



ELSEVIER

Performance Evaluation 50 (2002) 129–151

**PERFORMANCE  
EVALUATION**  
An International  
Journal

www.elsevier.com/locate/peva

# Product form solution for an insensitive stochastic process algebra structure

G. Clark, J. Hillston\*

*Laboratory for Foundations of Computer Science, The University of Edinburgh, Kings Buildings, Edinburgh, UK EH9 3JZ*

---

## Abstract

Recent research has investigated ways in which generally distributed random variables may be incorporated into stochastic process algebra (SPA). These proposals allow the arbitrary use of such variables, improving expressibility, but in general this makes performance evaluation difficult. Typically, simulation techniques must be employed. We attack the goal of generally distributed random variables from the opposite direction, using the stochastic property of *insensitivity*. In this paper we describe a construction which guarantees the insensitivity of certain concurrently enabled non-conflicting SPA *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 and the criteria we list do not match any of those currently proposed for SPA. We highlight our technique with an example drawn from the field of transaction processing systems. Our analysis uses the SPA PEPA, and its associated conventions.

© 2002 Elsevier Science B.V. All rights reserved.

*Keywords:* Stochastic process algebra; Insensitivity; Generally distributed random variables; Product form solution

---

## 1. Introduction

Stochastic process algebras (SPAs [1–4]) 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. When activity durations are assumed to be exponentially distributed, an SPA model can be reduced to a continuous time Markov chain (CTMC), from which performance measures may be calculated numerically.

---

\* Corresponding author. Present address: Department of Computer Science, University of Edinburgh, Mayfield Road James Clerk Maxwell Building, Edinburgh EH9 3JZ, UK. Tel.: +44-131-650-5199; fax: +44-131-667-7209.

*E-mail addresses:* gcla@dcs.ed.ac.uk (G. Clark), jeh@dcs.ed.ac.uk (J. Hillston).

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 generally distributed random variables 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 the tractability of the numerical solution of the underlying stochastic process. 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 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 SPA 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 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 [Section 2](#) we present a short overview of PEPA, the SPA used in our presentation, and some of its associated theory and conventions. In [Section 3](#) we discuss insensitivity, presenting the basic results, and review some process algebras with support for generally distributed random variables. [Section 4](#) describes the extent to which generally distributed random variables 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 7](#) concludes the paper.

## 2. PEPA

Classical process algebras disregard the notion of measuring time, and model functional behaviour only. PEPA is a *stochastic* process algebra, and extends classical process 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 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  $(\alpha, r)$ , where  $\alpha$  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 by  $\mathcal{A}$ , and  $\mathbb{R}^+$  is defined as the set of positive real numbers together

with the symbol  $\top$ . This symbol denotes an unspecified (or *passive*) 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  $\mathcal{Act}$ .

### 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 determined 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  $(\alpha, r)$  (an activity of type  $\alpha$ , 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. A choice over processes  $P_i$  for  $1 \leq i \leq n$  is denoted by  $\sum_{i=1}^n P_i$ .
- *Cooperation*. If  $P$  and  $Q$  are processes, and  $L$  is a set of action types, then  $P \bowtie_L Q$  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  $\alpha$  in its current state, the other component becomes *blocked* on all  $\alpha$  activities. If both are capable of performing an  $\alpha$  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  $\alpha$  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  $\tau$ . Activities whose types are in  $L$  are said to be *hidden*, and cooperation is not possible on  $\tau$  activities.

### 2.2. Terminology

In this section, we introduce our terminology and some useful definitions. Suppose a PEPA process  $P$  may perform an  $(\alpha, r)$  activity. We denote this fact by  $P \xrightarrow{(\alpha, r)}$ . Furthermore,  $P \rightarrow$  implies that  $P \xrightarrow{(\alpha, r)}$  for some  $(\alpha, r)$ . If  $P$  may evolve into  $Q$  via some 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  $P \stackrel{\text{def}}{=} Q$ , then  $Q \in ds(P)$ ;
- if  $Q \in ds(P)$  and  $Q \xrightarrow{(\alpha, r)} Q'$  for some  $(\alpha, r)$ , then  $Q' \in ds(P)$ .

Furthermore, for any PEPA process  $P$ , we denote the multiset of activities currently enabled by  $P$  as  $\mathcal{Act}(P)$ , and the set of types of these activities as  $\mathcal{A}(P)$ . The multiset of activities enabled by  $P$  over the course of its lifetime is given by  $\vec{\mathcal{Act}}(P)$ . This is defined more formally as  $\uplus_{P' \in \text{ds}(P)} \mathcal{Act}(P')$ . In similar fashion, the set of action types represented in  $\vec{\mathcal{Act}}(P)$  is denoted by  $\vec{\mathcal{A}}(P)$ . If process  $P$  is syntactically equivalent to  $Q$ , we write  $P \equiv Q$ . Finally, we represent sequences of objects by enclosing them in braces; e.g., we may denote an indexed sequence of processes  $P_i$  as  $\langle P_1, \dots, P_n \rangle$ .

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  $(S, T, \rightarrow)$  where  $S$  is a set of states,  $T$  a set of transition labels and  $\rightarrow \subseteq S \times T \times S$  is the transition relation. With PEPA, the states are syntactic process expressions, the transition labels are the  $(\alpha, r)$  activities, and the transition relation is defined inductively by the operational semantics. 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. Any PEPA model which enables a passive activity is termed *incomplete*, meaning it is underspecified for the purpose of performance analysis. Given a PEPA model which is *complete*, its multi-transition relation may be used to generate, a CTMC,  $Q$ , providing the stochastic process semantics of a PEPA model.

Given a derivative  $P$ , the *exit rate* (or *departure rate*) from  $P$  is the parameter of the exponential distribution governing the sojourn time in  $P$ . It is denoted as  $q(P)$ . The *transition rate* between two components  $P$  and  $P'$  is given by

$$q(P, P') = \sum_{\{(\alpha, r): P \xrightarrow{(\alpha, r)} P'\}} r. \quad (1)$$

These  $q(P, P')$  provide the off-diagonal elements of the infinitesimal generator matrix of the CTMC  $Q$ . Given that the current derivative is  $P$ , the *probability* that the successor derivative will be  $P'$  is given by

$$p(P, P') = \frac{q(P, P')}{\sum_{P'' \in \text{ds}(P)} q(P, P'')}. \quad (2)$$

### 3. Using generally distributed random variables

In this section, we discuss various approaches to stochastic modelling with generally distributed random variables. 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 incorporate generally distributed random variables into SPA.

#### 3.1. Insensitivity

An exponential random variable is often inappropriate in applications of performance modelling. Many features of systems, such as timeouts, are simply not memoryless. It would therefore be beneficial to be able to incorporate generally distributed random variables 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.

### 3.1.1. The generalised semi-Markov process

A generalised semi-Markov process (GSMP) is defined on a set of states  $\{g : g \in G\}$ . 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 residual 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 this model, Matthes [5] showed:

**Theorem** (Matthes). *Given a GSMP, 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.

### 3.1.2. Conditions for insensitivity

SPA and SPN [6] can be regarded as high-level performance modelling paradigms. These behavioural description techniques are enhanced with timing information by associating durations, specified by random variables, with some modelling features: *transitions* in SPN and *activities* in SPA. It is these durations for which it would be useful to relax the exponential assumption. However, exploiting insensitivity is not straightforward. This is because a feature which appears once in the high-level model may impact on many states in the underlying stochastic process. Conditions for insensitivity have been investigated in the context of SPNs. Some early work by Dugan et al. [7] 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.

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. These 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 non-conflicting transitions; there is no guarantee that the insensitivity balance equations are satisfied by the global balance equations.

As described by Henderson and Lucic [8], a further condition, that a transition in conflict may be generally distributed, but all others with which it conflicts must be exponentially distributed, can be understood as a practical restriction which means that the generated SMP, which features general distributions, may be solved with little computational effort. The restriction leads to tractable next state probabilities, due to a result presented by Ajmone et al. [9]. In principle, all concurrently enabled transitions which conflict may be generally distributed. When one fires, all others are disabled, and an appropriate 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 conflicting 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 [8] considered the conditions under which a GSMP may 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 SPN.

**Corollary 3.1** (Henderson and Lucic [8]). *The following two statements are equivalent:*

- (1) *The SPN model is insensitive to each generally distributed transition  $t$ .*
- (2) *If all generally distributed transitions  $t$  are assumed to be exponentially distributed with the same mean, then for all markings  $j$  that enable  $t$  the flux, in the underlying Markov process, 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 Rumsewicz and Henderson [10] 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.

In Section 4 we define a construction which guarantees the insensitivity of some PEPA activities. In Section 4.2 we present a corollary of Matthes' theorem in the SPA context, and use this, together with a GSMP semantics of PEPA to prove that the insensitivity result will hold whenever models are constructed in this way. However, in the case of conflicting activities, we encounter the same difficulty with respect

to time-averaged mean sojourn times and next state probabilities. This leads to a practical restriction on the application of the result, which is discussed in detail in [Section 4.3](#).

### 3.2. Process algebras with general distributions

An alternative and popular approach to incorporate the generally distributed random variables into SPA is to build a process algebra 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  $a \parallel b$  is *observationally* equivalent to  $a.b + b.a$ . However, when  $a$  and  $b$  are not modelled with ‘memoryless’ distributions, the interleaving approach is incorrect when a pre-emption with restart semantics is used. This is because after one activity completes, the choice does not represent the residual lifetime of the other activity.

When general distributions may be used arbitrarily in a process algebra model, it becomes very difficult in general to numerically solve the process for a steady-state distribution. Non-Markovian process algebra models correspond to less restricted stochastic processes. An early non-Markovian approach to process algebra was exemplified by Harrison and Strulo [11]. Their framework enhanced traditional process algebra with probabilistic and timed features, resulting in a model with three 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. [12]. Their calculus incorporates all the traditional process algebra combinators, including choice and parallel composition, and as in PEPA, typed actions are assumed to have a duration. In [12] they provide a mapping to a GSMP, the stochastic process introduced in [Section 3.1.1](#). GSMPA is provided with a ST semantics, meaning that the evolution of an action is represented as a combination of action start and action termination. In later work [13], Bravetti and Gorrieri describe an alternative process algebra, interactive generalised semi-Markov processes (IGSMP), incorporating generally distributed durations. Here, as in earlier work on interactive Markov chains (IMCs) by Hermanns [4], typed actions are assumed to have no duration, whilst delays are represented by anonymous actions with (generally) distributed durations. Bravetti and Gorrieri restrict IGSMPA so that it is only possible to synchronise on untimed actions. In [13], the authors show that it is possible to reduce models by amalgamating sequences of non-timed silent actions in such a way that certain properties of the algebra are preserved and the underlying stochastic process is still a GSMP. Furthermore, the authors list as further work extending their work such that collections of timed silent actions can be aggregated in a similar fashion. For example, a sequence of generally distributed  $\tau$ -actions could be reduced to a single  $\tau$ -action distributed as the convolution of the distributions of those in the sequence.

In [12], Bravetti et al. 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 of the GSMP model are amalgamated, and then derive a CTMC which they are able to solve conventionally for steady-state. The new derived combinator we present in this paper does not allow the modeller the freedom to arbitrarily use generally distributed activities. Their use is limited to certain



activities within the given construction pattern. However, insensitivity is guaranteed by construction, and does not need to be determined by the modeller on a model-by-model basis.

An alternative SPA is spades ( $\spadesuit$ ), introduced by D’Argenio et al. [14]. Once more, the authors choose to separate the stochastic timed behaviour of the model from the actions it performs. Again 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 0. This represents a state change in a GSMP when an active element reaches the end of its lifetime. Of course their calculus allows processes to be expressed in which clock settings persist over transitions, and thus residual 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.

Finally, El-Rayes et al. [15] propose a distinctive approach to using general distributions in process algebra. They propose  $\text{PEPA}_{\text{ph}}^{\infty}$ , a modification of PEPA, such that activities can be distributed by phase-type probability distributions, and furthermore, consider models representing potentially infinitely many customers in a queueing system. The solution of an infinite-state model relies on it having a restricted structure—it must be decomposable into an initial portion, and a repetitive portion. The stochastic process underlying such a restricted  $\text{PEPA}_{\text{ph}}^{\infty}$  model will then have a (infinite) generator matrix with a particular repeating structure, and can be solved by using matrix geometric methods. Despite features in  $\text{PEPA}_{\text{ph}}^{\infty}$  which are superficially similar to those present in this paper, the methods presented are quite dissimilar. The models we construct implicitly contain models of queues, but none may contain an arbitrary number of customers. On the other hand, the theory of *insensitivity* employed need not restrict attention to phase-type distributions only. Furthermore, the models built using our new combinator can be solved in an entirely conventional way, as if all activities were exponentially distributed, a property guaranteed by the theory of insensitivity.

#### 4. A structure for generally distributed activities

In this section, we present a derived combinator which allows construction of models containing concurrently enabled activities and which are insensitive. Therefore, these activities may be generally distributed. Simple sequential components directly generate an SMP and are insensitive to all activities; similarly collections of concurrent, non-interacting sequential components are insensitive. However, such models have little practical application. Our aim has been to introduce sufficient interaction to make our models useful whilst constraining the form of interaction so that the insensitivity balance equations might still be consistent with the global balance equations.

The form of interaction we introduce is indirect—concurrent sequential components do not interact directly but influence each other’s progress via an imposed *queueing discipline*. The derived combinator we use specifies a set of action types and to perform activities of these types components in the scope of the combinator are forced to cooperate with an arbiter process in a FCFS order. Whilst waiting for cooperation we regard components as being in a ‘queue’ and we term models constructed with this combinator, *queueing discipline models*. The combinator also imposes the rate at which queueing activities are completed. As we will demonstrate, such models are insensitive to all activities enabled outside the ‘queues’.



Section 4.1 proceeds to define a new combinator,  $Q_{A,\xi}(\cdot)$ , leading to two results. Theorem 1 demonstrates the general solution form of queueing discipline models and Theorem 2 proves the insensitivity of a particular set of activities used in queueing discipline models.

#### 4.1. A derived combinator for the queueing discipline

In this section, we introduce the new combinator,  $Q_{A,\xi}(\cdot)$ . It can be used to build PEPA models insensitive to the distributions associated with particular activities. Assume a collection of  $n$  complete sequential components  $S_i$ ,  $1 \leq i \leq n$ , upon which a queueing discipline is to be enforced. Now we present a derived combinator which defines a cooperating process  $R_A$  and a cooperation set  $M_A$ , dependent on a set of action types  $A$ . As for a PEPA cooperation set, the hidden action type  $\tau$  may not appear in  $A$ .

**Definition 1** (Simple queueing combinator).

$$Q_A(S_1, \dots, S_n) \stackrel{\text{def}}{=} \left( \prod_{i=1}^n S_i \right) \bowtie_{M_A} R_A. \quad (3)$$

An intuitive meaning for this notation is that  $Q_A(S_1, \dots, S_n)$  allows each  $S_i$ ,  $1 \leq i \leq n$ , to proceed in parallel, independent of each other, and only enforces synchronisation of each  $S_i$  with a distinguished process, on action types determined by each  $S_i$  and the set  $A$ . If each  $S_i$  of a subset of processes currently enables an activity whose action type is in  $A$ , each  $S_i$  must ‘queue’ to perform its activity. Therefore, at any one time, only one of the queueing  $S_i$  processes is allowed to proceed.

Let  $A$  be such that for each  $\alpha \in A$ , there exists a unique  $S_i$  such that  $(\alpha, r) \in \vec{Act}(S_i)$  for some  $r$ ; and for each  $S_i$ , there exists a unique  $\alpha \in A$  such that  $(\alpha, r) \in \vec{Act}(S_i)$  for some  $r$ . Each  $S_i$  will be forced to queue to perform its activity whose type is in  $A$ . The restrictions mean that within a life-cycle, each process interacts with the arbiter only once and can be uniquely identified by the activity it seeks cooperation on.

Next we give some definitions which allow the specification of a cooperation set  $M_A$  called the *arbiter synchronisation set*. This procedure is mechanical, and could be simply automated.

**Definition 2** (Enabling action type). Let  $S$  be a sequential component, and  $a$  an activity. Then  $\beta$  is an enabling action type for  $a$  in  $S$  if

- for all derivatives  $S'$  of  $S$  such that  $S' \xrightarrow{a}$ , there exists a derivative  $S'' \not\equiv S'$  such that  $S'' \xrightarrow{(\beta,r)} S'$  for some  $r$ ;
- for every derivative  $S''$  of  $S$  such that for some  $r$ ,  $S'' \xrightarrow{(\beta,r)} S'$ , it is the case that  $S' \xrightarrow{a}$ .

An enabling action type for  $a$  can be viewed as the type of an activity which may only be performed immediately prior to the model enabling  $a$ , and which if performed, always leads to  $a$  being enabled. The set  $e_A(S)$  is a set of enabling action types for those activities of  $S$  with types present in  $A$ .

**Definition 3** (Arbiter synchronisation set). Let  $P \equiv Q_A(S_1, \dots, S_n)$  be a process with a queueing discipline. The arbiter synchronisation set  $M_A$  of  $P$  is given by  $(\bigcup_{i=1}^n e_A(S_i)) \cup A$ .

We make the assumption that for  $i, j$ , the enabling action types of process  $S_i$  are distinct from those of  $S_j$ , i.e.,  $e_A(S_i) \cap e_A(S_j) = \emptyset$  for  $i \neq j$ . This ensures there is no confusion over which component is about to enter or leave the queue. Furthermore, for the models studied in this paper, we assumed that for every

queue activity  $a$  that may be enabled by a process  $S$ , there exists an enabling action type for  $a$  in  $S$ . Next, a process  $R_A$  is defined which enforces the required queuing discipline.  $R_A$  is called an *arbiter process*.

**Definition 4** (Arbiter process). Let  $P \equiv Q_A(S_1, \dots, S_n)$  be a process with a queuing discipline. Let  $\vec{Act}_A(P) \stackrel{\text{def}}{=} \{(\alpha, r) \in \vec{Act}(P) : \alpha \in A\}$  denote the set of queue activities belonging to process  $P$ . The arbiter process for  $P$  is given by  $R_{A, \langle \rangle}$ , defined as

$$\begin{aligned} R_{A, \langle \rangle} &\stackrel{\text{def}}{=} \sum_{1 \leq i \leq n} \sum_{\alpha \in A} \sum_{\beta \in e_\alpha(S_i)} (\beta, \top) \cdot R_{A, \langle S_i \rangle}, \\ R_{A, \langle S_j, S_m, \dots, S_n \rangle} &\stackrel{\text{def}}{=} \sum_{\substack{1 \leq i \leq n \\ i \notin \{j, m, \dots, n\}}} \sum_{\alpha \in A} \sum_{\beta \in e_\alpha(S_i)} (\beta, \top) \cdot R_{A, \langle S_j, S_m, \dots, S_n, S_i \rangle} + \sum_{(\alpha, r) \in \vec{Act}_A(S_j)} (\alpha, \top) \cdot R_{A, \langle S_m, \dots, S_n \rangle}, \\ |\{S_j, S_m, \dots, S_n\}| &< n, \\ R_{A, \langle S_j, S_m, \dots, S_n \rangle} &\stackrel{\text{def}}{=} \sum_{(\alpha, r) \in \vec{Act}_A(S_j)} (\alpha, \top) \cdot R_{A, \langle S_m, \dots, S_n \rangle}, \quad |\{S_j, S_m, \dots, S_n\}| = n. \end{aligned}$$

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.

#### 4.1.1. 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—all queued processes *must* perform their queue activities with a rate fixed for the particular arbiter process and number of customers in the queue (i.e., the mean duration of the queue activity of a queued process is common to all queued processes, and may vary only with queue length). The restriction on the rates of the queue activities emphasises that these activities are really in the control of the implicit arbiter process. As we will see in the example presented in [Section 5](#), such a restriction is natural in some applications. The effect of the restriction is that when the insensitivity balance equations are constructed, they can be shown to be consistent with the global balance equations of the process.

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

$$Q_A(S_1, \dots, S_n) \stackrel{\text{def}}{=} \left( \prod_{i=1}^n S_i \right) \boxtimes_{M_A} R_A,$$

where  $R_A$  is the arbiter process and  $M_A$  the arbiter synchronisation set. Now a simple translation of the current queuing discipline model is defined 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 right to individual behaviour in the queue is removed, by making the queue activity

passive, so that the queue defines at what rate processes may pass through. Note that the queue activity is made passive inside the process itself, although this would be carried out transparently to the modeller.

**Definition 5** (Rate replacement). Let  $P$  be a PEPA process and  $A$  a set of action types that does not contain  $\tau$ . Then  $P_{A \rightarrow \top}$  is the PEPA process where each occurrence of a non-passive activity  $(\alpha, r)$  such that  $\alpha \in A$  is modified such that it becomes passive, i.e., it is changed to  $(\alpha, \top)$ .  $P_{A \rightarrow \top}$  is defined on the structure of  $P$  as

$$\begin{aligned} P &\equiv Q \bowtie_L R : Q_{A \rightarrow \top} \bowtie_L R_{A \rightarrow \top}, & P &\equiv Q + R : Q_{A \rightarrow \top} + R_{A \rightarrow \top}, \\ P &\equiv Q/L : Q_{A \rightarrow \top}/L, & P &\equiv (\alpha, r).Q : (\alpha, \top).Q_{A \rightarrow \top} \text{ if } \alpha \in A, \\ P &\equiv (\alpha, r).Q : (\alpha, r).Q_{A \rightarrow \top} \text{ if } \alpha \notin A, & P &\equiv Q : Q' \text{ where } Q' \stackrel{\text{def}}{=} R_{A \rightarrow \top} \text{ if } Q \stackrel{\text{def}}{=} R. \end{aligned}$$

Then the following modification of the arbiter process is made.

**Definition 6** (Rate restricted arbiter process). Let  $P \equiv Q_A(S_1, \dots, S_n)$  be a process with a queueing discipline. Let  $\xi$  be a sequence of rates  $\langle \kappa_1, \dots, \kappa_n \rangle$ , which defines the rate at which any  $S_i$  will perform its queue activity dependent on the current queue length. Then the arbiter process rate restricted by  $\xi$ ,  $R_{A, \langle \rangle}^\xi$ , is defined as

$$\begin{aligned} R_{A, \langle \rangle}^\xi &\stackrel{\text{def}}{=} \sum_{1 \leq i \leq n} \sum_{\alpha \in A} \sum_{\beta \in e_\alpha(S_i)} (\beta, \top).R_{A, \langle S_i \rangle}^\xi, \\ R_{A, \langle S_j, S_m, \dots, S_n \rangle}^\xi &\stackrel{\text{def}}{=} \sum_{\substack{1 \leq i \leq n \\ i \notin \{j, m, \dots, n\}}} \sum_{\alpha \in A} \sum_{\beta \in e_\alpha(S_i)} (\beta, \top).R_{A, \langle S_j, S_m, \dots, S_n, S_i \rangle}^\xi \\ &\quad + \sum_{(\alpha, r) \in \bar{Act}_A(S_j)} (\alpha, \kappa_k).R_{A, \langle S_m, \dots, S_n \rangle}^\xi, \quad |\langle S_j, S_m, \dots, S_n \rangle| = k < n, \\ R_{A, \langle S_j, S_m, \dots, S_n \rangle}^\xi &\stackrel{\text{def}}{=} \sum_{(\alpha, r) \in \bar{Act}_A(S_j)} (\alpha, \kappa_n).R_{A, \langle S_m, \dots, S_n \rangle}^\xi, \quad |\langle S_j, S_m, \dots, S_n \rangle| = n. \end{aligned}$$

Now it is a simple task to write out the definition of the rate restricted queueing discipline model.

$$Q_{A, \xi}(S_1, \dots, S_n) \stackrel{\text{def}}{=} \left( \prod_{i=1}^n S_{iA \rightarrow \top} \right) \bowtie_{MA} R_{A, \langle \rangle}^\xi. \quad (4)$$

#### 4.1.2. Multiple queues

Now we define a model with *several* queueing disciplines, i.e. there are several notional queues which sequential processes may enter during a life-cycle. This is a straightforward extension of the model structure used to build models with one queueing discipline. Assume that there are  $N$  queueing disciplines and for  $1 \leq i \leq N$ , let  $A_i$  be a set of action types such that for  $1 \leq i < j \leq N$ ,  $A_i \cap A_j = \emptyset$ ;  $A_* = \bigcup_{i=1}^N A_i$  and  $\xi_i$  a finite sequence of rates as before.

**Definition 7** (Extended queueing combinator).

$$Q_{\langle A_1, \xi_1, \dots, A_N, \xi_N \rangle}(S_1, \dots, S_n) \stackrel{\text{def}}{=} \left( \dots \left( \left( \prod_{i=1}^n S_{iA_* \rightarrow \top} \right) \boxtimes_{M_{A_1}} R_{A_1}^{\xi_1} \right) \dots \boxtimes_{M_{A_N}} R_{A_N}^{\xi_N} \right). \quad (5)$$

This notation is cumbersome, and so the shorthand  $Q_\chi(S_1, \dots, S_n)$  is used to represent

$$Q_{\langle A_1, \xi_1, \dots, A_N, \xi_N \rangle}(S_1, \dots, S_n).$$

#### 4.2. Mapping a PEPA queueing discipline model to a GSMP

To study the insensitivity of activities in queueing discipline models, this section provides a translation to a GSMP. Such a mapping was given by Hillston [16] for earlier work on examining insensitivity of PEPA models. In that paper, Hillston gives a mapping for a PEPA model as a configuration of *top-level* components, e.g., for a cooperation of two sequential components. The active elements of the GSMP are multisets of enabled activities. Such a cooperation 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

$$((\alpha, r_1).P_1 + (\gamma, r_2).P_2) \boxtimes_{\{\alpha, \beta\}} ((\alpha, s_1).Q_1 + (\beta, s_2).Q_2 + (\gamma, s_3).Q_3). \quad (6)$$

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

$$(\{(\gamma, r_2)\}, \{(\gamma, s_3)\}, \{(\alpha, t_1)\}) \text{ where } t_1 = \min(r_1, s_1). \quad (7)$$

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. Such a mapping is appropriate since the PEPA semantics uses a pre-emption with restart policy. The PEPA model structure studied in this paper is of a restricted form, and so the GSMP mapping provided is more limited, and in particular does not need to take account of arbitrary cooperation in PEPA models.

The approach used in this paper is based on the consideration of PEPA models of the form described in Section 4.1.2, i.e., a collection of concurrent sequential components with queueing disciplines. In this approach, an active element of the GSMP is constructed from a sequential component as a multiset of pairs, where each pair is an enabled activity and the PEPA derivative that results. Without a queueing discipline, only simple models need be considered; these consist of the unrestricted cooperation of sequential PEPA components. For example, consider the component given below:

$$((\alpha, r_1).P_1 + (\gamma, r_2).P_2) \parallel (\beta, s_1).Q_1. \quad (8)$$

Under the new mapping, this component would be represented by the GSMP state

$$(\{((\alpha, r_1), P_1), ((\gamma, r_2), P_2)\}, \{((\beta, s_1), Q_1)\}). \quad (9)$$

Since we wish to be able to uniquely determine the GSMP successor state resulting from each active element, we now include the derivatives as well as the activities in the active element. This means that if concurrent active elements arise from activities with the same name we can still distinguish between them.

The queueing discipline consists of a component, the arbiter process, 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 and Thomas in [17]. 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. Now a more formal definition is presented.

**Definition 8** (GSMP state). The GSMP state corresponding to derivative  $P' \equiv Q_\chi(S'_1, \dots, S'_n)$  is given by

$$\prod_{i=1}^n \{(a, Q_\chi(S''_i)) : Q_\chi(S'_i) \xrightarrow{a} Q_\chi(S''_i)\}. \quad (10)$$

Note that this is a conventional Cartesian product over sets. The GSMP state representation of a component  $P$  is denoted  $\hat{P}$ .

This definition gives rise to a GSMP with some simple properties. If we consider the derived form of the queueing discipline models, it is clear that each activity of  $Q_\chi(S_1, \dots, S_n)$  is either an independent activity of some  $S_i$  or a shared activity of  $S_i$  in cooperation with the appropriate arbiter process. Moreover, the completion of an activity will always result in the death of an active element, even if the active element contains more than one activity. Thus, the death of an active element corresponding to  $S_i$  for some  $i$  will not cause the death of any active elements for components  $S_j$ ,  $j \neq i$ . Since interest will be limited to studying when activities, and thus active elements, can be generally distributed, this property is important—it ensures that the remaining lifetime of an active element representing an activity will persist over a state change, unless the state change corresponds to the completion or disabling (by competition) of that activity. A further related property is that any transition which can be made by  $Q_\chi(S_1, \dots, S_n)$  can be inferred from a transition made by  $Q_\chi(S_i)$ , for some  $i$ .

Since this is an alternative model for a PEPA process, it is important that it preserves the performance properties of the original model too. The lifetime of an active element is defined next.

**Definition 9** (Active element lifetime). Let  $s$  be an active element present in state  $\hat{P}$ . The lifetime of  $s$  is exponentially distributed with mean

$$\left( \sum_{\{((\alpha, r), Q_\chi(S')) \in s\}} r \right)^{-1}. \quad (11)$$

The rate of departure from state  $\hat{P}$  is denoted by  $\hat{q}(\hat{P})$ , and given by  $\sum_{i=1}^n \sum_{(\alpha, r) \in \text{Act}(Q_\chi(S_i))} r$ .

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. When an active element  $s$  of a state  $\hat{P}$  dies, the probability that the next state is  $\hat{P}'$  is given by a probability distribution  $p(\hat{P}'; \hat{P}, s)$ . This is defined as follows.

**Definition 10** (GSMP next-state probability). Let  $P \equiv Q_\chi(S_1, \dots, S_n)$ , and  $P' \equiv Q_\chi(S_1, \dots, S'_j, \dots, S_n)$  such that  $P \rightarrow P'$ . Given that  $s_i$  is the active element that dies in state  $\hat{P}$ , the probability that the next state is  $\hat{P}'$  is given by

$$p(\hat{P}'; \hat{P}, s_i) = \mathbf{1}(j = i) \left( \frac{\sum_W r}{\sum_V r} \right), \quad (12)$$

where  $W = \{((\alpha, r), Q_\chi(S'_i)) \in s_i : Q_\chi(S_i) \xrightarrow{(\alpha, r)} Q_\chi(S'_i)\}$  and  $V = \{((\alpha, r), Q_\chi(S'_i)) \in s_i\}$ .

The probability that the next state is  $\hat{P}'$  is denoted by  $\hat{p}(\hat{P}, \hat{P}')$ , and is given by  $\sum_{i=1}^n p(\hat{P}'; \hat{P}, s_i)$ .

Although not concise, these definitions are intuitive. An active element  $s$  of a state  $\hat{P}$  is a multiset of pairs of activities and derivatives. The activities constitute a particular subset of the enabled activities of  $P$ . These are used to specify the mean lifetime of  $s$ . The derivatives capture how the GSMP can evolve on the death of  $s$ , and are used to define the successor state probability distribution.

Now some simple results about the GSMP model are illustrated. The GSMP model is a faithful representation of the original PEPA model meaning in particular that performance properties are preserved.

**Lemma 4.1.** *Let  $P \equiv Q_\chi(S_1, \dots, S_n)$ . Then  $q(P) = \hat{q}(\hat{P})$ .*

**Lemma 4.2.** *Let  $P \equiv Q_\chi(S_1, \dots, S_n)$  and  $P \xrightarrow{(\alpha, r)} P'$  (therefore,  $P' \equiv Q_\chi(S_1, \dots, S'_j, \dots, S_n)$  for some  $j$ ). Then  $p(P, P') = \hat{p}(\hat{P}, \hat{P}')$ .*

These results are almost self-evident by deliberate construction of the GSMP model, but they illustrate that the GSMP performance model will faithfully represent the PEPA model.

For a PEPA model, this GSMP mapping would result in active elements with exponential lifetimes. However, in particular circumstances, generally distributed active elements will be allowed, and it is shown that these do not affect the steady-state solution. At the process algebra level, the modeller would wish to use general distributions when describing the duration of an activity. In an insensitive model, a generally distributed active element representing a set of (albeit mutually disabling) activities results in a next state probability which is well-defined [10], but which is in general difficult to calculate. However, 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, an analogous form of Matthes' theorem can be stated for PEPA models.

**Corollary 4.1.** *The following two statements are equivalent:*

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

Comparing this with Corollary 3.1, it can be seen that the condition for the flux out of a state looks to be different. The explanation for this is that each state of the GSMP for a queueing discipline model is constructed as a multiset of pairs of activities and PEPA terms, and the theory of insensitivity deals with the death of active elements of the GSMP. If an active element of a queueing discipline model 'dies', one of the activities that makes up the active element completes, and all the others are disabled.

#### 4.3. Queueing discipline structure, insensitivity and product form

Given the process algebra definitions of the queueing discipline framework above, we can now formally analyse the structure of the models it produces. The aim of the work in this paper is to examine conditions



for insensitivity of activities in process algebra models. The first result gives insight into why the structure of models built with the new combinator is related to insensitivity; we show that the solution of such models is a product form over both the solutions to the individual sequential components, and the current queue lengths.

Let there be a size- $n$  set of sequential PEPA components  $S_i$ , i.e., a set of components each without the syntactic cooperation or hiding operators. Now assume a model of the form  $Q_\chi(S_1, \dots, S_n)$  where  $\chi = \langle A_1, \xi_1, \dots, A_N, \xi_N \rangle$ . Therefore,  $A_j$  defines the set of activities by which components  $S_i$  may leave the  $j$ th queue, and also implicitly defines which components  $S_i$  may enter the queue. Recall from Section 4.1 that we assume that for any  $0 \leq i < j \leq N$ , the arbiter cooperation sets of  $S_i$  and  $S_j$  are pairwise disjoint. This condition ensures that at each point in the evolution of each  $S_i$ , it may enter either one queue only, or none at all, and that there is no confusion over which process will leave a queue. Furthermore, we assume that if a sequential process,  $S_i$ , enables a queue activity, then it has no potential to perform another activity; effectively, it must wait its turn in the queue. Currently, processes may not leave a queue such that they move directly into another queue, i.e., for any  $0 \leq i < j \leq N$ ,  $e_{A_i} \cap A_j = \emptyset$ . This restriction is only in place for simplicity, and indeed it is conjectured that the restriction is unnecessary. However, the extension remains as future work.

Let  $\Theta$  represent the set of processes currently queueing. At any point, a number of processes may be queued due to the  $j$ th queue. Let  $q_j$  represent the current queue length and  $d_j$  the maximum length (i.e.  $d_j = |\xi_j|$ , recalling that  $\xi_j$  is a sequence), such that  $0 \leq q_j \leq d_j$ . If  $S_i$  is currently at the front of the  $j$ th queue, it may perform its queue activity  $a$  with action type in  $A_j$  at a rate  $\kappa_{jq_i}$ . The notation is used with respect to the state currently being considered. For convenience,  $\kappa_{j0}$  is defined to be 0 for each  $j$ .

Finally, each sequential component is a PEPA process in its own right, and as such, its underlying stochastic process has a steady-state probability distribution. Let  $\pi_i(\cdot)$  denote the steady-state probability distribution of  $S_i$ ; then if  $S'_i \in ds(S_i)$ , the long-run probability of being present in  $S'_i$  is given by  $\pi_i(S'_i)$ . With all notation in place, the theorem can now be stated:

**Theorem 1.** *Let  $P \equiv Q_\chi(S_1, \dots, S_n)$ . The steady-state solution of  $P$  is given by*

$$\pi(P) = \frac{1}{G} \prod_{i=1}^n \pi_i(S_i) \prod_{i=1}^N \prod_{j=q_i+1}^{d_i} \kappa_{ij} \prod_{\substack{i=1 \\ S_i \in \Theta}} \sum_{(\alpha, r) \in \text{Act}(S_i)} r,$$

where  $1/G$  is a normalising constant.

Due to space limitations, the proof for the result above is omitted though it can be found in full in [18]. The approach taken was standard, that being to propose a general form of solution, and then show that it satisfies the global balance equations of the stochastic process underlying  $P$ . Now that we have exhibited the general form of the steady-state solution for models built with the queueing combinator, we can easily study further sets of balance equations. The next result relies on *insensitivity balance* equations for particular activities present in queueing combinator models.

**Theorem 2.** *Given a PEPA model  $P \equiv Q_\chi(S_1, \dots, S_n)$ ,  $P$  is insensitive to each activity in  $\vec{\text{Act}}(S_i)$ , for  $1 \leq i \leq n$ , with the exception of those activities in the set  $(\bigcup_{i=1}^n e_{A_*}(S_i)) \cup A_*$ , where  $A_* = \bigcup_{i=1}^N A_i$ .*

Again, the proof is omitted but can be found in [18]. It proceeds by constructing balance equations, namely the insensitivity balance equations for the non-queue activities. These equations are 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. [Theorem 1](#) guarantees the form of the solution for our queueing models, and we have shown that this is also a solution to the insensitivity balance equations for states which do not enable queue activities. Therefore, 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.

**Remark.** The application of [Theorem 2](#) raises some technical difficulties. The aim of the work is to consider the insensitivity of PEPA activities, but when a particular derivative enables more than one activity, in the context of a choice, it is difficult in general to construct the time-averaged distribution of the minimum of the distributions of these activities. This means that if a derivative enables more than one activity, then these activities may still be insensitive to the forms of their distributions, but calculating the parameters of the corresponding Markov process may be difficult. Therefore, for practical purposes, the result can be considered to be limited to activities which are not in competition.

## 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, e.g., user queries. Models of transaction processing systems abound in the literature; we refer the interested reader to [19] 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,

$$\begin{aligned}
TxnC_i &\stackrel{\text{def}}{=} (\text{think}_i, t_i).Txn_i \\
Txn_i &\stackrel{\text{def}}{=} \sum_{j=1}^{N_i} (\text{work}, w \times p_j).(\text{writelock}_{ij}, r_{ij}).(\text{commit}_{ij}, \top).Loop_i \\
&\quad + (\text{work}, r).(\text{dirtyread}, d).(\text{accessdb}, rl).Loop_i \\
Loop_i &\stackrel{\text{def}}{=} (\text{nexttxn}_i, r \times a_i).TxnC_i \\
&\quad + (\text{nexttxn}_i, (1 - r) \times a_i).Txn_i
\end{aligned}$$

Fig. 1. PEPA model of a transaction class.

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 Fig. 1 which represents a class of transactions; i.e., 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  $i$ th 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 [19]. Written changes are committed via the  $(\text{commit}_{ij}, \top)$  activity. This is passive because intuitively, the database manager will be the determining factor in how long such commits will take.

Assuming that the transaction requires exclusive locks (such that other transactions must wait to access the locked objects), the additional factor  $p_j$  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 [19], 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 (probability  $r$ )—if so, another is generated, and if not, we await another job from the  $i$ th user.

The next requirement is to model the central database itself. This model allows access to  $M$  classes of write-locked database objects, and enforces a two-phase locking concurrency control scheme. Given the above definition, modelling the transaction processing system using a queueing discipline is straightforward. First, we construct the types of the queue activities for each lock manager:

$$\mathcal{A}_i = \{\text{writelock}_{ij} : 1 \leq i \leq M\}, \quad 1 \leq j \leq n.$$

Secondly, we require that the database controls how long a commit takes to complete. We have the freedom to allow this to vary depending on how many transactions are currently waiting to commit their changes. Let

$$\xi_j = \langle \mu_{j0}, \mu_{j1}, \dots, \mu_{jM} \rangle \quad \text{for } 1 \leq j \leq n,$$

where  $\xi_j$  is a vector of rates; a commit on the  $j$ th database lock is exponentially distributed with rate  $\mu_{ji}$  when  $i$  transactions are currently waiting to commit their changes. The needs of the modeller may be such that only fixed rates are required independent of the number of transactions queued, i.e.  $\mu_{ji} = \mu_{jk}$  for all  $i, k$ .

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 leads to a representation of TPS as

$$\text{TPS} \stackrel{\text{def}}{=} Q_\chi(\text{TxnC}_1, \dots, \text{TxnC}_M) \text{ where } \chi = \langle A_1, \xi_1, \dots, A_N, \xi_N \rangle. \quad (13)$$

There are elements of this model which it may not be appropriate to model using an exponential distribution. One such activity may be the think time of each user, i.e.  $\text{think}_i$ ,  $1 \leq i \leq n$ , e.g. if the ‘user’ in one case is a computer program generating transactions at a constant rate. In this case it is possible to choose a generally distributed duration of activity, since the TPS has been modelled by making use of the new queuing combinator. Rate restriction is in place, since each lock manager controls the rate at which waiting transactions may commit changes to the database, i.e.  $\text{commit}_{kj}$  activities happen at a constant rate. Therefore, the rate is fixed independent of the number of blocked transactions, which is actually stricter than the queue-length dependent rates required. This is not unreasonable, since access times will depend on the database objects, not on the transactions themselves. [Theorem 2](#) allows us to model the transaction processing system in PEPA, yet with no assumption of the stochastic nature of each  $\text{think}_i$  other than each mean.

## 6. Related and further work

Ours is not the first syntactic characterisation of criteria for SPA models which generate product form solutions. Over recent years several such criteria have appeared in the literature [[17,20,21](#)]. Given the clear compositional structure of SPA models, it is a natural goal to seek to identify those cases when the model structure can be exploited for efficient decomposed solution. In all such cases the models are restricted in form, and the previous approach has been to formulate these restrictions in terms of syntactic conditions against which any given model can be checked. Our approach in this paper is distinct in that we provide a new combinator which *guarantees* that models satisfy the necessary restriction. Below, we briefly survey the previous work on product form SPA and discuss how each of the captured criteria relates to our class of models.

An initial study of SPA models giving rise to reversible Markov processes was presented by Bhabuta et al. in [[22](#)]. This paper largely considered the problem at the level of the underlying state space. In [[20](#)], Hillston and Thomas, identify syntactic conditions which an SPA model must satisfy in order for the underlying process to be reversible. First, a basic class of sequential components which give rise to reversible structures are identified. Then, assuming that a known class of reversible SPA components exist, the authors investigate under what circumstances the conditions for reversibility will be preserved if reversible components are composed using the combinators of the SPA. Fundamental to the basic class of reversible sequential components is the notion of a *reverse pair*. A pair of action types  $(\alpha, -\alpha)$  form a reverse pair if, in any state, any  $\alpha$  transition leads to a state in which a  $-\alpha$  transition leading back to the original state. This is an abstraction of the notions of *arrival* and *departure* in a queue. In [[20](#)] various canonical forms for sequential reversible components are described as well as conditions under which reversible components can be composed without losing the reversibility property. These criteria rely on

a *resource- or server-centric* view of the system, in which system behaviour is captured by representing the local states of the resources of the system. As we will explain below, our class of models present a *customer-centric* view of systems. Syntactically these two views are quite distinct.

Harrison and Hillston [23] explore SPA structure which give rise to *quasi-reversible* (QR) models. Again, a basic class of QR sequential components is identified and then conditions under which they can interact whilst preserving the QR property are investigated. Again, a resource-centric view of the system is assumed, making the models very different in appearance to those considered in this paper. The restrictions placed on the form of interaction are strong. Not surprisingly, given the origins in queueing networks, the form of admissible interaction is a *flow cooperation*. This means that the ‘positive’ half of a reverse pair in one component is carried out in cooperation with the ‘negative’ half of a reverse pair in another. The ‘positive’ actions correspond to the input process in the definition of quasi-reversibility, while the ‘negative’ correspond to the output process.

In [21], Sereno derives product form criteria for SPA models based on earlier work on product form criteria for SPN [24–26]. He defines a Markov process in which the states correspond to the actions of the SPA model, all of which are assumed to be distinct. This is called the *routing process*. The conditions for this process to exist place severe restrictions on the forms of synchronisation and resource contention which can be represented in the model. For each action, the sequential components which participate in the action must, as a result enter local states that collectively enable another action; moreover, they must be the only participants necessary to complete the other action. Our insensitive structure, generated by the queueing discipline PEPA models, are not subject to these restrictions. In particular, the action which removes a component from a queue, involving the arbiter in addition to the component, will result in the enabling of two or more distinct actions, one by the arbiter and one by the component.

In [27], Boucherie describes a class of Markov processes which are formed using a simple exclusion mechanism for product processes. In [17], Hillston and Thomas give a syntactic characterisation of these processes as PEPA models. The models they describe are perhaps the closest to the models presented in this paper. As in this paper, the models considered take a customer-centric view of the system modelled. In both cases the characterised models represent the competition, of otherwise independent components, over resources. However, the exclusion mechanism at the basis of Boucherie’s class of processes imposes a powerful restriction on the behaviour of competing components. If two components compete over a resource, and one of them is currently holding the resource, then the other cannot make any state change, no matter where it is in its state space. This restriction is not necessary, and is not imposed, for the models presented in this paper. Although two components may be subject to the same queueing discipline, the fact that one of them is currently ‘using the resource’ (enabling a shared activity with the appropriate arbiter) does not prevent the other from enabling its own activities outside the queue.

### 6.1. The new combinator and BCMP queues

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 [28]. 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-pre-emptive resume (LCFS-PR).

and which also contain multi-class traffic having 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, which helps to explain the product form result we have obtained. To illustrate this, we consider a model built using our combinator, and show how it can be interpreted as a closed queueing network.

Let  $\chi = \langle A_1, \xi_1, \dots, A_N, \xi_N \rangle$ . By definition, a model  $Q_\chi(S_1, \dots, S_n)$  is of the form

$$\left( \dots \left( \left( \prod_{i=1}^n S_{iA_* \rightarrow \top} \right) \boxtimes_{M_1} R_1^{\xi_1} \right) \dots \boxtimes_{M_N} R_N^{\xi_N} \right),$$

where  $A_* = \bigcup_{i=1}^N A_i$ . We assume that for  $1 \leq i \leq N$ , if both  $(\alpha, r) \in \vec{Act}(S_i)$  and  $(\alpha, s) \in \vec{Act}(S_j)$ , then  $r = s$ . Further, we assume that for any  $i \neq j$ ,  $\vec{Act}(S_i) \cap \vec{Act}(S_j) = \emptyset$ . Each  $S_i$  has the capacity to perform individual activities, except where specified by the sets  $A_j$ ,  $1 \leq j \leq N$ . This individual behaviour can be separated and captured in the following way. Let  $B_i = \vec{Act}(S_i) \setminus \{(\alpha, r) : \alpha \in A_*\}$ . Now consider the following PEPA definition:

$$IS_i \stackrel{\text{def}}{=} \sum_{(\alpha, r) \in B_i} (\alpha, r).IS_i. \quad (14)$$

Each sequential component  $S_i$  is now altered such that each of its activities is passive. Let  $C_i = \vec{Act}(S_i) \setminus A_*$ ; put together with  $IS_i$ , we recover a sequential component isomorphic to  $S_i$ :

$$S_i = S_{iA_* \rightarrow \top} \boxtimes_{C_i} IS_i. \quad (15)$$

This construction explicitly separates the sequential component from its timing behaviour, which is now specified by  $IS_i$ . We suggestively rename each arbiter process  $R_i^{\xi_i}$  to  $FCFS_i^{\xi_i}$ , meaning the entire model can now be constructed as

$$Q_\chi(S_1, \dots, S_n) = \left( \dots \left( \left( \prod_{i=1}^n S_{iA_* \rightarrow \top} \boxtimes_{\mathcal{X}(S_i) \setminus A_i} IS_i \right) \boxtimes_{M_1} FCFS_1^{\xi_1} \right) \dots \boxtimes_{M_N} FCFS_N^{\xi_N} \right). \quad (16)$$

This can be understood as a queueing network, with a marked separation in the modelling of each of its features.

The roles of each  $IS_i$  are intuitive—any behaviour of  $S_i$  that was originally individual to  $S_i$  is interpreted as the passage of customer  $i$  through an *infinite server* node. Customer  $i$  is held up for a (stochastically) fixed delay, and there is no queueing.  $FCFS_j^{\xi_j}$  is a shared resource, representing a *first come first served* node. Depending on the action types enabled by  $S_i$ , as it models customer  $i$ 's passage through the network, customer  $i$  may be forced to queue at this node for service; of course it may have to wait its turn behind other customers which arrived prior to it.  $\xi_j$  specifies a vector of rates, and  $FCFS_j^{\xi_j}$  determines the rate at which a customer receives service depending on how many customers are currently queued. The FCFS nodes mark the points in the queueing network where customers meet.

This provides a PEPA translation of a subset of possible BCMP queueing networks. Our translation is *customer-centric*, in that we construct the network from the point of view of the passage of classes



of customer. This is in contrast to the traditional view of a queueing network which identifies states with vectors of natural numbers, denoting the number of customers present at each service centre. We have only considered *closed* networks; there are no external arrivals. While external arrivals could be modelled in PEPA, our translation relies on the fact that the queues may only hold a bounded number of customers. Furthermore, we have not discussed the translation of BCMP queues with LCFS-PR or PS nodes.

## 7. 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 [16] was motivated by a desire to remove behaviourally inert sequences of silent  $\tau$  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  $\tau$  activity. Use of the work in this paper for aggregation would also be a legitimate pursuit; e.g., 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

The authors would like to express their gratitude to the referees for their detailed and extensive comments which greatly improved the paper. This work is supported by the Engineering and Physical Sciences Research Council via the COMPA project (G/L10215).

## References

- [1] J. Hillston, A Compositional Approach to Performance Modelling, Cambridge University Press, Cambridge, 1996.
- [2] N. Götz, U. Herzog, M. Rettelbach, Multiprocessor and distributed system design: the integration of functional specification and performance analysis using stochastic process algebras, in: Proceedings of the Performance'93, 1993.

- [3] M. Bernardo, R. Gorrieri, A tutorial on EMPA: a theory of concurrent processes with nondeterminism, priorities, probabilities and time, *Theor. Comput. Sci.* 202 (1–2) (1998) 1–54 (Tutorial).
- [4] H. Hermans, Interactive Markov chains, Ph.D. Thesis, University of Erlangen-Nürnberg, Germany, 1998.
- [5] K. Matthes, Zur Theorie der Bedienungsprozesse, in: *Trans. 3rd Prague Conference Inf. Th., Stat. Dec. Fns. Rand. Proc.*, 1962, pp. 513–528.
- [6] M.K. Molloy, Performance analysis using stochastic Petri nets, *IEEE Trans. Comput. C-31* (1982) 913–917.
- [7] J.B. Dugan, K.S. Trivedi, R. Geist, V. Nicola, Extended stochastic Petri nets: applications and analysis, Technical Report DUKE-TR-1984-16, Department of Computer Science, Duke University, 1984.
- [8] W. Henderson, D. Lucic, Applications of generalised semi-Markov processes to stochastic Petri nets, in: *Performance of Distributed and Parallel Systems, Proc. IFIP TC7/WG 7.3 International Seminar on Performance of Distributed and Parallel Systems*, Kyoto, Japan, 7–9 December 1988, North Holland, Amsterdam, 1989.
- [9] M. Ajmone Marsan, C. Chiola, On Petri nets with deterministic and exponentially distributed firing times, in: *Lecture Notes in Computer Science “Advances in Petri Nets”*, Vol. 266, Springer, Berlin, 1987, 132–145.
- [10] M. Rumsewicz, W. Henderson, Insensitivity with age-dependent routing, *Adv. Appl. Prob.* 21 (1989) 398–408.
- [11] P.G. Harrison, B. Strulo, Stochastic process algebra for discrete event simulation, *Esprit Basic Research Series*, Springer, Berlin, 1995, pp. 18–37.
- [12] M. Bravetti, M. Bernardo, R. Gorrieri, Towards performance evaluation with general distributions in process algebras, *Lect. Notes Comput. Sci.* 1466 (1998) 405–422.
- [13] M. Bravetti, R. Gorrieri, The theory of interactive generalized semi-Markov processes, *Theor. Comput. Sci.* 282 (1) (2002) 5–32.
- [14] P.R. D’Argenio, J.-P. Katoen, E. Brinksma, A stochastic automata model and its algebraic approach, in: *Proceedings of the Fifth Process Algebras and Performance Modelling Workshop*, 1997, pp. 1–16.
- [15] A. El-Rayes, M. Kwiatkowska, G. Norman, Solving infinite stochastic process algebra models through matrix geometric methods, in: J. Hillston, M. Silva (Eds.), *Proceedings of the Seventh Workshop on Process Algebras and Performance Modelling*, Prensas Universitarias de Zaragoza, September 1999, pp. 41–62.
- [16] J. Hillston, PEPA: performance enhanced process algebra, Technical Report CSR-24-93, Department of Computer Science, The University of Edinburgh, 1993.
- [17] J. Hillston, N. Thomas, Product form solution for a class of PEPA models, *Perform. Eval.* 35 (1999) 171–192.
- [18] G. Clark, Techniques for the construction and analysis of algebraic performance models, Ph.D. Thesis, The University of Edinburgh, 2000.
- [19] A. Thomasian, Concurrency control: methods, performance, and analysis, *ACM Comput. Sur.* 30 (1) (1998) 70–119.
- [20] J. Hillston, N. Thomas, A syntactical analysis of reversible PEPA models, in: C. Priami (Ed.), *Proceedings of the Sixth Annual Workshop on Process Algebra and Performance Modelling*, Nice, France, September 1998, Università degli studi di Verona, pp. 37–49.
- [21] M. Sereno, Towards a product form solution for stochastic process algebras, *Comput. J.* 38 (7) (1995) 622–632.
- [22] M. Bhabuta, P.G. Harrison, K. Kanani, Detecting reversibility in Markovian process algebras, in: *Proceedings of the 11th UK Performance Engineering Workshop for Computer and Telecommunication Systems*, Springer, Berlin, 1995.
- [23] P. Harrison, J. Hillston, Exploiting quasi-reversible structures in Markovian process algebra models, *Comput. J.* 38 (7) (1995) 510–520.
- [24] W. Henderson, D. Lucic, P.G. Taylor, A net level performance analysis of stochastic Petri nets, *J. Aust. Math. Soc., Ser. B* 31 (1989) 176–187.
- [25] W. Henderson, P.G. Taylor, Embedded processes in stochastic Petri nets, *IEEE Trans. Software Eng.* 17 (2) (1991) 108–116.
- [26] R.J. Boucherie, M. Sereno, On the traffic equations of batch routing queueing networks and stochastic Petri nets, Technical Report, European Research Consortium for Informatics and Mathematics, 1994.
- [27] R.J. Boucherie, A characterisation of independence for competing Markov chains with applications to stochastic Petri nets, *IEEE Trans. Software Eng.* 20 (7) (1994) 536–544.
- [28] F. Baskett, K.M. Chandy, R.R. Muntz, F.G. Palacios, Open, closed and mixed networks of queues with different classes of customers, *J. ACM* 22 (2) (1975) 248–260.



**Graham Clark** is a software engineer at Citrix Systems Inc. in Columbia, MD. He received his PhD degree in Computer Science from the LFCS at the University of Edinburgh, Scotland. Graham's research focussed on extending the usefulness of stochastic process algebras, and he recently developed support for PEPA within the Möbius tool at the PERFORM group of the University of Illinois at Urbana–Champaign.



**Jane Hillston** received the BA and MSc degrees in Mathematics from the University of York and Lehigh University, respectively. After a brief period working in industry, she joined the Department of Computer Science at the University of Edinburgh, as a research assistant in 1989. She received the PhD degree in computer science from that university in 1994. In 1995 she became lecturer, and in 2001 a reader, in computer science. She is a member of the Laboratory for Foundations of Computer Science. Her principal research interests are in the use of process algebras to model computer systems and the investigation of issues of compositionality with respect to Markov processes.