introduction to GSMPs we refer to [21,92]. We consider discrete-state GSMPs, *generalised semi-Markov chains* (GSMCs, for short).

*The model of GSMCs.* A GSMCs is a state automaton where transitions are triggered by the occurrence of stochastically timed events. A set of active events is associated to each state. These are the events that are possible in that state, i.e., that can cause a transition outgoing from that state. The remaining time

until the possible occurrence of an event is determined by its clock; we thus have one clock per event. Clocks are initialised according to a continuous probability distribution function and run backwards, all with the same pace. In the following we assume a set of clocks $\mathcal{C}$ is given.

**Definition 1.** $(Z, z_0, \mathbf{E}, E, C, \mathsf{next})$ *is a* generalised semi-Markov chain *(GSMC) with*[2]

- $Z$*, a non-empty set of* states *with* initial state $z_0 \in Z$*,*
- $\mathbf{E}$*, a non-empty set of* events*,*
- $E : Z \to \mathcal{P}_f(\mathbf{E})$*, the* event-assignment *function s.t.* $E(z) \neq \varnothing$ *for all* $z \in Z$*,*
- $C : \mathbf{E} \to \mathcal{C}$*, the* clock-assignment *function, with continuous distribution* $F_{C(e)}$ *for any* $e \in E$*,*
- $\mathsf{next} : Z \times \mathbf{E} \to Z$*, the partial* next-state *function that assigns to each state* $z$ *and event* $e \in E(z)$ *a next state* $\mathsf{next}(z, e)$*.*

*Example 1.* Consider a queueing system in which jobs arrive and wait until they are executed by a single server. An infinite population of jobs is assumed. Jobs arrive with an inter-arrival time that is determined by a continuous probability distribution $F$ while the delay between the processing of two successive jobs is controlled by distribution $H$. The serving discipline is FCFS (first come first served). This system is known as a $G/G/1/\infty$-queue, where $G$ stands for general distribution of the arrival and service process, respectively, 1 indicates the number of servers, and $\infty$ denotes the infinite buffer capacity. A typical GSMC description of such queueing system is defined in the following way. We let the state space $Z = \{0, 1, 2, \ldots\}$ where the state number indicates the number of jobs that are currently in the system, i.e., in the queue or currently being processed. The initial state $z_0$ is 0, the empty system. In each state, possible events are the arrival of a job (denoted $a$) and the completion of a job (denoted $c$); thus, $\mathbf{E} = \{a, c\}$. In the initial state no job completion is possible. Thus, $E(i) = \{a, c\}$ for $i > 0$ and $E(0) = \{a\}$. The arrival of a job causes a transition from state $i$ to state $i{+}1$. Completion of a job leads to a transition from state $i{+}1$ to state $i$. Thus, $\mathsf{next}(i, a) = i{+}1$ and $\mathsf{next}(i{+}1, c) = i$. The state-transition structure of this GSMC is depicted in Fig. 3. Clocks are initialised as follows. On entering state $i{+}1$ after an arrival, the clock of the next arrival $a$ is initialised

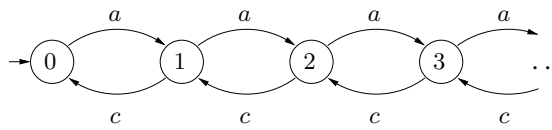



**Fig. 3.** GSMC for a G/G/1/$\infty$ queueing system

[2] We adopt the following notational convention. For a set $A$, $\mathcal{P}(A)$ denotes the set of all subsets of $A$, and $\mathcal{P}_f(A)$ denotes the set of finite subsets of $A$.

according to distribution $F$, the job inter-arrival time. On entering state $i$ after a job completion, the clock of the next job completion is initialised according to distribution $H$, the service delay. Accordingly, we let $C(a) = x$, $C(b) = y$ and $F_x = F$ and $F_y = H$.

*The behaviour of a GSMC.* The dynamics of a GSMC are described as follows in a procedural way. Note that for any state $z$, to each active event $e \in E(z)$ a clock value $val(C(e)) \geqslant 0$ is associated. Initially, all active events are initialised according to their distribution function, i.e., $val(C(e)) = F_{C(e)}(\cdot)$ if $e \in E(z_0)$. Intuitively, $F_{C(e)}(\cdot)$ denotes "taking a sample from distribution $F_{C(e)}$". Then:

1. determine the set of active events $E(z)$ in the current state $z$
2. determine the clock value $d^*$ such that $d^* = \min\{\, val(C(e)) \mid e \in E(z) \,\}$
3. determine event $e^*$ with $val(C(e^*)) = d^*$ and state $z' = \mathsf{next}(z, e^*)$
4. determine the new clock values $val'$ in $z'$ as follows:

$$val'(C(e)) = \begin{cases} val(C(e)) - d^* & \text{if } e \in E(z) \cap (E(z') - \{\, e^* \,\}) \\ F_{C(e)}(\cdot) & \text{if } e \in E(z') - (E(z) - \{\, e^* \,\}) \\ \infty & \text{otherwise} \end{cases}$$

5. go back to the first step of the procedure with current state $z'$.

Note that there always exists an event $e^*$, since $E(z)$ is non-empty for every state $z$. Since all clocks are initialised by continuous distributions, the event $e^*$ is guaranteed to be *unique* with probability one, as the probability of sampling two continuous distributions with the same value is 0. Once event $e^*$ with minimal clock value has been determined, the next state $z'$ is determined (step 3.) and the new clock values are calculated as follows (step 4.). For each active event $e$ in state $z$ that remains active in $z'$, the clock value is decreased by the elapsed time $d^*$. For each event $e$ that becomes active in $z'$, the clock value is determined by sampling the clock-distribution function $F_{C(e)}$; for all other events the clock value equals $\infty$, indicating that these events are inactive.

The above procedure is also known as variable time-advance procedure [92] which is characterised by time steps of varying length and an event occurrence in every time step. This procedure is controlled by the occurrence of "next events" and the time between the occurrence of two events is "skipped". This principle is reflected by the fact that clock values do not increase as time passes, but only increase if the next event happens (see step 4.).

*Example 2.* Fig. 4 presents an example execution of the GSMC of Fig. 3 by depicting the values of the clocks of events $a$ (solid line) and $c$ (dashed line) (on the y-axis) while time evolves (x-axis). Each time a clock expires, a transition to the next state is taken. Below the x-axis, for each time instant the current state is indicated. Note that in this execution, event $c$ becomes enabled when moving from the initial state to state 1 and stays enabled while visiting states 2 and 3, before triggering the transition back to state 2.

*Restrictions.* Actually, we consider a subclass of GSMCs as introduced in [41]. The main restriction that we impose is that the next state is *deterministically*

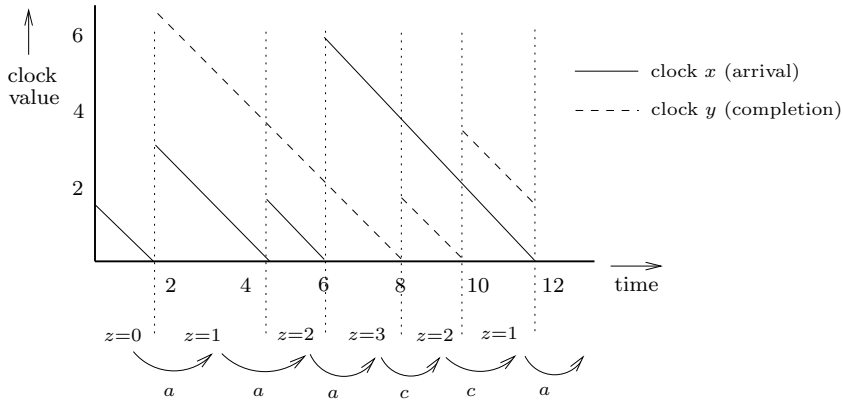



**Fig. 4.** A sample evolution of the GSMC of Fig. 3

determined by the present state and the triggered event, whereas in the original GSMCs the next state is chosen in a discrete probabilistic fashion from a set of possible next states. In addition, we consider *time-homogeneous* GSMCs. Such processes are invariant under time-shifts. Intuitively, the probability to be in state $z'$ at time $t'$ given that the system is in state $z$ at time $t < t'$, is equal to the probability that the system is in state $z'$ at time $t'-\Delta$ given that the system is in state $z$ at time $t-\Delta$ (for any $\Delta \leqslant t$). This is a rather common restriction. Finally, clocks in GSMCs are allowed to have different (possibly state-dependent) rates whereas in our case all clocks proceed with the same speed. Different rates are not very usual in discrete-event simulation, and moreover, under certain conditions, such multi-rated GSMCs can be represented by GSMCs where all clock rates equal one. The notion of GSMC considered here can thus be summarised as *mono-rated, deterministic, time-homogeneous* GSMCs.

*GSMCs versus CTMCs and SMCs.* In order to better understand the link with related models, we briefly address the relationship of GSMCs to continuous-time Markov chains (CTMCs) [95] and semi-Markov chains (SMCs) [86,54]. To put it in a nutshell, a CTMC is a GSMC in which all clocks are governed by negative exponential distributions, i.e., for each clock $x$ we have $F_x(t) = 1-e^{-\lambda t}$, for some non-negative real $\lambda$. A CTMC possesses the Markov property: the probability of making a transition to a next state only depends on the current state and not on previous states ("absence of state memory"). The memoryless property of exponential distributions further implies that the probabilities of taking next transitions do not depend on the amount of time spent in the current state ("absence of age memory"). The residence time of a state is exponentially distributed with a rate that equals the sum of the rates of its outgoing transitions.

A SMC is a GSMC in which all clocks are initialised on each state change. As exponential distributions are memoryless, cf. equation (3), there is no difference between setting clocks on each state change or not. Each CTMC is thus an SMC. An SMC is, however, not a CTMC. It possesses the Markov property,

but does not conform to the "absence of age memory" principle: probabilities of taking next transitions do depend on the amount of time spent in the current state. For an SMC, the state residence times are generally distributed and are explicitly specified for each state. In a GSMC, the residence time of a state is determined implicitly by the distributions of the set of active events in a state. As a result, the state residence times in a GSMC may be history-dependent. This phenomenon is the essential difference with SMCs, where state residence times are governed by an a priori, fixed random variable.

## 4 Why a Process Algebra for GSMCs?

In this section, we motivate the need for a process algebraic language for the specification of GSMCs by discussing a couple of examples. For each example a short informal description is given, a GSMC, and a process algebraic specification. Although using the above description the dynamics of our example GSMCs can be determined, it is in absence of any further explanation not easy to understand. As we will illustrate, this is basically due to the fact that the individual system components are hard to recognize from the overall system structure. This problem becomes more apparent if we consider GSMCs modeling systems of more realistic magnitude. We say that GSMC specifications lack *compositionality*. The idea that we shall pursue here is to specify GSMCs in a compositional way and to exploit this compositional structure by re-using components. We start with a process algebra specification for the G/G/1 queueing system of Example 1.

### 4.1 The Simple Queueing System

In process algebra, the specification of our queueing system can be obtained in a hierarchical manner, starting from the specifications of the individual components: the queue, the server, and the arrival process. The buffer of infinite capacity can be specified by the set of processes:

$$\begin{aligned} \mathit{Queue}_0 &= a; \mathit{Queue}_1 \\ \mathit{Queue}_i &= a; \mathit{Queue}_{i+1} + b; \mathit{Queue}_{i-1} \text{ for } i > 0 \end{aligned}$$

where the process indices indicate the number of jobs in the buffer, action $a$ denotes enqueueing a job, and action $b$ denotes dequeuing a job. Similarly to GSMCs, clocks can be used to model probabilistic delays. We obtain for the arrival and server processes:

$$\begin{aligned} \mathit{Arrival} &= \mathsf{set}\ x_F\ \mathsf{in}\ (\mathsf{when}\ x_F \mapsto a; \mathit{Arrival}) \\ \mathit{Server} &= b; \mathsf{set}\ y_H\ \mathsf{in}\ (\mathsf{when}\ y_H \mapsto c; \mathit{Server}) \end{aligned}$$

In the *Arrival* process clock $x$ is initialized and starts counting down. Once it has reached the value 0, it expires and action $a$ is enabled. The overall system is described by:

$$\mathit{System}_{G/G/1/\infty} = (\mathit{Arrival} \,||_{\varnothing}\, \mathit{Server}) \,||_{\{a,b\}} \mathit{Queue}_0$$

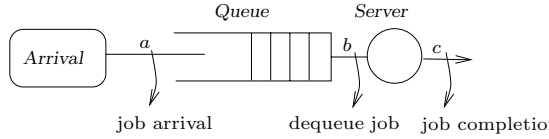



**Fig. 5.** Compositional specification of G/G/1 queueing system

In process $p \,||_A\, q$, where $A$ is a set of actions, $p$ and $q$ perform actions autonomously, but actions in $A$ should be synchronously performed by both. Action $a$, for instance, can only take place when the processes *Arrival* and *Queue* are both ready to participate; note that *Server* does not need to participate in this action. The resulting specification of the $G/G/1/\infty$ system closely resembles the structure of the system itself (cf. Fig. 5), it is easy to understand, and it is readily modifiable. For instance, a system with two servers is obtained in the following way:

$$System_{G/G/2/\infty} = (Arrival ||_\varnothing Server ||_\varnothing Server) \,||_{\{a,b\}} Queue_0$$

As an alternative extension, a specification of the $G/G/1/K$ queueing system, where $K$ ($K > 0$) denotes the finite capacity of the queue, is obtained by replacing the $Queue_0$ process by a slightly modified buffer $FQueue_0$:

$$\begin{aligned} FQueue_0 &= a; FQueue_1 \\ FQueue_i &= a; FQueue_{i+1} + b; FQueue_{i-1} \text{ for } 0 < i < K \\ FQueue_K &= a; FQueue_K + b; FQueue_{K-1} \end{aligned}$$

where it is assumed that when the queue is full, a job arrival is neglected and lost. The loss of a job is reflected by the fact that for process $FQueue_K$ the capacity is unchanged on arrival of a job.

### 4.2 A Queueing System with Two Classes of Jobs

*Informal description.* Consider now a single-server queueing system with a finite queueing capacity $K > 0$, cf. Fig. 6. Two types of jobs are considered. They arrive independently according to different arrival processes. Jobs of class $i$ ($i = 0, 1$) arrive with an inter-arrival time $F_i$; the processing delay of a job of class $i$ is controlled by distribution $H_i$. We assume that the service times are mutually independent and are independent of the arrival process. The single server thus takes the job (if any) at the head of the queue and services it with a delay according to its class. The serving discipline is FCFS. (This example is adopted from [21].)

*A GSMC description.* The possible events in this system are the arrival or completion of a job of class $i$ ($i = 0, 1$); accordingly, the set of events equals $\{ a_0, a_1, c_0, c_1 \}$ where the index of the actions indicate the class they are related to. The description of the state of the system is a bit more involved than for

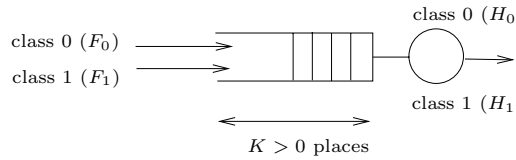



**Fig. 6.** G/G/1-queueing system with two job classes

the G/G/1/$\infty$ queueing system. It no longer suffices to consider only the queue length, but we also have to keep track of the class of the $j$-th job in the queue. Thus, a state in the GSMC is a $K$-tuple $(j_1, \ldots, j_K)$ where $j_m = -$ if the $m$-th position in the queue is empty, and $j_m = 0$ (1) if this position contains a job of class 0 (1), for $0 < m \leqslant K$. The oldest job (if any) is kept at position 1. This gives rise to $2^{K+1} - 1$ states. The GSMC for $K$ equal to 3 is depicted in Fig. 7 where empty positions in the queue are indicated as blanks. Note that states at depth $i$ indicate scenarios where $i$ jobs are currently in the system.

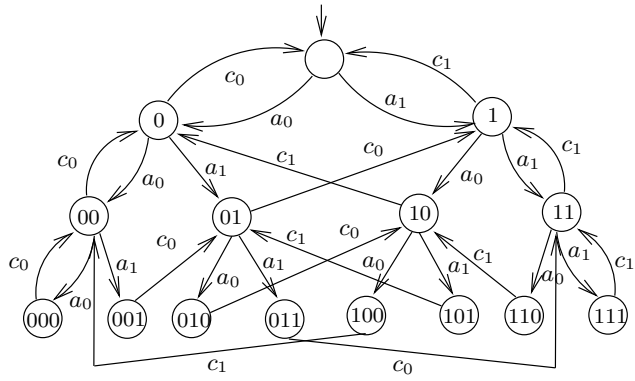



**Fig. 7.** GSMC of a G/G/1-queueing system with two job classes

*A compositional approach.* In order to obtain a process algebraic specification of this queueing system we adapt the specification of the simple G/G/1/$\infty$-system in a component-wise manner. The arrival process for each class of jobs is simply an instantiation of the *Arrival* process before; the server is slightly adapted in order to be able to deal with both classes of jobs:

$$\begin{aligned} \mathit{Arrival}_i &= \mathsf{set}\ x_{F_i}\ \mathsf{in}\ (\mathsf{when}\ x_{F_i} \mapsto a_i; \mathit{Arrival}_i) \text{ for } i = 0, 1 \\ \mathit{Server2} &= \quad b_0; \mathsf{set}\ y_{H_0}\ \mathsf{in}\ (\mathsf{when}\ y_{H_0} \mapsto c_0; \mathit{Server2}) \\ &\quad + b_1; \mathsf{set}\ y_{H_1}\ \mathsf{in}\ (\mathsf{when}\ y_{H_1} \mapsto c_1; \mathit{Server2}) \end{aligned}$$

For the finite queue, we extend the specification of the finite queue given above such that, besides the current occupancy of the queue, we keep track of the class

of the $j$-th job in the queue. The latter is carried out by adding a bit-vector (as superscript) to the process *FQueue* in the following way:

$$\begin{aligned} OQueue_0 &= a_0; OQueue_1^0 + a_1; OQueue_1^1 \\ OQueue_i^{0w} &= a_0; OQueue_{i+1}^{0w0} + a_1; OQueue_{i+1}^{0w1} + b_0; OQueue_{i-1}^{w} \text{ for } 0 < i < K \\ OQueue_K^{0w} &= a_0; OQueue_K^{0w} + a_1; OQueue_K^{0w} + b_0; OQueue_{K-1}^{w} \end{aligned}$$

The processes $OQueue_i^{1w}$ are similar to $OQueue_i^{0w}$ $(0 < i \leqslant K)$ and are omitted here. The overall system is now described by:

$$System_K = (Arrival_0 \,||_{\varnothing}\, Arrival_1 \,||_{\varnothing}\, Server2) \,||_{\{a_0,a_1,b_0,b_1\}} OQueue_0 \qquad (5)$$

Note the resemblance with the structure of the G/G/1/$\infty$-specification.

### 4.3 A Simple Queueing Network

Suppose now that we combine the two queueing systems above in the following (open) queueing network, cf. Fig. 8. The arrival processes of the two classes of jobs in the latter system, $System_K$, are the outputs generated by two finite G/G/1 queueing systems of size $K$ (for class 0) and $N$ (for class 1), respectively. For obtaining the GSMC of this system, the GSMCs of the individual compo-

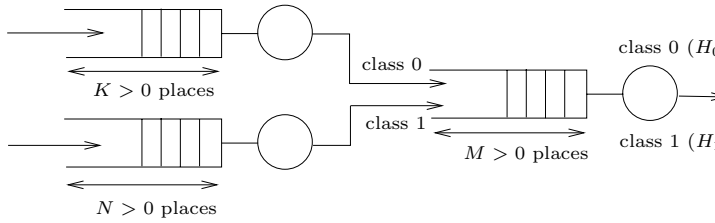



**Fig. 8.** A simple open queueing network

nents need to be combined in an appropriate way. This is not a straightforward exercise.

To obtain a specification of this system in the process algebraic setting, we first observe that the structure of the queueing network resembles that of $System_K$, except that the input streams are the output streams of two finite G/G/1 systems. Thus, the two *Arrival* processes in specification (5) are replaced by the specifications of the G/G/1-queues. It now remains to "link" the output of the G/G/1 systems to the input of the 2-class buffer system. This is established by means of renaming. Let $f$ be a function that maps action names to action names. Process $p[f]$ behaves like process $p$ except that actions are renamed according to $f$. For instance, process $System_{G/G/1/K}[c/a_0]$ denotes the finite G/G/1 queue with $K$ places where action $c$ (output) is renamed into $a_0$.

Thus, a job completion in this system is turned into an arrival of a class 0 job. Using this renaming operator we obtain:

$$\Big(\mathit{System}_{G/G/1/K}[c/a_0] \,||_\varnothing\, \mathit{System}_{G/G/1/N}[c/a_1] \,||_\varnothing\, \mathit{Server2}\Big) \;||_A \mathit{OQueue}_0$$

with $A = \{a_0, a_1, b_0, b_1\}$.

### 4.4 Non-determinism

Recall that in each state of a GSMC, the next event is determined in a unique way. We like to point out that in the process algebra this is (deliberately) not the case. For instance in a specification like

$$\mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto\ a; q + \mathsf{when}\ x \mapsto\ a; r)$$

it is not uniquely determined whether to move to $q$ or to $r$ once clock $x$ has expired. A similar scenario appears in $\mathit{System}_{G/G/2/\infty}$: if the buffer contains a job and both servers are idle, it is non-deterministically determined which server dequeues the job. This phenomenon, known as non-determinism, appears if two (or more) equally labelled transitions become enabled simultaneously. This concept is usually absent in stochastic discrete-event systems – how to analyze their performance? – but has been widely accepted in the computer science community for the purpose of under-specification. This is useful for modeling, for instance [58]:

- Implementation freedom: non-determinism allows to specify freedom of implementation; for instance, if two possible alternatives are described in the specification, a valid implementation would just realize one of them
- Scheduling freedom: if several processes run in parallel, there is a freedom of choice in selecting the next process that performs a move (interleaving)
- External environment: actions represent interaction opportunities with the context in which the process is considered; the interaction capabilities of the environment then influence how the choice is determined

Non-determinism is useful for under-specifying "how often" an alternative is chosen. This information is usually not available in the early steps of the design, or is deliberately left unspecified. If we are to study the performance of such system specifications, this non-determinism will be resolved by adversaries, see Section 7.

## 5 Stochastic Automata

This section introduces *stochastic automata*, a mixture of timed automata [2] and GSMCs. Stochastic automata are strongly related to GSMCs and incorporate, apart from the necessary (slightly generalized) ingredients to model GSMCs, the possibility of specifying non-determinism. An extensive treatment of stochastic automata is given in [27].

**Definition 2.** *A* stochastic automaton $SA = (\mathcal{S}, s_0, \mathbf{A}, \mathcal{C}, \rightarrow, \kappa)$ *where*

- $\mathcal{S}$ *is a non-empty set of* locations *with initial location* $s_0 \in \mathcal{S}$
- $\mathbf{A}$ *is a set of* actions
- $\mathcal{C}$ *is a set of* clocks *with distribution function* $F_x$ *for each* $x \in \mathcal{C}$
- $\rightarrow \subseteq \mathcal{S} \times (\mathbf{A} \times \mathcal{P}_f(\mathcal{C})) \times \mathcal{S}$ *is a set of* edges
- $\kappa : \mathcal{S} \rightarrow \mathcal{P}_f(\mathcal{C})$ *is a* clock-setting function

$(s, a, C, s') \in \rightarrow$ is denoted $s \xrightarrow{a,C} s'$. To each location $s$ a finite set of clocks $\kappa(s)$ is associated. As soon as location $s$ is entered, any clock $x$ in $\kappa(s)$ is initialized according to its probability distribution function $F_x$. Once initialized, the clock variables count down at the same rate of letting time pass (like in a GSMC). A clock expires if it has reached 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 $s$, action $a$ is offered as soon as all clocks in the set $C$ have expired. In this situation, the edge $s \xrightarrow{a,C} s'$ is called enabled. After taking the edge, the system evolves to location $s'$. If, after the expiration of a (possibly empty) set of clocks, more than one edge outgoing from $s$ is enabled, an enabled edge is selected non-deterministically.

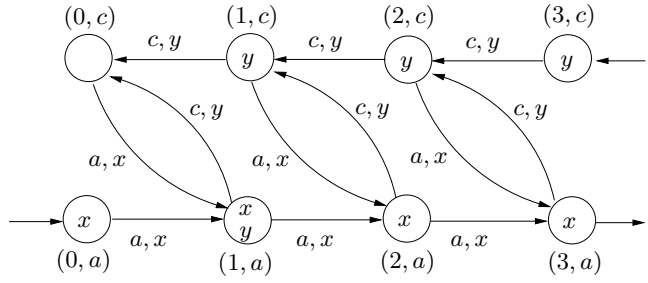



**Fig. 9.** Stochastic automaton of a $G/G/1/\infty$-system

*Example 3.* The stochastic automaton $SA_{G/G/1/\infty}$ (cf. Fig. 9) describes the behavior of the $G/G/1/\infty$ queue. Here, we represent a location $s$ as a circle containing the clocks that are to be set in $s$, and denote edges by arrows. The initial location is represented by a circle equipped with a small ingoing arrow (leftmost circle in second row). We often omit curly brackets for singleton sets. $SA_{G/G/1/\infty}$ is defined by: $\mathcal{S} = \{(i, \alpha) \mid i \geqslant 0, \alpha \in \{a, c\}\}$, $s_0 = (0, a)$, $\mathbf{A} = \{ a, c \}$, $\mathcal{C} = \{ x, y \}$ with $F_x = F$ and $F_y = H$, and

$$\kappa(i, a) = \begin{cases} \{ x \} & \text{if } i \neq 1 \\ \{ x, y \} & \text{if } i = 1 \end{cases} \quad \text{and} \quad \kappa(i, c) = \begin{cases} \{ x \} & \text{if } i \neq 0 \\ \varnothing & \text{if } i = 0 \end{cases}$$

and $(i, \alpha) \xrightarrow{a,x} (i{+}1, a)$ and $(i{+}1, \alpha) \xrightarrow{c,x} (i, c)$ for $i \geqslant 0$ and $\alpha \in \mathbf{A}$.

This stochastic automaton can be understood as follows. Locations are pairs where the first component indicates the number of jobs in the system (i.e., queue and server), and the second component indicates whether the last action was an arrival (action $a$) or a completion of a job (action $c$). Note that after the occurrence of action $a$ a location is reached in which clock $x$ is set. Similarly, after the occurrence of action $c$, clock $y$ is set (except for location $(0, c)$). Clock $x$ thus controls the job inter-arrival time while clock $y$ controls the service delay. In location $(1, a)$, the job that has just arrived is to be served. Thus both the time of the next job arrival and the time until the job is serviced are determined. Accordingly, clocks $x$ and $y$ are set in location $(1, a)$. In location $(0, c)$, however, the last job has just been served and the delay until the next job arrival has decided before. Accordingly, no clock is set in location $(0, c)$.

### 5.1 Probabilistic Transition Systems

The formal interpretation of stochastic automata is defined in terms of probabilistic transition systems. Probabilistic transition systems are labelled transition systems that contain two disjoint sets of states: probabilistic and non-deterministic states. A non-deterministic state has zero or more outgoing transitions. These transitions determine how the system evolves from the state to a possibly non-deterministically determined next probabilistic state. A probabilistic state has a single outgoing probabilistic transition. A probabilistic transition is a function that maps a probabilistic state onto a probability space whose sample space ranges over the set of non-deterministic states. Paths through a probabilistic transition system are thus sequences of alternating non-deterministic and probabilistic states.

**Definition 3.** *Let* $Prob(H)$ *denote the set of probability spaces* $(\Omega, \mathcal{F}, \Pr)$ *such that* $\Omega \subseteq H$.[3] *A* probabilistic transition system *(PTS, for short) is a structure* $PTS = (N, P, \mathcal{L}, T, \longrightarrow, \sigma_0)$ *where*

- $N$ *is a set of* non-deterministic *states*
- $P$ *is a set of* probabilistic *states with* $N \cap P = \varnothing$
- $\mathcal{L}$ *is a set of* labels
- $T : P \to Prob(N)$ *is a (total) function called* probabilistic transition relation
- $\longrightarrow \subseteq N \times \mathcal{L} \times P$ *is the* labelled (or non-deterministic) transition relation
- $\sigma_0 \in P$ *is the* initial *(probabilistic) state*

$(\sigma', \ell, \sigma) \in \longrightarrow$ will be denoted by $\sigma' \xrightarrow{\ell} \sigma$. In the setting of this paper, the set of labels is $\mathcal{L} = \mathbf{A} \times \mathbb{R}_{\geqslant 0}$, where $\mathbf{A}$ is a set of action names. The reals denote the (relative) time at which an action takes place. Transition $\sigma \xrightarrow{a,d} \sigma'$ denotes that the system evolves from non-deterministic state $\sigma$ to probabilistic state $\sigma'$ by offering action $a$ after idling precisely $d$ time in state $\sigma$.

[3] A probability space $(\Omega, \mathcal{F}, \Pr)$ consists of a (sample) set $\Omega$, a $\sigma$-algebra $\mathcal{F}$ and a probability measure $\Pr : \mathcal{F} \to \mathbb{R}_{\geqslant 0}$. More details can be found in [70].

*Example 4.* Consider an automatic switch that controls a light. Assume that the delay between two successive on-button switchings is governed by a negative exponential distribution with mean 30 minutes and that the switch automatically switches off the light in case the on-button has not been switched for exactly 2 minutes. The behavior of the switch is modeled by the following PTS:

- $N = \mathbb{R}^2 \cup \mathbb{R}$
- $P = \{\sigma_{init}, \sigma_{on}\} \cup (\{\sigma_{off}\} \times \mathbb{R})$
- $\mathcal{L} = \{on, off\} \times \mathbb{R}_{\geqslant 0}$
- $T(\sigma_{init}) = \mathcal{R}(F_{e,30})$, $T(\sigma_{on}) = \mathcal{R}(F_{e,30}, D_2)$, and $T(\sigma_{off}, d) = Triv(d)$, and
- $\longrightarrow$ is the smallest relation such that:
  - $(d, d') \xrightarrow{on(d)} \sigma_{on} \Leftrightarrow 0 \leqslant d \leqslant d'$
  - $(d, d') \xrightarrow{off(d')} (\sigma_{off}, d - d') \Leftrightarrow 0 \leqslant d' \leqslant d$
  - $d \xrightarrow{on(d)} \sigma_{on} \Leftrightarrow 0 \leqslant d$
- $\sigma_0 = \sigma_{init}$

where $\mathcal{R}(F_{e,30})$ is the probability space on the real line with the unique probability measure obtained from $F_{e,30}(t) = 1 - e^{-\frac{t}{30}}$, $\mathcal{R}(F_{e,30}, D_2)$ is the probability space on the real plane obtained from $F_{e,30}$ and $D_2(t) = 0$ for $t < 2$ and 1 otherwise, and $Triv(d)$ is the trivial probability space on $\{d\}$.

The PTS is explained as follows. The system starts in $\sigma_{init}$, the state in which the light is off. By taking a probabilistic transition, the time $d$ until switching the light on is determined according to distribution $F_{e,30}$ and the system evolves to state $d$, where the switch waits until it is switched on. If the light is switched on, the system moves to the probabilistic state $\sigma_{on}$. On taking the probabilistic transition from $\sigma_{on}$ two time instants are randomly determined: time $d$ until the light is switched on (again) and time $d'$ until the light turns itself off. The next non-deterministic state is thus $(d, d')$. Note that $d' = 2$ with probability one. Now two scenarios may occur. If the light is switched on first (i.e., $d \leqslant d'$), the on-button is switched: $(d, d') \longrightarrow \sigma_{on}$ labelled with action $on(d)$. In this state, both time values will be determined again. In the other case, the light turns off while reaching state $(\sigma_{off}, d - d')$ via performing action $off(d')$, where $d - d'$ is the remaining time until the next switching. From state $(\sigma_{off}, d)$ the non-deterministic state $d$ is reached with probability one, where the switch waits until it is switched on.

### 5.2 Interpretation of Stochastic Automata in Terms of PTSs

The formal semantics of a stochastic automaton is defined in terms of a PTS. The relation between stochastic automata and PTSs is similar to the relation between timed automata and timed transition systems. We present the mapping of stochastic automata onto PTSs in an informal way; a formal treatment can be found in [27,31]. The states of the PTS keep track of the current location and the values of all clocks involved. Values of clocks are determined by a *valuation*, a function that assigns to each clock $x \in \mathcal{C}$ its real value $v(x) \in \mathbb{R}$. (We let

$\mathcal{V}$ denote the set of valuations.) States are thus pairs $(s, v)$. To distinguish non-deterministic and probabilistic states we write $[s, v]$ for a non-deterministic state, and use $(s, v)$ for probabilistic states. As above, the labelled transition is labelled with pairs $(a, d)$, for action $a$ and delay $d$.

Performing an edge in the stochastic automaton is represented by a sequence of two steps. Suppose there is an edge from location $s$ to $s'$ labelled with $a, C$, and assume the current state is $[s, v]$. If after delaying for some time $d$, say, all clocks in $C$ have expired, then $[s, v] \xrightarrow{a,d} (s', v')$ with $s'$ the location just reached in the stochastic automaton and $v'$ such that for all clocks $d$ time units have passed: $v'(x) = v(x) - d$. While entering location $s'$ though, the clocks in $\kappa(s')$ need to be set. This is accomplished by a subsequent probabilistic transition starting from $(s', v')$, the state just reached. For the sake of simplicity, assume that $\kappa(s') = \{ x, y \}$. Then, $T(s', v')$ is a unique probability space induced by the distributions of the clocks, $F_x$ and $F_y$, in a Borel space on a two-dimensional real hyper-space. The clocks that are not in $\kappa(s')$ keep their value while the clocks $x$ and $y$ are initialized according to their distributions. Thus, in the resulting state $[s', v'']$ the system is still in location $s'$ (as expected), and $v''(z) = v'(z)$ for each $z$ different from $x$ and $y$, while $v''(x) = F_x()$ and $v''(y) = F_y()$. Note the resemblance of this recipe with step 4. of the procedure in Section 3 that described the dynamics of a GSMC.

There is a subtlety though in the first step of the recipe. According to the above procedure, $[s, v] \rightarrow (s', v')$ if all clocks in $C$ have expired. However, we did not make clear yet whether to take such transition *as soon as* all clocks in $C$ have expired, or whether it is allowed to take it at any time point once they have expired. In the first scenario, no delay is allowed once the clocks have expired – it adheres to the so-called "maximal progress" philosophy – while in the second such delay is allowed. Both interpretations have their own use:

- the notion of maximal progress is appropriate when the stochastic automaton is a *closed* system, i.e., a system which is complete by itself and which needs no further synchronization with other automata. As no further synchronizations are envisaged, there is no need to delay transitions any further once they are enabled, since there will be no further (external) processes that can delay their execution. Such perspective is useful, for instance, to determine the model's performance characteristics.
- to study reachability properties like freedom from deadlock, it is important to observe how the system behaves in an arbitrary context. That is, the interaction of a system with a certain "well-behaved" component may not induce a deadlock, while a "badly-behaved" component could take the system through an undesired path that will end in a deadlock situation. In this *open* system perspective, an action that is enabled may not be executed until the environment is also ready to execute such an action. Therefore, it may not take place as soon as it is enabled.

In the sequel, let $\mathcal{O}[\![ SA ]\!]^v$ denote the open interpretation of $SA$ with initial valuation $v$, and $\mathcal{C}[\![ SA ]\!]^v$ denote the closed interpretation of $SA$. Note that the

only difference between the closed and open interpretation is how to treat the expiration of clocks: either a delay is allowed once they expire (open), or it is not (closed). All other components of the PTSs $\mathcal{O}[\![ SA ]\!]$ and $\mathcal{C}[\![ SA ]\!]$ are the same.

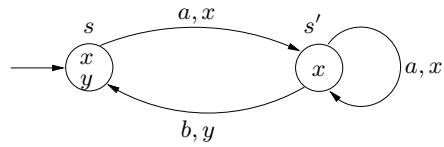



**Fig. 10.** A simple stochastic automaton

*Example 5.* To illustrate the difference between the open and closed interpretation, consider the stochastic automaton depicted in Fig. 10. In the closed interpretation the following non-deterministic transitions are present:

$$\begin{array}{ll}
[s,(d,d')] \xrightarrow{a,d} (s',(0,d'-d)) & \text{if and only if } \ d \geqslant 0 \\
[s',(d,d')] \xrightarrow{b,d'} (s,(d-d',0)) & \text{if and only if } \ 0 \leqslant d' \leqslant d \\
[s',(d,d')] \xrightarrow{a,d} (s,(0,d'-d)) & \text{if and only if } \ 0 \leqslant d \leqslant d'
\end{array}$$

where $(d,d')$ denotes valuation $v$ with $v(x) = d$ and $v(y) = d'$. Note the relationship between the clock values of $x$ and $y$ for taking an edge outgoing from location $s'$. In fact, if $F_x$ and $F_y$ would, for instance, be uniformly distributed on the intervals $[0,5]$ and $[11,12]$, respectively, it follows that in the closed interpretation the edge leading from $s'$ to $s$ will never be enabled. This follows from the fact that there is no valuation $(d,d')$ in $s'$ such that $d' \leqslant d$. Instead on entering location $s'$, clock $x$ is reset and will always expire before clock $y$ expires. In the open interpretation we obtain:

$$\begin{array}{ll}
[s,(d,d')] \xrightarrow{a,d^*} (s',(d-d^*,d'-d^*)) & \text{if and only if } \ d^* \geqslant 0 \wedge d^* \geqslant d \\
[s',(d,d')] \xrightarrow{b,d^*} (s,(d-d^*,d'-d^*)) & \text{if and only if } \ d^* \geqslant 0 \wedge d^* \geqslant d' \\
[s',(d,d')] \xrightarrow{a,d^*} (s,(d-d^*,d'-d^*)) & \text{if and only if } \ d^* \geqslant 0 \wedge d^* \geqslant d
\end{array}$$

In the open interpretation there is no relationship between clock $x$ and clock $y$. Instead, the only requirement is that the time $d^*$ of taking the transition is positive and beyond the time at which the edge becomes enabled. It follows that in this interpretation the edge from $s'$ to $s$ can indeed be taken for the uniform distributions given above.

### 5.3 How to Compare Stochastic Automata?

A key question in formal methods (and in practice) is whether a system implementation meets its specification, i.e., when is an implementation proper? In our setting this question can be dealt with in the following way: first model

the behavior of the specification as a stochastic automaton, do the same for the implementation, and then compare these two stochastic automata. Depending on the notion of "being a proper implementation", different comparisons can be made. For instance, if performance is not of much interest one may compare solely on the structure of the two automata while neglecting the probabilistic information. If, on the other hand, performance is of importance, such comparison is insufficient, and probabilistic information should be taken into account. Thus, for different perspectives, different notions of comparison are of interest. As a result, several equivalence relations (i.e., notions of comparison), are defined on stochastic automata.

*Isomorphism.* The first equivalence notions that we consider is isomorphism. Two stochastic automata are isomorphic if there exists a bijective function that maps locations from one to locations of the other without disturbing the structure of the automaton.

**Definition 4.** *Stochastic automata* $SA_1 = (\mathcal{S}_1, s_0^1, \mathcal{C}, \mathbf{A}, \rightarrow_1, \kappa_1)$ *and* $SA_2 = (\mathcal{S}_2, s_0^2, \mathcal{C}, \mathbf{A}, \rightarrow_2, \kappa_2)$ *are* isomorphic*, denoted* $SA_1 \sim_{iso} SA_2$*, iff there exists a bijection* $\phi : \mathcal{S}_1 \rightarrow \mathcal{S}_2$ *such that*

- $\phi(s_0^1) = s_0^2$*, and*
- $s \rightarrow_1 s'$ *if and only if* $\phi(s) \rightarrow_2 \phi(s')$*, and*
- $\kappa_1(s) = \kappa_2(\phi(s))$

Note: for simplicity we have assumed that the clocks and actions of both automata are identical; it is not difficult to extend this notion with an isomorphism on clocks and actions.

*Structural bisimulation.* In concurrency theory, one of the most interesting equivalence relations for labelled transition systems is *bisimulation* [77]: two states are bisimilar if they can mimic each other while evolving into bisimilar states. Two labelled transition systems are bisimilar if and only if their initial states are bisimilar. The notion of structural bisimulation decides the equivalence of stochastic automata by inspecting their structure only.

**Definition 5.** *Let* $(\mathcal{S}, s_0, \mathcal{C}, \mathbf{A}, \rightarrow, \kappa)$ *be a stochastic automaton.* $R \subseteq \mathcal{S} \times \mathcal{S}$ *is a* structural bisimulation *if* $R$ *is symmetric and whenever* $s_1 R s_2$*:*

1. $\forall a \in \mathbf{A}, C \subseteq \mathcal{C}.\ s_1 \xrightarrow{a,C} s_1'$ *implies* $\exists s_2'.\ s_2 \xrightarrow{a,C} s_2'$ *and* $s_1' R s_2'$
2. $\kappa(s_1) = \kappa(s_2)$

*If* $R$ *is a structural bisimulation such that* $s_1 R s_2$*, we write* $s_1 \sim s_2$ *and call* $s_1$ *and* $s_2$ *structural bisimilar.*

Stochastic automata $SA_1$ and $SA_2$ are *structural bisimilar*, notation $SA_1 \sim SA_2$, if their respective initial locations are structural bisimilar on the disjoint union of $SA_1$ and $SA_2$. If we omit the clock-related information, structural bisimulation reduces to usual (strong) bisimulation on labelled transition systems [77].

*Example 6.* Isomorphic stochastic automata are structural bisimilar, but the reverse is not true, cf. Fig. 11. In the left-hand process, there is a non-deterministic choice in the initial location to move (while emitting $a$ on expiration of clock $x$) to an absorbing location, i.e., a location without outgoing transitions. There is no bijection that maps the absorbing locations to the single absorbing location in the right-hand process, and thus, these processes are not isomorphic. The automata are structural bisimilar since the relation $R = \{ (s_1, t_1), (s_2, t_2), (s_3, t_2) \}$ is a structural bisimulation.

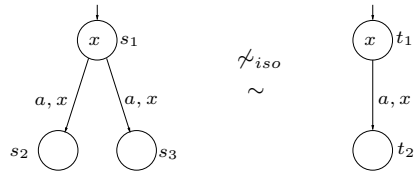



**Fig. 11.** Two structural bisimilar but non-isomorphic stochastic automata

*Probabilistic bisimulation.* Structural bisimulation is a simple notion of equivalence and does not take any probabilistic information into account. In order to define a probabilistic variant of bisimulation we consider a notion of *probabilistic bisimulation* on PTSs. Subsequently, we lift this to stochastic automata and illustrate its usage.

**Definition 6.** *Let $(N, P, \mathcal{L}, T, \longrightarrow)$ be a PTS with $S \subseteq N$ and $\sigma \in P$ such that $T(\sigma) = (\Omega, \mathcal{F}, \Pr)$. Then $\mu : P \times \mathcal{P}(N) \to [0, 1]$ is defined by:*

$$\mu(\sigma, S) = \begin{cases} \Pr(S \cap \Omega) & \text{if } S \cap \Omega \in \mathcal{F} \\ 0 & \text{otherwise} \end{cases}$$

*Let $R$ be an equivalence relation on $N \cup P$ such that if $\langle \sigma_1, \sigma_2 \rangle \in R$ then either $\sigma_1, \sigma_2 \in N$ or $\sigma_1, \sigma_2 \in P$. Let $N/R$ be the set of equivalence classes in $N$ induced by $R$. $R$ is a* probabilistic bisimulation *if for all $\langle \sigma_1, \sigma_2 \rangle \in R$:*

1. *$\forall S \subseteq N/R$: $\mu(\sigma_1, \bigcup S) = \mu(\sigma_2, \bigcup S)$, whenever $\sigma_1, \sigma_2 \in P$*
2. *$\forall \ell \in \mathcal{L}$: $\sigma_1 \xrightarrow{\ell} \sigma_1'$ implies $\sigma_2 \xrightarrow{\ell} \sigma_2'$ and $\langle \sigma_1', \sigma_2' \rangle \in R$, for some $\sigma_2' \in P$, whenever $\sigma_1, \sigma_2 \in N$*

*States $\sigma_1$ and $\sigma_2$ are* probabilistically bisimilar*, denoted $\sigma_1 \sim_p \sigma_2$, if there exists a probabilistic bisimulation $R$ with $\langle \sigma_1, \sigma_2 \rangle \in R$. $PTS_1$ and $PTS_2$ with initial state $\sigma_1$ and $\sigma_2$, respectively, are probabilistic bisimilar if $\sigma_1 \sim_p \sigma_2$ in the (disjoint) union of $PTS_1$ and $PTS_2$.*

Due to the involvement of continuous distributions, the definition is slightly more involved than existing definitions that consider only discrete distributions [44,71,90]. Nevertheless, both types of probabilistic bisimulation coincide

on the discrete case. A proof of this fact, together with a proof that $\sim_p$ is an equivalence relation, can be found in [27]. Probabilistic bisimulation is lifted to stochastic automata in the following way:

- $SA_1$ and $SA_2$ are *open* probabilistic bisimilar, denoted $SA_1 \sim_p^\circ SA_2$ if and only if $\mathcal{O}[\![ SA ]\!]^{v_0} \sim_p \mathcal{O}[\![ SA ]\!]^{v_0}$, for any initial valuation $v_0$;
- $SA_1$ and $SA_2$ are *closed* probabilistic bisimilar, denoted $SA_1 \sim_p^\bullet SA_2$ if and only if $\mathcal{C}[\![ SA ]\!]^{v_0} \sim_p \mathcal{C}[\![ SA ]\!]^{v_0}$, for any initial valuation $v_0$.

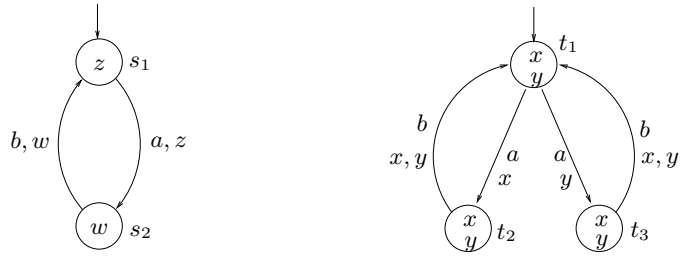



**Fig. 12.** Two open p-bisimilar stochastic automata

*Example 7.* The stochastic automata in Fig. 12 are open p-bisimilar if

$$F_z(t) = F_{\min\{x,y\}}(t) = 1 - (1 - F_x(t))(1 - F_y(t)) \text{ and}$$
$$F_w(t) = F_{\max\{x,y\}}(t) = F_x(t) \cdot F_y(t)$$

since

$$\begin{aligned} R = \quad & \{\, ((s_1, v), (t_1, u)) \mid u, v \in \mathcal{V} \,\} \\ & \cup \{\, ([s_1, v], [t_1, u]) \mid u, v \in \mathcal{V}, v(z) = \min\{u(x), u(y)\} \,\} \\ & \cup \{\, ((s_2, v), (t_i, u)) \mid u, v \in \mathcal{V}, i \in \{\, 2, 3 \,\} \,\} \\ & \cup \{\, ([s_2, v], [t_i, u]) \mid u, v \in \mathcal{V}, i \in \{\, 2, 3 \,\}, v(w) = \max\{u(x), u(y)\} \,\} \end{aligned}$$

is a probabilistic bisimulation between their open PTSs. This can intuitively be seen as follows. On entering location $t_1$ in the right-hand automaton, a race takes place between clocks $x$ and $y$. According to the clock that expires first — this corresponds to the minimum of their distributions — a transition is made to either $t_2$ or $t_3$ while performing $a$ and setting clocks $x$ and $y$. The same behavior can take place in location $s_1$ in the left-hand stochastic automaton where $a$ takes place after expiration of clock $z$ with $F_z = F_{\min\{x,y\}}$. From the locations $t_2$ and $t_3$ a $b$-transition is possible back to location $t_1$ when both clocks $x$ and $y$ — this corresponds to the maximum of their distributions — have expired. This is mimicked by going from $s_2$ to $s_1$ after the expiration of clock $w$ with $F_w = F_{\max\{x,y\}}$. Note that it is crucial that both clocks $x$ and $y$ are set in locations $t_2$ and $t_3$; otherwise the automata would not have been open probabilistic bisimilar.

**Theorem 1.** $\sim_p^\bullet \subset \sim_p^\circ \subset \sim \subset \sim_{iso}$.

This result relates the different notions of equivalence that were introduced: isomorphism is the strongest notion, whereas $\sim_p^\bullet$ is the weakest one.

The inclusions in Theorem 1 are strict as exemplified by Fig. 13. Stochastic automata (a) and (b) are structural bisimilar, but not isomorphic, as we have seen before; (b) and (c) are not structural bisimilar, since e.g., the initial locations cannot be related (due to different clocks), but are open probabilistic bisimilar, as both automata can perform an $a$-action after an equal stochastic delay while evolving to equivalent locations. Finally, the stochastic automata (d) and (e) are closed probabilistic bisimilar as both automata perform an $a$ immediately, but are not open probabilistic bisimilar, because the maximal progress condition does not apply. For instance, a context that is able to participate in $b$ (after imposing a possible extra delay) but forbids action $a$ distinguishes the two processes.

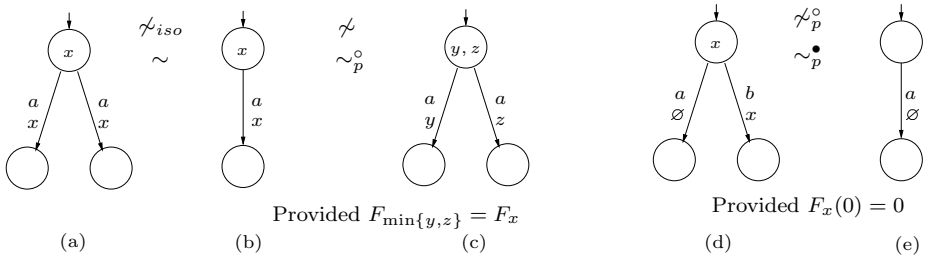



**Fig. 13.** Structural vs. open vs. closed probabilistic bisimulation

Structural bisimulation is defined directly on stochastic automata, but does not consider any stochastic information. Open and closed probabilistic bisimilarity do take the probabilistic behavior into account, but are defined in terms of the underlying, infinite PTS. Probabilistic information can be considered at a symbolic level too by considering a form of structural bisimulation [27]. The treatment of this notion falls outside the scope of this tutorial.

### 5.4 GSMCs versus Stochastic Automata

The relation between stochastic automata and GSMCs is shown by providing a mapping, denoted gsmc2sa, from GSMCs onto stochastic automata. The existence of this mapping indicates that GSMCs are properly included in stochastic automata.

The basic idea of the mapping of a GSMC onto a stochastic automaton is to introduce a location as a pair $(z, E)$ where $z$ is a state of the GSMC and $E$ is the set of events that are already active. The initial location is $(z_0, \varnothing)$. For each active event in state $z$, there is an outgoing edge from any location $(z, E)$. This

edge is labelled with event $e$ (i.e., the action) and the set of clocks $\{C(e)\}$. So, events are considered as actions and active events of $z$ are

$$E(z) = \bigcup \{ e \mid (z, E) \xrightarrow{e,\{C(e)\}} \}.$$

Since in a GSMC exactly one clock is associated to an event, we obtain singleton sets as triggering conditions in the automaton. In the following, we use $C$ on sets of events in the usual way, i.e., $C(E) = \{ C(e) \mid e \in E \}$.

**Definition 7.** *For GSMC* $\mathcal{G} = (Z, z_0, \mathbf{E}, E, C, \mathsf{next})$ *the stochastic automaton* $\mathsf{gsmc2sa}(\mathcal{G}) = (\mathcal{S}, s_0, \mathcal{C}, \mathbf{A}, \longrightarrow, \kappa)$ *is defined by:*

- $\mathcal{S} = Z \times \mathcal{P}_f(\mathbf{E})$ *with* $s_0 = (z_0, \varnothing)$,
- $\mathcal{C} = C(\mathbf{E})$,
- $\mathbf{A} = \mathbf{E}$,
- $\kappa(z, E) = C(E(z) - E)$, *and*
- $\longrightarrow$ *is defined by the rule*

$$\frac{e \in E(z)}{(z, E) \xrightarrow{e,\{C(e)\}} (\mathsf{next}(z, e), E(z) - \{e\})}$$

Due to the fact that $E(z) \neq \varnothing$ for any $z$, the condition $e \in E(z)$ is always satisfied. There are many locations $(z, E) \in \mathcal{S}$ that are unreachable via $\longrightarrow$. All reachable locations have the form $(\mathsf{next}(z, e), E(z) - \{e\})$ for every (reachable) $z \in Z$ and $e \in E(z)$. Note that for $z' = \mathsf{next}(z, e)$ we have $\kappa(z', E(z) - \{e\}) = C(E(z') - (E(z) - \{e\}))$, the set of clocks for all newly active events in $z'$.

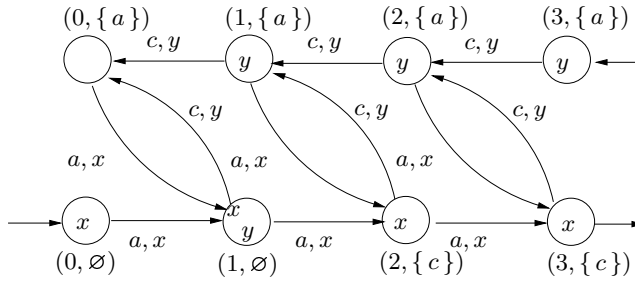



**Fig. 14.** Stochastic automaton generated from example GSMC of Fig. 3

*Example 8.* Consider the GSMC of the G/G/1/$\infty$-system depicted in Fig. 3. The stochastic automaton that corresponds to this GSMC is obtained in the following way: the initial location is $(0, \varnothing)$, the state in which there are no jobs in the queue, and there is no active event; $\kappa(0, \varnothing) = C(E(0)) = \{ x \}$; since $a \in E(0)$, and $C(a) = x$, the initial location has a single outgoing edge labelled $a, x$ leading to location $(\mathsf{next}(a, 0), E(0) - \{ a \}) = (1, \varnothing)$ with $\kappa(1, \varnothing) = C(E(1)) = \{ x, y \}$.

It is easy to check that this location has an edge labelled $c, y$ to location $(0, \{ a \})$ and an edge labelled $a, x$ to $(2, \{ c \})$. If we continue this reasoning we obtain the automaton of Fig. 14. Note that this automaton is isomorphic to Fig. 9.

The correctness of the translation gsmc2sa is assessed in [27]. As argued before, stochastic automata are more expressive than GSMCs, since stochastic automata do allow non-determinism (two or more outgoing edges that are enabled at the same time), whereas GSMCs do not. Therefore a reverse translation does not make much sense. In addition, in the stochastic automaton model clocks may be initialized by general distributions — including discrete distribution functions — without any restriction.

## 6 The Stochastic Process Algebra

The basic idea of the stochastic process algebra SPADES (Stochastic Process Algebra for Discrete-Event Simulation), symbolized by , is to separate the three ingredients that are present in stochastic automata at a syntactical level. We thus distinguish in  explicitly between:

- the start of a probabilistic delay (denoted set $C$ in $p$),
- the completion of a probabilistic delay (denoted when $C \mapsto p$), and
- the occurrence of immediate actions (denoted $a; p$).

As we will see, this separation allows us to obtain a straightforward expansion law.

### 6.1 Syntax and Semantics

*Syntax.* Let $\mathbf{A}$ be a set of *actions*, $\mathbf{V}$ a set of *process variables*, and $\mathcal{C}$ a set of clocks with $(x, G) \in \mathcal{C}$ for $x$ a clock name and $G$ an general probability distribution function. We abbreviate $(x, G)$ by $x_G$.

**Definition 8.** *The syntax of  is defined by:*

$$p ::= \mathbf{0} \mid a; p \mid \mathsf{when}\ C \mapsto p \mid p + p \mid \mathsf{set}\ C\ \mathsf{in}\ p \mid p \,||_A\, p \mid p[f] \mid X.$$

*where $C \subseteq \mathcal{C}$ is finite, $a \in \mathbf{A}$, $A \subseteq \mathbf{A}$, $f : \mathbf{A} \to \mathbf{A}$, and $X \in \mathbf{V}$. A* recursive specification *$E$ is a set of recursive equations of the form $X = p$ for each $X \in \mathbf{V}$, where $p \in$ .*

Besides the operations used in Section 4 the language incorporates the basic process $\mathbf{0}$, the process that cannot perform any action. A few words on $p + q$ are in order. $p + q$ behaves either as $p$ or $q$, but not both. At execution the fastest process, i.e. the process that is enabled first, is selected. This is known as the race condition. If this fastest process is not uniquely determined, a non-deterministic selection among the fastest processes is made.

Operators when $C \mapsto \ldots$, set $C$ in $\ldots$, prefixing and renaming have the highest binding precedence, followed by choice and parallel composition.

*Semantics.* To associate a stochastic automaton $SA(p)$ to a given term $p$ in the language, we define the different components of $SA(p)$[4]. In order to define the automaton associated to a parallel composition, we introduce the additional operation nock. $\mathsf{nock}(p)$ is a process that behaves like $p$ except that no clock is set at the very beginning. As usual in structured operational semantics, a location corresponds to a term. The clock setting function $\kappa$ is defined as the smallest set satisfying the equations in Table 1. The set of edges $\longrightarrow$ between locations is

| | | | |
|---|---|---|---|
| $\kappa(\mathbf{0})$ | $= \varnothing$ | $\kappa(\mathsf{set}\ C\ \mathsf{in}\ p)$ | $= \kappa(p) \cup C$ |
| $\kappa(a; p)$ | $= \varnothing$ | $\kappa(p \,\|\|_A\, q)$ | $= \kappa(p) \cup \kappa(q)$ |
| $\kappa(\mathsf{when}\ C \mapsto\ p)$ | $= \kappa(p)$ | $\kappa(p[f])$ | $= \kappa(p)$ |
| $\kappa(p + q)$ | $= \kappa(p) \cup \kappa(q)$ | $\kappa(X)$ | $= \kappa(p) \quad (X = p)$ |
| $\kappa(\mathsf{nock}(p))$ | $= \varnothing$ | | |

**Table 1.** Clock setting function for

defined as the smallest relation satisfying the rules in Table 2. The function $F$ is defined by $F(x_G) = G$ for each clock $x$ in $p$. The other components are defined as for the syntax of .

Let us briefly explain the operational rules from Table 2.

- There is no rule for the process $\mathbf{0}$ as it cannot perform any action.
- The action-prefixed process $a; p$ can immediately perform an $a$ while evolving into $p$. Since $a$ is performed immediately, there is no need to wait for the expiration of a clock. So, the transition is labelled with an empty set of clocks.
- Process when $C \mapsto\ p$ can perform any action that $p$ can perform, with the restriction that it has to wait until all clocks in the set $C$ have expired. So, if $p$ has to wait for the expiration of all clocks in $C'$ to perform action $a$, then process when $C \mapsto\ p$ has to wait for all clocks in $C \cup C'$.
- Processes set $C$ in $p$ and $\mathsf{nock}(p)$ can mimic $p$; their difference with $p$ is solely in the clock-setting function, cf. Table 1.
- $p + q$ behaves like either $p$ or $q$ (but not both).
- $p[f]$ behaves like $p$ except that all actions are renamed according to $f$.
- Process $X$ behaves like $p$, provided it is defined as $X = p$.

[4] Here we assume that $p$ does not contain any name clashes of clock variables. This is not a severe restriction since terms that suffer from such name clash can always be properly renamed into a term without such name clash [27].

$$a;p \xrightarrow{a,\varnothing} p \qquad \frac{p \xrightarrow{a,C'} p'}{\mathsf{when}\ C \mapsto\ p \xrightarrow{a,C\cup C'} p'} \qquad \frac{p \xrightarrow{a,C'} p'}{\mathsf{set}\ C\ \mathsf{in}\ p \xrightarrow{a,C'} p'}$$

$$\frac{p \xrightarrow{a,C} p'}{\begin{array}{l} p+q \xrightarrow{a,C} p' \\ q+p \xrightarrow{a,C} p' \end{array}} \qquad \frac{p \xrightarrow{a,C} p'}{p[f] \xrightarrow{f(a),C} p'[f]} \qquad \frac{p \xrightarrow{a,C} p'}{\mathsf{nock}(p) \xrightarrow{a,C} p'} \qquad \frac{p \xrightarrow{a,C} p'}{X \xrightarrow{a,C} p'}(X=p)$$

$$\frac{p \xrightarrow{a,C} p'}{\begin{array}{l} p\,||_A\,q \xrightarrow{a,C} p'\,||_A\,\mathsf{nock}(q) \\ q\,||_A\,p \xrightarrow{a,C} \mathsf{nock}(q)\,||_A\,p' \end{array}}(a\notin A) \qquad \frac{p \xrightarrow{a,C} p' \quad q \xrightarrow{a,C'} q'}{p\,||_A\,q \xrightarrow{a,C\cup C'} p'\,||_A\,q'}(a\in A)$$

**Table 2.** Structured operational semantics for

- For parallel composition two situations are distinguished:
  - In case a synchronization takes place, i.e., some action $a \in A$ is performed, both involved processes must be ready to perform $a$. So, all clocks needed to perform $a$ in both processes have to be expired.
  - If a process carries out an action not in $A$, it does so autonomously. Naively, this yields the following traditional operational rule:

$$\frac{p \xrightarrow{a,C} p'}{p\,||_A\,q \xrightarrow{a,C} p'\,||_A\,q}$$

    for $a \notin A$. This would, however, lead to a situation in which all clocks in $p'$ and $q$ are reset in the resulting location. This is incorrect for the clocks in $q$, since now the elapse of time since the clocks of $q$ were set (when reaching $p\,||_A\,q$) is neglected, cf. equation (4) and Fig. 2. To solve this problem, the state $p'\,||_A\,\mathsf{nock}(q)$ is reached instead where the use of $\mathsf{nock}(q)$ avoids the setting of the clocks in $q$, i.e., $\kappa(\mathsf{nock}(q)) = \varnothing$, cf. Table 1.

*Example 9.* Using this recipe it can be shown that the semantics of the process algebraic G/G/1/$\infty$ specification boils down to the (at first sight somewhat complicated) stochastic automaton depicted in Fig. 15. Here, empty sets are omitted; in particular $b$ stands for $b, \varnothing$. Note that the in locations that are reached after the server has just completed servicing a job, no clocks are set. Although the state space of this automaton is somewhat larger than that of the direct representation in Fig. 14, this does not have a serious impact on the efficiency of stochastic simulation, as will see later on. Since in our semantics a state corresponds to a term, simulation can be carried out on the basis of expressions rather than using their semantic representations. This will be further discussed in Section 7.

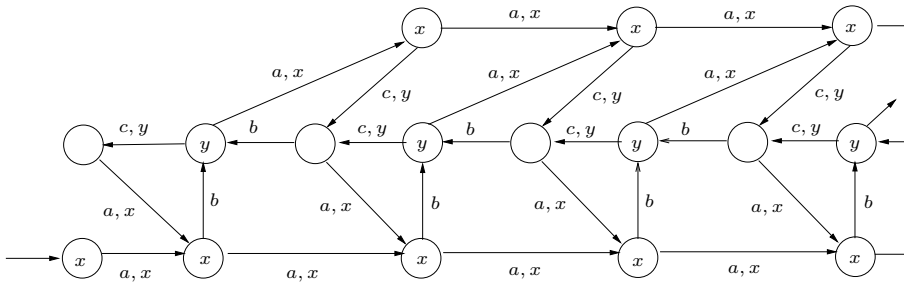



**Fig. 15.** Stochastic automaton of the compositional $G/G/1/\infty$ specification

*Expressive power of stochastic automata versus* ⏢. Stochastic automata and ⏢ are equally expressive. This means that for any stochastic automaton a corresponding term in ⏢ can be given whose reachable part of its stochastic automaton is identical to the stochastic automaton at hand, up to renaming of clocks.

**Theorem 2.** *For every stochastic automaton SA there exists a recursive specification E with root p in* ⏢ *such that the reachable part of SA and the reachable part of* $SA(p)$ *are isomorphic.*

As a result of this theorem and the fact that GSMCs are a proper subset of stochastic automata, it follows that each GSMC is representable by a ⏢ specification.

*Example 10.* Consider the stochastic automaton of Fig. 9. The reader is invited to check that this stochastic automaton is isomorphic to the following recursive ⏢ specification for $i \geqslant 0$:

$$
\begin{aligned}
P_0 &= \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto a; P_1) \\
P_1 &= \mathsf{set}\ x, y\ \mathsf{in}\ (\mathsf{when}\ y \mapsto c; Q_0 + \mathsf{when}\ x \mapsto a; P_2) \\
P_{i+2} &= \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ y \mapsto c; Q_{i+1} + \mathsf{when}\ x \mapsto a; P_{i+3}) \\
Q_0 &= \mathsf{set}\ \varnothing\ \mathsf{in}\ (\mathsf{when}\ x \mapsto a; P_1) \\
Q_{i+1} &= \mathsf{set}\ y\ \mathsf{in}\ (\mathsf{when}\ x \mapsto a; P_{i+2} + \mathsf{when}\ y \mapsto c; Q_i)
\end{aligned}
$$

The indices the processes indicate the number of jobs that are currently in the system, i.e., that are either queued or currently being processed. Note that the structure of each process definition is of the same shape: first some clocks are set, then their expiration is awaited, and an action takes place before invoking a process instance.

### 6.2 Notions of Congruence

The equivalence notions defined for stochastic automata (cf. Section 5.3) can be lifted to terms in ⏢ in the following straightforward way. Terms $p$ and $q$ are

structurally bisimilar, denoted $p \sim q$, if and only if $SA(p) \sim SA(q)$. In a similar way we define: $p \sim_p^\circ q$ iff $SA(p) \sim_p^\circ SA(q)$ and $p \sim_p^\bullet q$ iff $SA(p) \sim_p^\bullet SA(q)$. Here, we will investigate whether these equivalence notions are congruences. Recall that an equivalence relation is a congruence if two equivalent terms behave indistinguishable in any context. We have:

**Theorem 3.** *$\sim$ and $\sim_p^\circ$ are congruences with respect to all operators in* $\clubsuit$.

(It should be noted that $\sim$ is also a congruence for recursion and $\mathsf{nock}$.) Due to this result, replacing in a $\clubsuit$ specification a sub-term $p$ by its bisimilar equivalent $q$ results in a bisimilar $\clubsuit$ specification. The proof of this result for $\sim$ is rather simple and follows from the fact that the operational rules of Table 2 obey a certain syntactic format (the so-called path format of [7]); the proof for $\sim_p^\circ$ is rather involved and tedious as it requires manipulations on Borel spaces, cf. [27]. $\sim_p^\bullet$ is not a congruence for parallel composition as illustrated by the following example.

*Example 11.* Consider the processes

$$p = a;\mathbf{0} + \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto\ b;\mathbf{0})$$
$$q = a;\mathbf{0} + \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto\ c;\mathbf{0})$$

where $b$ and $c$ are distinct actions. Note that $q = p[b/c]$. We have $p \sim_p^\bullet q$ if $F_x(0) = 0$, since then in both processes only action $a$ at time 0 can be performed due to the maximal progress principle. However,

$$p \,||_a\, \mathbf{0} \not\sim_p^\bullet q \,||_a\, \mathbf{0}$$

In the context process $\mathbf{0}$, the execution of action $a$ is disabled, since there is no possible synchronization, and therefore $b$ or $c$ will happen (at a certain positive time instant). This example is depicted in terms of stochastic automata in Fig. 16.

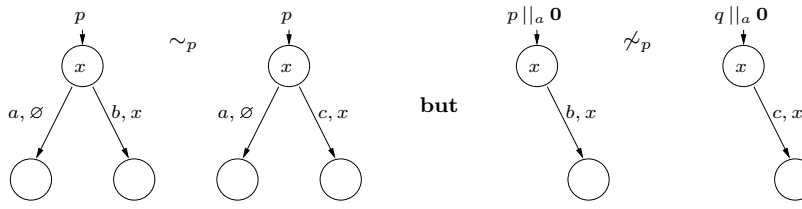



**Fig. 16.** Closed probabilistic bisimulation is not a congruence for $||_A$

The reader is invited to check that $\sim_p^\bullet$ is also not a congruence for the operator $\mathsf{when}\ C \mapsto p$.

### 6.3 Equational Reasoning

Rather than proving that $p$ and $q$ are e.g., structural bisimilar using the semantic interpretations $SA(p)$ and $SA(q)$ it is often more convenient to use rules defined on the syntax of $p$ and $q$ that are known to preserve (in this case) $\sim$. This enables the transformation and comparison of terms at a purely syntactic level. In this section, we present a sound and complete axiomatization for structural bisimulation of the core calculus of 🔔, i.e., the fragment of 🔔 consisting of the basic operators, i.e., excluding recursion and parallel composition. In addition, some sound axioms for open probabilistic bisimulation will be presented.

*Axiomatization of structural bisimulation.* Let the set $fv(p)$ of free (clock) variables of $p$ be defined as the smallest set satisfying the equations in Table 3. The

| | | | |
|---|---|---|---|
| $fv(\mathbf{0})$ | $= \varnothing$ | $fv(\mathsf{set}\ C\ \mathsf{in}\ p)$ | $= fv(p) - C$ |
| $fv(a;p)$ | $= fv(p)$ | $fv(p \,\|\|_A\, q)$ | $= fv(p) \cup fv(q)$ |
| $fv(\mathsf{when}\ C \mapsto p)$ | $= C \cup fv(p)$ | $fv(p[f])$ | $= fv(p)$ |
| $fv(p+q)$ | $= fv(p) \cup fv(q)$ | $fv(X)$ | $= fv(p)$ (X = p) |
| $fv(\mathsf{nock}(p))$ | $= fv(p) \cup \kappa(p)$ | | |

**Table 3.** Free variables of terms in 🔔

axioms for structural bisimulation are given in Table 4 and can be explained as follows. As in traditional process algebra, the choice is commutative (**A1**) and associative (**A2**), and **0** is the neutral element for $+$ (**A4**). Axiom **A3** is a distinguishing law for stochastic process algebra (also for Markovian process algebra) and can be regarded as a weak version of the traditional idempotence axiom of choice ($p + p = p$). The reason of not having this axiom of choice is that in case of two competing processes such as

$$(\mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto p)) + (\mathsf{set}\ y\ \mathsf{in}\ (\mathsf{when}\ y \mapsto p))$$

the resolution of the choice is controlled by the minimum of two random variables with the distributions of $x$ and $y$, respectively. As a result, the term $p$ is reached "faster" than in either of the sub-terms of the choice. Accordingly,

$$\mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto a;p) + \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto a;p) \neq \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto a;p) \tag{6}$$

Later on, we will make this more precise when discussing some axioms of open probabilistic bisimulation. Note that, however, due to **T5** and **A3** we have:

$$\mathsf{set}\ x\ \mathsf{in}\ ((\mathsf{when}\ x \mapsto a;p) + (\mathsf{when}\ x \mapsto a;p)) = \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto a;p)$$

| | | |
|---|---|---|
| **A1** | $p + q = q + p$ | |
| **A2** | $(p + q) + r = p + (q + r)$ | |
| **A3** | $a; p + a; p = a; p$ | |
| **A4** | $p + \mathbf{0} = p$ | |
| **T1** | $\mathsf{when}\ C \mapsto \mathbf{0} = \mathbf{0}$ | |
| **T2** | $\mathsf{when}\ \varnothing \mapsto p = p$ | |
| **T3** | $\mathsf{when}\ C \mapsto (\mathsf{when}\ C' \mapsto p) = \mathsf{when}\ (C \cup C') \mapsto p$ | |
| **T4** | $\mathsf{when}\ C \mapsto (\mathsf{set}\ C'\ \mathsf{in}\ p) = \mathsf{set}\ C'\ \mathsf{in}\ (\mathsf{when}\ C \mapsto p)$ | if $C \cap C' = \varnothing$ |
| **T5** | $\mathsf{when}\ C \mapsto (p + q) = \mathsf{when}\ C \mapsto p + \mathsf{when}\ C \mapsto q$ | |
| **C1** | $\mathsf{set}\ \varnothing\ \mathsf{in}\ p = p$ | |
| **C2** | $\mathsf{set}\ C\ \mathsf{in}\ (\mathsf{set}\ C'\ \mathsf{in}\ p) = \mathsf{set}\ (C \cup C')\ \mathsf{in}\ p$ | |
| **C3** | $(\mathsf{set}\ C\ \mathsf{in}\ p) + (\mathsf{set}\ C'\ \mathsf{in}\ q) = \mathsf{set}\ (C \cup C')\ \mathsf{in}\ (p + q)$ | if $C \cap (fv(q) \cup \kappa(q)) = C' \cap (fv(p) \cup \kappa(p)) = \varnothing$ |

**Table 4.** Axioms for structural bisimulation on 

where the setting of clock $x$ is common to both terms. In fact, one can show that the idempotence law $p + p = p$ is obtained if the structure of $p$ is restricted such that all clock setting operations of the form $\mathsf{set}\ C\ \mathsf{in}\ r$ in $p$ occur in a sub-term of the form $a; q$. Axioms **T1**–**T5** show the way in which triggering conditions can be simplified. In particular, **T3** defines how to reduce nested triggering conditions into a single one, and axioms **T4** and **T5** state how to move clock settings and summations out of the scope of a guard. Axiom **C1** expresses that it is irrelevant to set an empty set of clocks. **C2** gathers all the clocks settings in a single operation and **C3** moves clocks settings out of the scope of a summation. (The auxiliary notion of free variables in a term is defined in Table 3.)

For axioms for the static operators such as renaming and parallel composition we refer to [27].

*Expansion law.* Using the complete and sound axiomatization of structural bisimulation for  one can show that any (finite) term $p$ can be converted into a normal form which has the shape

$$\mathsf{set}\ C\ \mathsf{in}\ \left(\sum \mathsf{when}\ C_i \mapsto a_i; p_i\right)$$

where $p_i$ are terms in normal form and $\sum$ is the usual generalization of choice: $\sum_{0<i\leqslant n} p_i$ equals $p_1 + \ldots + p_n$ for $n{>}0$, and $\mathbf{0}$ for $n{=}0$. The reader is invited to check that this format is the same as the one used in Example 10.

As we have argued in the introduction, an essential law in traditional process algebras is the expansion law. This law allows to reduce parallel composition in terms of prefix and choice, and has proven to be of crucial importance for process

algebraic verification purposes. A stochastic equivalent of this law is defined by the following result.

**Theorem 4.** *Let* $p, q \in \hourglass$ *such that* $p = \mathsf{set}\ C\ \mathsf{in}\ p'$ *and* $q = \mathsf{set}\ C'\ \mathsf{in}\ q'$ *with* $p' = \sum(\mathsf{when}\ C_i \mapsto a_i; p_i)$ *and* $q' = \sum(\mathsf{when}\ C'_j \mapsto b_j; q_j)$. *Suppose* $p \,||_A\, q$ *does not contain name clashes. Then* $p \,||_A\, q$ *equals*

$$\begin{aligned}\mathsf{set}\ (C \cup C')\ \mathsf{in}\ \Big( & \textstyle\sum_{a_i \notin A} \mathsf{when}\ C_i \mapsto a_i; (p_i \,||_A\, q') \\ & + \textstyle\sum_{b_j \notin A} \mathsf{when}\ C'_j \mapsto b_j; (p' \,||_A\, q_j) \\ & + \textstyle\sum_{a_i = b_j \in A} \mathsf{when}\ (C_i \cup C'_j) \mapsto a_i; (p_i \,||_A\, q_j)\ \Big).\end{aligned}$$

In fact, the expansion law is inherent in our model and follows at the language level from the way in which we have distinguished between 3 activities in the syntax: (1) starting a delay, i.e., setting a clock, (2) finishing a delay, i.e., expiration of a clock, and (3) the occurrence of (immediate) actions.

*Example 12.* By means of the above expansion law we would like to discuss the way in which delayed actions are synchronized in $\hourglass$. For that purpose we consider:

$$(\mathsf{set}\ x\ \mathsf{in}\ \mathsf{when}\ x \mapsto a; p) \,||_a\, (\mathsf{set}\ y\ \mathsf{in}\ \mathsf{when}\ y \mapsto a; q)$$

Using the expansion law, this term can be rewritten into the equivalent term

$$\mathsf{set}\ x, y\ \mathsf{in}\ (\mathsf{when}\ x, y \mapsto a; (p \,||_a\, q))$$

This entails that action $a$ can happen after both clocks $x$ and $y$ have expired. In stochastic terms, this means that two random variables are competing, viz. the maximum of the random variables with distributions $F_x$ and $F_y$. As the maximum of two statistically independent random variables is distributed according to the product of their distribution (cf. Example 7), we obtain that synchronization of delayed actions in $\hourglass$ is based on the product of distributions. In most Markovian process algebras (with the notable exception of [49,51]), this policy is not taken, since the class of exponential distributions is not closed under product.

*Open probabilistic bisimulation.* Here, we present some sound axioms for open probabilistic bisimulation ($\sim_p^\circ$). The axioms presented in Table 5 are incomplete, but are useful for reducing the number of clocks in a $\hourglass$ expression. Axiom **P1** allows to eliminate redundant clock settings by stating that it is not necessary to set clocks that do not occur free in process $p$. Since these clocks do not freely occur in $p$ they are either not used or are set again once they will be used. Axiom **P2** states that clocks that have been used, i.e, that have expired, are not useful anymore (at least not before being set again), and thus can safely be removed. Axiom **P3** expresses that clocks that are set to a positive value with probability 0 cannot affect the timing of a process. Finally, the most involved axiom is **A3′**. This axiom is a weak variant of the idempotence axiom **A3** (i.e.,

| | | |
|---|---|---|
| **P1** | $\mathsf{set}\ C\ \mathsf{in}\ p = p$ | if $C \cap fv(p) = \varnothing$ |
| **P2** | $(\mathsf{when}\ C \mapsto a); (\mathsf{when}\ C \mapsto p) = \mathsf{when}\ C \mapsto a; p$ | |
| **A3′** | $\mathsf{set}\ x, y\ \mathsf{in}\ (\mathsf{when}\ x \mapsto p + \mathsf{when}\ y \mapsto p) = \mathsf{set}\ z\ \mathsf{in}\ (\mathsf{when}\ z \mapsto p)$ if $\{ x, y, z \} \cap fv(p) = \varnothing \wedge \forall t \in \mathbb{R}_{\geqslant 0}.\, F_z(t) = F_{\min\{x,y\}}(t)$ | |
| **P3** | $\mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto p) = \mathsf{set}\ x\ \mathsf{in}\ p$ | if $F_x(0) = 1$ |

**Table 5.** Some axioms for open probabilistic bisimulation on 

$a; p + a; p = a; p$) for structural bisimulation. Axiom **A3′** states that in case of two competing processes such as

$$(\mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto p)) + (\mathsf{set}\ y\ \mathsf{in}\ (\mathsf{when}\ y \mapsto p))$$

which, according to **C3**, is equivalent to

$$\mathsf{set}\ x, y\ \mathsf{in}\ (\mathsf{when}\ x \mapsto p + \mathsf{when}\ y \mapsto p)$$

the resolution of the choice is controlled by the minimum of two independent random variables with the distributions of $x$ and $y$. The process can thus be replaced by

$$\mathsf{set}\ z\ \mathsf{in}\ (\mathsf{when}\ z \mapsto p)$$

where $z$ is a fresh clock (i.e., $\{ x, y, z \} \cap fv(p) = \varnothing$) with distribution $min(F_x, F_y)$. Note that in case $x$ and $y$ are exponentially distributed, axiom **A3′** reduces to the idempotence law for Markovian process algebra [52]:

$$\lambda \mapsto p + \mu \mapsto p = (\lambda + \mu) \mapsto p$$

as stated in the notation used in the introduction.

*Example 13.* Using axiom **A3′** we can now deduce that indeed inequation (6) is valid in general:

$$\begin{aligned}
& \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto p) + \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto p) \\
= \quad & \{\ \alpha \text{ conversion}\ \} \\
& \mathsf{set}\ x\ \mathsf{in}\ (\mathsf{when}\ x \mapsto p) + \mathsf{set}\ y\ \mathsf{in}\ (\mathsf{when}\ y \mapsto p) \\
= \quad & \{\ \text{axiom } \mathbf{C3}\ \} \\
& \mathsf{set}\ x, y\ \mathsf{in}\ ((\mathsf{when}\ x \mapsto p) + (\mathsf{when}\ y \mapsto p)) \\
= \quad & \{\ \text{axiom } \mathbf{A3'}\ \} \\
& \mathsf{set}\ z\ \mathsf{in}\ (\mathsf{when}\ z \mapsto p)
\end{aligned}$$

where $F_z = F_x \cdot F_y = F_x^2$. Since $F_x^2 \neq F_x$ for any non-trivial random variable $x$, we obtain the inequality (6).

## 7 Analysis of Specifications

Until so far, we have introduced a process algebra for describing GSMCs (with possible non-determinism) while concentrating on notions like formal semantics, equivalences, axiomatisation and so on. In this section, we focus our attention on the *analysis* of specifications, both in a quantitative and qualitative sense. For the first type of analysis we consider discrete-event simulation, for the qualitative analysis we consider the checking of reachability properties.

### 7.1 Quantitative Analysis: Discrete-Event Simulation

By means of quantitative analysis, we obtain insight in questions related to the performance (in measures like throughput or response times) and dependability of systems (in terms of failure rates, mean time between failure, and so on). For a restricted set of stochastic automata – those that correspond to insensitive GSMCs [88] – numerical methods can be used to assess their steady-state (i.e., long run) behavior. Alternatively, in case of absence of non-determinism, approximation results can be employed, and arbitrary distributions can be approximated by phase-type distributions [79]. This results in a continuous-time Markov chain for which efficient numerical methods exist [95].

A general approach towards assessing quantitative properties is simulation, in particular *discrete-event simulation* [21,92]. In this simulation technique state changes take place at discrete points in time (but time is continuous), as opposed to continuous-time simulation techniques. In a simulation, runs (sample paths in the simulation jargon) are generated, and on the basis of these runs data is gathered and analyzed to determine an estimate of the desired measure of interest. The accuracy of the estimate is given by a confidence interval.

In the rest of this section, we focus on the following issues that occur when applying discrete-event simulation on specifications:

- how to generate sample paths from stochastic automata, and
- how to resolve non-determinism in a stochastic automaton prior to the simulation.

More details about simulation of specifications can be found in [27,33].

*Runs and adversaries.* A run of a labelled transition system is simply a walk through the state space by traversing transitions starting from the initial state. This also applies to PTSs except that we focus on traversals that are "probable".

**Definition 9.** *A* run $\rho$ *of PTS* $(N, P, \sigma_0, \mathcal{L}, T, \longrightarrow)$ *is a (finite or infinite) sequence* $\rho = \sigma_0\sigma'_0\ell_1 \ldots \ell_n\sigma_n\sigma'_n$ *for* $n \in \mathbb{N} \cup \{\infty\}$ *such that, for all* $0 \leqslant i < n$*:*

- $\sigma'_i$ *is in the support set of the probability measure of* $T(\sigma_i)$[5]
- $\sigma'_i \xrightarrow{\ell_{i+1}} \sigma_{i+1}$*, and*
- *if* $\rho$ *is finite, then* $\rho$ *ends in a non-deterministic state.*

[5] Intuitively, the support set of a probability measure is the smallest closed (measurable) set which has probability one.

Non-determinism is resolved by probabilistic adversaries: if during a run a state has been reached which has several non-deterministic possibilities, such adversary will make the choice in a (discrete) probabilistic way. An adversary is similar to a policy in Markov decision processes [34]. An *adversary* $\mathcal{A}$ on PTS $\mathcal{T}$ is a partial function that maps a finite run $\rho$ of PTS $\mathcal{T}$ to a discrete probability space on the non-deterministic transition relation. The sample space of $\mathcal{A}$ is a non-empty subset of the set of outgoing transitions from the final state of the run $\rho$. An adversary $\mathcal{A}$ of stochastic automaton $SA$ is a partial function on the runs of its closed semantics $\mathcal{C}[\![ SA ]\!]^{v_0}$.[6] Note that an adversary selects the next transition on the entire run $\rho$, i.e., on the entire history of the system. Usually, adversaries are considered that make a selection based on the current state only. For pragmatic reasons (space efficiency) we confine ourselves to such memoryless adversaries.

*Example 14.* To illustrate the notion of adversary, consider the -specification of the queueing system with two servers from Section 4 where we assume that all clocks in the system are governed by continuous distribution functions. The overall system specification was given by:

$$System_{G/G/2/\infty} = (Arrival \,||_{\varnothing}\, Server \,||_{\varnothing}\, Server) \,||_{\{a,b\}} Queue_0$$

This system contains non-determinism in case the queue contains one or more jobs and both servers are ready to process a job — which server will take the job out of the queue? In case we do not have any preference of one server over the other (e.g., they are both equally fast), an adversary that resolves this non-deterministic situation with equal probability is appropriate. Then $\mathcal{A}$ is defined such that $\mathcal{A}(\rho)$ equals $\frac{1}{2}$ for the first server taking the job out of the queue and $\frac{1}{2}$ for the second server taking the job out of the queue. In case we would prefer one server over the other, a different choice for $\mathcal{A}$ can be made.

*Discrete-event simulation.* Once we have resolved the present non-determinism (if any), the resulting system is fully probabilistic, that is to say, a stochastic automaton with an adversary that resolves all its non-determinism yields a GSMC with probabilistic branching. Hence, discrete-event simulation of such systems basically takes place according to the operational procedure as described in Section 3 for GSMCs with the minor difference that the trigger event $e^*$ may have several possible next states that are selected (by the adversary) in a probabilistic way. The user should bear in mind that the simulation results that will be obtained must be considered with respect to the specified adversary as different results are obtained (in general) for different adversaries!

An interesting aspect of simulation is that the state space is generated in an "on-the-fly" fashion – the state space is constructed dynamically and thus requires minimal storage as only the current state needs to be stored. This means that we are not forced to construct the entire stochastic automaton prior to the

[6] There are some conditions on the sample space that we omit here for the sake of simplicity; the interested reader may consult [27,33].

simulation. Instead, it suffices to store only the current state and generate new states on a call-by-need basis. As a consequence, simulation is not restricted to finite stochastic automata, but is also applicable to infinite-state stochastic automata. Besides, the generation of a state is very straightforward for a specification as a state uniquely corresponds to a process algebra expression.

*Example 15.* Consider the compositional -specification of the G/G/1-queueing system. Let the distribution $F$ of clock $x$ be such that $F(0) = 0$. The simulation will start by generating the initial term. Then clock $x$ is initialized and an arrival (action $a$) is generated once $x$ expires. The next term (i.e., location) is generated, and a new sample for $x$ is taken. Since $F_x(0) = 0$, this sample is positive. As in this location both an immediate action ($b$) and a non-immediate action ($a$) are possible, the latter will never be taken in a simulation since it has to be delayed first. Accordingly action $b$ happens and the next term is reached (after initializing clock $y$). This procedure continues ad infinitum. Note that the stochastic automaton as depicted in Fig. 14 is actually generated location by location, and that in each step only the current location (plus the valuation of all clocks involved) are needed to determine the next possible steps. Due to choices between immediate actions and non-immediate actions, in fact, only the sub-automaton of Fig. 17 is generated step-by-step. The reduction from Fig. 14 to Fig. 17 can also be carried out in an equational way while preserving probabilistic bisimulation [27].

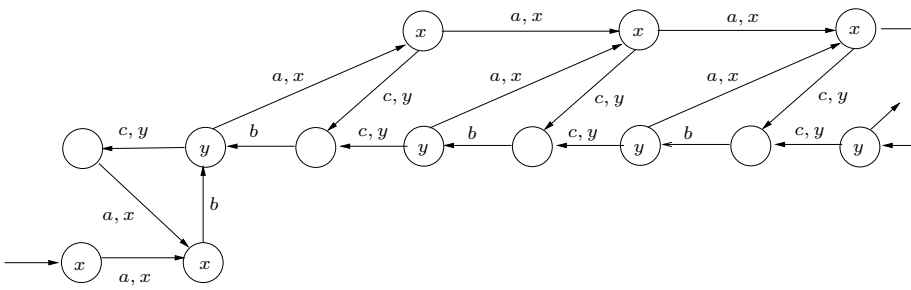



**Fig. 17.** Reduced stochastic automaton generated by simulation (if $F(0) = 0$)

### 7.2 Qualitative Analysis: Reachability Properties

Complementary to the quantitative analysis described above, we discuss in this section a classical analysis technique for functional correctness — reachability analysis. Reachability analysis is the key technique in proving safety properties (often characterized as properties of the type "something bad can never happen"). A typical reachability property is the absence of a deadlock, which is a state from which no further progress can be made. A naive strategy would

be to use the simulation approach just described and consider several simulation runs. As the coverage of such simulations is rather low – if no deadlock is reached during simulation, this does not guarantee absence of deadlocks – more systematic approaches are needed. In order to systematically check such properties for stochastic automata, and thus terms, the underlying semantics in terms of PTSs needs to be examined. However, even for finite terms, these transition systems are infinite due to the fact that distributions are continuous. We therefore consider a *symbolic* reachability analysis. Using a symbolic analysis we can avoid having to build and examine the infinite underlying PTS. Instead, we check reachability at the level of stochastic automata. We investigate this for finite stochastic automata.

*Reachability.* A (non-deterministic) state in a PTS is reachable if there exists a finite run that ends in that state. Let $\mathit{Reach}(\mathcal{T})$ denote the set of reachable states of PTS $\mathcal{T}$.

**Definition 10.** *Let* $SA = (\mathcal{S}, s_0, \mathcal{C}, \mathbf{A}, \rightarrow, \kappa, F)$ *be a stochastic automaton. A* symbolic run *of* $SA$ *is a finite sequence* $s_0 a_1 C_1 s_1 \ldots s_{n-1} a_n C_n s_n$, $n \geqslant 0$, *such that, for all* $0 < i \leqslant n$, $s_{i-1} \xrightarrow{a_i, C_i} s_i$.

Location $s$ is reachable if and only if there exists a symbolic run starting from $s_0$ that ends in $s$. The set of reachable locations of $SA$ is denoted $\mathit{Reach}(SA)$. The relationship between runs in a stochastic automaton, and the runs in the underlying (open and closed) PTSs is as follows. Every symbolic run of $SA$ has a corresponding finite run in $\mathcal{O}[\![ SA ]\!]$, and vice versa:

**Theorem 5.** $s \in \mathit{Reach}(SA) \Leftrightarrow \exists v \in \mathcal{V}.\, [s, v] \in \mathit{Reach}(\mathcal{O}[\![ SA ]\!]^{v_0})$

Recall that, as opposed to the open interpretation, in the closed interpretation an edge can only be taken, if there is no earlier point in time at which the current location can be left (maximal progress). As a consequence, certain edges present in the stochastic automaton need not result in a transition in the underlying closed PTS, since there exist competitive edges that are "faster" and thus will be taken instead. As a result, every finite run of $\mathcal{C}[\![ SA ]\!]$ has a corresponding symbolic run of $SA$,but not the reverse:

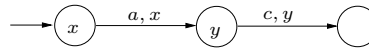
**Theorem 6.** $s \notin \mathit{Reach}(SA) \Rightarrow \forall v \in \mathcal{V}.\, [s, v] \notin \mathit{Reach}(\mathcal{C}[\![ SA ]\!]^{v_0})$.

This result is e.g., sufficient to check for freedom of deadlock: if $\mathcal{C}[\![ SA ]\!]$ does not have a reachable deadlock state, $SA$ is deadlock-free.

These results allow us to carry out systematic reachability analysis at a purely symbolic level, i.e., without the construction of the underlying infinite PTS and without using the clock information in the stochastic automaton. In this way, we can exploit existing tools like SPIN [59] for carrying out the reachability analysis.

*Timed reachability properties.* For checking reachability of locations, the stochastic information may not be of any importance. However, we like to obtain more information about timing aspects – e.g. "is a certain state reachable within $t$ time units?". In this case, the precise probabilistic value of the occurrence time

of some action is not relevant. Instead, it suffices to consider whether the action is possible or not at a particular time instant. Therefore, only a small bit of the information about the probability distributions involved is necessary. In fact, it suffices to know the "support set". In other words, we only need to consider within which boundaries a delay is possible. For instance, in a simple stochastic automaton like:




where $F_x$ and $F_y$ are uniform distributions on intervals $[10, 20]$ and $[0, 12]$, say, one can infer only from the bounds of the distributions that the right-most location is reachable within 32 time units. This idea is formalized in [28] by a compositional translation from stochastic automata into timed automata with deadlines [14]. This translation preserves safety properties in the sense that any state (i.e. a location and a clock valuation) that is not reachable in the generated timed automaton is also not reachable in the original stochastic automaton. Thus, timed safety properties are preserved.

An overview of the analysis techniques for specifications is given in Fig. 18.

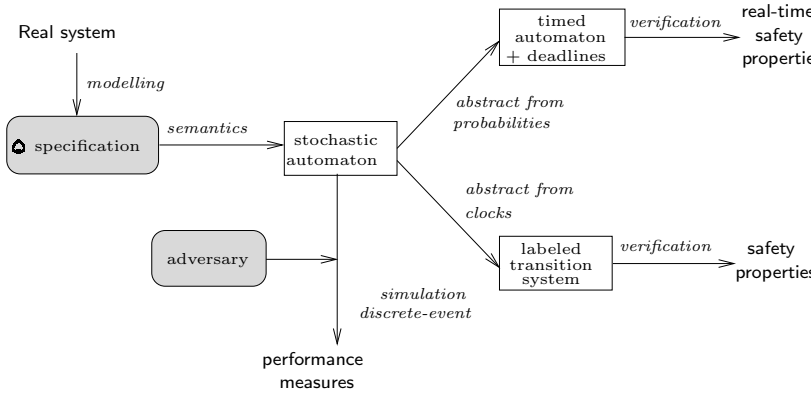



**Fig. 18.** Analysis of a specification

## 8 Case Study: The IEEE 1394 Root Contention Protocol

This section discusses a small case study where we applied to specify the root contention phase of the IEEE 1394 protocol in a compositional way, and used discrete-event simulation to analyze the time until contention resolution.

### 8.1 Informal Problem Description

*IEEE 1394.* The IEEE 1394 high performance serial bus has been developed for interconnecting computer and consumer equipment such as VCRs, CD players and multimedia PCs. It supports the addition and removal of equipment at any time ("hot-pluggable") and allows quick, reliable and inexpensive high-bandwidth transfer of digitized video and audio. The bus has been originally developed by Apple (FireWire) and has been standardized by IEEE in 1996 [62]. A revision of this standard is currently in progress [63]. As several companies have joined in the development of the 1394 bus, there is a good chance that this will become the future standard for connecting digital multimedia equipment; in fact, the bus is currently also being used in areas like aerospace equipment. Various parts of the standard have been specified and verified formally, e.g. [35,85,96,91]. Here, we consider the so-called root contention protocol.

*Leader election protocol.* (This description is adopted from [96].) The IEEE 1394 standard consists of a stack of protocols. The physical layer, the lowest in the protocol hierarchy, consists of a number of phases. During the tree identify phase, which is entered on occurrence of a bus reset, it is checked whether the network topology is a tree, and if so, a leader is elected among the nodes in the tree. The leader election takes place while constructing a spanning tree in the network and electing the root of the tree as leader. Informally, the basic idea of the protocol is as follows: leaf nodes send a "parent request" message to their neighbor. When a node has received a parent request from all but one of its neighbors it sends a parent request to its remaining neighbor. In this way the tree grows from the leafs to a root. If a node has received parent requests from all its neighbors, it knows that it has been elected as the root of the tree. It is possible that at the end of tree identify phase two nodes send parent request messages to each other; this situation is called *root contention*, cf. Fig. 19. To resolve this situation, a root contention protocol is run. After completion of this protocol, one of the two nodes has become root of the network.

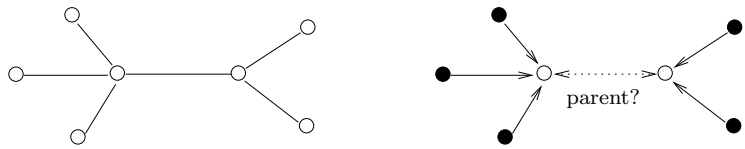



**Fig. 19.** Initial network topology (a) and root contention (b)

*Root contention protocol.* The protocol roughly works as follows. Suppose that the two contending nodes are node 0 and node 1. When node $i$ ($i = 0, 1$) has detected root contention it first flips a coin. If head comes up it waits a short period of time, somewhere in the interval $[\delta_{fast}, \Delta_{fast}]$; otherwise, it waits a long period of time somewhere in the interval $[\delta_{slow}, \Delta_{slow}]$. It is assumed that

$$0 \leqslant \delta_{fast} \leqslant \Delta_{fast} < \delta_{slow} \leqslant \Delta_{slow}$$

If after the waiting period no message has been received from the contender, the node sends a request message to its contender and declares itself to be a child, i.e., it assumes the contender to become a leader. Otherwise, it acknowledges the receipt of the message and declares itself to be the leader. If a node that has sent a request subsequently receives a request, then it concludes that there is a root contention again, and the protocol restarts.

The basic idea behind the protocol is that if the outcomes of the coin flips are different, the node with outcome tail (i.e., the slow one) will become root. And since with probability one the outcomes of the two coin flips will eventually be different, the root contention protocol will terminate (with probability one). This has been formally proven in [96] by manually verifying a model of the protocol in timed probabilistic I/O automata. An automatic verification of the root contention protocol has recently been reported [94], while parameterized verifications have been considered in [8,60].

### 8.2 Compositional Specification of the Root Contention Protocol

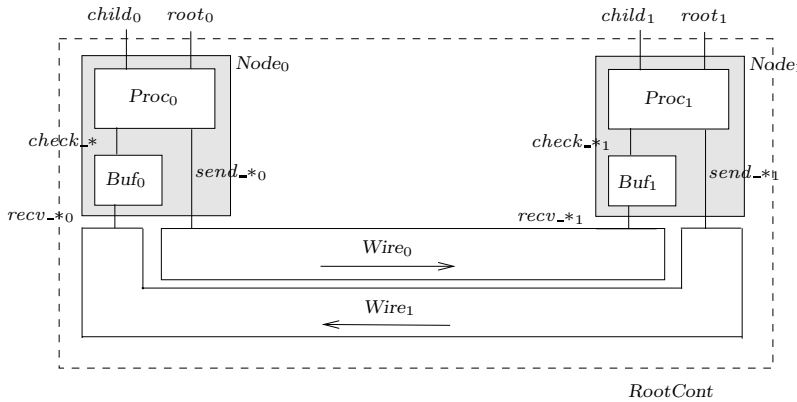



**Fig. 20.** Overview of the IEEE 1394 specification in

Fig. 21 presents the specification of the root contention protocol in . The model itself is based on [96]. A schematic overview of the processes involved and their synchronizations is given in Fig. 20. The $Node_i$ processes are connected to each other by two $Wire_i$ processes, that represent the communication lines between the components. Each $Node_i$ process has a $Buf_i$ process which can hold a single message from the other $Node_{(1-i)}$. New messages from $Node_{(1-i)}$ will simply overwrite the old message. Both nodes start (via $Proc_i$) to wait $F_{x_i}()$ units of time. If after waiting, the buffer is still empty (i.e. $check\_emp_i$), the node will send a $send\_req_i$ to its contender and will subsequently wait for an

acknowledgement. If this acknowledgement (i.e. $check\_ack_i$) arrives, $Node_i$ will declare itself a child using action $child_i$. On the other hand, if after waiting $F_{x_i}()$ units of time, $Node_i$ receives a $check\_req_i$ action, it declares itself to be the leader using action $root_i$. The delay of the communication line is modelled by clock $y_i$.

$$RootCont = (Node_0 \,||_{\varnothing}\, Node_1) \,||_A\, (Wire_0 \,||_{\varnothing}\, Wire_1)$$

with the following process definitions ($i = 1, 2$):

$$\begin{array}{lcl}
Node_i & = & (Proc_i \,||_B\, Buf_i) \\
Proc_i & = & \mathsf{set}\ x_i\ \mathsf{in}\ (\mathsf{when}\ x_i \mapsto\ (check_emp_i; SndReq_i + check_req_i; SndAck_i)) \\
SndReq_i & = & send_req_i; (check_req_i; Proc^i + check_ack_i; child_i; \mathbf{0}\ ) \\
SndAck_i & = & send_ack_i; root_i; \mathbf{0} \\
Buf_i & = & check_emp_i; Buf_i + recv_req^i; BufReq_i + recv_ack^i; BufAck_i \\
BufReq_i & = & check_req_i; Buf_i + recv_req^i; BufReq_i + recv_ack^i; BufAck_i \\
BufAck_i & = & check_ack_i; Buf_i + recv_req^i; BufReq_i + recv_ack^i; BufAck_i \\
Wire_i & = & send_req^i; WireReq_i + send_ack_i; WireAck_i \\
WireReq_i & = & \mathsf{set}\ y_i\ \mathsf{in}\ (\mathsf{when}\ y_i \mapsto\ recv_req_{(1-i)}; Wire_i) + Wire_i \\
WireAck_i & = & \mathsf{set}\ y_i\ \mathsf{in}\ (\mathsf{when}\ y_i \mapsto\ recv_ack_{(1-i)}; Wire_i) + Wire_i
\end{array}$$

and the following synchronization sets:

$$\begin{array}{lcl}
A & = & \{\, send_req_i, send_ack_i, recv_req_i, recv_ack_i \,\} \\
B & = & \{\, check_emp_i, check_req_i, check_ack_i \,\}
\end{array}$$

**Fig. 21.** ⌂ specification of the root contention protocol

*The clock distributions.* The delay after detection of a root contention in $Node_i$ is governed by clock $x_i$ ($i = 0, 1$) whose distribution is given by:

$$F_{x_i}(t) = \begin{cases}
0 & \text{if } t < \delta_{fast} \\
\frac{1}{2} \cdot \left( \frac{t - \delta_{fast}}{\Delta_{fast} - \delta_{fast}} \right) & \text{if } \delta_{fast} \leqslant t < \Delta_{fast} \\
\frac{1}{2} & \text{if } \Delta_{fast} \leqslant t < \delta_{slow} \\
\frac{1}{2} \cdot \left( 1 + \frac{t - \delta_{slow}}{\Delta_{slow} - \delta_{slow}} \right) & \text{if } \delta_{slow} \leqslant t < \Delta_{slow} \\
1 & \text{if } \Delta_{slow} \leqslant t
\end{cases}$$

This distribution corresponds to first choosing either a short delay in the interval $[\delta_{fast}, \Delta_{fast}]$ with probability $\frac{1}{2}$, or choosing a long delay in the interval $[\delta_{slow}, \Delta_{slow}]$ with probability $\frac{1}{2}$. In fact, $F_{x_i}$ is the combination of uniform distributions over the two respective intervals $[\delta_{fast}, \Delta_{fast}]$ and $[\delta_{slow}, \Delta_{slow}]$. The IEEE 1394 standard [62] only specifies the lower- and upper-bounds of these

intervals, but states nothing about mean, variances, or the like. Approximating these non-deterministic intervals in the above way is the most inderminate approximation conform to the principle of maximizing the entropy, see Section 2.2. A pictorial representation of $F_{x_i}$ is given in Fig. 22.

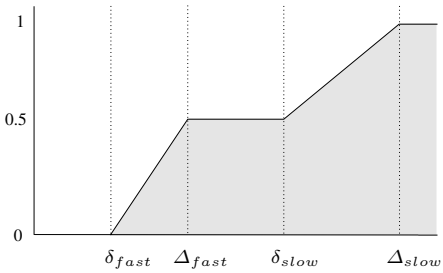



**Fig. 22.** Distribution of delay after detection of root contention

The transmission delay of $Wire_i$ is determined by clock $y_i$. This delay depends on the length of the cable. We assume that the transmission delay is between $v/2$ and a maximal velocity $v$ with an average of $\frac{4}{5}v$. The transmission delay is approximated by a beta-distribution with parameters $\beta_1 = 2$ and $\beta_2 = 6$. To be more precise, for cable-length $m$, the probability density function of clock $y_i$ $i = 0, 1)$, is given by:

$$f_{y_i}(t) \;=\; \begin{cases} \dfrac{\left(\frac{v}{m}t - 1\right)\left(2 - \frac{v}{m}t\right)^5}{\beta(2,6)} & \text{if } \frac{m}{v} \leqslant t < 2\frac{m}{v} \\ 0 & \text{if } t < \frac{m}{v} \;\text{ or }\; 2\frac{m}{v} \leqslant t \end{cases}$$

where $\beta$ is a function that is well-known from statistics and probability theory, see e.g., [93].

### 8.3 The Time until Contention Resolution

*Simulation parameters.* The verification study in [96] revealed that the correctness of the root contention protocol – a leader is eventually elected with probability one – is only guaranteed if the maximum transmission delay is smaller than $\delta_{fast}$ and $(\delta_{slow} - \Delta_{fast})/2$. In our case, the maximum transmission delay is assumed to be the lowest upper-bound of the support set of $F_{y_i}$, i.e., $2m/v$. This corresponds to the distance the message may travel divided by the slowest speed. To guarantee the correctness of the protocol we thus assume:

$$\frac{2\,m}{v} \;<\; \min\left\{\delta_{fast}, \frac{\delta_{slow} - \Delta_{fast}}{2}\right\}$$

For $v = 198$ $mtr/\mu sec$, the maximum velocity according to the IEEE 1394 standard, we obtain $m < \min\{99 \cdot \delta_{fast}, 49.5 \cdot (\delta_{slow} - \Delta_{fast})\}$. For the discrete-event

simulation we have taken the values of $\delta_{fast}$, $\Delta_{fast}$, $\delta_{slow}$, and $\Delta_{slow}$ from the specifications in the IEEE 1394 standard [62] and the draft IEEE 1394a proposal [63], cf. Table 6.

| | $\delta_{fast}$ | $\Delta_{fast}$ | $\delta_{slow}$ | $\Delta_{slow}$ | $m$ |
|---|---|---|---|---|---|
| IEEE 1394 | 0.24 $\mu sec$ | 0.26 $\mu sec$ | 0.57 $\mu sec$ | 0.60 $\mu sec$ | $< 15.345$ $mtr$ |
| IEEE 1394a | 0.76 $\mu sec$ | 0.80 $\mu sec$ | 1.60 $\mu sec$ | 1.64 $\mu sec$ | $< 39.6$ $mtr$ |

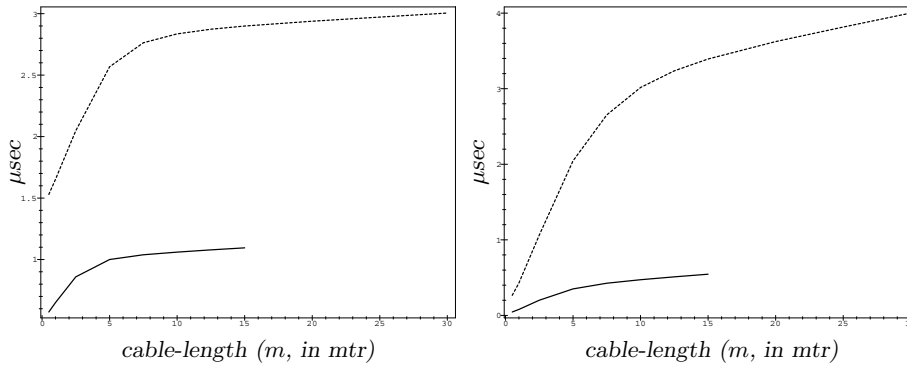
**Table 6.** Parameters of the root contention protocol

*Simulation results.* The closed behavior of the compositional specification is deterministic (with probability 1). Thus, no adversary is needed to resolve any non-determinism. The discrete-event simulation has been carried out by a prototype implementation in the functional programming language Haskell; for more details see [33]. For each setting, five series of 250,000 simulations have been carried out, each starting with different seeds for the random number generator. For IEEE 1394, the length $m$ of the cable ranged to up to 15 meters; for IEEE 1394a, lengths of up to $m$=30 meters were considered. The average and variance of the time until contention resolution are reported in Fig. 23 where the lower curves refer to IEEE 1394 while the upper curves refer to IEEE 1394a. As expected, the time increases on increasing cable length $m$. It appears that the new (draft) version of the protocol shows a lower performance. For a given cable length, the average time until contention resolution for IEEE 1394a is about a factor two higher than for standardized IEEE 1394. This phenomenon is due to the longer delay after flipping a coin. It should be remarked, though, that the IEEE 1394a protocol guarantees the correctness of the root contention protocol over longer distances.

## 9 Related Work

In this section we give an overview of stochastic process algebras that incorporate general distributions and compare these proposals to ♤.

- TIPP [43] is the earliest approach addressing general distributions in a process algebra. Its syntax has the integrated prefix $a_F; p$ which in ♤ corresponds to set $x_F$ in when $x_F \mapsto a; p$. Its semantics is based on labelled transition systems in which transitions are decorated with the associated distribution function and, to keep track of the execution of parallel processes, a number that indicates how many times an action has not been chosen to execute. This number introduces infinite semantic objects, even for simple regular processes.




(a) Average (b) Variance

**Fig. 23.** Time until contention resolution in IEEE 1394 (lower curve) and IEEE 1394a (upper curve)

- [1] extends LOTOS with several kinds of (stochastic) timers. The semantics is given in terms of a (variety of) timed transition system, therefore abstracting from probabilities. Separately, the authors consider the stochastic model as a kind of GSMP. This work proposed very interesting ingredients at language level. However, the treatment is not algebraic – despite the use of structured operational semantics. The authors impose strong syntactic restrictions, do not provide an adequate equivalence relation and, as a consequence, neither algebraic laws.
- [45,46,97] introduced a process algebra for discrete event simulation. The concerns of randomly setting a timer, expiration of such a timer, and actual activity are split in a rather similar way to ours. The semantic model is similar to our probabilistic transition systems where non-deterministic transitions are instead split into discrete transitions and timed transitions. As a consequence, semantic objects contain usually uncountably many states and transitions. Their process algebra includes an urgent and a delayable prefixing, so its interpretation combines both views of closed and open system.
- [18] studies a semantics for a process algebra similar to TIPP in terms of a stochastic extension of event structures. This model seems to be more natural to deal with general distributions since activities that are not causally dependent (i.e. concurrent activity) are not related in the model, contrarily of what occurs in interleaving based models. However, recursive processes always have associated an infinite semantic object. Recent investigations indicate, however, that finite objects can be obtained (for finite-state process algebra terms) from which stochastic task graph models are generated that can be analyzed numerically [87].
- [83] followed an approach similar to [43]. The author gives semantics to a stochastic extension of the $\pi$-calculus. In this case transitions are decorated with locality information to keep track which process performed it.

- A general semi-Markovian process algebra based on EMPA is discussed in [16]. Terms in this process algebra have semantics in a semi-interleaving model that allows for refinement of actions (the so-called ST-semantics). Although this calculus preserves finiteness of semantic objects in a reasonable way (like ours), the manner in which semantics is given is not straightforward. A nice characteristic of this process algebra is that it represents the GSMP model in a complete way.
- Recently, [17] adapted the previous work to consider —like in our case— the separation between beginning of a delay, ending of it, and execution of an action. In so doing, the authors obtained a neater semantics as well as a complete axiomatization for the weak equivalence relation they consider.
- Another recent work includes [74]. In this paper the authors pursue ideas similar to [43,83]. To keep track of the time that has passed, the transition is labelled with the action, its respective delay distribution, and the time it takes. Like [1], they also use an "age" function in order to update the distribution of the remaining delay of the concurrent events. The fact of labelling the transition with the occurrence time makes the semantic object infinite.

Among the stochastic process algebras enumerated above, [45,46,97] is the closest to  . Like , [45,46,97], [49], and [17] allow non-determinism. In all the other cases (including the Markovian process algebras), choice is always solved either by the race policy, by the pre-selection policy, or by a combination of both.

Other works that relate discrete-event simulation models (or languages) to process algebra are [82,10]. In these works, the approach is different: rather than generating a simulation model automatically from a process algebra specification (as in this paper), they use process algebra as a semantic model for simulation languages. In addition, these works do not take probabilistic timing into consideration.

## 10 Epilogue

In this tutorial, we have given an informal overview of incorporating general distributions in a process algebraic framework. We discussed its complications and gave an overview of possible solutions that have been suggested so far in the literature. We recapitulated the model of generalized semi-Markov chains (GSMCs) and justified the need for a process algebraic treatment of such models. In the second part of the paper, we have presented stochastic automata, a model that subsumes GSMCs, includes non-determinism and is amenable to composition. A process algebra named  has been presented to describe these stochastic automata in a modular way. As a result, stochastic automata (and thus GSMCs) can be generated automatically from  specifications. Analysis methods for  specifications have been treated. The proposed approach has been illustrated by specifying the root contention phase within the standardized IEEE 1394 serial bus protocol. As opposed to other studies of this protocol that focus on its func-

tional correctness we have studied the delay until root contention resolution. Below we conclude by listing some interesting research topics for future work.

*Weak bisimulation.* Thanks to the efforts of several research groups, most semantic issues of incorporating general distributions into a process algebraic setting have been satisfactorily solved. One of the most important (semantic) issues for future work is the investigation of weak equivalence relations such as variants of weak or branching bisimulation. A weak equivalence relation that allows to abstract from immediate invisible actions has been defined recently [17]. To our knowledge, the replacement of a sequence of generally distributed delays by a single delay labelled with the convolution of the delays on the sequence (i.e., the sum of the random variables involved), has not yet been captured in a congruence relation.

*Model checking.* As an alternative to discrete-event simulation techniques, model checking could be applied to stochastic automata. Using such techniques one would be able to check properties of the form "with probability at most 0.99 the system will deadlock within 2 hours" in a fully automated way. Such quantitative model checking algorithms have been recently considered for models that are closely related to stochastic automata: for probabilistic variants of the duration calculus on continuous semi-Markov processes [61] and for a continuous probabilistic variant of timed automata [69]. It would be interesting to see how these techniques can be effectively applied to stochastic automata and  specifications. Besides, the combination of these techniques with discrete-event simulation could be investigated.

*User-oriented specification languages.* Performance engineers do consider process algebra as being (too) complicated. In order to bridge the gap towards practitioners that e.g. use stochastic automata for modeling real-time multi-media systems [11,20], efforts should be made towards making the specification formalism more user-friendly. Interesting developments in that respect are the usage of stochastic automata for stochastic extensions of UML (Universal Modeling Language) Statecharts [42] and the activities on MoDeST (a Model Description language for Stochastic Timed Systems) that builds upon  , Promela [59], and timed automata and for which tool-support is currently under development.

## Acknowledgements

Ed Brinksma, Holger Hermanns, Ulrich Herzog, Ric Klaren, Rom Langerak, Diego Latella, Mieke Massink and Theo Ruys are all kindly acknowledged for their support and their joint experiences with us over the last years on the research on general distributions in process algebra. We thank Boudewijn Haverkort for providing us with information concerning entropies. The second author is supported by the Dutch Technology Foundation (STW) on the PROGRESS project HaaST (Verification of Hard and Softly Timed Systems).

## References


1. M. Ajmone Marsan, A. Bianco, L. Ciminiera, R. Sisto and A. Valenzano. A LOTOS extension for the performance analysis of distributed systems. *IEEE/ACM Trans. on Networking*, **2**(2), 151–164, 1994.
2. R. Alur and D.L. Dill. A theory of timed automata. *Th. Comp. Sc.*, **126**: 183–235, 1994.
3. S. Asmussen, O. Nermand and M. Olsson. Fitting phase-type distributions via the EM algorithm. *Scand. J. Statist.*, **23**: 420–441, 1996.
4. J. Baeten (ed). *Applications of Process Algebra*. Cambridge Tracts in Theoretical Computer Science, Cambridge Univ. Press, 1990.
5. J. Baeten and J. Bergstra. Real time process algebra. *Form. Asp. of Comp.*, **3**(2):142–188, 1991.
6. J. Baeten, J. Bergstra and S.A. Smolka. Axiomatizing probabilistic processes: ACP with generative probabilities. *Inf. & Comp.*, **121**: 234–255, 1995.
7. J. Baeten and C. Verhoef. A congruence theorem for structured operational semantics. In E. Best, ed, *Concurrency Theory*, LNCS 715: 477–492, Springer, 1993.
8. G. Bandini, R. Lutje Spelberg, and W.J. Toetenel. Parametric verification of the IEEE 1394a root contention protocol using LPMC, 2000 (submitted).
9. M. Bernardo and R. Gorrieri. A tutorial on EMPA: a theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Th. Comp. Sc.*, **202**: 1–54, 1998.
10. G. Birtwistle and C. Tofts. A denotational semantics for a process-based simulation language. *ACM Trans. on Modeling and Computer Simulation*, **8**(3): 281–305, 1998.
11. L. Blair, T. Jones and G. Blair. Stochastically enhanced timed automata. In S.F. Smith and C.L. Talcott, eds, *Proc. IFIP Conf. on Formal Methods for Open Object-based Distributed Systems IV*, pp. 327–350, Chapman & Hall, 2000.
12. H. Bohnenkamp and B.R. Haverkort. Stochastic event structures for the decomposition of stochastic process algebra models. In J. Hillston and M. Silva, eds, *Proc. 7th Workshop on Process Algebras and Performance Modelling*. Prensas Universitarias de Zaragoza, 1999.
13. T. Bolognesi, F. Lucidi and S. Trigila. Converging towards a timed LOTOS standard. *Comp. Standards & Interfaces*, **16**: 87–118, 1994.
14. S. Bornot, J. Sifakis and S. Tripakis. Modeling urgency in timed systems. In: W.-P. de Roever, H. Langmaack and A. Pnueli, eds, *Compositionality: The Significant Difference*, LNCS 1536: 103–129. Springer, 1998.
15. A. Bouajjani, S. Tripakis and S. Yovine. On-the-fly symbolic model-checking for real-time systems. In *Proc. IEEE Real-Time Systems Symp.*, pp. 25–34, IEEE CS Press, 1997.
16. M. Bravetti, M. Bernardo and R. Gorrieri. Towards performance evaluation with general distributions in process algebras. In D. Sangiorgi and R. de Simone, eds, *Concurrency Theory*, LNCS 1466: 405–422, Springer, 1998.
17. M. Bravetti and R. Gorrieri. The theory of interactive generalized semi-Markov processes. *Th. Comp. Sc.*, **286**, 2001 (to appear).
18. E. Brinksma, J.-P. Katoen, R. Langerak and D. Latella. A stochastic causality-based process algebra. *The Comp. J.*, **38**(7): 552–565, 1995.
19. E. Brinksma and H. Hermanns. Process algebra and Markov chains. This volume.

20. J. Bryans, L. Blair, H. Bowman, and J. Derrick. Specification and analysis of automata-based designs. In W. Grieskamp, T. Santen and B. Stoddart, eds, *Integrated Formal Methods*, LNCS 1945: 176–193, 2000.
21. C.G. Cassandras. *Discrete Event Systems – Modeling and Performance Analysis.* Irwin Inc. and Aksin Associates Inc., 1993.
22. G. Clark. Stochastic process algebra structure for insensitivity. In J. Hillston and M. Silva, eds, *Proc. 7th Workshop on Process Algebras and Performance Modelling*, pp. 63–82. Prensas Universitarias de Zaragoza, 1999.
23. G. Clark. *Techniques for the Construction and Analysis of Algebraic Performance Models.* PhD thesis, University of Edinburgh, 2000.
24. R.W. Cleaveland and S.A. Smolka. Strategic directions in concurrency research. *Computing Surveys*, **28**(4): 607–625, 1996.
25. D.R. Cox. A use of complex probabilities in the theory of stochastic processes. *Proc. Cambridge Philosophy Society*, **51**: 313–319, 1955.
26. M.E. Crovella. Performance evaluation with heavy tailed distributions (extended abstract). In B. Haverkort, H. Bohnenkamp and C. Smith, eds, *Computer Performance Evaluation*, LNCS 1786: 1–9, Springer, 2000.
27. P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems.* PhD thesis, University of Twente, 1999.
28. P.R. D'Argenio. A compositional translation of stochastic automata into timed automata. CTIT Technical Report 00-08, 32 pp., 2000.
29. P.R. D'Argenio and E. Brinksma. A calculus for timed automata (extended abstract). In B. Jonsson and J. Parrow, eds, *Formal Techniques in Real-Time and Fault Tolerant Systems*, LNCS 1135, pp. 110-129, Springer, 1996.
30. P.R. D'Argenio, J.-P. Katoen and E. Brinksma. A stochastic automata model and its algebraic approach. In E. Brinksma and A. Nymeyer, eds, *Proc. 5th Int. Workshop on Process Algebra and Performance Modelling*, CTIT Technical Report 97-14, pp. 1–16, 1997.
31. P.R. D'Argenio, J.-P. Katoen and E. Brinksma. An algebraic approach to the specification of stochastic systems (extended abstract). In D. Gries and W.-P. de Roever, eds, *Proc. IFIP Conf. on Programming Concepts and Methods.* Chapman & Hall, pp. 126–147, 1998.
32. P.R. D'Argenio, J.-P. Katoen and E. Brinksma. A compositional approach to generalised semi-Markov processes. In A. Guia, M. Spathopoulos and R. Smedinga, eds, *Proc. 4th Int. Workshop on Discrete-Event Systems*, pp. 391–397. IEE Press, 1998.
33. P.R. D'Argenio, J.-P. Katoen and E. Brinksma. Specification and analysis of soft real-time systems: Quantity and quality. In *Proc. IEEE Real-Time Systems Symp.*, pp. 104–114, IEEE CS Press, 1999.
34. C. Derman. *Finite-State Markovian Decision Processes.* Academic Press, 1970.
35. M. Devillers, D. Griffioen, J. Romijn and F. Vaandrager. Verification of a leader election protocol: formal methods applied to IEEE 1394. *Form. Methods in Sys. Design*, **16**(3): 307–320, 2000.
36. J. Engelfriet. Branching processes of Petri nets. *Acta Inf.*, **28**: 575–591, 1991.
37. D. Ferrari. Considerations on the insularity of performance evaluation. *IEEE Trans. on Soft. Eng.*, **12**(6): 678–683, 1986.
38. A. Feyzi Ates, M. Bilgic, S. Saito, and B. Sarikaya. Using timed CSP for specification, verification and simulation of multimedia synchronization. *IEEE J. on Sel. Areas in Comm.*, **14**:126–137, 1996.
39. W. Fokkink. *Introduction to Process Algebra.* Texts in Theoretical Computer Science, Springer, 2000.

40. R.J. van Glabbeek and F.W. Vaandrager. Petri net models for algebraic theories of concurrency. In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, eds, *PARLE — Parallel Architectures and Languages Europe*, LNCS 259: 224–242. Springer, 1987.
41. P.W. Glynn. A GSMP formalism for discrete event simulation. *Proc. of the IEEE*, **77**(1):14–23, 1989.
42. S. Gnesi, D. Latella and M. Massink. A stochastic extension of a behavioural subset of UML statechart diagrams. In: Proc. High Assurance System Engineering, IEEE CS Press, 2000.
43. N. Götz, U. Herzog and M. Rettelbach. Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras. In L. Donatiello and R. Nelson, eds, *Performance Evaluation of Computer and Communication Systems*, LNCS 729: 121–146. Springer, 1993.
44. H. Hansson and B. Jonsson. A calculus for communicating systems with time and probabilities. In *Proc. 11th IEEE Real-Time Systems Symposium*, pp. 278–287, 1990.
45. P.G. Harrison and B. Strulo. Stochastic process algebra for discrete event simulation. In F. Baccelli, A. Jean-Marie and I. Mitrani, eds, *Quantitative Methods in Parallel Systems*, pp. 18–37. Springer, 1995.
46. P.G. Harrison and B. Strulo. SPADES: a stochastic process algebra for discrete event simulation. *J. of Logic and Computation*, **10**(1): 3–42, 2000.
47. C. Harvey. Performance engineering as an integral part of system design. *Br. Telecom Technol. J.*, **4**(3): 142–147, 1986.
48. W. Henderson and D. Lucic. Aggregation and disaggregetion through insensitivity in stochastic Petri nets. *Perf. Ev.*, **17**: 91–114, 1993.
49. H. Hermanns. *Interactive Markov Chains.* PhD. thesis, U. Erlangen-Nürnberg, 1998.
50. H. Hermanns, U. Herzog and J.-P. Katoen. Process algebra for performance eveluation. *Th. Comp. Sci.*, 2001 (to appear).
51. H. Hermanns and J.-P. Katoen. Automated compositional Markov chain generation for a plain-old telephone system. *Sci. of Comp. Programming*, **36**(1): 97–127, 2000.
52. H. Hermanns and M. Rettelbach. Towards a superset of LOTOS for performance prediction. In [84], pp. 77–94.
53. U. Herzog. A concept for graph-based stochastic process algebras, generally distributed activity times, and hierarchical modelling. In [84], pp. 1–20.
54. D.P. Heyman and M.J. Sobel. *Stochastic Models in Operations Research, volume 1 - Stochastic Processes and Operating Characteristics.* McGraw-Hill, 1982.
55. J. Hillston. On the nature of synchronisation. In U. Herzog and M. Rettelbach, eds, *Proc. 2nd Int. Workshop on Process Algebra and Performance Modelling*, Arbeitsbericht **27**(4): 51–71, University of Erlangen, 1994.
56. J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.
57. J. Hillston and M. Ribaudo. Stochastic process algebras: a new approach to performance modeling. In K. Bagchi and G. Zobrist, eds, *Modeling and Simulation of Advanced Computer Systems.* Gordon Breach, 1998.
58. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.
59. G. Holzmann. The model checker SPIN. *IEEE Trans. on Softw. Eng.*, **23**(5), 279–295, 1997.

60. T. Hune, J. Romijn, M.I. Stoelinga and F. Vaandrager. Linear parametric model checking of timed automata. In: T. Margaria and W. Yi, eds, *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 2031: 189–204. Springer, 2001.
61. D. van Hung and Z. Chaochen. Probabilistic duration calculus for continuous time. *Formal Aspects of Computing*, **11**: 21–44, 1999.
62. IEEE Computer Society. IEEE Standard for a High Performance Serial Bus. Std 1394-1995, 1996.
63. IEEE Computer Society. P1394a Draft Standard for a High Performance Serial Bus (Supplement). Draft 2.0, 1998.
64. ISO IS 8807. LOTOS – A formal description technique based on the temporal ordering of observational behaviour. 1989.
65. B. Jonsson, W. Yi and K.G. Larsen. Probabilistic extensions of process algebras. In J. Bergstra, A. Ponse and S.A. Smolka, eds, *Handbook of Process Algebra*, North-Holland, pp. 685–710, 2001.
66. K. Kanani. *A Unified Framework for Systematic Quantitative and Qualitative Analysis of Communicating Systems.* PhD thesis, Imperial College, Univ. of London, 1998.
67. J.-P. Katoen, E. Brinksma, D. Latella and R. Langerak. Stochastic simulation of event structures. In [84], pp. 21–40.
68. J. Kemeny and J. Snell. *Finite Markov Chains.* Van Nostrand, 1960.
69. M.Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Verifying Quantitative properties of continuous probabilistic timed automata. In C. Palamadessi, ed, *Concurrency Theory*, LNCS 1877: 123–137. Springer, 2000.
70. S. Lang. *Real and Functional Analysis*, volume 142 of *Graduate Texts in Mathematics.* Springer, 1993.
71. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. and Comp.*, **94**(1): 1–28, 1992.
72. K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Int. J. on Softw. Tools for Technol. Transf.*, **1**(1/2): 134–152, 1997.
73. L. Léonard and G. Leduc. An introduction to ET-LOTOS for the description of time-sensitive systems. *Comp. Netw. & ISDN Sys.* **29**(3): 271–292, 1997.
74. N. Lopez and M. Núnez. NMSPA: A non-Markovian model for stochastic processes. In *Proc. 1st Int. Workshop on Distributed System Validation and Verification*, IEEE CS Press, 2000.
75. G. Lowe. Probabilistic and prioritized models of timed CSP. *Th. Comp. Sci.*, **138**: 315–352, 1995.
76. C. Miguel, A. Fernández and L. Vidaller. LOTOS extended with probabilistic behaviours. *Form. Asp. of Comp.*, **5**: 253–281, 1993.
77. R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.
78. F. Moller and C. Tofts. A temporal calculus of communicating systems. In J. Baeten and J-W. Klop, eds, *Concur'90: Theories of Concurrency – Unification and Extension*, LNCS 458: 401–415. Springer, 1990.
79. M.F. Neuts. *Matrix-geometric Solutions in Stochastic Models – An Algorithmic Approach.* The Johns Hopkins University Press, 1981.
80. X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In J.W. de Bakker et al, eds, *Real-Time: Theory in Practice*, LNCS 600: 526–548. Springer, 1992.
81. M. Nielsen, G.D. Plotkin and G. Winskel. Petri nets, event structures and domains. *Th. Comp. Sci.*, **13**(1):85–108, 1981.

82. R.J. Pooley. Integrating behavioural and simulation modelling. In *Quantitative Evaluation of Computing and Communication Systems*, LNCS 977: 102–116. Springer, 1995.
83. C. Priami. Stochastic $\pi$-calculus with general distributions. In [84], pp. 41–57.
84. M. Ribaudo, ed. *Proc. 4th Int. Workshop on Process Algebra and Performance Modelling*. C.L.U.T. Press, 1996.
85. J. Romijn. *Analysing Industrial Protocols with Formal Methods.* PhD thesis, Univ. of Twente, 1999.
86. S.M. Ross. *Stochastic Processes*. John Wiley & Sons, New York, 1983.
87. T. Ruys, R. Langerak, J.-P. Katoen, D. Latella and M. Massink. First passage time analysis of stochastic process algebra using partial orders. In: T. Margaria and W. Yi, eds, *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 2031: 220–235. Springer, 2001.
88. R. Schassberger. Insensitivity of steady-state distributions of GSMPs. *Ann. Probability*, **5**: 87–89, 1977.
89. S. Schneider. An operational semantics for timed CSP. *Inf. & Comp.*, **116**:193–213, 1995.
90. R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic J. of Computing*, **2**(2):250–273, 1995.
91. C. Shankland and M.B. van der Zwaag. The tree identify protocol of IEEE 1394 in $\mu$CRL. *Form. Asp. of Comp.*, **10**: 509–531, 1998.
92. G.S. Shedler. *Regenerative Stochastic Simulation*. Academic Press, Inc., 1993.
93. A.N. Shiryaev. *Probability*. Graduate Texts in Mathematics, Springer-Verlag, 1989.
94. D. Simons and M. Stoelinga. Mechanical verification of the IEEE 1394a root contention protocol using Uppaal2k. Tech. Rep. CSI-R0009, University of Nijmegen, 2000.
95. W. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton Univ. Press, 1994.
96. M.I. Stoelinga and F.W. Vaandrager. Root contention in IEEE 1394. In J.-P. Katoen, ed, *Formal Methods for Real-Time and Probabilistic Systems*, LNCS 1601: 53–74. Springer, 1999.
97. B. Strulo. *A Process Algebra for Discrete-Event Simulation*. PhD. thesis, Imperial College, U. London, 1993.
98. M.Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proc.* $26^{th}$ *IEEE Symp. on Foundations of Comp. Sc.*, pp. 327–338. IEEE CS Press, 1985.
99. W. Yi. Real-time behaviour of asynchronous agents. In J. Baeten and J-W. Klop, eds, *Concur'90: Theories of Concurrency – Unification and Extension*, LNCS 458: 502–520. Springer, 1990.
100. W. Whitt. Continuity of generalized semi-Markov processes. *Math. Oper. Res.*, **5**: 494–501, 1980.