A Class of PEPA Models Exhibiting Product Form Solution over Submodels

Jane Hillston

February 1998

Abstract

The advantages of the compositional structure within 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 [1], Boucherie showed that some of 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. Moreover, 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 [2, 3, 4, 5]. Most of these approaches stress compositionality, i.e. the system is decomposed into subsystems that are smaller and more easily modelled. In particular, in recent years, several stochastic process algebras (SPAs) have been presented. These include PEPA [6, 7], MTIPP [8, 9], and EMPA [10]. Initially the benefits of the compositional structure within these languages have been investigated for model construction and model simplification [11, 12, 13, 14]. However the real strength of this approach will result from the exploitation of this structure to aid model solution.

A variety of decompositional or structural techniques have been proposed to aid in the solution of large Markov processes. Recently several preliminary results have been published which show that, at least for some particular cases, there is a clear relationship between these 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 [15, 16], whilst the application of time scale decomposition techniques to SPA models is described in [17, 18], and investigations of structures within SPA models which give rise to product form equilibrium distributions are reported in [19, 20, 21].

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 study belong to the class of competing Markov processes identified by Boucherie in [1]. The advantage of characterising this class of models in PEPA is that by "lifting" the definition from the stochastic process level to a formally defined high-level modelling paradigm we can facilitate the automatic detection of these structures when they occur. The models presented in [1], 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 using formally defined procedures.

The rest of the paper is organised as follows: in the next section we present a brief overview of PEPA (Performance Evaluation Process Algebra), a process algebra in which exponentially distributed delays are associated with every action. Section 3 introduces the notion of *resource* within the context of PEPA models while Section 4 explores when models comprised of components competing for a resource are susceptible to product form solution. This solution technique is demonstrated on several examples in Section 5. Finally in Section 6 we summarise the results of the paper and outline directions for further work.

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 [22], 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.

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: $\boldsymbol{(\alpha, r).P}$ 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: $\boldsymbol{P + Q}$ 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: $\boldsymbol{P \underset{L}{\bowtie} Q}$ 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 [23]. In PEPA the shared activity occurs at the rate of the slowest participant (see Appendix A for details). 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 [6] 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 [6] for a formal definition).

It will also sometimes be 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 $\vec{\mathcal{A}}(C)$.

Definition 2.1 (Complete Action Type Set) *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 witnessed in the evolution of component C.

Definition 2.2 (Distinct Components) *Two components, C_1 and C_2, are said to be* distinct *if they have no actions in common, i.e. $\vec{\mathcal{A}}(C_1) \cap \vec{\mathcal{A}}(C_2) = \emptyset$.*

Example 1: Simple Processing System as Cooperating Components Consider a simple system in which a process repeatedly carries out some task. In order to complete its task the process needs the cooperation of a subsidiary process for part, but not all, of the time. Thus the task can be regarded as being in two stages. The subsidiary process meanwhile has only two activities, it is available for use except for a short period after use while it is reset. We model the process and the subsidiary process as two separate components: *Process* and *Sub* respectively. The process will undertake two activities consecutively: *use* with some rate r_1, in cooperation with the subsidiary process, and *task* at rate r_2, representing the remainder of its processing task. Similarly the subsidiary process will engage in two activities consecutively: *use*, at rate r_3 and *reset*, at rate r_4.

$$\begin{array}{ll} \mathit{Process} \stackrel{\text{def}}{=} (\mathit{use}, r_1).\mathit{Process}' & \mathit{Sub} \stackrel{\text{def}}{=} (\mathit{use}, r_3).\mathit{Sub}' \\ \mathit{Process}' \stackrel{\text{def}}{=} (\mathit{task}, r_2).\mathit{Process} & \mathit{Sub}' \stackrel{\text{def}}{=} (\mathit{reset}, r_4).\mathit{Sub} \end{array}$$

$$\mathit{System} \stackrel{\text{def}}{=} \mathit{Process} \underset{\{\mathit{use}\}}{\bowtie} \mathit{Sub}$$

The complete action type sets of these components are:

$$\begin{array}{rcl} \vec{\mathcal{A}}(\mathit{Process}) & = & \{\mathit{use}, \mathit{task}\} \\ \vec{\mathcal{A}}(\mathit{Sub}) & = & \{\mathit{use}, \mathit{reset}\} \\ \vec{\mathcal{A}}(\mathit{System}) & = & \{\mathit{use}, \mathit{task}, \mathit{reset}\} \end{array}$$

Note that we can easily extend the model to represent a system with two primary processes, independent of each other but competing for the use of the subsidiary process: $(\mathit{Process} \parallel \mathit{Process}) \underset{\{\mathit{use}\}}{\bowtie} \mathit{Sub}$.

2.2 Execution Strategy

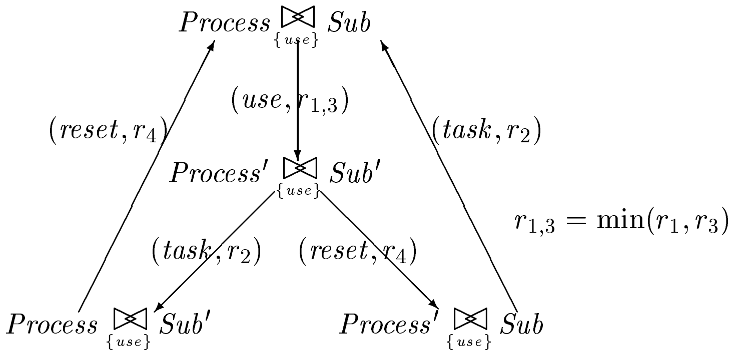
A *race condition* governs the dynamic behaviour of a model whenever more than one activity is enabled. This has the effect of replacing the non-deterministic branching of classical process algebra with probabilistic branching. The probability that a particular activity completes is given by the ratio 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 Operational Semantics and the Underlying CTMC

The semantics of PEPA, presented in the structured operational semantics style, are given in Appendix A. The underlying transition system also characterises the Markov process represented by the model. PEPA is the labelled *multi*-transition system $(\mathcal{C}, \mathcal{A}ct, \{\!| \xrightarrow{(\alpha,r)} \mid (\alpha, r) \in \mathcal{A}ct |\!\})$ where $\mathcal{C}$ is the set of components, $\mathcal{A}ct$ is the set of activities and the multi-relation $\xrightarrow{(\alpha,r)}$ is given by the rules in Appendix A.

The *derivation graph* is a graph in which syntactic terms form the nodes, and arcs represent the possible transitions between them: the operational rules define the form of this graph. Since $\xrightarrow{(\alpha,r)}$ is a multi-relation, the graph is a multigraph. This derivation graph describes the possible behaviour of any PEPA component and provides a useful way to reason about a model. It is also the basis of the construction of the underlying Markov process.

Example 1: Simple Processing System – Derivation Graph



Definition 2.3 (Derivatives) *If* $P \xrightarrow{(\alpha,r)} P'$, *then* P' *is a* (one-step) derivative *of* P. *In general,* P' *is a* derivative *of* P *if* $P \xrightarrow{(\alpha_1,r_1)} \cdots \xrightarrow{(\alpha_n,r_n)} P'$.

These derivatives are the states of the labelled multi-transition system (and of the underlying Markov process). The set of derivatives which can evolve from a component is defined recursively.

Definition 2.4 (Derivative Set) *The* derivative set *of a PEPA component* C *is denoted* $ds(C)$ *and defined as the smallest set of components such that*

- *if* $C \stackrel{\text{def}}{=} C_0$ *then* $C_0 \in ds(C)$;
- *if* $C_i \in ds(C)$ *and there exists* $a \in \mathcal{A}ct(C_i)$ *such that* $C_i \xrightarrow{a} C_j$ *then* $C_j \in ds(C)$.

Thus the derivative set is the set of components which capture all the reachable states of the system. These form the nodes of the derivation graph.

Definition 2.5 (Derivation Graph) *Given a PEPA component* C *and its derivative set* $ds(C)$, *the* derivation graph $\mathcal{D}(C)$ *is the labelled directed multigraph, whose set of nodes is* $ds(C)$, *and whose multiset of arcs,* A, *is defined as follows:*

- *The elements of* A *are taken from the set* $ds(C) \times ds(C) \times \mathcal{A}ct$;
- $\langle C_i, C_j, a \rangle$ *occurs in* A *with the same multiplicity as the number of distinct inference trees which imply* $C_i \xrightarrow{a} C_j$.

The initial component C_0, where $C \stackrel{\text{def}}{=} C_0$, forms the initial node of the graph.

It is sometimes useful to consider the relative derivative set of a component—the set of derivatives which may be reached before a given derivative.

Definition 2.6 (Relative Derivative Set) *The* relative derivative set *of a PEPA component C with respect to its derivative C' is denoted $ds_{C'}(C)$ and is defined as the smallest set of components such that*

- *if $C \stackrel{\text{def}}{=} C_0$ and $C_0 \not\equiv C'$ then $C_0 \in ds_{C'}(C)$;*
- *if $C_i \in ds_{C'}(C)$ and there exists $a \in \mathcal{A}ct(C_i)$ such that $C_i \xrightarrow{a} C_j$ and $C_j \not\equiv C'$ then $C_j \in ds_{C'}(C)$.*

Similarly, we will sometimes wish to refer to the set of action types which may be used in the transition sequence progressing from one component to another.

Definition 2.7 (Relative Action Set) *The* relative action set *of a PEPA component C with respect to its derivative C' is denoted $\vec{\mathcal{A}}_{C'}(C)$ and is defined as the set of action types such that*

- *if $C_i \in ds_{C'}(C)$ and there exist $(\alpha, r) \in \mathcal{A}ct(C_i)$ such that $C_i \xrightarrow{(\alpha,r)} C_j$ and $C_j \not\equiv C'$ then $\alpha \in \vec{\mathcal{A}}_{C'}(C)$.*

To form the underlying Markov process a state is associated with each node of the derivation graph, and the transitions between states are derived from the arcs of the graph. This use of the derivation graph is analogous to the use of the reachability graph in stochastic extensions of Petri nets such as SPNs [24]. We assume that the model is finite so that the number of nodes in the derivation graph is finite.

The *transition rate* between two components C_i and C_j, denoted $q(C_i, C_j)$, is the sum of the activity rates labelling arcs connecting node C_i to node C_j in the derivation graph, i.e. $q(C_i, C_j) = \sum_{a \in \mathcal{A}ct(C_i|C_j)} r_a$ where $\mathcal{A}ct(C_i|C_j) = \{\!|\, a \in \mathcal{A}ct(C_i) \mid C_i \xrightarrow{a} C_j \,|\!\}$. Typically this multiset will only contain one element. If C_j is not a one-step derivative of C_i, then $q(C_i, C_j) = 0$. The $q(C_i, C_j)$, or q_{ij}, are the off-diagonal elements of the infinitesimal generator matrix of the Markov process, $\mathbf{Q}$. Diagonal elements are formed as the negative sum of the non-diagonal elements of each row.

2.4 Cyclic PEPA

Unlike the earlier papers [19, 20], 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 equilibrium 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 [6, 25]. These conditions imply that the model must be a *cyclic* PEPA component.

Definition 2.8 (Cyclic Components) *A PEPA component is* cyclic*, or* irreducible*, if it is a derivative of all the components in its derivative set.*

$$C \in ds(C_i) \textit{ for all } i \textit{ such that } C_i \in ds(C)$$

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 this means that for every choice, whichever one-step derivative is chosen the model must eventually return to the point where the choice can be made again, possibly with a different outcome. If we consider the layering imposed on a component by cooperation combinators, this implies that choice combinators may only be introduced at the lowest level of a cyclic component since syntactic terms are associated with states. 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.

As stated earlier there is a strong relationship between cyclic PEPA components and irreducibility in the underlying Markov processes. This is formalised in the following theorem.

Theorem 2.1 *The Markov process underlying a PEPA model is irreducible, and therefore ergodic, if, and only if, the initial component of the model is cyclic.*

As explained in the previous section the "states" of a PEPA model as it evolves are the syntactic terms, or derivatives, which the model will go through. 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[1]. 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. This suggests a more compact representation of any particular state.

Definition 2.9 (State Vector) *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.

The subscript P is required since knowledge of the static structure of P must be retained in order to reason about the model's behaviour. However, it will be omitted when it is clear from the context which P is intended.

Definition 2.10 (Redundancy (within the state vector representation)) *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.

Definition 2.11 (Redundancy (within the model)) *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 removing 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. A component which is redundant within a model will also be redundant within the corresponding state vector representation.

[1]However they will not necessarily all change with every transition of the derivation graph.

Example 2: A simple $M/M/1/N/N$ queue Consider a single server queue with buffer capacity N and customer population N. Assume that customers arrive at rate λ and service will occur at rate μ. We can represent the queue as two interacting components: a *Server* and a *Line*. The *Server* will engage in a *serve* activity at rate μ whenever it is able.

$$Server \stackrel{\text{def}}{=} (serve, \mu).Server$$

The *Line* models the buffer. When the buffer is not full *Line* will *accept* a customer at rate λ. When the buffer is non-empty a customer will be available for service at a rate determined by the server (the activity $(serve, \top)$).

$$\begin{array}{rcll}
Line_0 & \stackrel{\text{def}}{=} & (accept, \lambda).Line_1 & \\
Line_i & \stackrel{\text{def}}{=} & (accept, \lambda).Line_{i+1} + (serve, \top).Line_{i-1} & \quad 1 \leq i \leq N-1 \\
Line_N & \stackrel{\text{def}}{=} & (serve, \top).Line_{N-1} & \\
 & & & \\
Queue_0 & \stackrel{\text{def}}{=} & Line_0 \underset{\{serve\}}{\bowtie} Server &
\end{array}$$

Figure 1: Example 2: A simple $M/M/1/N/N$ queue

Server and *Line* are both sequential components. The behaviour of the queue as a whole is determined by their cooperation.

The state vector representation of $Queue_j$ for any j, $1 \leq j \leq N$, is $(Line_j, Server)$. However, since the derivative of *Server* is the same in all derivatives of *Queue*, clearly *Server* is redundant in the state vector representation. The reduced state vector representation of $Queue_j$ is simply $(Line_j)$. However, it should be noted that the component *Line* alone does not represent the system; indeed it does not form a complete model since the passive activity $(serve, \top)$ has no active component to cooperate with. The component *Server* is redundant only in the sense of the state representation.

Example 3: A simple multi-processor shared memory system Consider a simple system in which two processors compete for access to a shared memory via a bus. The processors are independent and both follow the same pattern of behaviour—a simple repetitive cycle of actions: each will compute, acquire the bus, send the message, and then release the bus. The bus meanwhile will be available for transmitting (waiting to be acquired) or in use (waiting to be released). The memory is simply ready to transmit whenever access to it is acquired.

The state vector representation of any derivative $Machine'$ is

$$(Proc_{i_1}, Proc_{i_2}, Bus', Mem).$$

$$
\begin{array}{rcl}
\mathit{Proc} & \stackrel{\text{def}}{=} & (\mathit{compute}, r_1).(\mathit{acquire}, r_2).(\mathit{transmit}, r_3).(\mathit{release}, r_4).\mathit{Proc} \\
\mathit{Bus} & \stackrel{\text{def}}{=} & (\mathit{acquire}, \top).(\mathit{release}, \top).\mathit{Bus} \\
\mathit{Mem} & \stackrel{\text{def}}{=} & (\mathit{transmit}, \top).\mathit{Mem} \\
\\
\mathit{Machine} & \stackrel{\text{def}}{=} & \Big((\mathit{Proc} \parallel \mathit{Proc}) \underset{\{\mathit{acquire},\ \mathit{release}\}}{\bowtie} \mathit{Bus}\Big) \underset{\{\mathit{transmit}\}}{\bowtie} \mathit{Mem}
\end{array}
$$

Figure 2: Example 3: A simple multi-processor shared memory system

The current state of the memory is always *Mem* and so clearly this component is redundant in the state vector representation and can be eliminated. Similarly the current state of the bus, Bus' can always be deduced from the current state of the two processors. For example if

$$\mathit{Proc}_{i_1} \equiv \mathit{Proc} \quad \text{and} \quad \mathit{Proc}_{i_2} \equiv (\mathit{acquire}, r_2).(\mathit{transmit}, r_3).(\mathit{release}, r_4).\mathit{Proc}$$

then it follows that $\mathit{Bus}' \equiv \mathit{Bus}$. Hence the reduced state vector representation of $\mathit{Machine}'$ is $(\mathit{Proc}_{i_1}, \mathit{Proc}_{i_2})$. Clearly no further reduction is possible. Although the processors compete for the use of the bus and therefore constrain each other's behaviour we cannot deduce the current state of one processor given the state of the other. Similarly, the current derivatives of one processor and the bus do not contain sufficient information to be able to deduce the current derivative of the second processor.

Note that the component *Mem* is redundant within the model: it does not affect the behaviour since the necessary scheduling is already imposed by the *Bus*. The component *Bus*, however, is essential and cannot be omitted even if the memory component is retained.

Example 4: A small railway system Consider a simple railway system with the arrangement of tracks and stations as shown below. There are two trains: Train_1 which circulates round stations A, B and C, and Train_2 which has a choice of two routes, either round D, B, and C, or round D, E and F (see Figure 3).

The PEPA model of the system is shown in Figure 4.

In the first example the redundancy of the *Server* component within the state vector representation can be attributed to the fact that it is unchanging. The redundancy of the *Mem* component in the second example is similar. However note that in this second case, unlike the first, the component *Mem* can be eliminated *from the model* without changing the behaviour, or leaving the model incomplete. The *Bus* component is sufficient to ensure the correct exclusion is maintained between the two processors. The *Bus* component in the second example and the *Signal* component in the third, are redundant because they

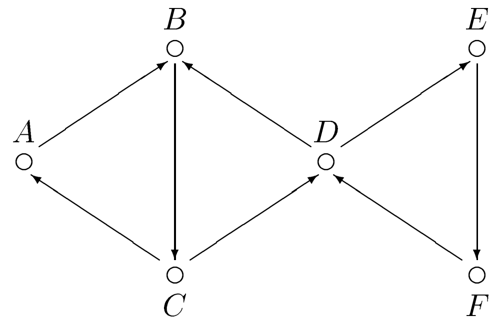


Figure 3: Schematic representation of the small railway system

never act independently and its current derivative can always be inferred directly from the rest of the derivatives. We will see in the following section that this form of redundancy is closely related to the notion that a sequential component can be identified as a *resource* to be used by the rest of the model. In fact we will see that *Server*, *Bus*, *Mem* and *Signal* are all *resource components* in their respective model components.

3 Resource Components

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 sometimes important to be able to distinguish whether a given sequential component R is a subcomponent within a model, and if so 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 action types outside this set.

In the following we will formalise these ideas. First, we define a partial order, $\prec$, over components, which captures the notion of *being a subcomponent*:

$$
\begin{array}{rcl}
\mathit{Train}_{1A} & \stackrel{\mathrm{def}}{=} & (\mathit{track}_{AB}, t_1).(\mathit{stop}_B, s_1).\mathit{Train}_{1B} \\
\mathit{Train}_{1B} & \stackrel{\mathrm{def}}{=} & (\mathit{track}_{BC}, t_1).(\mathit{stop}_C, s_1).\mathit{Train}_{1C} \\
\mathit{Train}_{1C} & \stackrel{\mathrm{def}}{=} & (\mathit{track}_{CA}, t_1).(\mathit{stop}_A, s_1).\mathit{Train}_{1A} \\
\mathit{Train}_{2D} & \stackrel{\mathrm{def}}{=} & (\mathit{track}_{DB}, t_2).(\mathit{stop}_B, s_2).\mathit{Train}_{2B} + (\mathit{track}_{DE}, t_2).(\mathit{stop}_E, s_2).\mathit{Train}_{2E} \\
\mathit{Train}_{2B} & \stackrel{\mathrm{def}}{=} & (\mathit{track}_{BC}, t_2).(\mathit{stop}_C, s_2).\mathit{Train}_{2C} \\
\mathit{Train}_{2C} & \stackrel{\mathrm{def}}{=} & (\mathit{track}_{CD}, t_2).(\mathit{stop}_D, s_2).\mathit{Train}_{2D} \\
\mathit{Train}_{2E} & \stackrel{\mathrm{def}}{=} & (\mathit{track}_{EF}, t_2).(\mathit{stop}_F, s_2).\mathit{Train}_{2F} \\
\mathit{Train}_{2F} & \stackrel{\mathrm{def}}{=} & (\mathit{track}_{FD}, t_2).(\mathit{stop}_D, s_2).\mathit{Train}_{2D} \\
& & \\
\mathit{Signal} & \stackrel{\mathrm{def}}{=} & (\mathit{track}_{AB}, \top).\mathit{Signal}_1 + (\mathit{track}_{DB}, \top).\mathit{Signal}_2 \\
\mathit{Signal}_1 & \stackrel{\mathrm{def}}{=} & (\mathit{track}_{CA}, \top).\mathit{Signal} \\
\mathit{Signal}_2 & \stackrel{\mathrm{def}}{=} & (\mathit{track}_{CD}, \top).\mathit{Signal} \\
& & \\
\mathit{Railway} & \stackrel{\mathrm{def}}{=} & \left(\mathit{Train}_{1A} \parallel \mathit{Train}_{2D}\right) \underset{L}{\bowtie} \mathit{Signal} \\
\text{where } L & = & \{\mathit{track}_{AB}, \mathit{track}_{CA}, \mathit{track}_{DB}, \mathit{track}_{CD}\}
\end{array}
$$

Figure 4: Example 4: A small railway system

Definition 3.1 (Subcomponents)

1. $R \prec P$ *if* $R \in ds(P)$
2. $R \prec P + Q$ *if* $R \prec P \vee R \prec Q$
3. $R \prec P \underset{L}{\bowtie} Q$ *if* $R \prec P \vee R \prec Q$
4. $R \prec P/L$ *if* $R \prec P$
5. $R \prec A$ *if* $A \stackrel{\mathrm{def}}{=} P \wedge R \prec P$

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 3.2 (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* $R \not\prec R'$

It is interesting to consider the case when all the possible actions of a sequential component are constrained by its interface. In this case 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 3.3 (Resource Components) *A sequential component R in a model C ($R \prec C$) 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.*

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

To see the need to restrict to a sequential component which has only a single instantiation within a model consider the following model:

$$\begin{aligned}
R &\stackrel{\text{def}}{=} (\alpha, r_\alpha).(\beta, r_\beta).R \\
P &\stackrel{\text{def}}{=} (\alpha, r_\alpha).P \\
Q &\stackrel{\text{def}}{=} (\beta, r_\beta).Q \\
\mathit{System} &\stackrel{\text{def}}{=} (R \underset{\{\alpha\}}{\bowtie} P) \parallel (R \underset{\{\beta\}}{\bowtie} Q)
\end{aligned}$$

Here $\vec{\mathcal{A}}(R) = \{\alpha, \beta\}$ and $\mathcal{I}(\mathit{System} :: R) = \{\alpha, \beta\}$ but it is not true to say that R cannot act independently: one instance can act independently on α and the other can act independently on β.

Throughout the rest of the paper we will use the term "resource component" in this technical sense, which will not always coincide with our intuitive notion of a resource. For example, in Example 1 the subsidiary process *Sub* could be regarded intuitively to be a resource but it is not a resource in the technical sense because it performs the activity (reset, r_4) independently.

3.1 Resource Components and Redundancy

As we can see from the examples and subsequent discussion in Section 2.4, resource components are closely related to the notion of redundancy. In general a redundant sequential component within the state vector representation will be a resource component. The only exception will be components for which all independent actions do not result in a change of derivative. For example, consider the simple component

$$A \stackrel{\text{def}}{=} (action, r_a).A.$$

This component may be placed within any model M to form $M_a \equiv M \parallel A$. Here A is not a resource component within the model since it does not cooperate on any activities. However, it is redundant since the state vector of any derivative M'_a of M_a will consist of the corresponding derivative M' of M with the additional component A. We will call such components, with only a single derivative, *trivially redundant*. The more general case of components whose interface contains all state-changing derivatives are called *semi-resources*. They will not be considered further in this paper.

It is not the case that all resource components within a model correspond to redundant components within the state vector representation of the model. For example, consider the following system:

Example 5: A simple system with a faulty resource Suppose that there is a simple processor, P, which repeatedly carries out a two phase task: $phase_1$ it carries out alone; for $phase_2$ it requires the cooperation of a resource, *Res*. The resource is unreliable and will sometimes fail (with probability p), after completing $phase_2$, needing to be repaired before it can be used again. There is a *Repairman* who carries out the *repair* whenever the resource has failed.

$$\begin{aligned}
P &\stackrel{\text{def}}{=} (phase_1, r_1).(phase_2, r_2).P \\
Res &\stackrel{\text{def}}{=} (phase_2, (1-p) \times r_2).Res + (phase_2, p \times r_2).Failed_Res \\
Failed_Res &\stackrel{\text{def}}{=} (repair, \top).Res \\
Repairman &\stackrel{\text{def}}{=} (repair, r_3).Repairman \\
\\
System &\stackrel{\text{def}}{=} (P \parallel Repairman) \underset{\{phase_2,\, repair\}}{\bowtie} Res
\end{aligned}$$

Here both *Res* and *Repairman* are resource components, but only *Repairman* is a redundant component in the state vector representation (trivially, since it only has one derivative). If we consider the derivatives of P it is not possible to deduce the current derivative of *Res* because the resource may fail transparently to the processor.

Recall that a sequential component, $S \in \mathcal{S}$, is constructed using only prefix and choice. If the model component is to be cyclic the sequential components must clearly contain cyclic behaviour. In the simplest case, where only prefix is used to construct the component, be cyclic only if it consists of a single cycle of activities repeatedly carried out in turn. Many components take this basic form: for example, *Proc* and *Bus* in Example 3, $Train_{1A}$ in

Example 4, and P in Example 5. The derivation graph of such a component will have a ring structure.

Definition 3.4 (Basic Components) *A component $S \in \mathcal{S}$ 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.*

In general when the component is constructed also using choice, even with the restriction to cyclic behaviour, the derivation graph can take on many complex forms.

In this paper we are interested in sequential components which consist of a single choice of basic cyclic components constructed using only prefix: for example, Train_2D in Example 4, *Res* in Example 5. The derivation graph of such a component will have a central node and one loop corresponding to each basic cyclic component offered in the choice. We term such sequential components *simple*.

Definition 3.5 (Simple Components) *A sequential component $R \in \mathcal{S}$ 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 the basic component S_i returns to R and not S_i, for all $i, 1 \leq i \leq n$.*

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.

Proposition 3.1 *If a model P has distinct resource components $S_1, \ldots, S_n$ such that each S_i is basic or simple, then $S_1, \ldots, S_n$ are redundant within the state vector representation of P.*

Proof We prove the proposition by induction over n.

Case 1: $n = 1$ Suppose, without loss of generality, that the state vector representation of P is $(P_1, \ldots, P_\ell, S)$, where S is the resource component.

Case 1.1: S is a basic component. S consists of a single cycle of activities and since S is a resource component it cannot act independently of $P_1, \ldots, P_\ell$: each activity must be carried out in cooperation with one or more of the P_i. Moreover since S is a basic component we know that all its activities have distinct types. Thus, given the static structure of P and the current derivatives of $P_1, \ldots, P_\ell$, it will be apparent which activities of S have been completed. Therefore S is redundant in the state vector representation of P.

Case 1.2: S is a simple component. S consists of several competing cycles, each of which is distinct in terms of action types. Since S is a resource it cannot act independently of $P_1, \ldots, P_\ell$. Thus by similar reasoning to above the current derivative of S, will always be apparent from the current derivatives of $P_1, \ldots, P_\ell$, and S is redundant in the state vector representation of P.

Case 2: $n = k, k > 1$ Suppose, without loss of generality, that the state vector representation of P is $(P_1, \ldots, P_\ell, S_1, \ldots, S_k)$, where each S_i is a distinct simple or basic resource component, i.e. all the complete action type sets of the S_i are disjoint. Then, if S_k is a resource component in $(P_1, \ldots, P_\ell)$ it follows that S_k is a resource component in $(P_1, \ldots, P_\ell, S_1, \ldots, S_{k-1})$ and by the same argument as above S_k is redundant in this state vector representation. Moreover, by the induction hypothesis, it follows that each of the S_i, $1 \leq i < k$, is redundant within $(P_1, \ldots, P_\ell)$. □

4 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 [12] 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 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 [19, 17, 20]. One class of CTMCs which are susceptible to such an efficient solution technique are those which exhibit a *product form* 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.

Product form distributions have been widely used in the analysis of queueing networks and, due to their efficient solution, have contributed to the popularity of queueing networks for performance analysis. For example, Jackson networks [26] and their generalisation BCMP-networks [27] have been widely employed. In contrast stochastic Petri nets have rarely been found amenable to such efficient equilibrium solution, except when some of the expressibility of the formalism is reduced by excluding resource sharing and competition over resources in a general form [28].

For SPN there have been contrasting approaches. Henderson and Taylor develop product form over the *places* of the Petri net, to obtain a product form similar to that obtained for queueing networks [28]. Lazar and Robertazzi establish a first step towards a product form over *subnets*, characterising independence between subnets which compete for resources [29]. Donatelli and Sereno show how both these approaches are related to T-semiflows in the Petri net [30].

For stochastic process algebras two approaches to identifying models which give rise to product form 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:

$$\pi(C_i) = \frac{1}{B}\big(\pi_P(P_j) \times \pi_Q(Q_k)\big) \tag{4.1}$$

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, and B is a normalising constant. In [20], an extension to this class of product form models is found based on the notion of quasi-reversibility which underlies product form in queueing networks. Here a weak form of interaction between components is allowed but components are restricted to have a particular form. In [19] the application of Henderson and Taylor's results for SPN are explored within an SPA setting. Again 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 [21] an investigation of SPA models giving rise to reversible structures in the state space is presented.

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. In these cases the probability of a given model derivative will be the product of the probabilities of the corresponding derivatives in the lower level components, possibly subject to a normalising constant. The approach taken is analogous to Lazar and Robertazzi's approach with SPN, 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 4.1 above, by the introduction of one or more redundant resources[2] which impose indirect interactions between the components, but which may be eliminated from the state vector representation.

Recently Lazar and Robertazzi's result has been generalised by Boucherie [1]. In Section 4.1 we discuss the framework he introduced and consider its application to PEPA models in Section 4.2.

[2] For the remainder of this document we will mean *redundant* within the state vector representation whenever we refer to a "redundant resource" unless we explicitly state otherwise.

4.1 Boucherie's Framework

In [1], Boucherie aims to generalise the result of Lazar and Robertazzi 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 rate 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:

Definition 4.1 (Competition) *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 and enter 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.

Definition 4.2 (Product process) *The Markov process at state space $\mathcal{S} = \prod_{k=1}^{K} S_k$,* *with transition rates*

$$q(n,n') = \sum_{k=1}^{K} q_k(n_k, n'_k) \prod_{\ell=1,\ell\neq k}^{K} \mathbf{1}(n_\ell = n'_\ell)\mathbf{1}(\textit{if } n_\ell \in A_{\ell i} \textit{ then } k \notin C_{\ell i})\ ^3$$

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*.*

Observe that, as required, these transition rates imply that in each transition only one process can change its state, and that process k cannot access resource i when it 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. Moreover he establishes the following result:

Theorem 4.1 (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

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

where B *is a normalising constant, determined by the exact form of* $\mathcal{S}$*, and* $\pi_k(\cdot)$ *is the equilibrium 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 which is not in competition with ℓ ($\mathbf{1}(\text{if } i : n_\ell \in A_{\ell i} \text{ then } k \notin C_{\ell i}) = 1$) or process k wants to access the resource which ℓ occupies ($\mathbf{1}(\text{if } i : n_\ell \in A_{\ell i} \text{ then } k \notin C_{\ell i}) = 0$). In either case process k will satisfy its own global balance equations: these equations are trivially satisfied when the process is stopped and also true when the process is operating independently. It appears that the exclusion principle maintained by the transition rates of the product process imposes a protocol on the behaviour of the product process that ensures that the Markov processes in the collection behave as if they are independent.

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

[3] $\mathbf{1}$ is an indicator function: the value of $\mathbf{1}(\textit{statement})$ is 1 if the statement is true and 0 otherwise.

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 $q^t_k(n_k, n'_k)$, $t = 1, \ldots T_k$, such that

$$q_k(n_k, n'_k) = \sum_{t=1}^{T_k} q^t_k(n_k, n'_k) \tag{4.2}$$

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

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

Each process (k, t), $1 \le k \le K$, $1 \le t \le T_k$, is a Markov process in its own right. Thus for each k there may be several subprocesses $(k, t), 1 \le t \le T_k$, 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 $C_{(k,t)i} \subseteq \{(k, t) : 1 \le k \le K, 1 \le t \le T_k\}$ 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 4.2 (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 4.2. 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

$$q(n, n') = \sum_{k=1}^{K} \sum_{t=1}^{T_k} q^t_k(n_k, n'_k) \prod_{\ell=1, \ell \ne k}^{K} \mathbf{1}(n_\ell = n'_\ell) \prod_{p=1}^{T_\ell} \mathbf{1} \left(\textit{if } n_\ell \in A_{(\ell,p)i} \textit{ then } (k, t) \notin C_{(\ell,p)i} \right)$$

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.

4.2 PEPA Components Competing over Resources

Example 4 revisited The application of Boucherie's framework to PEPA models with resource components can be readily illustrated if we consider again the simple railway system (Example 4) introduced in Section 2.4 and shown in Figure 4.

Signal is a simple resource component within the *Railway*—all its activities are carried out in cooperation with one or other of the trains—and as we have already seen *Signal* is redundant within the state vector representation of *Railway*. The reduced state vector representation is $(\mathit{Train}_1', \mathit{Train}_2')$, and these two sequential components are independent, in the sense that there is no cooperation set in operation between them.

Let us associate Markov process S_1 with the first train and Markov process S_2 with the second; then

$$\begin{aligned}
S_1 &= \{n_{1_1} \equiv \mathit{Train}_{1A}, n_{1_2} \equiv (\mathit{stop}_B, s_1).\mathit{Train}_{1B}, n_{1_3} \equiv \mathit{Train}_{1B}, \\
&\quad n_{1_4} \equiv (\mathit{stop}_C, s_1).\mathit{Train}_{1C}, n_{1_5} \equiv \mathit{Train}_{1C}, n_{1_6} \equiv (\mathit{stop}_A, s_1).\mathit{Train}_{1A}\} \\
S_2 &= \{n_{2_1} \equiv \mathit{Train}_{2D}, n_{2_2} \equiv (\mathit{stop}_B, s_2).\mathit{Train}_{2B}, n_{2_3} \equiv \mathit{Train}_{2B}, \\
&\quad n_{2_4} \equiv (\mathit{stop}_C, s_2).\mathit{Train}_{2C}, n_{2_5} \equiv \mathit{Train}_{2C}, n_{2_6} \equiv (\mathit{stop}_D, s_2).\mathit{Train}_{2D} \\
&\quad n_{2_7} \equiv (\mathit{stop}_E, s_2).\mathit{Train}_{2E}, n_{2_8} \equiv \mathit{Train}_{2E}, n_{2_9} \equiv (\mathit{stop}_F, s_2).\mathit{Train}_{2F}, \\
&\quad n_{2_{10}} \equiv \mathit{Train}_{2F}, n_{2_{11}} \equiv (\mathit{stop}_D, s_2).\mathit{Train}_{2D}\}
\end{aligned}$$

In fact we can separate S_2 into $S_2^{(1)}$ and $S_2^{(2)}$ as follows:

$$\begin{aligned}
q_2^{(1)}(n_{2_i}, n_{2_j}) &= q_2(n_{2_i}, n_{2_j}) \qquad \text{if } i,j \in \{1,2,3,4,5,6\} \\
q_2^{(2)}(n_{2_i}, n_{2_j}) &= q_2(n_{2_i}, n_{2_j}) \qquad \text{if } i,j \in \{1,7,8,9,10,11\}
\end{aligned}$$

and it is clear to see that the corresponding Markov processes $S_2^{(1)}$ and $S_2^{(2)}$ satisfy the local balance property with respect to this separation.

Let the set of resources I be $\{0, \mathit{signal}\}$, where 0 denotes the notional resource held during independent operation, and *signal* denotes working in cooperation with the resource component. Note that the Markov processes S_1 and $S_2^{(2)}$ do not compete at all, whereas S_1 and $S_2^{(1)}$ compete for use of the signal (thus ensuring safety in the system). Then the state spaces of the individual processes S_1 and S_2 can be partitioned as follows:

$$\begin{aligned}
A_{10} &= \{n_{1_1}\} & A_{1\mathit{signal}} &= \{n_{1_2}, n_{1_3}, n_{1_4}, n_{1_5}, n_{1_6}\} \\
A_{(2,1)0} &= \{n_{2_1}\} & A_{(2,1)\mathit{signal}} &= \{n_{2_2}, n_{2_3}, n_{2_4}, n_{2_5}, n_{2_6}\} \\
A_{(2,2)0} &= \{n_{2_1}, n_{2_7}, n_{2_8}, n_{2_9}, n_{2_{10}}, n_{2_{11}}\} & A_{(2,2)\mathit{signal}} &= \emptyset
\end{aligned}$$

$C_{10} = C_{(2,1)0} = C_{(2,2)0} = \emptyset$; $C_{1\mathit{signal}} = \{(2,1)\}$; $C_{(2,1)\mathit{signal}} = \{1\}$; $C_{(2,2)\mathit{signal}} = \emptyset$.

It can be readily seen that the definition of rates of shared activities in PEPA components imposes the correct transition rates in the product process. There is no direct interaction between the two trains and so they have no shared activities, i.e. only one of them will change state at a time. Moreover these actions either proceed unhindered (at the normal rate) or are blocked (because the *Signal* is not available to synchronise with). Thus the Markov process generated by the PEPA component *Railway* is exactly the product process

$$(S_1 \times S_2) \setminus (A_{1signal} \times A_{(2,1)signal})$$

and Boucherie's more general result (Theorem 4.2) applies. It follows that the equilibrium probability of any derivative $\mathit{Railway}' = (\mathit{Train_1}', \mathit{Train_2}')$ can be derived from the equilibrium probability of $\mathit{Train_1}' \underset{L}{\bowtie} \mathit{Signal}'$ and the equilibrium probability of $\mathit{Train_2}' \underset{L}{\bowtie} \mathit{Signal}''$. The equilibrium probability distribution of $(\mathit{Train_1} \parallel \mathit{Train_2}) \underset{L}{\bowtie} \mathit{Signal}$ can be derived directly from the derivation graph and a renormalisation of the product of probabilities. Note that *Signal* is a passive resource in both subsystems, and the Markov processes underlying $\mathit{Train_i} \underset{L}{\bowtie} \mathit{Signal}$ and $\mathit{Train_i}$ are identical, i.e. *Signal* is redundant within the model when these subsystems are considered in isolation.

Considering the trains in isolation it is straightforward to calculate that the equilibrium distributions are:

$$\begin{aligned}
\pi_1(n_{1_i}) &= \begin{cases} \dfrac{s_1}{3(s_1+t_1)} & \text{if } i \in \{1,3,5\} \\ \dfrac{s_1}{3(s_1+t_1)}\left(\dfrac{t_1}{s_1}\right) & \text{if } i \in \{2,4,6\} \end{cases} \\
\pi_2(n_{2_j}) &= \begin{cases} \dfrac{s_2}{5s_2+6t_2} & \text{if } j \in \{1,3,5,8,10\} \\ \dfrac{s_2}{5s_2+6t_2}\left(\dfrac{t_2}{s_2}\right) & \text{if } j \in \{2,4,6,7,9,11\} \end{cases}
\end{aligned}$$

Thus the equilibrium probability that the railway is in state $(\mathit{Train_1}', \mathit{Train_2}')$ is

$$B\pi_1(n_{1_i}) \times \pi_2(n_{2_j})$$

where B is the normalising constant and n_{1_i} and n_{2_j} are the states corresponding to derivatives $\mathit{Train_1}'$ and $\mathit{Train_2}'$ respectively. □

In this example it is clear that the resource, *Signal*, enforces a Boucherie-style exclusion over the Markov processes corresponding to the first train and the first loop of the second train. However, in general we need to impose some more 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 4.3 (Guarding Resource with respect to a Sequential Component) *Let $R \equiv R_1 \parallel \cdots \parallel R_m$, where the R_j are distinct simple components, be a redundant resource in the model component $S \underset{L}{\bowtie} R$. Then R is a* guarding resource *with respect to the sequential component S if, for each α_i such that $S \equiv \sum(\alpha_i, r_i).S_i$ there exists $R_j \prec R$ such that $\alpha_i \in \mathcal{A}(R_j)$ or $\vec{\mathcal{A}}_S(S_i) \cap \vec{\mathcal{A}}(R) = \emptyset$.*

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_j, one of the subcomponents of 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 each potential first activity requires the cooperation of one of the subcomponents R_j or it initiates a sequence of activities which can return to the derivative S without any cooperation with R.

Definition 4.4 (Returning Resource with respect to a Sequential Component) *Let $R \equiv R_1 \parallel \cdots \parallel R_m$, where the R_j are distinct simple components, be a redundant resource in the model component $S \underset{L}{\bowtie} R$. Then 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 $R' \equiv R_1 \parallel \cdots \parallel (\alpha, s).R_j \parallel \cdots \parallel R_m$ for some s and $R_j \prec R$ and $\alpha \in L$, or $R' \equiv R$ and there is a sequence of activities from S to S' which are independent of R.*

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 4.3 *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 equilibrium 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 equilibrium 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 $S_k \underset{L}{\bowtie} R$.

Proof The proof 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.

First, we eliminate the resource component R from the state vector representation of M. This is possible because R is redundant within the model and so the reduced state vector without it contains enough information to capture the current state of the model.

Then 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.

It remains to show that the competition imposed by the PEPA semantics, over the resource component R, does indeed provide the Boucherie exclusion. Since we are considering a single simple resource component the resources in the model will consist of the set $I = \{0, r\}$ where I is the notional resource representing independent activity and r denotes using the resource component R.

Suppose $S_k \equiv \sum_{t=1}^{T_k}(\alpha_t, r_t).S_{kt}$. R is a guarding and returning resource with respect to S_k. This implies that the S_{kt} form distinct behaviours within $ds(S_k)$ and it follows that the state space of the corresponding Markov process can be separated into T_k Markov processes which satisfy the local balance property: $S_k^{(t)} = \{S_k\} \cup ds_{S_k}(S_{kt})$. The state space of $S_k^{(t)}$ can be partitioned into sets $A_{(k,t)0}$ and $A_{(k,t)r}$, representing sets in which $S_k^{(t)}$ is operating independently and states in which $S_k^{(t)}$ has control of R, respectively. Since R is a guarding and returning resource with respect to S_k, if $\alpha_t \in L \cap \mathcal{A}(R)$ then $A_{(k,t)0} = \{S_k\}$ and $A_{(k,t)r} = ds_{S_k}(S_{kt})$; otherwise, $A_{(k,t)0} = S_k^{(t)}$ and $A_{(k,t)r} = \emptyset$. Let $C_{(k,t)r}$ be the set of behaviours within other sequential components S_j which are also guarded by R. Then the state space of the product process generated by M when R has been eliminated from the state vector representation is:

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

Now, the S_k are mutually independent, and the semantics of the $\underset{L}{\bowtie}$ combinator ensures that the activities of R are shared with just one of the S_k at a time. Since R is a guarding resource component with respect to each of the S_k, if R is used by S_k to start a cycle $S_{(k,t)}$ the state of R will be changed and so the resource will be unavailable for all the competing behaviours $C_{(k,t)r}$. By the semantics of PEPA, the corresponding activities will simply not be enabled until R returns again to its initial state. Moreover since R is a returning resource with respect to S_k this will not happen until this component has itself returned to its initial state again. Thus, the transition rates of the underlying Markov process are:

$$q(M', M'') = \sum_{k=1}^{K} \sum_{t=1}^{T_k} q_k^t(S'_k, S''_k) \prod_{\ell=1, \ell \neq k}^{K} \mathbf{1}(S'_\ell = S''_\ell) \prod_{p=1}^{T_\ell} \mathbf{1}\left(\text{if } S'_\ell \in A_{(\ell,p)r} \text{ then } (k,t) \notin C_{(\ell,p)r}\right)$$

i.e. the locally balanced Markov processes, can either proceed as normal, satisfying their usual balance equations, or are completely blocked, and Boucherie's exclusion property is imposed. Thus the equilibrium probability of any derivative M' which has reduced state vector representation $(S'_1, \ldots, S'_K)$ is given by the product form

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

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

Before going on to more useful generalisations of the theorem we state the following trivial corollary:

Corollary 4.1 *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 4.3, with the additional condition that R is a passive resource. Then the equilibrium probability of any derivative M which has state vector representation $(S'_1, \ldots, S'_K, R')$ is given by the product form*

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

where B is the normalising constant, π_k is the equilibrium probability distribution of S_k considered in isolation.

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

Corollary 4.2 *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 4.3, 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 equilibrium 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.

Proof As previously, we have to show that if we eliminate the redundant resource component from the state vector representation and form the underlying Markov process based on the reduced state vector then it is a product process satisfying the conditions of Theorem 4.2. Since R is a guarding and returning resource with respect to each S_k, it follows that each R_i is a guarding and returning resource with respect to each S_k. Moreover, each simple resource component R_i corresponds to a notional resource, r_i. Thus, the set of notional resources is $I = \{0, r_1, \ldots, r_m\}$ where 0 denotes independent activity without the cooperation of a resource and r_i denotes activity with the cooperation of the resource component R_i.

As before assume that $S_k \equiv \sum_{t=1}^{T_k} (\alpha_t, r_t).S_{kt}$ and that the set $\{S_k^1, \ldots, S_k^{T_k}\}$ denotes the set of locally balanced Markov processes within S_k. The state space of each $S_k^{(t)}$ can be partitioned according to the set $\{0, r_1, \ldots, r_m\}$ and the sets of competing processes $C_{(k,t)}$ can be readily identified by considering the initial actions of the S_k . The PEPA semantics ensures that the state space is

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

and that the transition rates are

$$q(M', M'') = \sum_{k=1}^{K} \sum_{t=1}^{T_k} q_k^t(S_k', S_k'') \prod_{\ell=1, \ell \neq k}^{K} \mathbf{1}(S_\ell' = S_\ell'') \prod_{p=1}^{T_\ell} \mathbf{1}(\text{if } S_\ell' \in A_{(\ell,p)i} \text{ then } (k,t) \notin C_{(\ell,p)i})$$

Thus, by Theorem 4.2, we can conclude that the equilibrium distribution Π of M' is given by

$$\Pi(S_1', \ldots, S_K', R_1', \ldots, R_m') = B \prod_{k=1}^{K} \pi_k(S_k', R_{1k}', \ldots, R_{mk}')$$

where B is a normalising constant. □

Another generalisation can be achieved if we consider models in which the competing processes are not restricted to be sequential components but may themselves be model components. In order to satisfy the Boucherie conditions these model components must still be mutually independent but each model component may consist of the cooperation of several sequential components. The definitions for guarding and returning resources can be extended from sequential components to model components in a natural way. We require that the resource component is a guarding (returning) resource with respect to each sequential component considered in isolation, and moreover, that if one of a pair of interacting sequential components requires the resource in order to start a behavioural loop then the other must do so also.

Definition 4.5 (Guarding resource for a model component) *Consider a model component P which consists of n sequential components which cooperate over arbitrary sets of* *action types:* $P \equiv S_1 \underset{L_1}{\bowtie} S_2 \underset{L_2}{\bowtie} \cdots \underset{L_{n-1}}{\bowtie} S_n$. *Let* R *be a redundant resource in the model* $P \underset{L}{\bowtie} R$ *such that* R *is a guarding resource with respect to each* S_k *in the model* $S_k \underset{L}{\bowtie} R$. *Without loss of generality assume each* S_k *has the form* $S_k \equiv \sum_{t=1}^{T_k} (\alpha_{kt}, r_{kt}).S_{kt}$. *Then* R *is a* guarding resource with respect to P *if whenever, for any* k *and* ℓ *there exist* t *and* t' *such that* $\vec{\mathcal{A}}_{S_k}(S_{kt}) \cap \vec{\mathcal{A}}_{S_\ell}(S_{\ell t'}) \cap \mathcal{I}(P :: S_k) \cap \mathcal{I}(P :: S_\ell) \neq \emptyset$ *then* $\alpha_{kt} = \alpha_{\ell t'}$ *and* $\alpha_{kt} \in \mathcal{A}(R) \cap \mathcal{I}(P :: S_k) \cap \mathcal{I}(P :: S_\ell)$.

Definition 4.6 (Returning resource for a model component) *Consider a model component* P *which consists of* n *sequential components which cooperate over arbitrary sets of action types:* $P \equiv S_1 \underset{L_1}{\bowtie} S_2 \underset{L_2}{\bowtie} \cdots \underset{L_{n-1}}{\bowtie} S_n$. *Let* R *be a redundant resource in the model* $P \underset{L}{\bowtie} R$ *such that* R *is a returning resource with respect to each* S_k *in the model* $S_k \underset{L}{\bowtie} R$. *Without loss of generality assume each* S_k *has the form* $S_k \equiv \sum_{t=1}^{T_k} (\alpha_{kt}, r_{kt}).S_{kt}$. *Then* R *is a* returning resource with respect to P *if whenever, for any* k *and* ℓ *there exist* t *and* t' *such that* $\vec{\mathcal{A}}_{S_k}(S_{kt}) \cap \vec{\mathcal{A}}_{S_\ell}(S_{\ell t'}) \cap \mathcal{I}(P :: S_k) \cap \mathcal{I}(P :: S_\ell) \neq \emptyset$ *then there is a derivative of* S_{kt}, $S'_{kt} \equiv (\beta_{kt}, r_{\beta k}).S_k$ *and a derivative of* $S_{\ell t'}$, $S_{\ell t'} \equiv (\beta_{\ell t'}, r_{\ell t'}).S_\ell$ *such that* $\beta_{kt} = \beta_{\ell t'}$ *and* $\beta_{kt} \in \vec{\mathcal{A}}(R) \cap \mathcal{I}(P :: S_k) \cap \mathcal{I}(P :: S_\ell)$.

We are now in the position to state the general theorem:

Theorem 4.4 *Let* M *be a model component,* $M \equiv (P_1 \parallel P_2 \parallel \cdots \parallel P_K) \underset{L}{\bowtie} R$, *where* R *is a parallel composition of distinct simple components. Assume that the cooperation set* L *is such that each* R_i *is a redundant resource within the state representation vector of* M, *and moreover, when we consider the model components* $P_k \underset{L}{\bowtie} R$ *for all* $k, 1 \leq k \leq K$, R *is a guarding and returning resource with respect to* P_k. *Then the equilibrium probability of any derivative* M' *which has state vector representation* $(P'_1, \ldots, P'_K, R')$ *is given by the product form*

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

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

Proof As previously, we have to show that if we form the product process based on the P_k then it satisfies the conditions of Theorem 4.2. Assume that each model component P_k is comprised of sequential components $S_{k1}, \ldots, S_{kN_k}$. If each R is a guarding and returning resource with respect to each P_k, it follows that each R_i is a guarding and returning resource with respect to each $S_{kj}, 1 \leq j \leq N_k$. Each simple resource component, R_i, corresponds to a notional resource, r_i. Thus, the set of notional resources is $I = \{0, r_1, \ldots, r_m\}$ where 0 denotes independent activity without the cooperation of a resource and r_i denotes activity with the cooperation of the resource component R_i.

In the previous proofs, we associated each independent sequential component S_k with one product process in the underlying Markov process and separated that process according to the alternative behaviours of S_k which could be identified with its initial actions $\alpha_{k1}, \ldots, \alpha_{kT_k}$. Now we wish to associate each of the model components P_k with one product process in the underlying Markov process. The separation of each process into locally balanced processes in not now so straightforward.

Assume $P_k \equiv S_{k1} \underset{L_{k1}}{\bowtie} S_{k2} \underset{L_{k2}}{\bowtie} \cdots \underset{L_{kN_k-1}}{\bowtie} S_{kN_k}$ where each $S_{kj} \equiv \sum_{t=1}^{T_{kj}} (\alpha_{kjt}, r_{kjt}).S_{kjt}$. If we consider S_{kj} in isolation, since R is a guarding and returning resource with respect to S_{kj} the underlying Markov process can be separated into T_{kj} locally balanced processes. If we consider P_k, since R is a guarding and returning resource with respect to P_k, any pair of sequential components S_{kj} and S_{ki} only interact if they do so on the first activities of loops S_{kjt} and $S_{kit'}$ say. It follows that we can separate the Markov process underlying P_k into locally balanced Markov processes which correspond directly with the enabled activities of P_k, $\mathcal{A}ct(P_k) = \{\!|\, \alpha_{k1}, \ldots, \alpha_{km} \,|\!\}$. Thus we can denote the set of locally balanced Markov processes within P_k as $\{S_{P_k}^{(1)}, \ldots, S_{P_k}^{(m)}\}$.

The state space of each $S_{P_k}^{(t)}$ can be partitioned according to the set $\{0, r_1, \ldots, r_m\}$ and the sets of competing processes $C_{(k,t)}$ can be readily identified by considering the initial actions of the S_{kj} . The PEPA semantics ensures that the state space is

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

and that the transition rates are

$$q(M', M'') = \sum_{k=1}^{K} \sum_{t=1}^{T_k} q_k^t(P_k', P_k'') \prod_{\ell=1, \ell \neq k}^{K} \mathbf{1}(P_\ell' = P_\ell'') \prod_{p=1}^{T_\ell} \mathbf{1}(\text{if } P_\ell' \in A_{(\ell,p)i} \text{ then } (k,t) \notin C_{(\ell,p)i})$$

Thus, by Theorem 4.2, we can conclude that the equilibrium distribution Π of M' is given by

$$\Pi(P_1', \ldots, P_K', R_1', \ldots, R_m') = B \prod_{k=1}^{K} \pi_k(P_k', R_{1k}', \ldots, R_{mk}')$$

where B is a normalising constant. $\square$

5 Examples

5.1 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. The database locking protocol was introduced in [31] by Mitra and Weinberger.

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 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.

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

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 $Trans_k$, $k \in T(i)$.

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.

5.2 The Taxi Rank System

In [19] 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 simple 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.

5.3 Geographical Information System

Finally we consider an example in which the competing processes are not sequential components but more complex subsystems. A navigational device consists of a pair of sensors which maintain complementary data about geographical location. In order to keep the sensors' internal data structures in complementary states, they share data via a register. Each sensor gains access to the register and locks it while it reads the current data value; it then uses this information to adjust the equipment it controls while also recalculating a value for the shared register based on its own internal data structures. It then updates the value in the register and releases it. In addition, sensor 1 maintains an external monitor and will periodically gather data from this monitor and use it to recalculate its internal data structures.

Each sensor can be represented as a pair of sequential components: one responsible for resetting the equipment during each cycle, and one responsible for carrying out the data recalculation. In sensor 1, the recalculation component is assumed to be also responsible for interaction with the monitor. Thus the PEPA model of sensor 1 is:

$$\begin{aligned}
S_{110} &\stackrel{\mathrm{def}}{=} (\mathit{read_data}, r_1).S_{111} + (\mathit{gather_data}, r_5).S_{113} \\
S_{111} &\stackrel{\mathrm{def}}{=} (\mathit{recalculate}, r_2).S_{112} \\
S_{112} &\stackrel{\mathrm{def}}{=} (\mathit{update_data}, r_3).S_{110} \\
S_{113} &\stackrel{\mathrm{def}}{=} (\mathit{recalculate}, r_2).S_{110} \\
\\
S_{120} &\stackrel{\mathrm{def}}{=} (\mathit{read_data}, r_1).S_{121} \\
S_{121} &\stackrel{\mathrm{def}}{=} (\mathit{reset_apparatus}, r_4).S_{122} \\
S_{122} &\stackrel{\mathrm{def}}{=} (\mathit{update_data}, r_3).S_{120} \\
\\
\mathit{Monitor} &\stackrel{\mathrm{def}}{=} (\mathit{gather_data}, \top).\mathit{Monitor} \\
\\
\mathit{Sensor}'_{10} &\stackrel{\mathrm{def}}{=} (S_{110} \underset{L}{\bowtie} S_{120}) \underset{\{\mathit{gather_data}\}}{\bowtie} \mathit{Monitor}
\end{aligned}$$

where $L = \{read\_data, update\_data\}$. Inspection of the model makes it clear that the component *Monitor* is redundant within the model—removing it makes no difference to the underlying state space. Thus we can reduce the defining equation for $Sensor_{10}$ to be:

$$Sensor_{10} \stackrel{\text{def}}{=} S_{110} \underset{L}{\bowtie} S_{120}$$

The second sensor is defined similarly:

$$\begin{array}{rcl}
S_{210} & \stackrel{\text{def}}{=} & (read_data, r_1).S_{211} \\
S_{211} & \stackrel{\text{def}}{=} & (recalculate, r_6).S_{212} \\
S_{212} & \stackrel{\text{def}}{=} & (update_data, r_3).S_{210} \\
\\
S_{220} & \stackrel{\text{def}}{=} & (read_data, r_1).S_{221} \\
S_{221} & \stackrel{\text{def}}{=} & (reset_apparatus, r_7).S_{222} \\
S_{222} & \stackrel{\text{def}}{=} & (update_data, r_3).S_{220} \\
\\
Sensor_{20} & \stackrel{\text{def}}{=} & S_{210} \underset{L}{\bowtie} S_{220}
\end{array}$$

The data register is represented as a simple resource:

$$R \stackrel{\text{def}}{=} (read\_data, \top).(update\_data, \top).R$$

The complete system is represented as:

$$M \stackrel{\text{def}}{=} (Sensor_{10} \parallel Sensor_{20}) \underset{L}{\bowtie} R$$

If we consider $Sensor_{10} \underset{L}{\bowtie} R \equiv (S_{110} \underset{L}{\bowtie} S_{120}) \underset{L}{\bowtie} R$ it is straightforward to see that R is a guarding and returning resource with respect to $Sensor_{10}$. Similarly for $Sensor_{20}$. Moreover, since $Sensor_{10}$ and $Sensor_{20}$ cooperate on the action *read_data* and $Sensor_{10}$ acts independently within the cycle starting with the action *gather_data*, we can see that R is also a guarding and returning resource with respect to the model component $(Sensor_{10} \parallel Sensor_{20})$.

Considering the sensors in isolation we can deduce that their equilibrium probability distributions are as follows:

$$\begin{array}{rcl}
\pi_1(S_{110}, S_{120}) & = & r_2 r_3 r_4 (r_2 + r_4)/G_1 \\
\pi_1(S_{111}, S_{121}) & = & r_1 r_2 r_3 r_4 / G_1 \\
\pi_1(S_{112}, S_{121}) & = & r_1 r_2^2 r_3 / G_1 \\
\pi_1(S_{111}, S_{122}) & = & r_1 r_3 r_4^2 / G_1 \\
\pi_1(S_{112}, S_{122}) & = & r_1 r_2 r_4 (r_2 + r_4)/G_1 \\
\pi_1(S_{113}, S_{120}) & = & r_3 r_4 r_5 (r_2 + r_4)/G_1 \\
\text{where } G_1 & = & (r_2 r_4 + r_4^2)(r_1 r_2 + r_1 r_3 + r_2 r_3 + r_3 r_5) + r_1 r_2^2 r_3
\end{array}$$

$$\begin{aligned}
\pi_2(S_{210}, S_{220}) &= r_3 r_6 r_7 (r_6 + r_7)/G_2 \\
\pi_2(S_{211}, S_{221}) &= r_1 r_3 r_6 r_7/G_2 \\
\pi_2(S_{212}, S_{221}) &= r_1 r_3 r_6^2/G_2 2 \\
\pi_2(S_{211}, S_{222}) &= r_1 r_3 r_7^2/G_2 \\
\pi_2(S_{212}, S_{222}) &= r_1 r_6 r_7 (r_6 + r_7)/G_2 \\
\text{where } G_2 &= (r_6 r_7 + r_7^2)(r_1 r_3 + r_1 r_6 + r_3 r_6) + r_1 r_3 r_6^2
\end{aligned}$$

For the complete system, from Theorem 4.4 we can deduce that the equilibrium probability of any state $(S_{11i}, S_{12j}, S_{21k}, S_{22l}, R')$ is:

$$\pi(S_{11i}, S_{12j}, S_{21k}, S_{22l}, R') = B\pi_1(S_{11i}, S_{12j}) \times \pi_2(S_{21k}, S_{22l})$$

Here we have solved a six state model and a five state model instead of a 14 state model. However in general the ability to decompose the solution of a model in this way may mean the difference between the model being tractable and intractable.

6 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 [1], the PEPA models which exhibit product form can be detected automatically. Unlike the Markov processes presented in [1] all elements of the system, including the resources, are explicitly and formally represented in the model.

Furthermore, as we have seen in Section 5.2 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 contrast the queueing network with finite buffer capacity which is also presented in [19] fails to satisfy our criteria. 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 explored with respect to the class of models identified in this report. 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 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. 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 [1], the PEPA models which exhibit product form can be detected automatically. Unlike the Markov processes presented in [1] all elements of the system, including the resources, are explicitly and formally represented in the model.

Acknowledgements

This work was supported by the Engineering and Physical Sciences Research Council via the COMPA project (G/L10215). It originated from, and benefitted 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. I would also like to thank Stephen Gilmore and Graham Clark for helpful comments and discussions during the preparation of this report.

References

[1] R.J. Boucherie. A Characterisation of Independence for Competing Markov Chains with Applications to Stochastic Petri nets. In *Petri Nets and Performance Models*. IEEE Computer Society Press, 1993.

[2] W.H. Sanders and J.F. Meyer. Reduced base model construction methods for stochastic activity networks. *IEEE Journal on Selected Areas in Communications*, 9(1):25–36, January 1991.

[3] P. Buchholz. Hierarchical Markovian Models - Symmetries and Reduction. In R.J. Pooley and J. Hillston, editors, *Computer Performance Evaluation: Modelling Techniques and Tools*, volume 10 of *EDITS*, pages 234–246. Edinburgh University Press, August 1993.

[4] W.J. Stewart, K. Atif, and B. Plateau. The Numerical Solution of Stochastic Automata Network. Technical Report 6, LMC-IMAG, November 1993.

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

[6] J. Hillston. *A Compositional Approach to Performance Modelling.* PhD thesis, Department of Computer Science, University of Edinburgh, April 1994. CST-107-94.

[7] S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Appro ach to Performance Modelling. In G. Haring and G. Kotsis, editors, *Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, volume 794 of *LNCS*, pages 353–368. Springer-Verlag, 1994.

[8] N. Götz, U. Herzog, and M. Rettelbach. Multiprocessor and Distributed System Design: The Integration of Functional Specification and Performance Analysis using Stochastic Process Algebras. In *Performance'93*, 1993.

[9] H. Hermanns and M.L. Rettelbach. Syntax, Semantics, Equivalences and Axioms for MTIPP. In U. Herzog and M. Rettelbach, editors, *Proc. of 2nd Process Algebra and Performance Modelling Workshop*, 1994.

[10] M. Bernardo, L. Donatiello, and R. Gorrieri. Modelling and Analyzing Concurrent Systems with MPA. In U. Herzog and M. Rettelbach, editors, *Proc. of 2nd Process Algebra and Performance Modelling Workshop*, 1994.

[11] 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.

[12] J. Hillston. Compositional Markovian Modelling Using a Process Algebra. In W.J. Stewart, editor, *Numerical Solution of Markov Chains.* Kluwer, 1995.

[13] D.R.W. Holton. A PEPA Specification of and Industrial Production Cell. *The Computer Journal*, 38(6), 1995. Special Issue: Proc. of 3rd Process Algebra and Performance Modelling Workshop.

[14] M. Ribaudo. On the Aggregation Techniques in Stochastic Petri Nets and Stochastic Process Algebras. *The Computer Journal*, 38(6), 1995. Special Issue: Proc. of 3rd Process Algebra and Performance Modelling Workshop.

[15] P. Buchholz. Compositional Analysis of a Markovian Process Algebra. In U. Herzog and M. Rettelbach, editors, *Proc. of 2nd Process Algebra and Performance Modelling Workshop*, 1994.

[16] M.L. Rettelbach and M. Siegle. Compositional Minimal Semantics for the Stochastic Process Algebra TIPP. In U. Herzog and M. Rettelbach, editors, *Proc. of 2nd Process Algebra and Performance Modelling Workshop*, 1994.

[17] J. Hillston and V. Mertsiotakis. A Simple Time Scale Decomposition Technique for Stochastic Process Algebras. *The Computer Journal*, 38(6), 1995. Special Issue: Proc. of 3rd Process Algebra and Performance Modelling Workshop.

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

[19] M. Sereno. Towards a Product Form Solution of Stochastic Process Algebras. *The Computer Journal*, 38(6), 1995. Special Issue: Proc. of 3rd Process Algebra and Performance Modelling Workshop.

[20] P. Harrison and J. Hillston. Exploiting Quasi-reversible Structures in Markovian Process Algebra Models. *The Computer Journal*, 38(6), 1995. Special Issue: Proc. of 3rd Process Algebra and Performance Modelling Workshop.

[21] M. Bhabuta, P.G. Harrison, and K.Kanani. Detecting Reversibility in Markovian Process Algebras. In *Proc. of the 11th UK Performance Engineering Workshop for Computer and Telecommunication Systems*. Springer, 1995.

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

[23] J. Hillston. The Nature of Synchronisation. In U. Herzog and M. Rettelbach, editors, *Proc. of 2nd Process Algebra and Performance Modelling Workshop*, 1994.

[24] M.K. Molloy. Performance analysis using stochastic petri nets. *IEEE Transactions on Computers*, 31(9):913–917, September 1982.

[25] S. Gilmore, J. Hillston, and L. Recalde. Elementary Structural Analysis for PEPA. Technical Report ECS-LFCS-97-377, Laboratory of Foundations of Computer Science, University of Edinburgh, December 1997.

[26] J.R. Jackson. Jobshop-like Queueing Systems. *Management Science*, 10:131–142, 1963.

[27] F. Baskett, K.M. Chandy, R.R. Muntz, and F.G. Palacios. Open, Closed and Mixed Networks of Queues with Different Classes of Customers. *Journal of the ACM*, 22(2):248–260, April 1975.

[28] W. Henderson and P.G. Taylor. Embedded Processes in Stochastic Petri Nets. *IEEE Transactions on Software Engineering*, 17(2), 1991.

[29] A.A. Lazar and T.G. Robertazzi. Markovian Petri Net Protocols with Product Form Solution. *Performance Evaluation*, 12(1):67–77, January 1991.

[30] S. Donatelli and M. Sereno. On the Product Form Solution for Stochastic Petri Nets. In *Application and Theory of Petri Nets*, pages 154–172. Springer Verlag, 1992.

[31] B. Mitra and P.J. Weinberger. Probabilistic models of database locking: solutions, computational algorithms, and asymptotics. *Journal of the ACM*, 31:855–878, 1984.

A Structured Operational Semantics for PEPA

Before we state the inference rules which define PEPA we introduce the notion of *apparent rate*, which is needed to define the rate of a shared activity. The apparent rate at which an action type occurs within a component is of importance when comparing components or when defining how they interact. We assume that the apparent rate of an action type represents the totally capacity of a component to carry out activities of that type when it is in its current state.

Definition A.1 (Apparent Rate) *The* apparent rate *of action of type α in a component P, denoted $r_\alpha(P)$, is the sum of the rates of all activities of type α in $\mathcal{A}ct(P)$.*

1. $r_\alpha((\beta, r).P) = \begin{cases} r & \text{if } \beta = \alpha \\ 0 & \text{if } \beta \neq \alpha \end{cases}$
2. $r_\alpha(P + Q) = r_\alpha(P) + r_\alpha(Q)$
3. $r_\alpha(P/L) = \begin{cases} r_\alpha(P) & \text{if } \alpha \notin L \\ 0 & \text{if } \alpha \in L \end{cases}$
4. $r_\alpha(P \underset{L}{\bowtie} Q) = \begin{cases} \min(r_\alpha(P), r_\alpha(Q)) & \text{if } \alpha \in L \\ r_\alpha(P) + r_\alpha(Q) & \text{if } \alpha \notin L \end{cases}$

Note that an apparent rate may be unspecified: if P is defined as,

$$P \stackrel{\text{def}}{=} (\alpha, w_1\top).P_1 + (\alpha, w_2\top).P_2$$

then the apparent rate of α in component P is $r_\alpha(P) = (w_1 + w_2)\top$.

The apparent rate will be *undefined* for component expressions containing *unguarded* variables, i.e. variables which are not prefixed by an activity. Consequently we do not allow a component to be defined by such an expression.

Note that in cases of cooperation, the apparent rate of the shared activity will be the minimum of the apparent rates of the components involved, where $\min(\top, r) = r$ for all $r \in \mathbb{R}^+$. Thus we make an assumption that, in general, shared activities proceed at the rate of the slower of the two participating components. This is based on a notion that in general both components contribute some work to the shared activity. For a discussion of alternative assumptions see [23]. In the case of a passive action it is assumed that the corresponding component does not contribute at all to the work required to complete the shared activity.

The semantic rules, in the structured operational style of Plotkin, are presented here without comment; the interested reader is referred to [6] for more details. The rules are read as follows: if the transition(s) above the inference line can be inferred, then we can infer the transition below the line.

Prefix

$$\frac{}{(\alpha, r).E \xrightarrow{(\alpha,r)} E}$$

Choice

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E + F \xrightarrow{(\alpha,r)} E'} \qquad \frac{F \xrightarrow{(\alpha,r)} F'}{E + F \xrightarrow{(\alpha,r)} F'}$$

Cooperation

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E' \underset{L}{\bowtie} F} (\alpha \notin L) \qquad \frac{F \xrightarrow{(\alpha,r)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E \underset{L}{\bowtie} F'} (\alpha \notin L)$$

$$\frac{E \xrightarrow{(\alpha,r_1)} E' \quad F \xrightarrow{(\alpha,r_2)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,R)} E' \underset{L}{\bowtie} F'} (\alpha \in L), \qquad R = \frac{r_1}{r_\alpha(E)} \frac{r_2}{r_\alpha(F)} \min(r_\alpha(E), r_\alpha(F))$$

Hiding

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\alpha,r)} E'/L} (\alpha \notin L) \qquad \frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\tau,r)} E'/L} (\alpha \in L)$$

Constant

$$\frac{E \xrightarrow{(\alpha,r)} E'}{A \xrightarrow{(\alpha,r)} E'} (A \stackrel{\mathrm{def}}{=} E)$$

Figure 5: Operational Semantics of PEPA