A COMPOSITIONAL APPROACH TO GENERALISED SEMI-MARKOV PROCESSES

Pedro R. D'Argenio[a], Joost-Pieter Katoen[b], Ed Brinksma[a]

[a] Dept. of Computer Science, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
[b] Informatik VII, University of Erlangen-Nürnberg, Martensstrasse 3, D-91058 Erlangen, Germany

Abstract

This paper proposes a compositional approach to the specification and analysis of stochastic discrete-event systems. We present a process algebra that allows one to specify generalised semi-Markov processes (GSMPs) in a compositional way. The semantics of this process algebra is given in terms of stochastic automata, an extension of automata with clocks that are basically random variables of continuous or discrete nature. We show that GSMPs are a proper subset of stochastic automata and provide some example laws that are useful for the verification of such models.

Keywords: *GSMP, process algebra, discrete-event simulation, compositionality, equational reasoning*

1 Introduction

In discrete-event simulation the behaviour of a system as it evolves in time is described in terms of a simulation model. Simulation modelling is usually a time-consuming task and is mainly based on human ingenuity and experience. In addition, the difficulty of the design of a simulation model rapidly grows with the increasing magnitude and complexity of the system itself. To ease the description of simulation models, dedicated simulation languages have been developed, such as Demos, GPSS and Simscript, most of which are process-oriented.

State changes in discrete-event simulation models take place at discrete points in time, whereas the time of occurrence of activities is controlled stochastically, i.e. by means of random variables. A mathematical framework for the study of these models—known as stochastic discrete-event systems—is given by Glynn [8]. He presents a *generalised semi-Markov process* (GSMP) theory for such systems.

In this paper we propose a high-level specification language for stochastic discrete-event systems, in particular GSMPs. Our specification language is based on *process algebra*. In process algebras, like ACP, CCS, CSP and LOTOS, a concurrent system is syntactically represented using powerful composition operators which facilitate the development of well-structured specifications. The algebraic nature of the language allows to reason about specifications in an equational way, thus allowing transformation and verification. Traditionally, process algebras focussed on the specification and analysis of qualitative system properties, but in the last decade the interest in extensions with quantitative information has grown significantly. This integration facilitates the analysis of qualitative and quantitative properties in a single framework.

We use an extension of process algebra in which the time of occurrence of actions, the most primitive notions of activity in process algebra, is determined by a continuous or discrete probability distribution of arbitrary nature. To give a formal semantics to our language SPADES (*Stochastic Process Algebra for Discrete-Event Simulation*, symbolised by ♤) we introduce the concept of *stochastic automata*, an extension of labelled automata with clocks, that can be seen as a stochastic variant of timed automata by Alur & Dill [1]. We argue that GSMPs are a subclass of stochastic automata and give a stochastic extension of the expansion law known from (untimed) process algebra. This law is of central importance for the verification and correctness-preserving transformation of expressions in ♤ and, thus of GSMPs.

2 Motivation

A simple queueing system. 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.

© 1998 IEE, WODES98 – Cagliari, Italy
Proc. of the Fourth Workshop on Discrete Event Systems

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 ∞ denotes the buffer capacity.

A GSMP description. A typical GSMP description of such queueing system is defined in the following way. The basic ingredients are states and events. To each state z a (non-empty) set of active events $E(z)$ is associated denoting the set of events that can cause transitions out of z. In our example, we let the state space be $\mathbb{N} \times \{0, 1\}$, where the first component of a state indicates the number of jobs that are currently in the system, and the second component indicates the system status (1 = 'a job just arrived', and 0 = 'a job has just been processed'). Initial state is $(0, 1)$. In each state, possible events are the arrival of a job (denoted a) and the completion of a job (denoted c). In the initial state no job completion is possible. Thus, $E(i, j) = \{ a_{i+1}, c_{i+1} \}$ and $E(0, 1) = \{ a_1 \}$. The arrival of a job causes a transition from state $(i, 0)$ or from $(i, 1)$ to state $(i+1, 1)$. Completion of a job leads to a transition from $(i+1, 0)$ or $(i+1, 1)$ to state $(i, 0)$.

To each event $e \in E(z)$ a clock c_e is associated that indicates the amount of time until expiration. Clocks are initialised by probability distribution functions and run backwards. In each state the active event e^* with minimal clock value is selected for execution. The values of the events in z', the successor state of z' on executing e^*, are determined as follows. Clocks of events in $E(z) - \{ e^* \}$ are decreased by the value of c_{e^*}. The clock of any newly active event e in $E(z') - (E(z) - \{ e^* \})$ is set according to the clock-setting distribution $F(c_e)$. All other clocks in z' equal ∞. Due to the conditions imposed on clock-setting distributions, the event e^* in GSMPs is always uniquely determined [8].

In our example, clocks are initialised as follows. On entering state $(i, 1)$ the clock of the next arrival a_{i+1} is initialised according to distribution F, the job inter-arrival time. On entering state $(i, 0)$ with $i \neq 0$ the clock of the next possible job completion c_{i+1} is initialised according to distribution H, the service delay. The clock for c_1 is set in the same way in state $(1, 1)$.

A compositional approach. Although using this description the dynamics of our example GSMP can be determined, it is in absence of any further explanation not easy to understand. This is basically due to the fact that the individual system components, like arrival and server processes, are hard to recognise from the overall system structure. This problem becomes more apparent if we consider GSMPs modelling systems of more realistic magnitude. We say that the specification lacks *compositionality*. The idea that we shall pursue here is to specify GSMPs in a compositional way.

In *process algebra* the specification of our queueing system can be obtained in a hierarchical manner, starting from the specifications of the individual components. If we let $a;p$ denote a process that immediately can perform an action a and then behaves like process p, and $p + q$ denote the process that behaves either like p or like q, then a buffer of infinite capacity can be specified by the set of processes:

$$\begin{aligned} \mathit{Queue}_0 &= a; \mathit{Queue}_1 \\ \mathit{Queue}_{i+1} &= a; \mathit{Queue}_{i+2} + b; \mathit{Queue}_i \text{ for } i \geqslant 0 \end{aligned}$$

where the index indicates the number of jobs in the buffer. Similarly to GSMPs, clocks can be used to model probabilistic delays. Let C be a finite set of clocks. Using the primitives $C \mapsto p$, the process that after expiration of all clocks in C behaves like p, and $\{\!|C|\!\}p$, the process that behaves like p after any clock x in C is initialised according to some indicated distribution, we obtain for the arrival and server processes:

$$\begin{aligned} \mathit{Arrival} &= \{\!|x_F|\!\}\{x_F\} \mapsto a; \mathit{Arrival} \\ \mathit{Server} &= b; \{\!|y_H|\!\}\{y_H\} \mapsto c; \mathit{Server} \end{aligned}$$

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

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

Here, $||$ stands for parallel composition. In process $p \,||_A\, q$, where A is a set of actions, p and q perform actions autonomously, but actions in A should be performed by both. The resulting specification of the $G/G/1/\infty$ system closely resembles the structure of the system itself, is easy to understand, and readily modifiable (for instance, to a queue with finite capacity, or a system in which the service rate depends on the number of waiting jobs).

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) specifications. If, in addition, this relation is a *congruence*, then such transformation can be carried out component-wise. An equivalence relation is a congruence if for any term a sub-term may be replaced by an equivalent sub-term and an equivalent term results.

This compositional transformation reduces the complexity significantly. Finally, due to the algebraic nature of the formalism it is possible to define equational rules on the syntax that allow to perform transformation and simplification at a purely syntactical level, without any reasoning in semantical terms.

3 Stochastic automata and GSMPs

Stochastic automata. The semantics of our process algebra is defined in terms of stochastic automata. (For a formal interpretation of such automata in terms of measure theory, see [6].) This model is strongly related to GSMPs and incorporates, apart from the necessary ingredients to model GSMPs, the possibility of specifying non-determinism. Non-determinism appears if two transitions become enabled simultaneously. This concept is usually absent in stochastic discrete-event systems, but has been widely accepted in the computer science community for the purpose of under-specification in a step-wise design methodology [13]. For simulation purposes, the non-determinism can be resolved using so-called adversaries that schedule the different branches according to some discrete probability distribution [17]. In this way, a mechanism is obtained similar to the probabilistic branching in GSMPs (see also later on).

Definition 1. A *stochastic automaton* is a tuple $(\mathcal{S}, s_0, \mathcal{C}, \mathbf{A}, \longrightarrow, \kappa, F)$ where: $\mathcal{S}$ is a non-empty set of *locations* with $s_0 \in \mathcal{S}$ being the *initial location*, $\mathcal{C}$ is a set of *clocks*, $\mathbf{A}$ is a set of *actions*, $\longrightarrow \subseteq \mathcal{S} \times (\mathbf{A} \times \wp_{\mathrm{fin}}(\mathcal{C})) \times \mathcal{S}$ is the set of *edges*, $\kappa : \mathcal{S} \to \wp_{\mathrm{fin}}(\mathcal{C})$ is the *clock-setting function*, and $F : \mathcal{C} \to (\mathbb{R} \to [0,1])$ is the *clock-distribution function* such that $F(x)(t) = 0$ for $t < 0$.

We denote $(s, a, C, s') \in \longrightarrow$ by $s \xrightarrow{a,C} s'$, use x and y to denote clocks, and abbreviate $F(x)$ by F_x. To each location s a finite set of clocks $\kappa(s)$ is associated. As soon as location s is entered any clock x in this set is initialised according to its probability distribution function F_x. Once initialised, the clocks start counting down, all with the same rate. 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 can happen as soon as all clocks in the set C have expired. The next location will then be s'.

Example 2. The stochastic automaton that corresponds to the $G/G/1/\infty$ queue from Section 2 is depicted in Figure 1. 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). Notice that after an a-action always a location is reached in which clock x is set (according to distribution F), and after a c-action always clock y is set (apart from the first location in the upper row) according to H. Clock x thus controls the job inter-arrival time while y controls the service delay. The locations in the upper row represent the states $(i, 0)$ whereas the lower row represents the states $(i, 1)$. In state $(0, 0)$ there are no jobs in the system and a completion can only happen after a job arrives first. Therefore, in this state clock y is not set, but only after a job arrival (in state $(1, 1)$). □

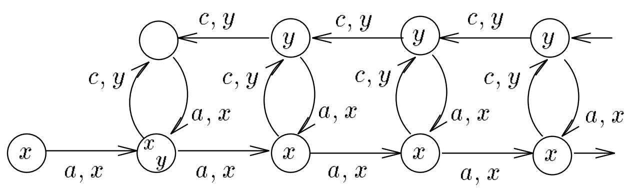

Figure 1: Stoch. automaton of a $G/G/1/\infty$-system

Generalised semi-Markov processes. In Section 2 we have seen a flavour of GSMPs. Actually we will consider a (large) subclass of GSMPs. The main restriction is that the next state is uniquely determined by the present state and the triggered event. In general GSMPs the next state is chosen probabilistically from a set of possible next states. In addition, we assign to each clock a fixed distribution function whereas in general GSMPs such distribution may depend on the history of the system. This is not a severe restriction since one can model each history of a general GSMP by a sequence of sufficiently many events such that each such event marks a relevant point in history. The class of GSMPs with history-independent distribution functions is known as *time-homogeneous* [8]. Finally, sometimes clocks are allowed to have different rates whereas in our case all clocks proceed with the same speed. Different rates are not very usual in simulation, and moreover, under certain conditions, such "multi-rated" GSMPs can be represented by GSMPs where all clock rates equal 1.

The notion of GSMPs that we consider is defined as follows, where it is assumed that initial state z_0 has a single active event e_0. $\mathcal{C}$ is a set of clocks. The dynamics of a GSMP in the following sense is as described in Section 2.

Definition 3. $(Z, z_0, \mathbf{E}, e_0, E, C, N, F)$ is a GSMP with Z, a non-empty set of *states* with $z_0 \in Z$, $\mathbf{E}$, a non-empty set of *events* with $e_0 \in \mathbf{E}$, $E : Z \to \wp_{\mathrm{fin}}(\mathbf{E})$, the *event-assignment* function with $E(z) \neq \varnothing$ for all $z \in Z$ and $E(z_0) = \{ e_0 \}$, $C : \mathbf{E} \to \mathcal{C}$, the *clock-assignment* function, $N : Z \times \mathbf{E} \to Z$, the *next-state* function, and $F : \mathcal{C} \to (\mathbb{R} \to [0,1])$, the *distribution assignment* function, such that $F(x)(0) = 0$.

GSMPs versus stochastic automata. The relation between stochastic automata and GSMPs (in the above sense) is shown by defining a mapping from GSMPs onto stochastic automata. The existence of this mapping indicates that GSMPs are properly included in stochastic automata. We have proven the correctness of the mapping in the sense that the underlying probabilistic transition systems (that are based on Borel spaces) of a GSMP and its associated stochastic automaton are probabilistically bisimilar [6].

The basic idea of the mapping is to introduce a location as a pair (z, E) where z is a state of the GSMP and E is the set of events that are already active. The initial location is $(N(z_0, e_0), \varnothing)$. 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

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

Definition 4. The associated stochastic automaton of GSMP $\mathcal{G} = (Z, z_0, \mathbf{E}, e_0, E, C, N, F)$ is defined by $\mathcal{S} = Z \times \wp_{\mathrm{fin}}(\mathbf{E})$ with $s_0 = (N(z_0, e_0), \varnothing)$, $\mathbf{A} = \mathbf{E}$, $\mathcal{C} = \{ C(e) \mid e \in \mathbf{E} \}$, $\kappa(z, E) = \{ C(e) \mid e \in E(z) - E \}$, and F is the same as for $\mathcal{G}$. $\longrightarrow$ is defined by the rule

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

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 $(N(z, e), E(z) - \{e\})$ for every (reachable) $z \in Z$ and $e \in E(z)$. Remark that for $z' = N(z, e)$ we have $\kappa(z', E(z) - \{e\}) = \{ C(e') \mid e' \in E(z') - (E(z) - \{e\}) \}$, the set of clocks for all newly active events in z'.

As argued before, stochastic automata are more expressive than GSMPs, since stochastic automata do allow non-determinism (two outgoing edges that are enabled at the same time), whereas GSMPs do not. In addition, in the stochastic automaton model clocks may be initialised by arbitrary distributions—including discrete distribution functions—without any restriction. In GSMPs it is required that in any set of active events there is at most one clock x such that $F_x(t)$ is not continuous as a function of t [8].

4 The stochastic process algebra $\spadesuit$

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 arbitrary probability distribution function. We abbreviate (x, G) by x_G.

Definition 5. The syntax of $\spadesuit$ is defined by:

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

where $C \subseteq \mathcal{C}$ 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 $p \in \spadesuit$.

Besides the operations used in Section 2 the language incorporates the basic process $\mathbf{0}$, the process that cannot perform any action, and the *renaming* operation $p[f]$, a process that behaves like p except that actions are renamed by function f. 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.

Semantics. To associate a stochastic automaton $SA(p)$ to a given term p in the language, we define the different components of $SA(p)$[1]. In order to define the automaton associated to a parallel composition, we introduce the additional operation $\overline{\mathsf{ck}}$. $\overline{\mathsf{ck}}(p)$ is a process that behaves like p except that no clock is set at the very beginning. As usual in structured operational semantics, a location corresponds to a term. Thus, the set of locations equals $\spadesuit \cup \{\overline{\mathsf{ck}}\}$. The clock setting function κ is defined by induction on the structure of expression: $\kappa(\mathbf{0}) = \kappa(a;p) = \kappa(\overline{\mathsf{ck}}(p)) = \varnothing$, $\kappa(C \mapsto p) = \kappa(p[f]) = \kappa(p)$, $\kappa(p{+}q) = \kappa(p\,||_A\,q) = \kappa(p) \cup \kappa(q)$, $\kappa(\{\!|C|\!\}p) = C \cup \kappa(p)$ and $\kappa(X) = \kappa(p)$ for $X = p$. The set of edges $\longrightarrow$ between locations is defined as the

[1]Here we assume that p does not contain any name clashes of clock variables. This not a severe restriction since terms that suffer from such name clash can always be properly renamed into a term without such name clash [6].

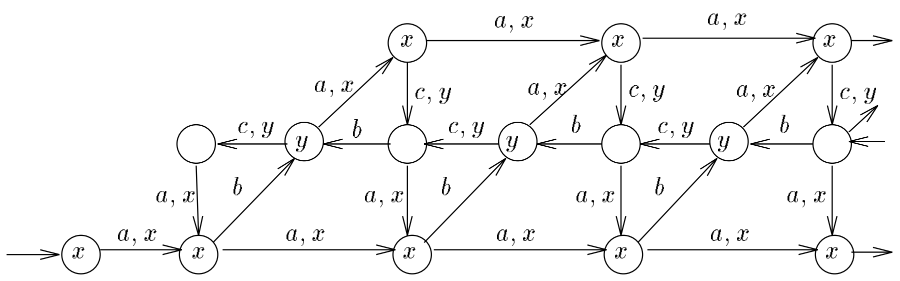

Figure 2: Stochastic automaton of the compositional $G/G/1/\infty$ specification

Table 1: Stochastic automata for ♤ $(X = p \ \in \ E)$

$$a;p \xrightarrow{a,\varnothing} p \qquad \frac{p \xrightarrow{a,C} p'}{\begin{array}{c} p+q \xrightarrow{a,C} p' \\ q+p \xrightarrow{a,C} p' \end{array}}$$

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

$$\frac{p \xrightarrow{a,C'} p'}{C \mapsto p \xrightarrow{a,C\cup C'} p'} \qquad \frac{p \xrightarrow{a,C} p'}{p[f] \xrightarrow{f(a),C} p'[f]}$$

$$\frac{p \xrightarrow{a,C} p'}{X \xrightarrow{a,C} p'} \qquad \frac{p \xrightarrow{a,C} p'}{\overline{\mathsf{ck}}(p) \xrightarrow{a,C} p'}$$

$$\frac{p \xrightarrow{a,C} p'}{\begin{array}{c} p \,||_A\, q \xrightarrow{a,C} p' \,||_A\, \overline{\mathsf{ck}}(q) \\ q \,||_A\, p \xrightarrow{a,C} \overline{\mathsf{ck}}(q) \,||_A\, p' \end{array}} \ (a \notin A)$$

$$\frac{p \xrightarrow{a,C} p' \quad q \xrightarrow{a,C'} q'}{p \,||_A\, q \xrightarrow{a,C\cup C'} p' \,||_A\, q'} \ (a \in A)$$

smallest relation satisfying the rules in Table 1. 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 ♤.

Example 6. Using this recipe it can be shown that the semantics of the *System* specification of Section 2 boils down to the (at first sight somewhat complicated) stochastic automaton depicted in Figure 2. Here, empty sets are omitted; in particular b stands for $b, \varnothing$. Although the state space of this automaton is somewhat larger than that of the direct representation in Figure 1, this does not have a serious impact on the efficiency of stochastic simulation. 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 allows on-the-fly simulation, that is, simulation while constructing the state space. In this approach the unreachable locations will not be visited. For instance, for locations in which both an immediate (i.e. an action equipped with no clocks) and a non-immediate action are enabled, the non-immediate transition will never be traversed. The corresponding reduction of the state space can also be obtained by syntactical transformation as shown in Example 9. □

It turns out that stochastic automata and the language ♤ are equally expressive [6]. This means that for any (finitely branching) stochastic automaton a corresponding (guarded recursive) term in the language can be given whose reachable part of its stochastic automaton is identical to the stochastic automaton at hand, up to renaming of clocks. A recursive specification E is *guarded* if $X = p \in E$ implies that all variables in p appear in a context of a prefix. A stochastic automaton is finitely branching if for every location the set of outgoing edges is finite.

Structural bisimulation. For process algebras many equivalences and pre-orders have been defined to compare specifications. One of the most interesting equivalence relations is bisimulation [15]. The following notion of bisimulation decides the equivalence of stochastic automata on the basis of their structure. Weaker notions of bisimulation are defined in [6].

Definition 7. Let $(\mathcal{S}, s_0, \mathcal{C}, \mathbf{A}, \longrightarrow, \kappa, F)$ be a stochastic automaton. $R \subseteq \mathcal{S} \times \mathcal{S}$ is a *structural bisimulation* if, R is symmetric and whenever $s_1 R s_2$, for all $a \in \mathbf{A}$, $C \subseteq \mathcal{C}$, we have:

1. $s_1 \xrightarrow{a,C} s_1'$ 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 $s_1 \mathrel{\underset{\sim}{\leftrightarrows}} s_2$ and call s_1 and s_2 structurally bisimilar.

Two stochastic automata SA_1 and SA_2 are *structurally bisimilar*, notation $SA_1 \leftrightarrows SA_2$, if their respective initial locations are structurally bisimilar on the disjoint union of SA_1 and SA_2. If we omit the clock-related information, we obtain the usual (strong) bisimulation relation on transition systems [15]. Terms p and q are structurally bisimilar if and only if $SA(p) \leftrightarrows SA(q)$. The relation $\leftrightarrows$ is a congruence for ♤ [6]. This means that for any term in our language a sub-term p may be replaced by its bisimilar equivalent q such that a bisimilar term results.

Equational reasoning. Rather than proving $p \leftrightarrows q$ using their semantical interpretation it is often more convenient to use rules defined on the syntax of p and q that are known to preserve (in our case) $\leftrightarrows$. This enables the transformation and comparison of terms at a purely syntactical level. Some typical axioms are

$$\begin{aligned}(p+q)+r &= p+(q+r)\\ C \mapsto (C' \mapsto p) &= C \cup C' \mapsto p\\ C \mapsto \{\!|C'|\!\}p &= \{\!|C'|\!\}C \mapsto p \text{ if } C \cap C' = \varnothing.\end{aligned}$$

In [6] a complete and sound axiomatisation of structural bisimulation for ♤ is presented for finite terms. Using these axioms any term p can be converted into a canonical form which has the shape $\{\!|C|\!\}(\sum C_i \mapsto a_i; p_i)$ where p_i are terms in canonical form and $\sum$ is the usual generalisation of choice: $\sum_{0<i\leqslant n} p_i$ equals $p_1 + \ldots + p_n$ for $n{>}0$, and $\mathbf{0}$ for $n{=}0$.

An essential law in traditional process algebras is the expansion law. This law allows one to reduce parallel composition in terms of prefix and choice, and has proven to be of crucial importance for verification purposes. A stochastic equivalent of this law can be derived for our language. In fact, the expansion law is inherent in our model and follows from the way in which parallel composition is defined. It can be derived using the axioms in [6].

Theorem 8. (Expansion Law) Let $p, q \in$ ♤ such that $p = \{\!|C|\!\}p'$ and $q = \{\!|C'|\!\}q'$ with $p' = \sum C_i \mapsto a_i; p_i$ and $q' = \sum C'_j \mapsto b_j; q_j$. Suppose $p \,||_A\, q$ does not contain name clashes. Then $p \,||_A\, q$ equals

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

Example 9. Using structural and probabilistic bisimulation [6] we are able to formally relate Figure 2 to Figure 1 in the following way. (The following transformations could also be carried out using equational laws, but this is omitted due to space reasons.) If in a location both an immediate action and a non-immediate action are possible, then the latter will never be taken since it has to be delayed first. This allows one to remove the locations in the lower row of Figure 2, except for the two leftmost locations. The thus obtained automaton is depicted in Figure 3. The only

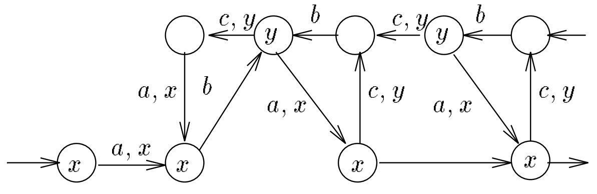

Figure 3: Reduced automaton

difference with Figure 1 are the b-transitions that are used for the sole purpose of synchronising the *Queue* and *Server* processes. If, as a last step, we would copy the leftmost location that contains clock y and subsequently aggregate locations appropriately such that the b-transitions only occur inside aggregates, then we obtain the automaton of Figure 1. This latter transformation can be formalised using the common notion of abstraction in process algebra and weak bisimulation equivalence [15], a notion of equivalence that allows one to abstract from internal moves. This approach is applied to Markovian queues in [10], and is for our setting an interesting subject for further work. □

5 Related work

Other stochastic process algebras. Since 1990 extensions of process algebras have been investigated in which the time of actions is determined by (continuous) distribution functions. In languages like TIPP [11], PEPA [12] and EMPA [2] the time of actions is controlled by exponential distributions. The elementary operator in these languages is $a_\lambda; p$ with rate λ. This corresponds to $\{\!|x_F|\!\}\{x_F\} \mapsto a; p$ with $F(t) = 1 - e^{-\lambda t}$. Due to the memoryless property of exponential distributions the semantics of these languages can adequately be described using labelled transition systems that closely resemble continuous-time Markov chains. Various useful links between process algebras and Markov chains have thus been established, e.g. between bisimulation and lumping [12].

In fact, our presented approach can be considered as a generalisation of this line of research using GSMPs rather than Markov chains. To our knowledge the use of a process algebra to specify GSMPs is novel. [14] used GSMPs indirectly: they map an extended process algebra onto event structures, and obtain for a subclass of event structures a GSMP. For recursive processes infinite event structures are obtained which makes this approach less suited for the use of efficient regenerative simulation techniques. The finite representations we obtain do not suffer from this problem.

Process algebra and discrete-event simulation. Harrison & Strulo [9] developed a stochastic process algebra to formally describe discrete-event simulation. Typically their semantic objects are highly infinite. Although their work is somehow related to ours, stochastic automata appear to be more intuitive and resemble more closely the conceptual ideas of simulation languages. In particular, measure theory only plays a role in our case when defining the formal interpretation of stochastic automata.

Pooley [16] investigates the mapping of a high-level language for describing discrete-event simulation models, baptised extended activity diagrams, onto the timed process algebra TCCS and the simulation language Demos [3]. Using this framework Pooley is able to check certain properties of a model a priori to simulation, by analysing the (T)CCS specification. In this work distribution functions are neglected and the use of process algebra is quite different from ours.

Birtwistle and Tofts use process algebras, basically CCS and its synchronous variant SCCS, to provide a denotational semantics of Demos [4]. They focus on analysing properties like absence of deadlock and livelock and do not consider any timing aspects.

6 Concluding remarks

In this paper we presented a novel process algebra suitable for specifying GSMPs in a compositional way. The concept of stochastic automata has been introduced and is shown to properly contain a large class of GSMPs. Since our process algebra ♤ and stochastic automata are equally expressive, this class of GSMPs is also a subset of ♤. Using equational laws—like the presented expansion law that allows to reduce parallel composition—GSMP specifications can be simplified and compared syntactically.

We have currently implemented a prototypical tool that allows us to simulate specifications written in ♤. The simulation algorithm takes a ♤ specification and an additional process to resolve possible non-determinism in this process as input, and automatically generates simulation runs. We applied our prototype to model and analyse a multi-processor system that is vulnerable to failures [7].

Acknowledgement. The first author is supported by the NWO/SION project 612-33-006.

References

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

[2] M. Bernardo and R. Gorrieri. Extended Markovian process algebra. In *Proceedings CONCUR'96*, LNCS 1119, pp 314–330. Springer-Verlag, 1996.

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

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

[5] P.R. D'Argenio and E. Brinksma. A calculus for timed automata (Extended abstract). In *Proceedings FTRTFT'96*, LNCS 1135, pp 110–129. Springer-Verlag, 1996.

[6] P.R. D'Argenio, J.-P. Katoen, and E. Brinksma. An algebraic approach to the specification of stochastic systems (Extended abstract). In *Proceedings PROCOMET'98*. Chapman & Hall, 1998.

[7] P.R. D'Argenio, J.-P. Katoen, and E. Brinksma. General purpose discrete-event simulation using ♤. Submitted for publication. 1998.

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

[9] P.G. Harrison and B. Strulo. Stochastic process algebra for discrete event simulation. In *Quantitative Methods in Parallel Systems*, pp 18–37. Springer, 1995.

[10] H. Hermanns, M. Rettelbach and T. Weiss. Formal characterisation of immediate actions in SPA with non-deterministic branching. *The Computer Journal*, 38(7):530–542, 1995.

[11] H. Hermanns, U. Herzog and V. Mertsiotakis. Stochastic process algebras – Between LOTOS and Markov chains. *Computer Networks & ISDN Systems*, 1998.

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

[13] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[14] J.-P. Katoen, E. Brinksma, D. Latella, and R. Langerak. Stochastic simulation of event structures. In *Proceedings PAPM'96*, pp 21–40. CLUT Press, 1996.

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

[16] R.J. Pooley. Integrating behavioural and simulation modelling. In *Quantitative Evaluation of Computing and Communication Systems*, LNCS 977, pp 102–116. Springer-Verlag, 1995.

[17] M.Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proceedings* 26th *FOCS*, pp 327–338. IEEE Comp. Soc. Press, 1985.