Stochastic Process Algebra Structure for Insensitivity

Graham Clark

Laboratory for Foundations of Computer Science,
The University of Edinburgh, Kings Buildings, Edinburgh EH9 3JZ.
Email: gcla@dcs.ed.ac.uk

Abstract. Recent research has investigated ways in which general probability distributions may be incorporated into stochastic process algebra. These proposals allow the arbitrary use of such distributions, improving expressibility, but in general this makes performance evaluation difficult. Typically, simulation techniques must be employed. We attack the goal of general probability distributions from the opposite direction, using the stochastic phenomenon of *insensitivity*. In this paper we describe a construction which guarantees the insensitivity of certain concurrently enabled non-conflicting stochastic process algebra *activities*. We give a derived combinator for constructing process algebra models. Use of this combinator guarantees that the stochastic process underlying the model is insensitive to a particular set of activities. Therefore, the user need not assume these activities are exponentially distributed, yet may still use familiar Markovian techniques to solve the model. We find that the model structure we identify has a product form solution which does not match any of those currently known to exist for stochastic process algebra. We highlight our technique with an example drawn from the field of transaction processing systems. Our analysis uses the stochastic process algebra PEPA, and its associated conventions.

1 Introduction

Stochastic process algebras (SPA) are modelling methodologies useful for the performance analysis of systems with concurrent behaviour. An SPA model is typically built compositionally from smaller components using a small set of combinators. The fundamental ability of an SPA component is to perform one or more enabled *activities*, which causes the model components to effect *transitions*. Since an SPA model may enable several activities, it exhibits the potential for concurrent behaviour. Furthermore, activities may 'compete' with others, whereby the completion of one activity cancels the others. An SPA model may be reduced to a continuous time Markov chain, from which performance measures may be calculated.

The exponential assumption inherent in Markovian analysis is regarded by some as a restriction in the application of SPA modelling. For example, deterministic random variables may be more appropriate in modelling time-outs in communication protocols. It would therefore be of great utility to be able to incorporate more general distributions into SPA performance models. Indeed several attempts to do this are underway, as discussed later in Section 3.2. However unlike previously published work, our aim is to introduce this increased modelling expressiveness only when it does not seriously impinge on model tractability. We do this by making use of the concept of *insensitivity.* A stochastic process is said to be insensitive if its steady-state distribution depends on the distribution of one or more of its state lifetime random variables *only through the mean.* In this paper, we will consider insensitivity with respect to the activities of an SPA model. The consequence of an SPA model being insensitive to a particular activity is that the activity may provably be arbitrarily distributed (with the same mean) without affecting the steady-state state probability distribution of the model. Therefore conditions under which activities can be shown to be insensitive are conditions under which more flexible modelling features, such as realistic time-outs and repair times, may be introduced.

In this paper, we present a simple class of PEPA models, which may be constructed using a new derived combinator. The class consists of a collection of simple subcomponents which interact in a 'weak' fashion. Despite this interaction, we show that insensitivity of residence time in particular sub-states (states of each subcomponent) is retained, and thus in our approach, insensitivity of particular SPA activities too. Therefore, in many cases, generally distributed activities may be used to build the subcomponents without affecting the equilibrium distribution of the model as a whole. Models built using our combinator are guaranteed to have our insensitivity properties. Furthermore, we show that the class of PEPA models we describe gives rise to a product form solution, and that the structure does not match the criteria for any of the currently known SPA product form classes.

The rest of this paper is organised as follows: in the next section, we present a short overview of PEPA, and some associated theory and conventions. In Section 3 we discuss insensitivity, presenting the basic results, and discuss some process algebras with support for general distributions. Section 4 describes the extent to which general distributions may be introduced, and the structure of PEPA models which permit this. The class of models is demonstrated with an example drawn from the field of transaction processing systems in Section 5. Section 6 then looks at existing product form solutions for process algebra, and compares them to our approach. Finally, Section 6.1 concludes.

2 PEPA

Classical process algebras disregard the notion of measuring time, and model functional behaviour only. PEPA extends these algebras by associating an exponentially distributed random variable, which represents a duration, with every action. There is a close relationship between a PEPA model and a continuous time Markov chain (CTMC), and it is from this underlying stochastic process that performance measures can be calculated.

PEPA models are constructed compositionally from *components*, which are able to interact with each other. Each of these components is capable of performing *activities*. Formally, an activity a is described as a pair (α, r), where α is the *action type* i.e. the *type* of the activity, and $r \in \mathbb{R}^+$ is the activity rate. The set of all action types is denoted $\mathcal{A}$, and $\mathbb{R}^+$ is defined as the set of positive real numbers together with the symbol $\top$. This symbol denotes an unspecified activity rate. Such an activity may only take place in synchrony with another of the same type, whose rate is specified. When r is specified, its value is the parameter of an exponential distribution and thus governs the duration of the activity. The set of all activities is $\mathcal{A} \times \mathbb{R}^+$, denoted by $\mathcal{A}ct$.

2.1 PEPA Syntax

PEPA models are built using only a small set of combinators. The way in which a model may behave is entirely characterised by the modeller's use of the combinators in building structure. Each PEPA combinator is described briefly below:

Prefix: if P is a process, then $(\alpha, r).P$ is a process that performs (α, r) (an activity of type α, which has a duration exponentially distributed with mean $1/r$) and then evolves into P.

Constant: if Q is a process and $P \stackrel{\text{def}}{=} Q$, then P is a process that behaves in exactly the same fashion as Q.

Summation/Choice: if P and Q are processes, then $P + Q$ is a process that expresses the conflicting competition of P and Q. The current activities of both P and Q are enabled; a *race condition* determines the first to complete and distinguishes the component into which the process evolves.

Cooperation: if P and Q are processes, and L is a set of action types, then $P \underset{L}{\bowtie} Q$ is a process that expresses the parallel and synchronising execution of both P and Q. Both components proceed independently on activities whose types are not in the cooperation set, L. However, those activities whose types are contained in L require the participation of both P and Q. If one of the components may not perform an activity of some type α in its current state, the other component becomes *blocked* on all α activities. If both are capable of performing an α activity, it may occur with a rate given as the minimum of the rates at which each would have had the capacity to perform an α activity in isolation.

If the cooperation set L is empty, we denote the parallel composition of P and Q as $P \parallel Q$. More generally, the parallel composition of a set of n processes is given as $\prod_{i=1}^{n} P_i$.

Hiding: if P is a process, and L is a set of action types, then P/L is the process that can behave exactly as P, except that if P would perform an activity of type $\alpha \in L$, then P/L would perform a *silent* activity denoted by the type τ. Activities whose types are in L are said to be *hidden*, and cooperation is not possible on τ activities.

2.2 Terminology

In this section, we introduce our terminology and some useful definitions. Suppose a PEPA process P may perform an (α, r) activity. We denote this fact by $P \xrightarrow{(\alpha,r)}$, and the fact that P is capable of evolving by $P \rightarrow$. Moreover, if P may evolve into Q by this activity, Q is called a (one-step) *derivative* of P, and we write $P \xrightarrow{(\alpha,r)} Q$. The relation denoted by $\xrightarrow{(\alpha,r)}$ is called the *transition relation.* For any PEPA process P, the *derivative set* of P, $ds(P)$, is the least set of derivatives closed under the transition relation, and it thus captures all the reachable states of the system i.e.

- if $Q \stackrel{\text{def}}{=} P$, then $Q \in ds(P)$
- if $Q \in ds(P)$, and $Q \xrightarrow{(\alpha,r)} Q'$ for some (α, r), then $Q' \in ds(P)$

Furthermore, for any PEPA process P, we denote the multi-set of activities currently enabled by P as $\mathcal{A}ct(P)$, and the types of these activities as $\mathcal{A}(P)$. The multi-set of activities enabled by P over the course of its lifetime is given by $\overrightarrow{\mathcal{A}ct}(P)$. This is defined more formally as $\biguplus_{P' \in ds(P)} \mathcal{A}ct(P')$. In similar fashion, the set of action types represented in $\overrightarrow{\mathcal{A}ct}(P)$ is denoted $\vec{\mathcal{A}}(P)$. We abuse notation and treat multi-sets as sets where required. If process P is syntactically equivalent to Q, we write $P \equiv Q$. Finally, we represent sequences of objects by enclosing them in braces; for example, we may denote an indexed sequence of processes P_i as $<P_1, \ldots, P_n>$.

PEPA has a formal operational semantics [1] and this associates a labelled *multi*-transition system with each PEPA process expression. A labelled transition system is a triple $(S, T, \longrightarrow)$ where S is a set of states, T a set of transition labels, and $\longrightarrow \subseteq S \times T \times S$ is the transition relation. With PEPA, the states are syntactic process expressions, the transition labels are the (α, r) activities, and the transition relation is defined inductively by the operational semantics. However a multi-transition relation must be used because the timing behaviour of a process will depend on the number of instances of an activity that are enabled.

3 General Distributions

In this section, we discuss various approaches to stochastic modelling with general probability distributions. We begin with an explanation of insensitivity, and formally introduce the stochastic model which was used as a vehicle for the insensitivity property. The conditions on the stochastic process that guarantee insensitivity are then described, and we discuss some work which has identified conditions under which stochastic Petri net (SPN) models may be insensitive to particular transitions. We conclude this section with a discussion of current approaches to incorporating general distributions into stochastic process algebra.

3.1 Insensitivity

The exponential assumption inherent in Markovian analysis is regarded by some as a restriction in the application of SPA modelling. It would therefore be beneficial to be able to incorporate more general distributions into SPA performance models. A stochastic process is said to be insensitive if its steady-state distribution depends on the distribution of one or more of the random variables representing residence time in a state *only through their mean*. Just as steady-state is characterised by a set of *balance equations* insensitivity of residence time in a state is characterised by a set of *insensitivity balance equations*. These are interpreted with respect to a particular model, which we introduce next.

The Generalised Semi-Markov Process A Generalised Semi-Markov Process (GSMP) is defined on a set of states $\{g \mid g \in G\}$. Within each of these states are active elements, $s \in S$. The current state g will contain a set of active elements, each element s with a *lifetime* which decays at the state dependent rate $c(s, g)$. When the lifetime of an active element s expires, the process moves to another state $g' \in G$ with probability $P(g, s, g')$. When the process changes from state g due to the death of an active element the remaining elements from $g \cap S^*$ retain their spent lifetimes. Active elements new to the current state are given new lifetimes, as samples drawn from their governing distribution functions. Let S', S^* be disjoint, and such that $S' \cup S^* = S$. If $s \in S'$ then the lifetime of s is exponentially distributed; if $s \in S^*$, s has a generally distributed lifetime. A restriction on the process's behaviour is that no two active elements from S^* may be activated or die simultaneously.

Insensitivity results were originally presented with respect to the GSMP. Working with ths model, Matthes [2] showed:

Theorem 1 (Matthes) *The following two statements are equivalent:*

1. *The process is insensitive to the elements of S^*. That is, the distributions of the lifetimes of the elements of S^* may be replaced by any other distribution with the same mean, while still retaining the same equilibrium distribution.*
2. *When all elements of S^* are assumed to be exponentially distributed, the flux out of each state due to the death of an element of S^* is equivalent to the flux into that state due to the birth of that element.*

The second statement describes the *insensitivity balance equations* for the GSMP.

Conditions for Insensitivity High level performance modelling paradigms provide features which are active, in some sense, for a randomly distributed length of time. For example, SPNs incorporate transitions, and SPAs typically provide activities. It is these features for which it would be useful to relax the exponential assumption. However exploiting insensitivity is not straightforward, and this is because a feature which appears once in a high-level model may be represented in many states of the underlying stochastic process. Conditions for insensitivity have been investigated in the context of SPNs. Some early work by Dugan *at al.* [3] considered SPNs which satisfied the following rules:

1. The firing time of non-conflicting transitions which are enabled concurrently must be exponentially distributed.
2. The firing time of an exclusive transition, a transition which is never enabled concurrently with another, may be generally distributed.
3. A transition in conflict may be generally distributed, but all others with which it conflicts must be exponentially distributed.

The intuition for these rules can be understood by examining the authors' choice of stochastic model, the *semi-Markov process* (SMP). An SMP is a renewal process that passes through a set of states S at successive renewal points according to a transition probability matrix (therefore a Markov chain). For a given sequence of states, the sojourn times in each state, given each successor, are independent. Therefore at transition times, this process has the Markov property.

The first two conditions can be readily understood by recourse to the balance equations of the stochastic process. The second condition listed above is a strong one; for such a transition t, every marking in which it is enabled represents a state in which no other transitions are enabled. However this means the insensitivity balance equations for t are immediately satisfied by the global balance equations for the states in which t is enabled. However the first condition rules out generally distributed concurrently enabled (CE) non-conflicting (NC) transitions; there is no guarantee that the insensitivity balance equations are satisfied by the global balance equations.

As described by Henderson and Lucic [4], the third condition can be understood as a practical concession which means that an SMP, which features general distributions, may still be generated and solved with little computational effort. The restriction leads to tractable next state probabilities, due to a result presented by Ajmone Marsan and Chiola [5]. In principle, all CE transitions which conflict may be generally distributed. When one fires, all others are disabled, and a realistic interpretation is that when re-enabled, each transition is assigned a 'new' time-to-live. This means that in any marking corresponding to a set of CE-NC transitions, the next marking will not require a record of any transition's residual lifetime. Since the transitions notionally compete, the next marking should be chosen based on the transition which is fastest to fire. Therefore, the distribution of the sojourn time in a state of the SMP is given by the minimum of the distributions associated with each transition.

Henderson and Lucic [4] consider the conditions under which a GSMP may instead be produced from an SPN. In their translation, SPN transitions are represented by active elements; in this setting, all concurrently enabled generally distributed transitions carry over their spent lifetimes to successor markings. The authors give a translation of Matthes theorem for use with stochastic Petri nets:

Corollary 1 (Henderson and Lucic) *The following two statements are equivalent:*

1. *The SPN model is insensitive to a generally distributed transition t.*
2. *The purely Markov process, i.e. when $S = S'$, has the property that for all markings j that enable transition t, the flux into j enabling t is balanced by the flux out of j due to the death of t.*

In general, the choice of successor state is time-dependent; consider two enabled transitions, one of which, t, is uniformly distributed between m and n. If n time units elapse, then t must definitely have fired. If a transition fires before m time units pass, it must not have been t. This property is called *age dependent routing*. Fortunately, a result due to Rumsiewicz and Henderson [6] states that the *time-averaged* stochastic process, with age independent transition probabilities, gives the same equilibrium distribution as the original process, under the condition that the time-averaged process is insensitive to its generally distributed transitions. The technical difficulty is constructing the time-averaged mean sojourn time, given a set of transitions with arbitrary distributions, and then the next state probabilities. Although our approach will focus on insensitivity of SPA activities, it also encounters this difficulty. We use Matthes theorem in an SPA setting in Section 4.1, where we give a GSMP semantics to PEPA, and thus illustrate what we mean by the insensitivity of PEPA activities.

3.2 Process Algebras with General Distributions

An alternative, and latterly popular, approach to incorporating general distributions is to build a process algebra calculus where every activity may have a generally distributed lifetime. Such an approach has both benefits and disadvantages. The obvious and greatest advantage is the extra modelling flexibility this affords to the user. There is no longer a requirement that model activities have durations which are exponentially distributed only.

One disadvantage is that some familiar process algebra rules are no longer applicable in general. For example, consider the parallel composition of two processes, each capable of performing a single activity. The familiar *expansion law* states, using informal notation, that $\mathtt{a} \mid\mid \mathtt{b}$ is *observationally* equivalent to $\mathtt{a.b} + \mathtt{b.a}$. However, when $\mathtt{a}$ and $\mathtt{b}$ are not modelled with "memoryless" distributions, the interleaving approach is incorrect. This is because after one activity completes, the choice does not represent the spent lifetime of the other activity.

When general distributions may be used arbitrarily in a process algebra model, it becomes very difficult in general to solve the process, if, for example, the user is interested in a steady-state probability distribution. Markovian models can be mapped to continuous time Markov chains, and these may be solved for steady-state using linear algebra. Non-Markovian process algebra models correspond to less restricted stochastic processes.

An early non-Markovian approach to process algebra was exemplified by Strulo and Harrison [7]. Their framework enhanced traditional process algebra with probabilistic and timed features, resulting in a model with several distinct transition relations. Their approach to performance evaluation was to show how their models could evolve over time via a discrete event simulation.

A recent example of a process algebra incorporating general probability distributions is *Generalised Semi-Markovian Process Algebra* (GSMPA), by Bravetti *et al.* [8]. Their calculus incorporates all the traditional process algebra combinators, including choice and parallel composition, and they provide a mapping to a GSMP, the stochastic process introduced in Section 3.1. The authors give an example of a simple queueing system modelled in GSMPA, where the queue has a deterministic service time. They determine that the resulting GSMP is insensitive when particular states are amalgamated, and from it derive a CTMC which they are able to solve conventionally for steady-state. We contrast this to the new derived combinator for PEPA we present in this paper—we do not allow the modeller the freedom to arbitrarily use generally distributed activities. However, we guarantee that the use of the combinator ensures that the modeller need not assume that any of a particular set of activities used in the model are exponentially distributed.

An alternative stochastic process algebra is *spades*(♠), introduced by D'Argenio *et al.* [9]. Their calculus differs from traditional approaches in that they separate the stochastic behaviour of the model from the *actions* it performs. This immediately gives a more visible correspondence with a GSMP. For example, if P is a spades process, then so is $\{\!|C|\!\}P$ where $\{\!|C|\!\}$ represents a set of *clocks*. Each clock has a distribution function, and thus corresponds to an active element of a GSMP. $\{\!|C|\!\}P$ represents a process where all clocks in $\{\!|C|\!\}$ are set according to their distribution functions, and begin counting down. The spades process $C \mapsto P$ represents the process that may become P if the clock C has reached zero. This represents a state change in a GSMP when an active element reaches the end of its lifetime. Notably, their calculus allows processes to be expressed in which clock settings persist over transitions, and thus spent lifetimes are respected. Moreover due to their separation of action and duration, they recover a form of the expansion law. However for performance evaluation, they do not attempt to use analytical techniques, and instead choose discrete event simulation.

4 A Structure for Generally Distributed CE NC Activities

In this section, we present a derived combinator which allows the limited construction of models containing CE activities which are insensitive, and thus may be generally distributed. Simple sequential components directly generate an SMP and are insensitive to all activities; however the models we construct are not sequential components in general. In essence, our constructed models consist of a set of concurrent sequential components, with a synchronisation discipline. This takes the form of an arbiter process which synchronises with two or more sequential components. The synchronisation enforces a *queueing* discipline and causes each sequential component to wait in a queue at a particular point in its execution. When a process leaves a particular queue, it does so with a fixed rate particular to that queue. However, despite such interaction, all activities not enabled while in the queue are insensitive to their distributions. In this section, we first state what is meant by the insensitivity of a PEPA activity. Section 4.2 proceeds to define a new combinator, $Q_A(.)$, leading to the statement of our result in Section 4.3 which guarantees the insensitivity of a particular set of the process's activities.

4.1 Mapping a PEPA Model to a GSMP

First, we give a meaning to the insensitivity of a PEPA activity by providing a mapping from a PEPA model to a GSMP. Such a mapping was provided by Hillston [10] for earlier work on examining insensitivity of PEPA models. In this setting, a mapping is given for a PEPA model as a configuration of *top-level* components, for example for a cooperation of two sequential components. The active elements of the GSMP are multisets of enabled activities. A cooperation which would in general enable three active elements—one for the multiset of individual activities that each component could perform individually, and one for the activities on which each component must cooperate. For example, consider the PEPA component

$$((\mathtt{a}, r_1).P_1 + (\mathtt{c}, r_2).P_2) \underset{\{\mathtt{a},\mathtt{b}\}}{\bowtie} ((\mathtt{a}, s_1).Q_1 + (\mathtt{b}, s_2).Q_2 + (\mathtt{c}, s_3).Q_3) \tag{4.1}$$

Under Hillston's mapping, this component is represented by the GSMP state

$$\begin{gathered} (\{(\mathtt{c}, r_2)\}, \{(\mathtt{c}, s_3)\}, \{(\mathtt{a}, t_1)\}) \\ \text{where } t_1 = \min(r_1, s_1) \end{gathered} \tag{4.2}$$

The active elements present represent the individual abilities of both sides of the cooperation to proceed, and one element to represent the shared ability of the component to proceed via cooperation. The PEPA model structure we study in this paper is of a restricted form, and so the GSMP mapping we provide is more limited, and in particular does not need to take account of arbitrary cooperation in PEPA models.

Our approach is based on the consideration of PEPA models of the form described above, that is a collection of concurrent sequential components with a queueing discipline. The model structure is defined precisely in Section 4.2. In our approach, again each multiset of activities enabled by a sequential component represents an active element of the GSMP. Without a queueing discipline, we consider simple models, which consist of the unrestricted cooperation of sequential PEPA components. For example, consider the component given below.

$$((\mathtt{a}, r_1).P_1 + (\mathtt{c}, r_2).P_2) \;||\; (\mathtt{b}, s_1).Q_1 \tag{4.3}$$

Under our mapping, this would be represented by the GSMP state

$$(\{(\mathtt{a}, r_1), (\mathtt{c}, r_2)\}, \{(\mathtt{b}, s_1)\}) \tag{4.4}$$

The queueing discipline consists of a component with the potential to interact with each of the sequential components; however the component has no individual ability, and must cooperate in order to evolve. In this respect, the queueing discipline provides a similar restriction to that described by Hillston in [11]. Its effect on the GSMP state is not to add active elements, but rather to reduce the number of activities which constitute existing active elements.

We proceed to give a more formal definition. Assume a PEPA model of the form $P \stackrel{\text{def}}{=} (\prod_{i=1}^{n} P_i) \underset{M}{\bowtie} R$, under the condition that for $R' \in ds(R)$, if $R' \rightarrow$, then it must be as part of a cooperation with $\prod_{i=1}^{n} P_i$. Each active element $s \in S$ is such that $s \in \mathbb{N}^{\mathcal{A}ct}$, the set of all sub-*multisets* of $\mathcal{A}ct$. Each state $g \in G$ is such that $g \in \prod_{i=1}^{n} S$, that is each state consists of a size n cartesian product of active elements, as illustrated in the example above. For the form of model we consider, there is one and only one GSMP state for each $P' \in ds(P)$.

Definition 1 (GSMP state). *Let $P \stackrel{\text{def}}{=} (\prod_{i=1}^{n} P_i) \underset{M}{\bowtie} R$. The GSMP state for derivative P is given by*

$$\prod_{i=1}^{n}(\mathcal{A}ct(P_i \underset{M}{\bowtie} R)) \tag{4.5}$$

This definition gives a GSMP with some simple properties. The completion of an activity results in the death of an active element; we can show that with the particular restrictions that our queueing discipline places on the sequential components that the death of an active element corresponding to P_i for some i will not cause the death of any active elements for components P_j, $j \neq i$. Since we will be interested in studying when activities, and thus active elements can be generally distributed, this property is important—it ensures that the remaining lifetimes of an active element representing an activity will persist over a state change, unless the state change corresponds to the completion of that activity.

Since an active element is a *multiset* of activities, then the death of an active element also corresponds to the disabling of more than one activity in general. This interpretation is correct, since the multiset will represent *competing*, and therefore mutually disabling, activities, as the result of a sequential component offering a choice.

For a PEPA model, this GSMP mapping would result in active elements with exponential lifetimes. However, in particular circumstances, we will allow generally distributed activities, and show that these do not affect the steady-state solution. When an active element is generally distributed, its firing results in a next state probability which is well-defined, but in general difficult to calculate. However, we note that where a sequential component does not currently enable a choice, then the active element is represented by a single activity. In such circumstances, the next state probability is trivial. Furthermore, despite the loss of behavioural independence of each sequential component due to the restrictions of the queueing discipline, it is still the case that such activities are provably insensitive. With this mapping, we can use an analogous form of Matthes theorem (1) for PEPA with which to prove our result.

Corollary 2 *The following two statements are equivalent:*

1. *The PEPA model is insensitive to a generally distributed activity* `a`.
2. *The purely Markov process, i.e. when $S = S'$, has the property that for all states P that enable transition* `a`*, the flux into P enabling* `a` *is balanced by the flux out of P due to the completion or disabling of* `a`.

4.2 A Derived Combinator for the Queueing Discipline

In this section, we proceed to describe our new combinator, $Q_A(.)$, with which we can build PEPA models insensitive to the distributions associated with particular activities. Assume an arbitrary PEPA model $\prod_{i=1}^{n} P_i$, therefore with no cooperations, upon which we wish to enforce our queueing discipline. We introduce a derived combinator, such that for a particular process R and cooperation set M which depend on a set of action types A, we have

$$Q_A(P_1, \ldots, P_n) \stackrel{\text{def}}{=} (\prod_{i=1}^{n} P_i) \underset{M}{\bowtie} R \tag{4.6}$$

An intuitive meaning for this notation is that $Q_A(P_1, \ldots, P_n)$ represents the parallel composition of each $P_i, 1 \le i \le n$ except in particular cases. These cases are where a subset of P_i processes each currently enable an activity $\mathtt{a_i}$ whose action type is in A. Then, the processes must "queue" to perform these activities. Therefore, at any one time, only one of the queueing P_i processes is allowed to perform its $\mathtt{a_i}$ activity.

Let A be such that for each $\mathtt{a} \in A$, there exists a unique P_i such that $\mathtt{a} \in \vec{\mathcal{A}}(P_i)$; and for each P_i, there exists a unique $\mathtt{a} \in A$ such that $\mathtt{a} \in \vec{\mathcal{A}}(P_i)$. Each P_i will be forced to queue to perform its activity whose type is in A. We proceed with some definitions which allow us to construct a cooperation set M which we call the *arbiter synchronisation set*. This procedure is mechanical, and could be simply automated.

Definition 2 (Enabling activity). *Let P be a process, and* $\mathtt{a}$ *an activity, such that* $\mathtt{a} \in \overrightarrow{Act}(P)$. *Then* $\mathtt{b}$ *is an* enabling activity for $\mathtt{a}$ *if for some derivative P' of P such that $P' \overset{\mathtt{a}}{\not\longrightarrow}$, it is the case that $P' \xrightarrow{\mathtt{b}} P''$, and $P'' \xrightarrow{\mathtt{a}}$.*

We can view an enabling activity for $\mathtt{a}$ as simply one which must be performed immediately before the model enables $\mathtt{a}$. We make the assumption that for each i, $\mathtt{a_i}$, the enabling activity of process P_i, is unique to components $P_i' \in ds(P_i)$. This ensures there is no confusion over which queueing component is next in line to leave the queue.

Definition 3 (Arbiter synchronisation set). *Let $S \equiv Q_A(P_1, \ldots, P_n)$ be a process with a queueing discipline. The* arbiter synchronisation set M *of S is given by $A \cup e_A$, where $e_A = \{\mathtt{b} \mid \mathtt{b} \in \vec{\mathcal{A}}(\prod_{i=1}^{n} P_i)$,* $\mathtt{b}$ *is an enabling activity for some* $\mathtt{a} \in A\}$.

From here, we define a process R which enforces the required queueing discipline. We call R an *arbiter process.*

Definition 4 (Arbiter process). *Let $S \equiv Q_A(P_1, \ldots, P_n)$ be a process with a queueing discipline. Let $e_{\mathtt{a}}(P)$ denote the set of enabling activities for* $\mathtt{a} \in A$ *of process P. Let $\overrightarrow{Act}_A(P) \stackrel{\text{def}}{=} \{(\mathtt{a}, r) \in \overrightarrow{Act}(P) \mid \mathtt{a} \in A\}$ denote the set of "queue" activities belonging to process P.*

The arbiter process *for S is given by $R_{<>}$, defined as*

$$
\begin{aligned}
R_{<>} &\stackrel{\text{def}}{=} \textstyle\sum_{1\le i\le n} \sum_{\mathtt{a}\in A} \sum_{\mathtt{b}\in e_{\mathtt{a}}(P_i)} (\mathtt{b}, \top).R_{<P_i>} \\
R_{<P_j,P_m,\dots,P_n>} &\stackrel{\text{def}}{=} \textstyle\sum_{\substack{1\le i\le n \\ j\notin\{j,m,\dots,n\}}} \sum_{\mathtt{a}\in A} \sum_{\mathtt{b}\in e_{\mathtt{a}}(P_i)} (\mathtt{b}, \top).R_{<P_j,P_m,\dots,P_n,P_i>} \\
&\quad + \textstyle\sum_{\mathtt{a}\in \overrightarrow{Act}_A(P_j)} (\mathtt{a}, \top).R_{<P_m,\dots,P_n>}, \qquad |\{P_j, P_m, \dots, P_n\}| < n \\
R_{<P_j,P_m,\dots,P_n>} &\stackrel{\text{def}}{=} \textstyle\sum_{\mathtt{a}\in \overrightarrow{Act}_A(P_j)} (\mathtt{a}, \top).R_{<P_m,\dots,P_n>}, \qquad |\{P_j, P_m, \dots, P_n\}| = n
\end{aligned}
$$

This arbiter process can be viewed as a PEPA definition of a queue, which enforces an ordering on the processes it controls. The size of the queue specified by the combinator is equal to the number of processes given as the combinator's arguments.

A Restriction on Activity Rates The arbiter process given in Definition 4 only performs activities which are passive with respect to the model with which it interacts. This simplifies the definition, and ensures that the arbiter does not affect the *rate* at which any activities are performed. However, in order to gain insensitivity results, an extra restriction is required:

Unless the queue is full, all queued processes *must* perform their queue activities with a rate fixed for the particular arbiter process (that is, the mean of the distribution with which a queued process performs its queue activity is common to all queued processes).

It is a mechanical task to alter a PEPA model with a queueing discipline such that it conforms to this restriction. Recall that the PEPA definition of a model with a queueing discipline over the action set A is given by

$$
Q_A(P_1, \dots, P_n) \stackrel{\text{def}}{=} (\prod_{i=1}^{n} P_i) \underset{M}{\bowtie} R
$$

for an appropriate arbiter process R and synchronisation set M. We now define a simple translation of the current queueing discipline model such that it conforms with the required rate restriction. Crucial to this translation is the use of PEPA's passive activities. Currently, the arbiter restricts the behaviour of processes, but does not affect the rate at which they perform activities. The idea behind the translation is that each process's right to individual behaviour in the queue is removed, by making the queue activity passive, so that in most circumstances, the queue defines at what rate processes may pass through. However, when the queue is full, the process at the head of the queue regains the right to perform its queue activity at its own specified rate. The translation is given below.

Definition 5 (Rate replacement). *Let P be a PEPA process and A a set of action types. Then $P_{A\to\top}$ is the PEPA process where each occurrence of a non-passive activity of type* $\mathtt{a}$ *is modified such that it becomes passive. $P_{A\to\top}$ is defined on the structure of P as*

$$\begin{array}{rl}
P \equiv Q \underset{L}{\bowtie} R : & Q_{A\to\top} \underset{L}{\bowtie} R_{A\to\top} \\
P \equiv Q + R : & Q_{A\to\top} + R_{A\to\top} \\
P \equiv Q/L : & Q_{A\to\top}/L \\
P \equiv (\mathtt{a}, r).Q : & (\mathtt{a}, \top).Q_{A\to\top} \ \textit{if}\ \mathtt{a} \in A \\
P \equiv (\mathtt{a}, r).Q : & (\mathtt{a}, r).Q_{A\to\top} \ \textit{if}\ \mathtt{a} \notin A \\
\textit{otherwise} : & Q_{A\to\top} \ \textit{where}\ P \stackrel{\text{def}}{=} Q
\end{array}$$

Now, given a queueing discipline model $Q_A(P_1, \dots, P_n)$, let r_i represent the rate at which P_i performs its queue activity. We make the following modification to the arbiter process:

Definition 6 (Rate restricted arbiter process). *Let $S \equiv Q_A(P_1, \dots, P_n)$ be a process with a queueing discipline. Let r be the function $\{(i, r_i) \mid i \in I\}$ which for process P_i gives the rate at which it performs its queue activity. Then the arbiter process* rate restricted by *s, R^s, is defined as*

$$\begin{aligned}
R^s_{<>} &\stackrel{\text{def}}{=} \textstyle\sum_{1\le i\le n} \sum_{\mathtt{a}\in A} \sum_{\mathtt{b}\in e_{\mathtt{a}}(P_i)} (\mathtt{b}, \top).R^s_{<P_i>} \\
R^s_{<P_j, P_m, \dots, P_n>} &\stackrel{\text{def}}{=} \textstyle\sum_{\substack{1\le i\le n \\ j\notin\{j,m,\dots,n\}}} \sum_{\mathtt{a}\in A} \sum_{\mathtt{b}\in e_{\mathtt{a}}(P_i)} (\mathtt{b}, \top).R^s_{<P_j, P_m, \dots, P_n, P_i>} \\
&\quad + \textstyle\sum_{\mathtt{a}\in \overrightarrow{Act}_A(P_j)} (\mathtt{a}, s).R^s_{<P_m, \dots, P_n>}, \qquad |{<P_j, P_m, \dots, P_n>}| < n \\
R^s_{<P_j, P_m, \dots, P_n>} &\stackrel{\text{def}}{=} \textstyle\sum_{\mathtt{a}\in \overrightarrow{Act}_A(P_j)} (\mathtt{a}, r(i)).R^s_{<P_m, \dots, P_n>}, \ |{<P_j, P_m, \dots, P_n>}| = n
\end{aligned}$$

Now we can simply write out the definition of the rate restricted queueing discipline model.

$$Q_A(P_1, \dots, P_n) \text{ rate restricted by } s = Q_{A,s}(P_1, \dots, P_n) \stackrel{\text{def}}{=} (\prod_{i=1}^{n} P_{iA\to\top}) \underset{M}{\bowtie} R^s \tag{4.7}$$

4.3 Insensitivity and the Queueing Discipline

With these definitions in place, we are now in a position to state our insensitivity result: that given a collection of sequential components which interact according to a particular queueing discipline, we can nevertheless guarantee the insensitivity of all activities except those involved in leaving a queue. The proof is straightforward in nature, and shows that the required insensitivity balance equations are satisfied by the process's global balance equations. Here we outline the proof, and refer the interested reader to [12] for details.

Let there be a finite set of sequential PEPA components P_i, $1 \le i \le n$, that is components without the syntactic cooperation or hiding operators. We assume for now that all action types are distinct, and we abuse notation by using the name $\mathtt{a}$ to represent both the type and the rate of an activity $\mathtt{a}$. We now assume a set of queueing disciplines each of which operates upon a subset of sequential components P_i. An m-place queue may operate upon an m-sized subset of components; we will identify a set of distinct activities from each P_i for which that component must queue in order to perform. Let $\{I_k \mid 1 \le k \le n\}$ be a set of subsets of I such that $i \in \cup_{1 \leq j \leq k} I_j, 1 \leq i \leq n$. Note that these subsets need not be distinct. Each I_j will be used to define a queueing discipline such that for each $P_i \in I_j$, P_i will be forced to enter a queue once on each cycle, leaving it with its designated queue activity. We denote the set of the types of queue activities of a set of processes $\{P_i \mid i \in I_j\}$ as A_j.

We respect the queueing rate restriction on each process P_i. At any point, a number of processes may be queued due to the jth queue (the queue length is state dependent). Let e_j represent the current queue length, such that $0 < e_j \leq |I_j|$, and where the queue contains process P_i. Then if P_i is at the front of the (non-full) queue, it may perform its queue activity $\mathtt{a}$ with action type in A_j at a rate μ_{je_j}. For convenience, we set $\mu_{j0} = 0$. We denote our PEPA process with a rate restricted queueing discipline as

$$S \stackrel{\text{def}}{=} Q_{A_1,\mu_{1e_1}}(Q_{A_2,\mu_{2e_2}}(\ldots Q_{A_k,\mu_{ke_k}}(P_1,\ldots,P_n)\ldots)) \equiv Q_\xi(P_1,\ldots,P_n)$$

We wish to show that despite the interaction imposed by the queues, S retains insensitivity of all activities except those which are queue activities, or those which are enabled by queue activities.

Theorem 2 *The process $S \stackrel{\text{def}}{=} Q_\xi(P_1,\ldots,P_n)$ is insensitive to all but its queue activities, with action types in $A_j, 1 \leq j \leq k$, and activities enabled by those queue activities.*

The Proof Method Due to space limitations, the proof for the result above is omitted though it can be found in full in [12]. Here we give a flavour of the proof, outlining the approach that was taken. The proof proceeds by constructing two sets of equations, for the process S, these being

- the *global balance equations*, satisfied by the process under Markovian assumptions when it is in steady-state, and
- the *insensitivity balance equations* for the non-queue activities, equations satisfied with respect to a state and an activity if it is the case that the steady-state solution of the process depends on the distribution of the activity rate *only* through its mean.

We propose a solution for the steady-state distribution of S, and show that this is also a solution to the insensitivity balance equations. The general form of the solution for a particular state is expressed in terms of the steady-state solutions of each component P_i, and the current queue lengths in that state. By this method, the process is shown to be insensitive to the residence time in these states, and therefore insensitive to the activities represented by the active elements.

5 A Simple Transaction Processing System

We illustrate our queueing discipline by using it to give a simple model of a transaction processing system. In recent years, the demand for these systems has grown rapidly, along with a strong focus on performance requirements. These systems consist of a centralised database, which controls a set of database *objects*; and a set of *transactions*, which attempt to access database objects, and thus perform, for example, user queries. Models of transaction processing systems abound in the literature; we refer the interested reader to [13] as a starting point.

The example we present is a simple transaction processing system, with components to represent a collection of transaction classes, and a manager to enforce locking rules on database objects. Broadly, each transaction class may launch a transaction, which may then attempt to access particular database objects, be they pages or even records. In a typical database system, each object will be in a certain state of access—perhaps currently being modified by a transaction, or perhaps not being used at all. A transaction in our model may either attempt to

- modify a particular database object, in which case it attempts to acquire a *write* lock for that object; or
- read a particular object in its current state, in which case it does so regardless of any locks present on that object; this is called a *dirty read*.

Dirty reads are useful when a set of data is required often and in real-time. Although they run the risk of occasionally reading out-of-date information, they have the great advantage that they do not impact on transaction concurrency, and so keep transaction throughput higher. However, if a transaction acquires a write lock on an object, no other transaction may also acquire a write lock until the first has *committed* its changes. This is modelled by forcing such transactions to block in an orderly fashion, sequentially gaining write access to the required object. According to the attributes of the class it belongs to, each transaction will choose to read from or write to an object with a fixed probability; and after the object access, another transaction of the same class will be immediately generated with another fixed probability. In this way, different transaction classes will be characterised in part by the mean number of transactions generated in a row.

We proceed by giving a model in Figure 1 which represents a class of transactions; that is, it can be used as part of a model when a sequence of transactions with particular characteristics are required. Component $TxnC_i$ represents the state in which no transactions of class C_i are presently being processed. From this point, we specify that there is a delay, a *think* time (mean $1/t_i$), before the user generates a job which requires access to the database. The think time represents the delay between jobs generated by the supposed ith user or application. From here, a transaction begins the process of accessing the database by attempting either a write lock, or a dirty read, which is modelled by a probabilistic choice over the `work` activity. The issue of deadlock is avoided since in our model, transactions cannot accumulate locks—locks are released immediately the processing of an object is complete. Our model is therefore closer to a *static* locking methodology [13].

$$
\begin{aligned}
TxnC_i &\stackrel{\text{def}}{=} (\mathtt{think_i}, t_i).Txn_i \\
Txn_i &\stackrel{\text{def}}{=} \textstyle\sum_{j=1}^{n} (\mathtt{work}, ww \times p_j).(\mathtt{writelock_{ij}}, r_{ij}).(\mathtt{commit_{ij}}, wl_{ij}).Loop_i \\
&\quad + (\mathtt{work}, wr).(\mathtt{dirtyread_r}, r).(\mathtt{accessdb}, rl).Loop_i \\
Loop_i &\stackrel{\text{def}}{=} (\mathtt{nexttxn_i}, n \times a_{ij}).TxnC_i \\
&\quad + (\mathtt{nexttxn_i}, (1-n) \times a_{ij}).Txn_i
\end{aligned}
$$

Fig. 1. PEPA model of a transaction class

Assuming that the transaction requires exclusive locks (such that other transactions must wait to access the locked objects), an additional factor in the sum over `work` governs which area of the database the transaction wishes to access. In this way, we capture the fact that parts of the database may be more frequently accessed than others. This phenomenon is known variously as the "b-c" rule and "80-20" rule [13], meaning that $b\%$ (20%) of the database is accessed on average $c\%$ (80%) of the time. Activities `writelock` and `dirtyread` model the point at which the transaction attempts to access an object. If a write lock is awarded, the object is used, and the transaction commits its changes. The work of the current transaction is now done, and the model now probabilistically determines if the current job features more transactions—if so, another is generated, and if not, we await another job from the ith user.

The next requirement is that we model the central database itself. This will be an abstraction, allowing access to database objects, and enforcing a two phase locking concurrency control scheme. We construct the database as a product of processes which control access to write-locked database objects. Each of these lock manager processes, and the collection constituting the database, are given by expressions the form of which appears in Figure 2.

Each $LockMgr$ process simply acts as a queue—a transaction performing a `writelock` claims exclusive access to a particular object, and any other transactions making subsequent conflicting requests are blocked until, by performing a `commit`, the original claimant releases the required resource. TPS gives the definition of the transaction processing system in its entirety.

In our transaction processing model, there are elements which we may wish not to model using an exponential distribution. One such activity may be the think time of each user, perhaps if the user in one case is not human, but a computer—thus $\mathtt{think_i}$, $1 \leq i \leq n$. If we accept a small restriction on the behaviour of the lock managers, it is possible to choose a generally distributed duration of action, since we may model our transaction processing system using our new combinator. The restriction is that each lock manager must control the rate at which waiting transactions may access the database i.e. each activity $\mathtt{commit_{kj}}$. This captures rate restriction, given in Section 4.2. We may argue that this is not unreasonable, since access times will depend on the database objects, not on the transactions themselves.

$$LockMgr_{j,<>} \stackrel{\text{def}}{=} \sum_{i=1}^{M} (\texttt{writelock}_{\texttt{ij}}, \top).LockMgr_{j,<i>}$$

$$LockMgr_{j,<k,a,\ldots,b>} \stackrel{\text{def}}{=} \sum_{\substack{i=1 \\ i \notin \{k,a,\ldots,b\}}}^{M} (\texttt{writelock}_{\texttt{ij}}, \top).LockMgr_{j,<k,a,\ldots,b,i>}$$

$$+ (\texttt{commit}_{\texttt{kj}}, \top).LockMgr_{j,<a,\ldots,b>} \qquad |\{k,a,\ldots,b\}| < n$$

$$TPS \stackrel{\text{def}}{=} (\prod_{i=1}^{M} TxnC_i) \underset{L}{\bowtie} (\prod_{i=1}^{n} LockMgr_{i,<>})$$

$$\text{where } L = \{\texttt{writelock}_{\texttt{ij}}, \texttt{commit}_{\texttt{ij}} \mid 1 \leq i \leq M, 1 \leq j \leq n\} \qquad (5.8)$$

Fig. 2. The system as a central database and a set of transactions

Given the above definitions, modelling the transaction processing system using a queueing discipline is straightforward. First, we construct the types of the queue activities for each lock manager:

$$A_j = \{\texttt{commit}_{\texttt{ij}}, i \leq i \leq M\} \text{ for } 1 \leq j \leq n$$

We assume that in a situation when a lock manager cannot accept any more transactions, it still controls the rate of access to the database. This gives an alternative representation of TPS as

$$TPS \stackrel{\text{def}}{=} Q_{A_1}(\ldots Q_{A_n}(TxnC_1, \ldots, TxnC_M)\ldots) \equiv Q_{\xi}(TxnC_1, \ldots, TxnC_M)$$

This proposition would allow us to model the transaction processing system in PEPA, yet with no assumption of the stochastic nature of each $\texttt{think}_{\texttt{i}}$ other than each mean.

6 Related and Further Work

An interesting fact about this insensitivity result is that our syntactic characterisation is distinct from the known criteria for SPA models which generate product form solutions (over submodels). A model which exhibits product form over submodels may be restricted in form, but allows for an efficient solution. The components of a model can be solved in isolation, and then combined to give a solution for the model as a whole. This approach avoids the computational burden involved in typical non-compositional techniques.

Harrison and Hillston [14] explore SPA structures which give rise to *quasi-reversible* (QR) models. Consider a stochastic process whose state is represented by a countable set of elements. An input process causes the stochastic process to change stage by changing the state of one of the elements; or an output process may reverse this state change by converting the element back. The process is QR if and only if for all times t, the current state of the process is independent of the input process after t, and the output process before t. The authors show that a PEPA model consisting of a cooperation of processes has a QR stochastic process if considered pairwise, each of the components form a *channel*. This means that a cooperation between two components leads to a derivative for which a cooperation is available to return the model to the original state.

Later work by Hillston and Thomas [11] takes advantage of a generalisation made by Boucherie [15], where several distinct product form results for SPN are shown to be special cases of a simple exclusion mechanism for a product process. In their setting, subcomponents compete over resources, with the restriction that if a subcomponent possesses a resource which another would like now, or at some point in its lifespan, then the rival is blocked until the resource is freed. The syntactic form of the PEPA models which generate such product form solutions look similar to the form we present in Section 4.2. Further work is required to determine if the two approaches will easily combine. However, resources claimed in the Boucherie setting are denied to others, which if they wish to use the resource now, or at some future point in their behaviours, must block. Clearly, the arbiter processes present in a queueing discipline model do not act as resources in this way, since they do not prevent the evolution of other components not currently queueing. Moreover, we note that our approach differs in that the structure is described entirely in terms of a new derived combinator, which allows the construction of models with our insensitivity property.

Sereno [16] take a different approach to finding product form solutions for PEPA models. He finds conditions on the *traffic equations* that guarantee the existence of a product form solution. The traffic equations for a PEPA model are analogous to those used in queueing theory, and are formed from the *routing process* generated by considering activities as states themselves. Again, we approach the problem in a different direction, in that we provide an operator which *guarantees* a product form, and with which insensitive models may be constructed. We note that the existing work on product form for process algebra contains many common ideas, and we speculate that it will be possible to capture much of it under one encompassing theory.

We note that the product form solution for our structure, along with the use of queueing disciplines, suggests a link to the well-known class of BCMP network queueing models [17]. This similarity is worth exploring further and formalising. The BCMP theorem states that queueing networks with nodes which can be classified as

- First come first served (FCFS); or
- Processor sharing (PS); or
- Infinite server (IS); or
- Last come first served–Premptive resume (LCFS-PR)

and which also contain multi-class traffic have a product form solution for the steady-state probability distribution of the node states. A comparison with the structure of models we present here suggests that it may be possible to view them too as networks of queues; each bottleneck point may represent a FCFS node, where the service time for each customer is identically distributed and dependent on the queue length; and the components between ordinary PEPA transitions could be seen to represent IS nodes, where notionally, each customer is delayed only by a service time. This comparison has not been made in any detail, and remains as interesting future work. However it may suggest directions in which to extend our construction—an obvious first step is to consider the PEPA analogues of IS and LCFS-PR nodes, and whether they may be simply integrated into the construction as it stands. Furthermore, this might provide a beginning for a comparison of process algebra and queueing networks in general.

6.1 Conclusion

To summarise, we have presented a new derived combinator for PEPA. This combinator is used to construct models of sequential processes which synchronise according to a queueing discipline. We show that despite such synchronisation, we can guarantee the insensitivity of residence time in particular states of the model, and thus can guarantee insensitivity of PEPA activities in these cases. Where the state does not enable a syntactic choice, we may choose to employ a non-exponential distribution to model the duration of the enabled activity. We believe this work should translate for use with other Markovian process algebras.

The structure we present is rigid, and the combinator builds PEPA models which are certainly a small subset of those possible with the complete algebra. However, this restrictive structure seems necessary in order to obtain insensitivity results; the insensitivity balance equations themselves place strict demands on the stochastic process. It is not possible to use the combinator to build arbitrary PEPA models; however we have shown in Section 5 that some real-life systems do exhibit the required structure at a suitable level of abstraction. For the future, we plan to further investigate syntactic structures in which insensitivity may be exploited.

An aspect of this work that we have not dealt with in this paper is *aggregation*. Hillston's original work on insensitivity for PEPA models [10] was motivated by a desire to remove behaviourally inert sequences of silent τ activities. Hillston showed that in particular circumstances the residence time spent in the set of states constituting the sequence was insensitive, and therefore the sequence could be treated as a single τ activity. Use of the work in this paper for aggregation would also be a legitimate pursuit; for example circumstances in which we may use a generally distributed activity could represent situations where part of the state space of the PEPA model has been aggregated.

Acknowledgements I would like to thank Jane Hillston both for the guidance with, and for the careful and patient proof-reading of this paper. Furthermore, I would like to thank the anonymous referees for many corrections and suggested improvements.

References

1. J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.
2. K. Matthes. Zur theorie der Bedienungsprozesse. In *Trans. 3rd Prague Conference Inf. Th., Stat. Dec. Fns. and Rand. Proc.*, pages 513–528, 1962.
3. J. B. Dugan, K. S. Trivedi, R. Geist, and V. Nicola. Extended stochastic Petri nets: Applications and analysis. Technical Report DUKE–TR–1984–16, Department of Computer Science, Duke University, 1984. Printed copies available from T.R. Librarian, Dept. of Computer Science, Duke University, Box 90129, Durham, NC 27708-0129.
4. W. Henderson and D. Lucic. Applications of Generalised Semi Markov Processes to Stochastic Petri Nets. *Performance of Distributed and Parallel Systems*, pages 315–328, 1989.
5. M. Ajmone Marsan and C. Chiola. On Petri nets with deterministic and exponentially distributed firing times. *Lecture Notes in Computer Science "Advances in Petri Nets", Springer-Verlag*, 266:132–145, 1987.
6. M. Rumsewicz and W. Henderson. Insensitivity with age-dependent routing. *Adv. Appl. Prob., Vol. 21*, pages 398–408, 1989.
7. P. G. Harrison and B. Strulo. *Stochastic Process Algebra for Discrete Event Simulation*, chapter 2, pages 18–37. Esprit Basic Research Series. Springer Verlag, 1995.
8. M. Bravetti, M. Bernardo, and R. Gorrieri. Towards performance evaluation with general distributions in process algebras. *Lecture Notes in Computer Science*, 1466:405–422, 1998.
9. P. R. D'Argenio, J.-P. Katoen, and E. Brinksma. A stochastic automata model and its algebraic approach. In *Proceedings of the Fifth Process Algebras and Performance Modelling Workshop*, pages 1–16, 1997.
10. J. Hillston. PEPA: Performance Enhanced Process Algebra. Technical Report CSR-24-93, Department of Computer Science, The University of Edinburgh, 1993.
11. J. Hillston and N. Thomas. Product Form Solution for a class of PEPA Models. *Performance Evaluation*, 35:171–192, 1999.
12. G. Clark. *Practical Techniques for the Analysis of Algebraic Models of Systems—to appear*. PhD thesis, The University of Edinburgh, 1999.
13. A. Thomasian. Concurrency control: methods, performance, and analysis. *ACM Computing Surveys*, 30(1):70–119, March 1998.
14. P. Harrison and J. Hillston. Exploiting quasi-reversible structures in Markovian process algebra models. *The Computer Journal*, 38(7):510–520, 1995.
15. R.J. Boucherie. A Characterisation of Independence for Competing Markov Chains with Applications to Stochastic Petri Nets. *IEEE Transactions on Software Engineering*, 20(7):536–544, July 1994.
16. M. Sereno. Towards a Product Form Solution for Stochastic Process Algebras. *Proceedings of PAPM'95*, 38(7):622–632, 1995.
17. F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed and mixed networks of queues with different classes of customers. *Journal of the ACM*, 22(2):248–260, April 1975.