Product Form Solution for a class of PEPA Models

Jane Hillston and Nigel Thomas
Department of Computer Science
The University of Edinburgh
United Kingdom
{jeh,nat}@dcs.ed.ac.uk

Abstract

The advantages of the compositional structure within the Markovian process algebra PEPA for model construction and simplification have already been demonstrated. In this paper we show that for some PEPA models this structure may also be used to advantage during the solution of the model. Several papers offering product form solutions of stochastic Petri nets have been published during the last ten years. In a recent paper [2], Boucherie showed that these solutions were a special case of a simple exclusion mechanism for the product process of a collection of Markov chains. The results presented in this paper take advantage of his observation. In particular we show that PEPA models that generate such processes may be readily identified and show how the product form solution may be obtained. Although developed here in the context of PEPA the results presented can be easily generalised to any of the other stochastic process algebra languages.

1. Introduction

The use of structured modelling paradigms for performance modelling, based on Markov processes, has been advocated by several authors for many years, and several such approaches have been developed and exploited [3, 6, 22]. Most of these approaches stress compositionality, i.e. the system is decomposed into subsystems that are smaller and more easily modelled. In particular several stochastic process algebras (SPAs) have been presented in recent years. These include PEPA [7, 12], MTIPP [9], and EMPA [1]. Initially the benefits of the compositional structure within these languages have been investigated for model construction and model simplification [8, 13, 16, 21]. However the real strength of this approach will result from the exploitation of this structure to aid model solution.

Recently several preliminary results have been published which show that, at least for some particular cases, there is a clear relationship between decompositional techniques and the SPA model descriptions. For example, the exploitation of the structure inherent in SPA models for solution based on tensor algebra has been proposed in [4, 20], whilst the application of time scale decomposition techniques to SPA models is described in [15, 18], and investigations of structures within SPA models which give rise to product form equilibrium distributions are reported in [10, 23].

In this paper we present an alternative way to exploit the compositional structure of stochastic process algebra models during solution. The previous work on product form SPA models has centred on components of a particular structure which interact in a restricted way, preserving a form of independence between the components. Here we characterise models which represent the competition of otherwise independent processes over resources and identify the cases in which these models exhibit a product form solution. This facilitates a solution technique in which components of the model are solved in isolation, these partial solutions subsequently being combined to give a solution of the complete model. These results are presented in the context of the stochastic process algebra PEPA but can easily be generalised to any of the other stochastic process algebra languages.

The models which we consider here belong to the class of competing Markov processes identified by Boucherie in [2]. The advantage of characterising this class of models in PEPA is that by elevating the definition to a formally defined high-level modelling paradigm we can automatically detect these structures when they occur. The models presented in [2], even those presented as stochastic Petri nets, relied on the insight of the modeller to detect the product form structure. Moreover, in the case of the stochastic Petri net models, a non-standard state representation had to be used in order to eliminate the resource from the model representation. The PEPA models do not have this disadvantage since the resource may (indeed, must) be represented explicitly and subsequently eliminated from the state representation automatically.

The class of models we present here does not extend that introduced by Boucherie. However we hope that the characterisation in PEPA will allow these results to be better understood and to be of greater practical use. Moreover, we aim to ultimately use the class as the target for formally defined (machine-assisted) model simplification techniques.

2. PEPA

Process algebras are mathematical theories which model concurrent systems by their algebra and provide apparatus for reasoning about the structure and behaviour of the model. In classical process algebras, such as CCS [19], time is abstracted away—actions are assumed to be instantaneous and only relative ordering is represented—and choices are generally nondeterministic. If an exponentially distributed random variable is used to specify the duration of each action the process algebra may be used to represent a Markov process. This is the approach taken by stochastic process algebras such as PEPA. Specifications written in PEPA represent Markov processes and can be mapped to a continuous time Markov chain (CTMC) for analytic or numerical solution.

The basic elements of PEPA are *components* and *activities*, corresponding to *states* and *transitions* in the underlying Markov process. Each activity has an *action type* (or simply *type*). Activities which are private to the component in which they occur are represented by the distinguished action type, τ. The duration of each activity is represented by the parameter of the associated exponential distribution: the *activity rate* (or simply *rate*) of the activity. This parameter may be any positive real number, or the distinguished symbol $\top$ (read as *unspecified*). Thus each activity, a, is a pair (α, r) where α is the action type and r is the activity rate. We assume that there is a countable set of components, which we denote $\mathcal{C}$, and a countable set, $\mathcal{A}$, of all possible action types. We denote by $\mathcal{A}ct \subseteq \mathcal{A} \times \mathbb{R}^+$, the set of activities, where $\mathbb{R}^+$ is the set of positive real numbers together with the symbol $\top$.

2.1. Syntax and Informal Semantics

PEPA provides a small set of combinators. These allow expressions, or terms, to be constructed defining the behaviour of components, via the activities they undertake and the interactions between them. The combinators, together with their names and interpretations, are presented informally below.

Prefix: $(\alpha, r).P$ Prefix is the basic mechanism by which the behaviours of components are constructed. The component carries out activity (α, r) and subsequently behaves as component P.

Choice: $P + Q$ The component represents a system which may behave either as component P or as Q: all the current activities of both components are enabled. The first activity to complete, determined by a *race condition*, distinguishes one component, the other is discarded. The choice combinator represents competition between components.

Cooperation: $P \underset{L}{\bowtie} Q$ The components proceed independently with any activities whose types do not occur in the *cooperation set* L (*individual activities*). However, activities with action types in the set L require the simultaneous involvement of both components (*shared activities*). These activities are only enabled in $P \underset{L}{\bowtie} Q$ when they are enabled in both P and Q. Thus one component may become blocked, waiting for the other component to be ready to participate. The cooperation combinator associates to the left but brackets may also be used to clarify the meaning. When the set L is empty, we use the more concise notation $P \parallel Q$ to represent $P \underset{\emptyset}{\bowtie} Q$.

The published stochastic process algebras differ on how the rate of shared activities are defined [11]. In PEPA the shared activity occurs at the rate of the slowest participant. If an activity has an unspecified rate in a component, the component is *passive* with respect to that action type. This means that the component does not influence the rate at which any shared activity occurs. A model which contains a passive activity without a partner for cooperation is considered to be *incomplete*.

Hiding: P/L The component behaves as P except that any activities of types within the set L are *hidden*, i.e. such an activity exhibits the unknown type τ and the activity can be regarded as an internal delay by the component. Such an activity cannot be carried out in cooperation with any other component: the original action type of a hidden activity is no longer externally accessible, to an observer or to another component; the duration is unaffected.

Constant: $A \stackrel{\text{def}}{=} P$ Constants are components whose meaning is given by a defining equation: $A \stackrel{\text{def}}{=} P$ gives the constant A the behaviour of the component P. This is how we assign names to components (behaviours). There is no explicit recursion operator but components of infinite behaviour may be readily described using sets of mutually recursive defining equations.

The action types which the component P may next engage in are the *current action types* of P, a set denoted $\mathcal{A}(P)$. This set is defined inductively over the syntactic constructs of the language (see [12] for a formal definition). For example, $\mathcal{A}(P+Q) = \mathcal{A}(P) \cup \mathcal{A}(Q)$. The activities which the component P may next engage in are the *current activities* of P, a multiset denoted $\mathcal{A}ct(P)$. When the system is behaving as component P these are the activities which are enabled. Note that the dynamic behaviour of a component depends on the number of instances of each enabled activity and therefore we consider *multisets* of activities as opposed

to *sets* of action types. For any component P, the multiset $\mathcal{A}ct(P)$ is defined inductively over the structure of P, as for $\mathcal{A}(P)$ (see [12] for a formal definition).

It is also necessary to refer to the complete set of action types which are used within the complete behaviour of a component C, i.e. all the possible action types which may be witnessed as a component evolves. This set will be denoted $\vec{\mathcal{A}}(C)$. The *complete action type set* of a component C is

$$\vec{\mathcal{A}}(C) = \bigcup_{C' \in ds(C)} \mathcal{A}(C').$$

where $ds(C)$ is the set of (syntactic) process terms in the evolution of component C. Two components, C_1 and C_2, are said to be *distinct* if they have no action types in common, i.e. $\vec{\mathcal{A}}(C_1) \cap \vec{\mathcal{A}}(C_2) = \emptyset$.

2.2. Execution strategy

A race condition governs the dynamic behaviour of a model whenever more than one activity is enabled. This has the effect of replacing the non-deterministic branching of classical process algebra with probabilistic branching. The probability that a particular activity completes is given by the ratio of the activity rate to the sum of the activity rates of all the enabled activities. Any other activities which were simultaneously enabled will be interrupted or aborted. The memoryless property of the exponential distribution makes it unnecessary to record the remaining lifetime in either case.

2.3. Cyclic PEPA

Unlike the earlier papers [10, 23], the aim of this paper is to consider models which exhibit a product form solution over the components of the model, even though it may be necessary to find the steady state solution of those components by numerical solution. Thus it is important that we ensure that the components within the model, as well as the model itself, are finite and ergodic. Necessary (but not sufficient) conditions for the ergodicity of the Markov process in terms of the structure of the PEPA model have been identified and can be readily checked [12]. These conditions imply that the model must be a *cyclic* PEPA component.

A cyclic component is one in which behaviour may always be repeated—however the model evolves from this component it will always eventually return to this component and this set of behaviours. In particular for every choice, whichever alternative is chosen the model must eventually return to the point where the choice can be made again, possibly with a different outcome. Since syntactic terms are associated with states, if we consider the layering imposed on a component by cooperation combinators, choice combinators may only be introduced at the lowest level of a cyclic component. In other words, a component which involves a choice combinator may subsequently be used in a cooperation, but a component involving a cooperation may not be subsequently used in a choice.

This leads us to formally define the syntax of PEPA expressions in terms of *sequential components* S and *model components* P:

$$\begin{array}{rcl} P & ::= & S \mid P \underset{L}{\bowtie} P \mid P/L \\ S & ::= & (\alpha, r).S \mid S + S \mid A \end{array}$$

For the remainder of the paper we will assume that all the models which we consider are cyclic.

The "states" of a PEPA model as it evolves are the syntactic terms, or *derivatives*, which the model will go through. The *derivative set* of a PEPA component C is denoted $ds(C)$ and is the set of components which capture all the reachable states of the system. When a model component is defined it consists of one or more cooperating components, and these cooperating components will be apparent in every derivative of the model. The sequential components which are involved in the model, and the cooperation sets in operation between them, will remain static throughout the evolution of the model. Only the particular derivatives exhibited by each of the sequential components may change.

Let P be a model component comprising sequential components $S_1, S_2, \ldots S_K$. Then a *state vector* of the model component P as derivative P_i is the vector $(S_{1_i}, S_{2_i}, \ldots, S_{K_i})_P$ where $S_{k_i}, 1 \leq k \leq K$ is the current derivative of S_k in P_i. This can be regarded as analogous to the state representation of a queueing network which consists of a vector $(n_1, n_2, \ldots, n_K)$, where n_i denotes the number of customers currently at queue i.

A sequential component S_k is *redundant within the state vector representation* of a model component P if S_k is a sequential component of P and for all derivatives $P_i \in ds(P)$ given the current derivatives of the other sequential components $S_{j_i}, j \neq k$, the current derivative of S_k, S_{k_i}, can be inferred. If a sequential component is shown to be redundant within the state vector, a *reduced state vector* may be formed in which the derivatives of this component have been eliminated. A sequential component S_k is *redundant within a model component* P if S_k is a sequential component of P and the Markov process generated by P is isomorphic to the Markov process generated by P', where P' is the model obtained by eliminating S_k from P.

Note that redundancy within the state vector representation is distinct from redundancy within the model. A component which is redundant within the state vector representation may be essential for correct behaviour of the model, imposing some form of scheduling between the other components. In contrast a component which is redundant within the model has no effect on the behaviour of the model at

all. For the remainder of the paper we will use the term *redundant resource* to refer to resource components redundant within the state vector representation.

3. Product Form Solutions

Stochastic process algebras impose a formally-defined compositional structure on the underlying Markov chain and the exploitation of this structure for model simplification has already been demonstrated. In [13] it is shown that components of a model can be considered and simplified in isolation, thereby avoiding the computational effort required to consider the model as a whole. This technique can be applied to models which exceed the capabilities of the existing techniques. Even greater advantage is gained when the compositional structure can used during model solution, i.e. if the CTMCs corresponding to the components could be solved separately and their solutions combined to obtain a solution, exact or approximate, of the whole CTMC [10, 15, 23]. One class of CTMCs which are susceptible to such an efficient solution technique are those which exhibit a *product form* (PF) equilibrium distribution.

Consider a Markov process $X(t)$, whose state space $\mathcal{S}$ is of the form $\mathcal{S} = S_1 \times S_2$, i.e. each state $\boldsymbol{s} = (s_1, s_2)$ contains two pieces of information capturing different aspects of the current state. In general, these aspects may be related in many ways. When the process $X(t)$ exhibits a product form solution, i.e. $\pi(\boldsymbol{s}) = \pi_1(s_1) \times \pi_2(s_2)$, it indicates that these different aspects of the state description are independent.

For stochastic process algebras two approaches to identifying models which give rise to PF equilibrium distributions have appeared in the literature. It is clear that when a PEPA model consists of completely independent sequential components, i.e. $C \stackrel{\text{def}}{=} P \parallel Q$, the equilibrium distribution over the state vector representation will have a product form:

$$\pi(C_i) = \pi_P(P_j) \times \pi_Q(Q_k) \tag{3.1}$$

where $C_i \equiv P_j \parallel Q_k$, π_P and π_Q are the steady state distributions over the derivatives of P and Q respectively. In [10] an extension to this class of models is found based on the notion of quasi-reversibility which underlies product form in queueing networks. A weak form of interaction between components is allowed but components are restricted to have a particular form. In [23] the application of earlier results for SPN is explored within an SPA setting. The class of components which may be used within product form models is found to be restricted, although these restrictions are now expressed in terms of the actions of the model and how they are distributed within the components.

In this paper we aim to identify cases when the CTMC underlying a PEPA model has a product form equilibrium distribution and there is no restriction over the possible form of the components which behave independently other than that they are sequential. In these cases the probability of a given model derivative will be the product of the probabilities of the corresponding derivatives in the sequential components, possibly subject to a normalisation constant. The approach taken is analogous to the approach used by Lazar and Robertazzi with SPN [17], since the aim is to find a product form in terms of submodels (subnets or components respectively). CTMCs may still need to be solved numerically to find the equilibrium distribution of the submodels but these Markov processes will be smaller and can be tackled separately. Unlike the other approaches we maintain the restriction that the subcomponents we consider in the state representation are independent of each other. However we extend from the simple case represented in equation 3.1 above, by the introduction of a redundant resource which imposes indirect interaction between the components, but which may be eliminated from the state vector representation.

3.1. Boucherie's Framework

In [2], Boucherie aims to generalise the result of Lazar and Robertazzi [17] to show that it can be regarded as a special case of a simple exclusion mechanism for the product process of a collection of Markov processes. His framework consists of a set of Markov processes which must compete over resources. This competition means that there are certain areas of the state space of the product process which cannot be entered—these areas would correspond to two processes holding a resource at the same time. He identifies circumstances in which, despite this indirect form of interaction, the product process exhibits a product form equilibrium distribution over the permissible states, suitably renormalised.

More formally the framework can be described as follows: Let S_k and q_k be the state space and transition rates, respectively, of the kth Markov process in a collection, $1 \leq k \leq K$. Let $\mathcal{S} = S_1 \times \cdots \times S_K$ denote the state space of the *product process*. It is assumed that in each transition of the product process only the state in one dimension changes, i.e. in each transition of the product process only one of the underlying Markov processes changes its state. The transition rates of the product process in dimension k are then given by the transition rates of the individual process, q_k. In other words, the product process consists of K individual processes which do not directly interact in any way.

Competition between the processes over resources introduces an indirect form of interaction between them. Let us assume that there is a set I of *notional resources*. For technical reasons this set may also contain a non-resource,

possession of which will indicate that a process is working without a resource. Then the state space of each Markov process can be partitioned into states corresponding to resource use. Only one notional resource may be held at a time but this does not imply that only one actual resource is held since different notional resources may correspond to a resource held individually, and a resource held in conjunction with another. Competition over resources is then defined as follows:

Let I be an index set. For each k, let $A_{ki}, i \in I$, be a set of mutually exclusive sets such that $\emptyset \neq A_{ki} \subset S_k$ and $\bigcup_{i\in I} A_{ki} = S_k$, $k = 1, \ldots, K$. Markov process k uses resource i if the Markov process is in state $n_k \in A_{ki}$. Markov processes k_1 and k_2 compete over resource i if

$$\{n_{k_1}, n_{k_2} : n_{k_1} \in A_{k_1 i}, n_{k_2} \in A_{k_2 i}\} = \emptyset$$

Let $C_{ki} \subset \{1, \ldots, K\}$ be the Markov processes that compete over resource i with Markov process k.

For any process k, $1 \leq k \leq K$, if the current state is in the subset A_{ki} it signifies that the process is presently using the resource i and no other process j, such that $j \in C_{ki}$, can gain access to i, entering its subset of states A_{ji}. Thus the competition and the sets C_{ki} define areas of the state space of the product process which are inaccessible. The transition rates of the product process are defined in a way which ensures this exclusion.

The Markov process at state space $\mathcal{S} = \prod_{k=1}^{K} S_k$, with transition rates

$$q(n, n') = \sum_{k=1}^{K} q_k(n_k, n'_k) \prod_{\ell=1, \ell\neq k}^{K} \mathbf{1}(n_\ell = n'_\ell)\mathbf{1}(\forall i, n_\ell \in A_{\ell i} \rightarrow k \notin C_{\ell i})$$

where $n = (n_1, \ldots, n_K)$, $n' = (n'_1, \ldots, n'_K)$, is called the *product process* of the collection of Markov processes $1, \ldots, K$, competing over resources I. The indicator function $\mathbf{1}$ is such that the value of $\mathbf{1}$(*statement*) is 1 if the statement is true and 0 otherwise.

Observe that, as required, these transition rates imply that in each transition only one process can change its state, and that process k cannot progress if resource i is being used by process ℓ if $k \in C_{\ell i}$, and vice versa. In the paper Boucherie shows that this mechanism of pair-wise relations imposed on the individual processes of a product process can be used to model various types of competition.

Theorem 3.1 (Product-form distribution)
The product process of the collection of Markov processes $1, \ldots K$ competing over resources I has equilibrium distribution π at $\mathcal{S}$, defined as

$$\mathcal{S} = \prod_{k=1}^{K} S_k \setminus \left(\prod_{k=1}^{K} \prod_{i\in I} \prod_{j\in C_{ki}} A_{ki} \times A_{ji} \right)$$

given by

$$\pi(n) = B \prod_{k=1}^{K} \pi_k(n_k) \qquad n \in \mathcal{S} \tag{3.2}$$

where B is a normalising constant, determined by the exact form of $\mathcal{S}$, and $\pi_k(\cdot)$ is the steady state distribution of process S_k.

The result holds because each process can either operate independently of the other processes or it is blocked. For all $n \in \mathcal{S}$, if process ℓ is in state n_ℓ and $\ell \neq k$ then process k either carries out a transition independently because it is not in competition with ℓ ($\mathbf{1}$(if $i : n_\ell \in A_{\ell i}$ then $k \notin C_{\ell i}$) = 1) or process k is blocked because it needs to access the resource which ℓ occupies ($\mathbf{1}$(if $i : n_\ell \in A_{\ell i}$ then $k \notin C_{\ell i}$) = 0). In either case process k will satisfy its own global balance equations: these equations are trivially satisfied when the process is stopped and are also true when the process is operating independently. It appears that the exclusion principle incorporated in the transition rates of the product process amounts to a redefinition of the product process such that the Markov processes in the collection operate as if they are independent.

Boucherie shows that the theorem can be generalised if we consider the case when each of the processes $1, \ldots, K$ is composed of several locally balanced Markov processes. Assume that the transition rates of Markov process k can be separated into T_k parts, labelled $t = 1, \ldots, T_k$. For $n_k, n'_k \in S_k$ we define the separated transition rates as $q_k^t(n_k, n'_k)$, $t = 1, \ldots T_k$, such that

$$q_k(n_k, n'_k) = \sum_{t=1}^{T_k} q_k^t(n_k, n'_k) \tag{3.3}$$

The Markov process k is said to be *locally balanced* with respect to this separation if the equilibrium distribution π_k satisfies

$$\sum_{n'_k \in S_k} \left\{ \pi_k(n_k) q_k^t(n_k, n'_k) - \pi_k(n'_k) q_k^t(n'_k, n_k) \right\} = 0 \qquad t = 1, \ldots, T_k.$$

Each process (k, t), $1 \leq k \leq K$, $1 \leq t \leq T_k$, is a Markov process in its own right. Thus for each k there may be several subprocesses $(k, t), 1 \leq t \leq T_k$, which all operate over the same state space S_k. Note that they do not necessarily all use the whole state space. We assume that for any k, subprocesses (k, t_i) and (k, t_j) do not compete over any resources. However, we extend the notion of competition between processes in the natural way: for each (k, t), let $A_{(k,t)i}$ be mutually exclusive sets such that $A_{(k,t)i} \subset S_k$, $\bigcup_{i\in I} A_{(k,t)i} \subseteq S_k$ and $\bigcup_{t=1}^{T_k} \bigcup_{i\in I} A_{(k,t)i} = S_k$. Similarly, let $C_{(k,t)i} \subseteq \{(k, t) : 1 \leq k \leq K, 1 \leq t \leq T_k\}$

be the set of Markov processes which compete with process (k,t) over resource i, i.e. if $(k',t') \in C_{(k,t)i}$ then $\{n_k \in A_{(k,t)i}, n'_k \in A_{(k',t')i}\} = \emptyset$.

Theorem 3.2 (Product process with local balance) *Assume that each Markov process in the collection $k = 1,\ldots,K$ satisfies local balance with respect to the separation 3.3. The Markov process at state space*

$$\mathcal{S} = \prod_{k=1}^{K} S_k \setminus \left(\prod_{k=1}^{K} \prod_{i \in I} \prod_{t=1}^{T_k} \prod_{(j,s) \in C_{(k,t)i}} A_{(k,t)i} \times A_{(j,s)i} \right)$$

with transition rates

$$q(n,n') = \sum_{k=1}^{K} \sum_{t=1}^{T_k} q_k^t(n_k, n'_k) \prod_{\ell=1, \ell \neq k}^{K} \mathbf{1}(n_\ell = n'_\ell) \prod_{p=1}^{T_\ell} \mathbf{1}(\forall i, n_\ell \in A_{(\ell,p)i} \rightarrow (k,t) \notin C_{(\ell,p)i})$$

has an equilibrium distribution Π at $\mathcal{S}$ given by

$$\Pi(n) = B \prod_{k=1}^{K} \pi_k(n_k)$$

where B is a normalising constant.

3.2. PEPA Components Competing over Resources

In order to apply Boucherie's framework we need to define the notions associated with a resource in PEPA. When a model component is constructed from sequential components via the cooperation combinator we can regard the model as being built up in layers or levels, each cooperation combining just two components. As suggested by the grammar those components may be sequential components or model components. Thus a sequential component may be within the scope of several cooperation sets because of the way the model has been constructed.

It is clearly useful to be able to establish which cooperation sets will affect the behaviour of the component. For example, in the component

$$X \stackrel{\text{def}}{=} (P \underset{L}{\bowtie} R) \underset{K}{\bowtie} (S \underset{N}{\bowtie} T)$$

the subcomponent R can act independently on any action types in the set N which do not occur in K or L, but must have the cooperation of other subcomponents to achieve actions in the set $L \cup K$, whereas the subcomponent S can act independently on any action types in the set $L \setminus (K \cup N)$, but must have the cooperation of other subcomponents to achieve actions in the set $K \cup N$. In the component

$$Y \stackrel{\text{def}}{=} (P \underset{L}{\bowtie} R) \parallel (S \underset{N}{\bowtie} T)$$

R must cooperate for actions in the set L but may act independently for all actions outside this set, and S must cooperate for actions in the set N but may act independently for all actions outside this set.

In the following we will formalise this idea. The *interface* of a sequential component within a component model is then defined to be the union of all the cooperation sets whose scope includes the component R.

Definition 3.1 (Interface) *For any sequential component R within a model component C (i.e. $R \prec C$) the* interface *of R within C, denoted $\mathcal{I}(C :: R)$, is the set of action types on which R is required to cooperate. It is defined in terms of the subsidiary function $\mathcal{I}'$: $\mathcal{I}(P :: R) = \mathcal{I}'(P :: R, \emptyset)$, where $\mathcal{I}'$ is defined as follows*

1. $\mathcal{I}'(R :: R, S) = S$
2. $\mathcal{I}'(P \underset{L}{\bowtie} Q :: R, S) = S' \cup S''$ *if* *a)* $\mathcal{I}'(P :: R, S \cup L) = S'$ *b)* $\mathcal{I}'(Q :: R, S \cup L) = S''$
3. $\mathcal{I}'(P/L :: R, S) = \mathcal{I}'(P :: R, S \setminus L)$
4. $\mathcal{I}'(A :: R, S) = \mathcal{I}(P :: R, S)$ *if* $A \stackrel{\text{def}}{=} P$
5. $\mathcal{I}'(R' :: R, S) = \emptyset$ *if* $R \not\prec R'$.

When all the possible actions of a sequential component are constrained by its interface the component is never free to act independently; it must cooperate with other components to complete *any* action. Such a component can be viewed as being subservient to the rest of the model, and is called a *resource component*.

Definition 3.2 (Resource Components) *A sequential component R in a model C is a* resource component *if there is only one instance of R within C and the complete action type set of R is a subset of its interface within C, i.e.*

$$\vec{\mathcal{A}}(R) \subseteq \mathcal{I}(C :: R)$$

A resource component is clearly closely related to the notion of redundancy. It is only possible for a sequential component to be redundant within a model while not being a resource component if all independent actions by the component do not result in a change of derivative.

A component S is termed a *basic component* if it is a cyclic component constructed using only prefix and each activity in the cycle is of a distinct type. A sequential component R is termed a *simple component* if

$$R \equiv S_1 + S_2 + \cdots + S_n$$

for some distinct basic components $S_1, S_2, \ldots S_n$, modified so that the last action of each loop returns to R and not S_i. If a resource component is simple it implies that it offers

alternative behaviours through its interface but once one of those behaviours is chosen (on the first action) the pattern of behaviour is set until the chosen cycle is completed and the choice is offered again.

To fit within Boucherie's framework we need to impose some restrictions on the syntactic form of the resource component, and its interaction with the rest of the model, in order to ensure that the exclusion property will hold.

Definition 3.3 (Guarding Resource)
A resource component, R, in the model component $S \underset{L}{\bowtie} R$ is a guarding resource *with respect to the sequential component S if for each α_i such that $S \equiv \sum(\alpha_i, r_i).S_i$ either $\alpha_i \in \mathcal{A}(R) \cap L$ or there is an activity sequence σ such that $S_i \xrightarrow{\sigma} S$ and $\sigma \in \mathcal{A}(R) = \emptyset$.*

Since S is a sequential component it is constructed using only prefix and choice, which means that in its initial state it either has one behaviour (prefix) or a set of alternative behaviours (choice). If it has a single behaviour, R is a guarding resource if S is required to cooperate with R to start its behaviour, or if the complete behaviour of S is independent of R, i.e. there is no cooperation between them. If S has a set of alternative behaviours represented by a choice of activities in the initial state, then R is a guarding resource if R's cooperation is required for the first activity or for any complete loop which S can start from this activity it can return to S without interaction with R. Note that this implies that the Markov processes underlying S can be separated into those requiring interaction with R and those not in such a way that the local balance property holds.

Definition 3.4 (Returning Resource)
A resource component, R, in the model component $S \underset{L}{\bowtie} R$ is a returning resource *with respect to the sequential component S if for each derivative $S' \underset{L}{\bowtie} R'$ such that $S' \equiv (\alpha, r).S$ then either $R' \equiv (\alpha, s).R$ for some s and $\alpha \in L$, or $R' \equiv R$ and there is a succession of derivatives from $S \underset{L}{\bowtie} R$ to $S' \underset{L}{\bowtie} R$ in which R does not change state.*

The condition on the returning resource is similar to that of the guarding resource but acting at the end of alternative behaviours instead of the start. For each possible loop of the sequential component, if it cooperates with the resource component at all it must do so on the last activity of the loop. In order to ensure Boucherie's exclusion mechanism we need resource components which are both guarding and returning resources with respect to the sequential components. We are now in the position to state the simplest class of PEPA models that satisfy Boucherie's condition.

Theorem 3.3
Let M be a model component,

$$M \equiv (S_1 \parallel S_2 \parallel \cdots \parallel S_K) \underset{L}{\bowtie} R$$

where each S_i is a sequential component and R is a simple component. Assume that the cooperation set L is such that R is a redundant resource within the state representation vector of M, and moreover, when we consider the model components $S_k \underset{L}{\bowtie} R$ for all $k, 1 \leq k \leq K$, R is a guarding and returning resource with respect to S_k. Then the steady state probability of any derivative M' which has state vector representation $(S'_1, \ldots, S'_K, R')$ is given by the product form

$$\Pi(S'_1, \ldots, S'_K, R') = B \prod_{k=1}^{K} \pi_k (S'_k \underset{L}{\bowtie} R'_k)$$

where B is the normalising constant, π_k is the steady state probability distribution of $S_k \underset{L}{\bowtie} R$ and R'_k is the derivative of R which can be inferred from S'_k, since R is redundant in the state vector representation of $S_k \underset{L}{\bowtie} R$.

The proof of Theorem 3.3 relies on showing that the Markov process underlying the model M satisfies Boucherie's conditions, i.e. it consists of the product of a number of competing Markov processes, such that only one competing process may change its state at a time and that they observe the exclusion property with respect to the resource. The mapping from the state representation of the PEPA model to the product process is straightforward. We associate one Markov process with each sequential component S_i in the model; the state space of the Markov process is simply the derivative set of the component. The PEPA semantics and the mutual independence of the sequential components ensures that only one product process will change state in any transition (using the reduced state vector representation). It is then not difficult show that the competition imposed by the PEPA semantics over the resource component does indeed provide the Boucherie exclusion (see [14]).

The restriction that the resource within the system is comprised of a single sequential resource component is unnecessarily strict. It is straightforward to make the result more general by considering a set of mutually independent resources over which the other components compete.

Corollary 3.1
Let M be a model component,

$$M \equiv (S_1 \parallel S_2 \parallel \cdots \parallel S_K) \underset{L}{\bowtie} R$$

as in Theorem 3.3, except that $R \equiv R_1 \parallel \cdots \parallel R_m$, and the R_i are distinct simple components, i.e. $\vec{\mathcal{A}}(R_i) \cap \vec{\mathcal{A}}(R_j) = \emptyset$ for all $i, j, 1 \leq i, j, \leq m$. Then the steady state probability of any derivative M which has state vector representation $(S'_1, \ldots, S'_K, R'_1, \ldots, R'_m)$ is given by the product form

$$\Pi(S'_1, \ldots, S'_K, R'_1, \ldots, R'_m) =$$
$$B \prod_{k=1}^{K} \pi_k (S'_k \underset{L}{\bowtie} (R'_{1k} \parallel \cdots \parallel R'_{mk}))$$

where B is the normalising constant, π_k is the steady state probability distribution of $S_k \underset{L}{\bowtie} R$ and R'_{jk} is the derivative of R_j which can be inferred from S'_k.

As previously, it is a straightforward matter to show that if we form the product process based on the S_k then it satisfies the conditions of Theorem 3.2.

4 Examples

4.1. Concurrent processing and database locking

We consider the database locking protocol modelled in Boucherie's paper directly as a product process of Markov processes. We show that the system can naturally be modelled using PEPA, generating the same set of competing Markov product processes. Moreover, this can be recognised without recourse to the state space since the PEPA model satisfies the syntactic conditions identified in the previous section.

We assume that the database consists of N items and that each transaction is associated with a list of items in the database—those items needed to process the transaction. Each such list can be partitioned into two sets, with items in the leading part requiring exclusive locks, and items in the trailing set requiring non-exclusive locks. We assume all transactions exclusively lock at least one item. Transaction processing requests arrive from the environment; on arrival, the database lock manager makes decides whether to grant or refuse the request on the basis of the locks required. Let W_d and R_d be the lists of exclusively locked and non-exclusively locked items respectively in the database at the time of arrival. Let W_a and R_a be the lists of items required, exclusively and non-exclusively, by the arriving request. The locks are granted and the processing request accepted if

$$(W_a \cap W_d) \cup (W_a \cap R_d) \cup (R_a \cap W_d) = \emptyset.$$

Otherwise the locks are denied and the request for transaction processing is blocked, cleared, and discarded. If the locks are granted, and the transaction is accepted for processing, the locks are not released until the entire processing of the transaction is complete.

There are K types of transactions, labelled $k = 1, \ldots, K$. Assume that a transaction of type k requires items $j_k \subset \{1, \ldots, N\}$ and $p_k \subset \{1, \ldots, N\}$ to be exclusively and non-exclusively locked respectively. If transactions of type k arrive at Poisson rate λ_k and are served at exponential rate μ_k, then the database lock protocol can be modelled as a PEPA process. We associate one PEPA component with each transaction type and each item in the database.

$$\begin{aligned} Trans_k &\stackrel{\text{def}}{=} (arrive_k, \lambda_k).(finish_k, \mu_k).Trans_k \\ Item_i &\stackrel{\text{def}}{=} \sum_{k \in T(i)} (arrive_k, \top).(finish_k, \top).Item_i \end{aligned}$$

where $T(i) = \{k : i \in j_k\}$, is the set of transactions which require exclusive access to item i.

The complete system is

$$(Trans_1 \parallel \cdots \parallel Trans_K) \underset{L}{\bowtie} (Item_1 \parallel \cdots \parallel Item_N)$$

where $L = \{arrive_k, finish_k \forall k\}$. Each $Item_i$ is a simple guarding and returning resource with respect to the components $Trans_k$, $1 \leq k \leq K$. Following Boucherie, we can see that the equilibrium distribution is

$$\pi(\bar{n}) = B \prod_{k=1}^{K} \left(\frac{\lambda_k}{\mu_k}\right)^{n_k}$$

where B is the normalising constant, and n_k is 1 if $Trans'_k \equiv (finish_k, \mu_k).Trans_k$, 0 otherwise.

4.2. The Taxi Rank System

In [23] a simple taxi rank system is considered. In this system there are a given number of customers and a given number of taxis. If there is an available taxi in the taxi rank a customer has two possibilities. He can take a taxi to the market; the taxi then waits for the customer to complete his shopping and then returns the customer to the taxi rank, becoming available for the next customer. Alternatively the customer can choose to walk through the garden and then go back to the taxi rank. If there are no taxis available at the rank the customer can only go for a walk.

A PEPA model of the customer is as follows:

$$\begin{aligned} C_1 &\stackrel{\text{def}}{=} (to_market, r_1).C_2 + (to_garden, r_2).C_3 \\ C_2 &\stackrel{\text{def}}{=} (from_market, r_3).C_1 \\ C_3 &\stackrel{\text{def}}{=} (from_garden, r_4).C_1 \end{aligned}$$

while the taxi can be represented as:

$$\begin{aligned} T_1 &\stackrel{\text{def}}{=} (to_market, \top).T_2 \\ T_2 &\stackrel{\text{def}}{=} (from_market, \top).T_1 \end{aligned}$$

A taxi rank system in which there are two customers and one taxi is modelled by

$$TS \stackrel{\text{def}}{=} (C_1 \parallel C_1) \underset{\substack{\{to_market \\ from_market\}}}{\bowtie} T_1$$

In this model it is trivial to see that the taxi is a guarding and returning resource with respect to each of the customers. Considering a customer in isolation we find that:

$$\begin{aligned}\pi_C(C_1) &= \frac{r_2 r_4}{r_2 r_4 + r_1 r_4 + r_2 r_3} \\ \pi_C(C_2) &= \frac{r_1 r_4}{r_2 r_4 + r_1 r_4 + r_2 r_3} \\ \pi_C(C_3) &= \frac{r_2 r_3}{r_2 r_4 + r_1 r_4 + r_2 r_3}\end{aligned}$$

Thus, we can deduce that the equilibrium distribution is:

$$\Pi(C_i, C_j, T_k) = B\pi_C(C_i) \times \pi_C(C_j)$$

where B is the normalising constant and the $\pi_C(C_i)$ are as above.

The significance of this example is that it shows that there is an intersection between the class of PEPA models which satisfies Sereno's criteria for product form and those which satisfy Boucherie's.

5 Conclusions and Further Work

We have demonstrated a class of PEPA models which satisfy Boucherie's conditions and so can be decomposed into submodels which are then solved separately without loss of accuracy. Unlike the Petri net models presented in [2], the PEPA models which exhibit product form can be detected automatically. Unlike the Markov processes presented in [2] all elements of the system, including the resources, are explicitly and formally represented in the model.

Furthermore, as we have seen in Section 4.2 the SPA context provides a framework for comparing models which satisfy different product form criteria. The taxi model presented in that section can clearly be seen to satisfy Sereno's criteria for product form as well as those presented in this paper. In future work we hope to use the SPA context to investigate fully the relationship between the different classes of product form model which have been identified.

Several issues remain to be explored with respect to the class of models identified in this paper. Perhaps the most important one is to establish an efficient method for calculating the normalising constant. We also intend to extend the class of models considered by removing the restriction that the independent submodels should be sequential and by allowing multiple instances of resource components. The advantage of using SPA for this work is the formality of the approach and the subsequent automation of the technique within a modelling tool, such as the PEPA Workbench [7].

In order to tackle models of the size and complexity needed for the performance analysis of the next generation of computer systems, efficient techniques for the construction, manipulation and solution of large Markov processes will be required. PEPA provides a formal, structured, yet expressive, high-level language for specifying such processes.

The class of models presented here is quite limited. However a much larger class of models *almost* fit this criteria. Through tool support within the PEPA Workbench we aim, in future, to be able to identify this wider class of models and use the decomposition technique presented here to provide reliable approximate solutions. This work forms part of a wider goal of implementing a wide range of decomposition techniques for product form and almost product form models. An account of the development of the PEPA Workbench in this regard can be found in [5].

Acknowledgements

This work was supported by the Engineering and Physical Sciences Research Council via the COMPA project (G/L10215). It originated from, and benefited greatly from, discussions with Matteo Sereno of University of Torino, made possible by funding from The British Council, CNR (ROM/889/94/9) and the QMIPS BRA project. We would also like to thank Graham Clark and Stephen Gilmore for helpful comments and discussions during the preparation of earlier drafts of this paper.

References

[1] M. Bernardo and R. Gorrieri, A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time, *Theoretical Computer Science*, **202**(1-2), pp. 1-54, 1998.

[2] R. Boucherie, A Characterisation of Independence for Competing Markov Chains with Applications to Stochastic Petri nets, *IEEE Transactions on Software Engineering*, **20**(7), pp. 536-544, 1994.

[3] P. Buchholz, Hierarchical Markovian Models - Symmetries and Reduction, in: R. Pooley and J. Hillston eds., *Computer Performance Evaluation: Modelling Techniques and Tools,* Edinburgh University Press, 1993.

[4] P. Buchholz, Compositional Analysis of a Markovian Process Algebra, in: U. Herzog and M. Rettelbach eds., *Proceedings of the 2nd Workshop on Process Algebra and Performance Modelling,* Regensburg, July, 1994.

[5] G. Clark, S. Gilmore, J. Hillston and N. Thomas, Experience with the PEPA Performance Modelling Tools, in: R. Pooley and N. Thomas eds., *Proceedings of the 14th UK Performance Engineering Workshop,* Edinburgh, July, 1998.

[6] S. Donatelli, Superposed Generalised Stochastic Petri Nets: Definition and Efficient Solution, in: M. Silva ed., *Proc. of 15th Int. Conference on Application and Theory of Petri Nets,* 1994.

[7] S. Gilmore and J. Hillston, The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling, in: G. Haring and G. Kotsis eds., *Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation,* Lecture Notes in Computer Science, vol. 794, pp. 353–368, Springer Verlag, 1994.

[8] S. Gilmore, J. Hillston, R. Holton, and M. Rettelbach. Specifications in Stochastic Process Algebra for a Robot Control Problem, *International Journal of Production Research,* December 1995.

[9] N. Götz, U. Herzog and M. Rettelbach, TIPP - a language for timed processes and performance evaluation, in: J. Hillston and F. Moller eds., *Proceedings of the Workshop on Process Algebra and Performance Modelling,* University of Edinburgh, UK, May, 1993.

[10] P. Harrison and J. Hillston, Exploiting quasi-reversible structures to find product form solutions in Markovian process algebra models, *The Computer Journal,* **38**(7), pp. 510-520, 1995.

[11] J. Hillston, The Nature of Synchronisation, in: U. Herzog and M. Rettelbach eds., *Proceedings of the 2nd Workshop on Process Algebra and Performance Modelling,* Regensburg, July, 1994.

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

[13] J. Hillston, Compositional Markovian Modelling Using a Process Algebra, in: W. Stewart ed., *Computations With Markov Chains,* Kluwer, 1995.

[14] J. Hillston, A Class of PEPA Models Exhibiting Product Form Solution over Submodels, Technical Report LFCS-98-382, Department of Computer Science, University of Edinburgh, 1998.

[15] J. Hillston and V. Mertsiotakis, A Simple Time Scale Decomposition Technique for Stochastic Process Algebras, *The Computer Journal,* **38**(7), pp. 566-577, 1995.

[16] D.R.W. Holton, A PEPA Specification of an Industrial Production Cell, *The Computer Journal,* **38**(7), pp. 542-551, 1995.

[17] A. Lazar and T. Robertazzi, Markovian Petri Net Protocols with Product Form Solution, *Performance Evaluation,* **12**(1), pp 67-77, 1991.

[18] V. Mertsiotakis, Time Scale Decomposition of Stochastic Process Algebra Models, in: *Proceedings of the 5th Process Algebra and Performance Modelling Workshop,* University of Twente, 1997.

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

[20] M. Rettelbach and M. Siegle, Compositional Minimal Semantics for the Stochastic Process Algebra TIPP, in: U. Herzog and M. Rettelbach eds., *Proceedings of the 2nd Workshop on Process Algebra and Performance Modelling,* Regensburg, July, 1994.

[21] M. Ribaudo, On the Aggregation Techniques in Stochastic Petri Nets and Stochastic Process Algebras, *The Computer Journal,* **38**(7), pp. 600-611, 1995.

[22] W. Sanders and J. Meyer, Reduced base model construction methods for stochastic activity networks. *IEEE Journal on Selected Areas in Communications,* **9**(1), pp. 25-36, 1991.

[23] M. Sereno, Towards a product form solution for stochastic process algebras, *The Computer Journal,* **38**(7), pp. 622-632, 1995.