Exploiting Structure in Solution: Decomposing Compositional Models

Jane Hillston

Division of Informatics, University of Edinburgh, jeh@dcs.ed.ac.uk

Abstract. Since their introduction in the early 1990s, compositionality has been reported as one of the major attractions of stochastic process algebras. The benefits that compositionality provides for model construction are readily apparent and have been demonstrated in numerous case studies. Early research on the compositionality of the languages focused on how the inherent structure could be used, in conjunction with equivalence relations, for model simplification and aggregation. In this chapter we consider how far we have been able to take advantage of compositionality when it comes to solving the Markov process underlying a Markovian process algebra model.

1 Introduction

At the time of the introduction of stochastic process algebras (SPA) [30] there was already a plethora of techniques for constructing performance models so the introduction of another one could have been deemed unnecessary if it were not for the fact that SPA offered something new—formally defined compositionality. Queueing networks, which have been widely used for performance modelling for more than thirty years, have an inherent compositionality but this is implicit and informal. Stochastic extensions of Petri nets have a semantic model but, in general, no clear compositional structure. In the process algebra the compositionality is explicit—provided by the combinators of the language—and formal—supported by the semantics and equivalence relations of the language.

It was immediately clear that having this explicit structure within models offers benefits for model construction:

- when a system consists of interacting components, the components, and the interaction, can each be modelled separately;
- models have a clear structure and are easy to understand;
- models can be constructed systematically, by either elaboration or refinement;
- the possibility of maintaining a library of model components, supporting model reusability, is introduced.

Many case studies demonstrating these and other benefits have appeared in the literature [33, 35, 55, 23, 41, 1, 21], and examples have been given in earlier chapters.

However, almost as quickly, it became clear that SPA models are prone to problems of state space explosion: making it easy for the modeller to represent systems in detail, coupled with the inherent complexity of the systems of interest, inevitably leads to models which are extremely large; in many cases, intractably so. In particular, coupled with the abstraction provided by the hiding combinator, compositionality allows a modeller to represent components of the system in detail, model their interaction in appropriate ways, and then abstract from the internal details of the combined component. This is a good technique for capturing the behaviour of systems. But note that although abstraction reduces the observability of actions, and in some languages reduces the measures that can be made on the model, it does not generally eliminate the internal states. Thus this attractive "feature" of SPA actually exacerbates the state space explosion problem.

In the context of Markovian process algebras (MPA), tackling this problem has been a major motivation of much research. Of course, there are two state spaces which may be considered—the state space of the MPA model, which is generated in the labelled transition system via the operational semantics; and the state space of the underlying Markov process, the model to be solved (see Figure 1). Using the most straightforward procedure for generating the underlying Markov process, there is an isomorphism between the two. We could try to attack the state space explosion problem at either level; indeed, there exists a substantial body of literature considering the problem of state space explosion at the level of the Markov process via a variety of techniques. But in order to benefit fully from the process algebra apparatus we choose to work at the level of the MPA.

Initial efforts to combat the state space explosion problem concentrated on model manipulation techniques—model simplification and model aggregation. These techniques aim to *improve*, from the perspective of solution, the underlying Markov process, via manipulations of the state space at the MPA level. The simplest form of improvement is a reduction in the number of states. However, there are other possibilities, as we will discuss at the end of the chapter. At the core of these techniques are equivalence relations, but compositionality also has an important role to play, as we will discuss in Section 3.

Unfortunately model manipulation alone still leaves us with a Markov process which must be solved numerically as a single entity. This leads to the inevitable question of how the compositional structure within MPA models relates to the various decomposed solution techniques which can be applied to Markov processes. We survey MPA work in this area in some detail in Section 5. Within the possible decomposed solution techniques we will focus primarily on *product form approaches*. These offer *exact solution* in the sense that the steady state distribution of the original model can be recovered, when the decomposition satisfies certain (stringent) criteria. Nevertheless we will also discuss some approximative decomposed solution techniques, which generally have wider applicability, in Section 6.

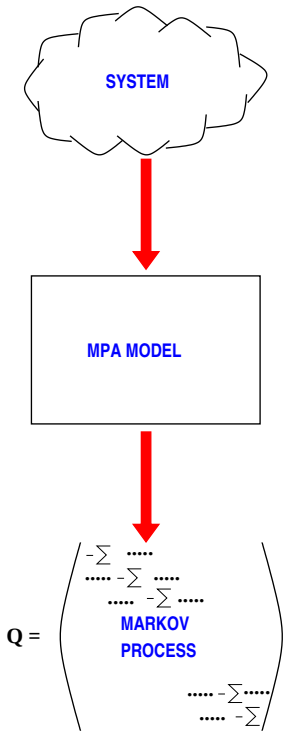


Fig. 1. Schematic view of MPA modelling methodology

In both approaches the work often exhibits common elements, although different researchers concentrate on different aspects of this general framework. In order for a decomposed solution technique to be fully developed and exploited in a tool the following steps have to be established:

- *Characterisation of structures within the MPA model which correspond to decomposable structure in the underlying Markov process.* The aim of this work is to identify those MPA models which have a structure which is amenable to decomposed solution. In some cases this is fairly informal; in others the aim has been to establish syntactic rules which may be applied automatically, meaning that a tool may analyse a given model and decide whether it falls within the decomposable class or not.
- *Development of revised algorithms to generate the decomposed Markov process (usually as a set of Markov processes) from the MPA model.* Identifying the set of Markov processes susceptible to efficient solution is only the first step. In the "standard" solution algorithm there is a straightforward mapping from the semantic model to the state space of the underlying Markov process. If this process is to be decomposed, more sophisticated Markov process generation algorithms may be needed. Once the composite Markov pro-

cesses are formed, in some cases known solution algorithms can be applied; in some existing techniques can be modified; in others, new approaches to decomposed solution have been suggested by the process algebra structure. In the latter case, the new solution algorithms must also be developed.
- *Implementation of related algorithms.* Prototype tools and implementations of the new techniques, for characterisation, Markov process generation and decomposed solution, allow them to be tested in practice, and eventually, put to practical use.

Each of the decomposed solution techniques is limited in its scope of application if corresponding results and performance measures are not to be too seriously compromised. However, the real opportunity to exploit the benefits of decomposed solution is likely to arise when the decomposition structure is used as a target for model manipulation. In this case a given model is manipulated into a form which is more amenable to efficient solution, not by reducing the number of states *per se*, but by manipulating its structure into a form suitable for a decomposed solution. This is a major area of current work and will be discussed in more detail in Section 7.

As mentioned earlier, the focus of this chapter will be a class of decomposed solution techniques which result in exact product form solution of the steady state distribution. All work in this area for MPA has been carried out in the context of the MPA language PEPA (Performance Evaluation Process Algebra). Consequently, in the following section, we give a brief introduction to PEPA; the interested reader is referred to [33] for more details.

2 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. The essential idea of these combinators will be familiar from earlier chapters, but they are included here for clarity.

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

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

Cooperation: $P \underset{L}{\bowtie} Q$ The components proceed independently with any activities whose types do not occur in the *cooperation set* L (*individual activities*). However, activities with action types in the set L require the simultaneous involvement of both components (*shared activities*). These activities are only enabled in $P \underset{L}{\bowtie} Q$ when they are enabled in both P and Q. 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 [31]. 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. 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: $A \stackrel{\text{def}}{=} P$ associates the constant A with 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 [33] 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.

Example: 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.

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

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

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: $(Process \parallel Process) \underset{\{use\}}{\bowtie} Sub$.

2.2 Execution strategy

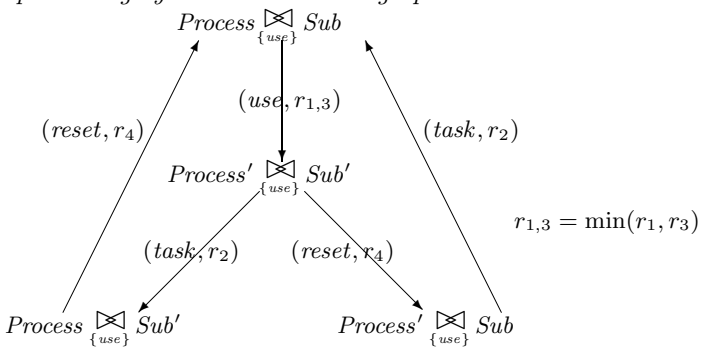
A *race condition* governs the dynamic behaviour of a model whenever more than one activity is enabled. This has the effect of replacing the non-deterministic branching of classical process algebra with probabilistic branching. The probability that a particular activity completes is given by the ratio 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 $(\mathcal{C}, \mathcal{A}ct, \{\!| \xrightarrow{(\alpha,r)} | \ (\alpha, r) \in \mathcal{A}ct \ |\!\})$ 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: Simple processing system – derivation graph



Definition 1 (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 (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 3 (Derivation Graph). *Given a PEPA component* C *and its derivative set* $ds(C)$*, the* derivation graph $\mathcal{D}(C)$ *is the labelled directed multi-graph, 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.*

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 [52] and has been discussed in an earlier chapter. 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 $\mathcal{A}ct(C_i|C_j)$ is defined to be $\{\!|a \in \mathcal{A}ct(C_i) \mid C_i \xrightarrow{a} C_j|\!\}$. 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

When aiming for a decomposed solution which may rely on the numerical solution of individual components within the model, it is important to ensure that these components, 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 [33, 24]. These conditions imply that the model must be a *cyclic* PEPA component.

Definition 4 (Cyclic Components). *A PEPA component is* cyclic*, or* irreducible*, if it is a derivative of all the components in its derivative set.*

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

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:

$$P ::= S \;\mid\; P \underset{L}{\bowtie} P \;\mid\; P/L$$
$$S ::= (\alpha, r).S \;\mid\; S + S \;\mid\; A$$

For the remainder of the chapter 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 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 subsection the "states" of a PEPA model as it evolves are the syntactic terms, or derivatives, which the model will go through. For a model component the sequential components it consists of will be apparent in every derivative of the model[1], i.e. the sequential components 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 compact representation of any particular state.

Definition 5 (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 6 (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.

3 Harnessing compositionality

As outlined in Section 1, initial attempts to exploit the compositionality of MPA languages focused on model manipulation—improving the model in some sense before the underlying Markov process is generated and solved.

There have been two principal approaches to model manipulation:

model simplification: Here an equivalence relation is used to establish behavioural or observational equivalence *between models*. The aim is to replace one model by an equivalent one which is more desirable from a solution point of view. Once the desirable model has replaced the original, the underlying Markov process is generated as usual, associating one state with each node in the labelled transition system generated by the semantics. Equivalence relations which have been used in this way are *weak isomorphism* in PEPA [33, 14], *Markovian bisimulation* and *weak bisimulation* in TIPP [47].

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

model aggregation: Here an equivalence relation is used to establish equivalence *between states within a model*. The aim is to use an alternative mapping from the labelled transition system, given by the semantics of the model, to the underlying Markov process. The equivalence relation is used to partition the nodes of the labelled transition system into equivalence classes. Then, instead of the usual one-to-one correspondence between nodes and states, one state in the underlying Markov process is associated with each equivalence class of nodes. The equivalence relation which has been used in this way is variously called *strong equivalence* (PEPA) [33], *Markovian bisimulation* (TIPP) [29], and *extended Markovian bisimulation equivalence* (EMPA) [4].

In model aggregation the introduction of equivalence classes will generally reduce the number of states in the underlying Markov process and will certainly never increase it. Thus the resulting Markov process is more amenable to solution because its size has been reduced. Similarly, in model simplification reducing the number of states is the most straightforward way to make a model more desirable from a solution point of view; this is the approach taken in the work on weak isomorphism in PEPA.

Note that in general, when model manipulation is based on an equivalence relation which captures all relevant aspects of the behaviour of a model the subsequent solution of the transformed model will be exact [33, 29, 4]. However, when a partial order relation or an equivalence relation which does not consider all aspects of represented behaviour is used, model manipulation may result in an approximation of the original model. This is the case when weak bisimulation is used with respect to a subset of TIPP in Mertsiotakis's work on throughput approximation (see Section 6.2): the weak bisimulation relation cannot capture the timing characteristics of the models.

In the context of model manipulation, the role of compositionality is perhaps secondary to the equivalence relations which are used to define the model transformations. Nevertheless it is an important role, distinguishing the use of the model manipulation techniques in the MPA setting from their direct application at the Markov process level. Using process algebra apparatus we can establish the congruence of an equivalence relation as a feature of a language. The consequences of this form model manipulation are significant: components within the model may be manipulated, and *improved*, in isolation. The modified model is formed as the composite of the modified components. Thus the state space of the complete model may never need to be constructed [33, 32, 29]. This greatly reduces the complexity of the procedure and ultimately, may make intractable models tractable.

Another approach which has been taken to exploit MPA model compositionality during Markov process generation, is the use of tensor algebra. Again, the objective is to tackle the problem of state space explosion but the strategy is to alter the representation of the underlying Markov process. Instead of capturing the Markov process as a single infinitesimal generator matrix, using tensor algebra it is represented as an expression of smaller matrices. Specially designed algorithms are able to take advantage of this expression and find the steady state

solution in terms of the expression, avoiding the construction of the complete matrix. This general approach has been pioneered by Plateau and others with *Stochastic Automata Networks* [53, 54, 58].

In [12], Buchholz identifies the relationship between the parallel composition operator in the SPA language, *MPA*, and tensor algebra expressions for the underlying Markov process. He shows how an expression for the complete model can be constructed in terms of smaller matrices representing the individual components and the synchronisation sets in operation between them. However, the form of synchronisation defined in the semantics of *MPA* is slightly unusual and tailored to the tensor representation. Similarly, in [55], Rettelbach and Siegle construct a minimal compositional semantics for a subset of TIPP, called TIPP^{MS}. Specifically, this language includes a synchronising replication operator but not parallel composition in its general form. In this work a matrix is defined for each language expression without recourse to an operational semantics and the associated labelled transition system. This is achieved by constructing the matrix from sub-matrices corresponding to terms in the expression using matrix operators corresponding to the process algebra combinators. In particular, the replication operator maps to the tensor sum of replicated copies of the matrix corresponding to the replicated process.

More recently, in [36], Hillston and Kloul present a representation of the Markov process underlying a PEPA model in terms of a tensor product of terms. Unlike the earlier work by Buchholz, Rettelbach and Siegle, this is not based on an especially designed, syntactically restricted, form of the process algebra. Whilst the representation is similar to previous representations of Stochastic Automata Networks and Stochastic Petri Nets, it has novel features, arising from the definition of the PEPA models. In particular, capturing the correct timing behaviour of cooperating PEPA activities relies on a new tensor operator defined within the paper [36].

We do not classify this work as decomposed solution since the underlying Markov process is still solved as a single entity although it is represented in a decomposed form. In the following sections we survey work which we classify as truly decomposed solution. In these cases the MPA model is used to generate not one Markov process but several, and these processes are solved separately.

4 Decomposed solution

A variety of decompositional or structural techniques have been proposed to aid in the solution of large Markov processes. Recently several results have been published which show that, at least for some particular cases, there is a clear relationship between these techniques and MPA model descriptions. In the following two sections we survey some of these recent results. In this section we aim to provide a general introduction and framework for decomposed solution of MPA models.

The overall approach is summarised schematically in Figure 2. A model is considered to be made up of a number of submodels $m_1, m_2, \ldots, m_n$, and any

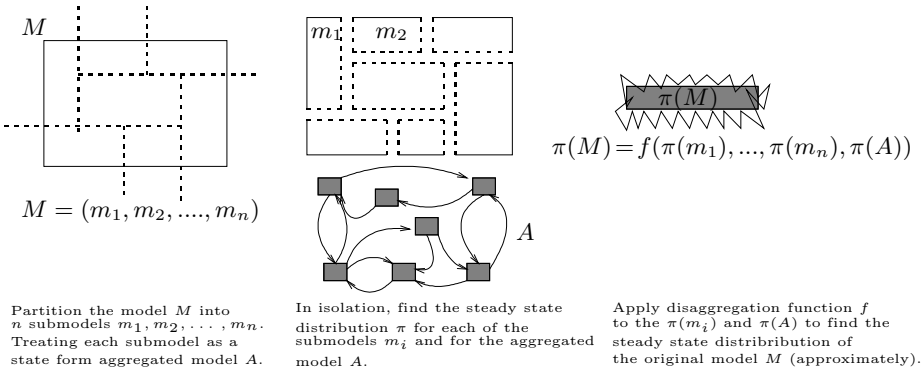


Fig. 2. Schematic overview of decomposed solution

state of the model can be expressed in terms of the local states in each of the submodels. Note that in general these submodels will correspond to regions of the state space of a model and may or may not correspond to components within the system being represented. Although they may not be independent, a decomposed solution will typically solve each of these submodels in isolation, generating a corresponding steady state distribution $\pi_i(\cdot)$ over the state space of the submodel m_i. In addition an *aggregated* model may also be formed, aiming to capture the pattern of interactions between submodels. In this model each submodel is represented by a single state and the transitions between states are intended to capture the points when the complete model moves between submodels (regions of the state space). A steady state distribution for this aggregated model is also generated. An approximate steady state distribution of the original model is then obtained by applying a *disaggregation function* to the results of the separate analyses. How the submodels are identified, the formation of the aggregated model and the definition of the disaggregation function are all particular to the decomposed solution technique.

Many such techniques are well-known at the Markov process level. The advantage of characterising the corresponding class of MPA models 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, thus avoiding the construction of the original Markov process.

4.1 Product form solutions

It is clear that there is great advantage to be gained if the compositional structure of a MPA model can used during model solution. This would mean that the submodels within the general scenario shown in Figure 2 would correspond directly to components within the MPA model, i.e. the components of the MPA model could be solved separately and their solutions combined to obtain a solution, exact or approximate, of the whole Markov process. One class of Markov

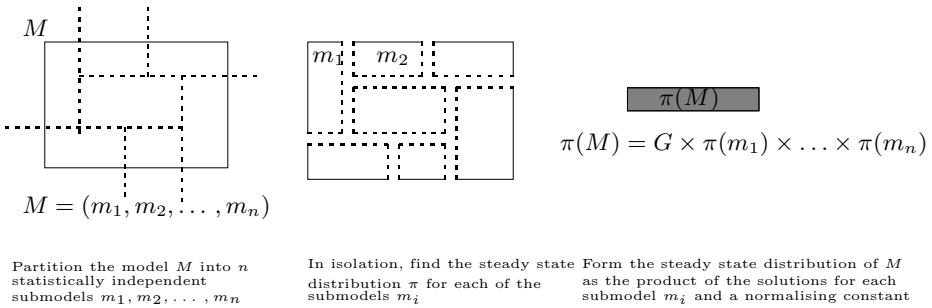


Fig. 3. Schematic overview of product form solution

processes which are susceptible to such an efficient solution technique are those which exhibit a *product form* steady state distribution. Significant effort has gone into identifying those classes of PEPA models in which the constituent components will generate a product form solution.

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. In this case the general scenario simplifies to that shown in Figure 3. Note that the aggregate model is no longer constructed or solved, and that disaggregation function is a simple multiple of the steady state probabilities of the submodels. Moreover this technique generates an *exact* solution up to a normalisation constant.

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 [42] and their generalisation, BCMP networks [3], have been widely employed. Here the underlying Markov process is known to have a reversible or quasi-reversible structure.

In contrast stochastic Petri nets (SPN) have rarely been found to be amenable to such efficient steady state solution, except when some of the expressiveness of the formalism is reduced, for example by excluding resource sharing and competition over resources in a general form. By imposing these restrictions, 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 [27]. Lazar and Robertazzi establish a first step towards a product form over *subnets*, characterising independence between subnets which compete for resources [45]. Donatelli and Sereno show how both these approaches are related to T-semiflows in the Petri net [20]. An excellent survey of product form results for Petri nets can be found in [57].

Work on finding PEPA models which give rise to product form solutions has drawn on the previous work on both queueing networks and SPNs. Essentially this can be seen as an investigation of when components *interact* and yet remain *statistically independent*. It is clear that when any MPA model consists of completely independent sequential components, i.e. $P \parallel Q$, the equilibrium distribution will have a product form:

$$\pi(P \parallel Q) = \pi_P(P) \times \pi_Q(Q) \tag{1}$$

where π_P and π_Q are the steady state distributions over the local states of P and Q respectively. However few real systems consist of components which are independent in this way, and if they did the state space explosion problem would not arise because it would be obvious that the components could be analysed separately. The challenge is to find circumstances in which components P and Q which synchronise, $P \underset{L}{\bowtie} Q$ in PEPA notation, still exhibit statistical independence.

4.2 Other decomposed solution techniques

In the following section we consider some techniques for *aggregated decomposed* solutions. Here, it is not simply a case of splitting the model into submodels or components in the style of product form. As well as a stochastic representation of each of the components, the decomposed solution involves a stochastic representation of the interactions between these components, the *aggregated model*. In most cases these stochastic representations will be Markov processes but in the work by Bohnenkamp and Haverkort on decomposition via synchronisation points semi-Markov processes are used [8]. In addition to work on characterising the class of MPA models for which the technique is appropriate, the research effort can also involve development of a good disaggregation function, which allows the results of the individual submodels and the aggregated model, to combined to approximate the solution of the original model. In several cases this effort has been based on previous work on SPN.

5 Product form PEPA models

5.1 Reversibility

Informally, a reversible Markov process is one which behaves identically when we observe it with time reversed as when we observe it with time flowing forward. At the Markov process level there are several ways to characterise these processes, but we state only the *local balance* condition. An irreducible, stationary Markov process $X(t)$ is *reversible* if it satisfies the detailed balance equations:

$$\pi(j)q(j,k) = \pi(k)q(k,j) \tag{2}$$

where $q(j,k)$ is the instantaneous transition rate from state j to state k and $\pi(\cdot)$ is the steady state probability distribution. A more complete account of reversibility can found in the book by Kelly [44].

An initial study of MPA models giving rise to reversible Markov processes was presented by Bhabuta *et al.* in [5]. This paper largely considered the problem at the level of the underlying state space. In [38], Hillston and Thomas, identify syntactic conditions which a PEPA model must satisfy in order for the underlying process to be reversible. The problem is tackled in two stages. First, a basic class of sequential components which give rise to reversible structures are identified. These are shown in Figure 4. Then, assuming that a known class of reversible PEPA components exist, the authors investigate under what circumstances the conditions for reversibility will be preserved if reversible components are composed using the combinators of PEPA.

Fundamental to the basic class of reversible sequential components is the notion of a *reverse pair*. A pair of action types $(\alpha, -\alpha)$ form a reverse pair if, in any state, any α transition leads to a state in which a $-\alpha$ transition leading back to the original state. This ability to "undo" any transition in the subsequent transition seems to be fundamental to reversibility. It clear to see that this is a necessary condition for equation (2) to be satisfied. From this starting pointing various canonical forms for sequential reversible components are described. The interesting conditions for when reversible components can be composed without losing the reversibility property relate to cooperation. In [38] detailed conditions, on the form of the cooperation set and the rates of the activities involved, are given. We refer the reader to that paper for more detail.

5.2 Quasi-reversibility

Like reversibility, the origins of *quasi-reversibility* are in queueing theory. It is the condition which allows a wide class of queueing networks to be separated into their individual queues and solved in isolation, provided traffic equations are solved to give appropriate arrival rates at each queue. Formally, a stationary Markov process $X(t)$ is *quasi-reversible* if, for all times t_0 the state $X(t_0)$ is independent of

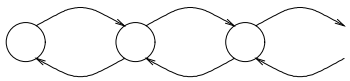
1. the input process after t_0 and
2. the output process before t_0.

Rather than the detailed balance equations which characterised reversibility, a quasi-reversible process satisfies *partial balance equations*:

$$\pi(i) \sum_{j \in S'} q(i,j) = \sum_{j \in S'} \pi(j) q(j,i) \tag{3}$$

for all states i and a corresponding subset of states S'. Again, more details of the definition of quasi-reversibility can be found in the book by Kelly [44].

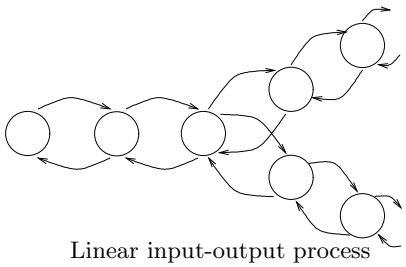
In [25], a PEPA characterisation of this class is presented. As in the work on reversibility, the approach is to first find simple instances of PEPA processes which give rise to quasi-reversible structure in their underlying Markov process (*QR components*). Then, conditions are established under which these components can be composed whilst maintaining the quasi-reversible property. Relative



Birth-death process

$$Gambler_0 \stackrel{\text{def}}{=} (borrow, r).Gambler_1$$
$$Gambler_n \stackrel{\text{def}}{=} (win, w).Gambler_{n+1} + (lose, l).Gambler_{n-1} \quad n \geq 1$$

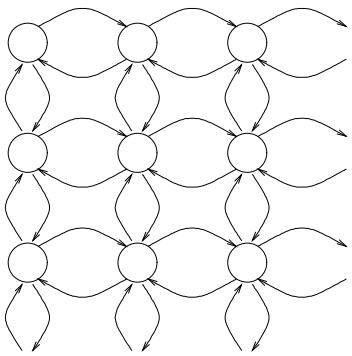
Birth-death component



Linear input-output process

$$Gambler_0 \stackrel{\text{def}}{=} (borrow, r).Gambler_1$$
$$Gambler_1 \stackrel{\text{def}}{=} (win, w).Gambler_2 + (lose, l).Gambler_1$$
$$Gambler_2 \stackrel{\text{def}}{=} (win_1, w_1).Cards_3 + (win_2, w_2).Slots_3 + (lose, l).Gambler_1$$
$$Cards_3 \stackrel{\text{def}}{=} (win, w).Cards_4 + (lose, l).Gambler_2$$
$$Slots_3 \stackrel{\text{def}}{=} (win, w).Slots_4 + (lose, l).Gambler_2$$
$$Cards_n \stackrel{\text{def}}{=} (win, w).Cards_{n+1} + (lose, l).Cards_{n-1}$$
$$Slots_n \stackrel{\text{def}}{=} (win, w).Slots_{n+1} + (lose, l).Slots_{n-1} \quad n \geq 4$$

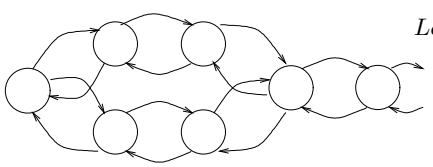
Linear input-output component



Parallel input-output process

$$Gambler1_0 \stackrel{\text{def}}{=} (borrow, r).Gambler1_1$$
$$Gambler1_n \stackrel{\text{def}}{=} (win, w).Gambler1_{n+1} + (lose, l).Gambler1_{n-1} \quad n \geq 1$$
$$Gambler2_0 \stackrel{\text{def}}{=} (borrow, r).Gambler2_1$$
$$Gambler2_m \stackrel{\text{def}}{=} (win, w).Gambler2_{m+1} + (lose, l).Gambler2_{m-1} \quad m \geq 1$$
$$Gamblers_{(n,m)} \stackrel{\text{def}}{=} Gambler1_n \parallel Gambler2_m$$

Parallel input-output component



Symmetrical branch-join process

$$Loser_0 \stackrel{\text{def}}{=} (borrow, r_1).BJ_1 + (borrow, r_2).21_1$$
$$BJ_1 \stackrel{\text{def}}{=} (win, w).BJ_2 + (lose, l).Loser_0$$
$$BJ_2 \stackrel{\text{def}}{=} (win, w).Spin_3 + (lose, l).BJ_1$$
$$21_1 \stackrel{\text{def}}{=} (win, w).21_2 + (lose, l).Gambler_0$$
$$21_2 \stackrel{\text{def}}{=} (win, w).Spin_3 + (lose, l).21_2$$
$$Spin_3 \stackrel{\text{def}}{=} (win, w).Spin_4 + (lose, l_1).BJ_2 + (lose, l_2).21_2$$
$$Spin_n \stackrel{\text{def}}{=} (win, w).Spin_{n+1} + (lose, l).Spin_{n-1} \quad n \geq 4$$

Symmetrical branch-join component

Fig. 4. Basic reversible structures

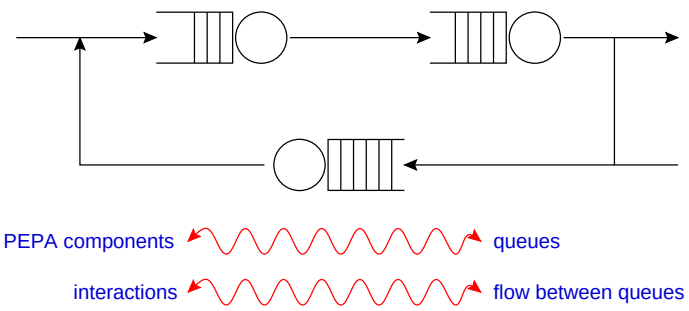


Fig. 5. Relating queueing network concepts to PEPA

to the simple product form PEPA case presented in equation (1), this does allow interaction between the components P and Q. But strong restrictions are placed on the form of this interaction. Again the notion of a *reverse pair* is important. Intuitively, it is easy to see the reverse pair action types as a generalisation of the input-output behaviour of a queue.

Not surprisingly, given the origins in queueing networks, the form of admissible PEPA interaction is a *flow cooperation*. This means that the "positive" half of a reverse pair in one component is carried out in synchronisation or cooperation with the "negative" half of a reverse pair in another. The "positive" actions correspond to the input process in the definition of quasi-reversibility, while the "negative" correspond to the output process. The subsequent theorems, stated below and developed in more detail in [25], correspond to those for open and closed queueing networks reported in [44].

Theorem 2 (Open Quasi-reversible Interactions). *An open flow cooperation of QR components has the following properties at steady state:*

- *The marginal states of the individual components are independent.*
- *For an individual component the steady state distribution and the distribution over states at the time of completion of a positive action of a given type are identical and are both as they would be if the component were in isolation with independent Poisson external sources with rates given by traffic equations.*
- *Under time reversal the model becomes another flow cooperation of QR components; in particular sources become sinks and vice versa.*
- *The underlying Markov process is quasi-reversible. Thus the completion of negative actions of each type at the sinks form independent Poisson processes and the state of the model at time t_0 is independent of the completion instants of the negative actions in the model prior to time t_0.*

Theorem 3 (Closed Quasi-reversible Interactions). *A closed flow cooperation of QR components, $C_1, C_2, \ldots, C_m$ has the following properties at steady state:*

– *Let $\pi(C_{k_i})$ be the steady state probability that component C_k is in state C_{k_i}, $1 \leq k \leq m$, when interacting only with its Poisson external sources with rates given by the traffic equations. Then the steady state probability distribution of the closed flow cooperation is*

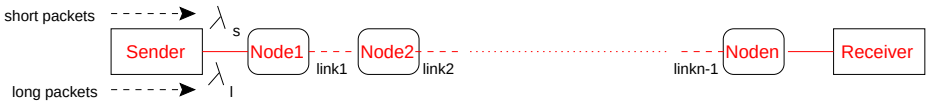
$$\prod_{i=1}^{m} \pi(C_{k_i})/B$$

where B is the normalising constant.

– *The distribution over states at the time of completion of an positive action of type α, which produces an $(\alpha, r)^n$-derivative in a component of the state vector representation, is identical to that above for the closed flow cooperation obtained by reducing n to $n-1$.*

– *Under time reversal the model becomes another closed flow cooperation of QR components.*

– *The underlying Markov process is quasi-reversible. Thus the completion of negative actions of each type at the sinks form independent Poisson processes and the state of the model at time t_0 is independent of the completion instants of the negative actions in the model prior to time t_0.*



$$System \stackrel{\text{def}}{=} Sender \underset{\{snd_{sht}, snd_{lg}\}}{\bowtie} Network \underset{\{dlvr_{sht}, dlvr_{lg}\}}{\bowtie} Receiver$$

$$Network \stackrel{\text{def}}{=} Node1_{0,0} \underset{\{lk^1_{sht}, lk^1_{lg}\}}{\bowtie} Node2_{0,0} \underset{\{lk^2_{sht}, lk^2_{lg}\}}{\bowtie} \cdots \underset{\{lk^{n-1}_{sht}, lk^{n-1}_{lg}\}}{\bowtie} Noden_{0,0}$$

$$\vdots \quad \vdots \quad \vdots$$

$$Nodek_{i,j} \stackrel{\text{def}}{=} (lk^{k-1}_{sht}, \top).Nodek_{i+1,j} + (lk^{k-1}_{lg}, \top).Nodek_{i,j+1} + (lk^{k}_{sht}, \mu_{k1}).Nodek_{i-1,j} + (lk^{k}_{lg}, \mu_{k2}).Nodek_{i,j-1}$$

$$\vdots \quad \vdots \quad \vdots$$

$$\pi\left(Node1_{i_1,j_1}, \ldots, Nodek_{i_k,j_k}, \ldots, Noden_{i_n,j_n}\right) = \prod_{k=1}^{N} \left[\left(\frac{\lambda_s}{\mu_{k1}}\right)^{i_k} \left(\frac{\lambda_l}{\mu_{k2}}\right)^{j_k} \left(1 - \frac{\lambda_s}{\mu_{k1}}\right) \left(1 - \frac{\lambda_l}{\mu_{k2}}\right) \right]$$

Fig. 6. Quasi-reversibility Example: A simple packet transmission network

Figure 6 shows a simple packet transmission network expressed as a PEPA model. This model satisfies the quasi-reversibility characterisation. Each node is a QR component, enabling two positive actions on its outgoing links and two corresponding reverse actions on its incoming links. The cooperation between $Node k$ and $Node(k+1)$ is a flow cooperation since a positive action in one cooperates with a negative action in the other. The sender is a source for the system and the receiver is a sink. Thus we have an open quasi-reversible interaction. As a consequence of the characterisation we know that the steady state distribution of the complete model can be expressed as a product of terms, one corresponding to each node in the network, as shown.

5.3 Routing process approach

Sereno's work, reported in [56], derives product form criteria for PEPA models based on earlier work on product form criteria for SPN [26, 27, 11]. The SPN results rely on defining a Markov chain whose states correspond to the transitions of the SPN, the so-called *routing chain*. The condition for this chain to exist is that the set of places into which tokens are placed when a transition fires should be exactly the input places of another transition. This condition places severe restrictions on the forms of synchronisation and resource contention which can be represented in the net.

In [56], Sereno uses a vector representation of the state of a PEPA model[2] in the characterisation of the class of models which have a product form based on the *routing process*. It is assumed that some preprocessing of the model is done in order to collect information and to aggregate the model, using one of the techniques outlined in Section 3. The information which is needed is the local state space of each component, and, for each action type of the model, which local states of participating components enable the action and which appear after it has been performed. These two sets of local states are called the *pre-set* and *post-set* of the action, respectively. The state vector representation is composed of sub-vectors, one corresponding to each defined sequential component; an element within the sub-vector corresponds to a local state within corresponding component. In the representation of any particular state the value of an element within the vector records the number of instances of each local state exhibited in the current syntactical state of the model. Storing the state in this form, together with the pre- and post-sets of actions represented as vectors, allows the effect of completing an action to be written down in vector form.

There are several restrictions placed on the PEPA models, in particular with respect to action types. An action can only have one pre-set—this implies that each action within the behaviour of a sequential component must have a distinct name. Moreover if actions of the same type occur within different sequential components they must be synchronised.

Sereno's approach for PEPA is completely analogous to the earlier work on SPN—he defines a Markov chain in which the states correspond to the actions of

[2] Note, however, that this differs from the state vector defined in Definition 5.

the PEPA model. This is called the *routing process*. The global balance equations of the routing process correspond to the traffic equations of queueing networks. If the state space of this process can be partitioned into equivalence classes of enabling actions (roughly speaking, one action *enables* another if the post-set of one is the pre-set of the other; we take the transitive closure of that relation), then a product form solution exists. Moreover the partition forms the basis for the decomposition.

5.4 Product form over submodels

As mentioned earlier, in [45] Lazar and Robertazzi investigate a product form in the context of SPN—the decomposition is carried out over subnets, which may still need to be solved by standard numerical techniques to find their local steady state.

In [10], Boucherie generalised their result and characterised the class of underlying Markov processes. Such a process is formed as the product of a set of Markov processes which compete over a set of resources. The resources are not explicitly represented but the competition has two important impacts on the state space and the transition rates of the product process. Firstly, if two constituent processes compete over a resource they cannot both enter the region of their state space representing possession of the resource at the same time. Thus areas of the state space of the product process are eliminated. It is assumed that the product process will change state in only one of the constituent processes at each state change. The second effect of the competition over resources is to limit this still further—if two processes compete over a resource, and one of them is currently holding the resource, then the other cannot make any state change, no matter where it is in its state space. Thus when a constituent process holds a resource, all competing processes are blocked.

This form of interaction is illustrated by the product process shown in Figure 7. Here we assume that we have two Markov processes S_1 and S_2. The state space of each process is partitioned into those states in which no resource is being used (A_{i0}) and those in which the resource r is being used (A_{ir}). Immediately we see that the bottom right hand corner of the product state space has been eliminated because the processes cannot access the resource r simultaneously. Moreover, when, say, S_1 holds the resource (is in partition A_{1r}), we see that process S_2 is unable to make any transitions at all, not even those not involving resource use.

Essentially the product form result holds because in each state of the product process each constituent process satisfies its own global balance equations. If it can make a transition it is free to act as if it were independent; alternatively, it is completely blocked and satisfies its global balance equations trivially.

In [34, 39], Hillston and Thomas characterise this class of Markov processes in PEPA. The class of models that they identify consist of independent components, which give rise to the constituent Markov processes of the underlying Markov process. These components interact indirectly by synchronisation with *resource components*. Compared with the simple product form model presented

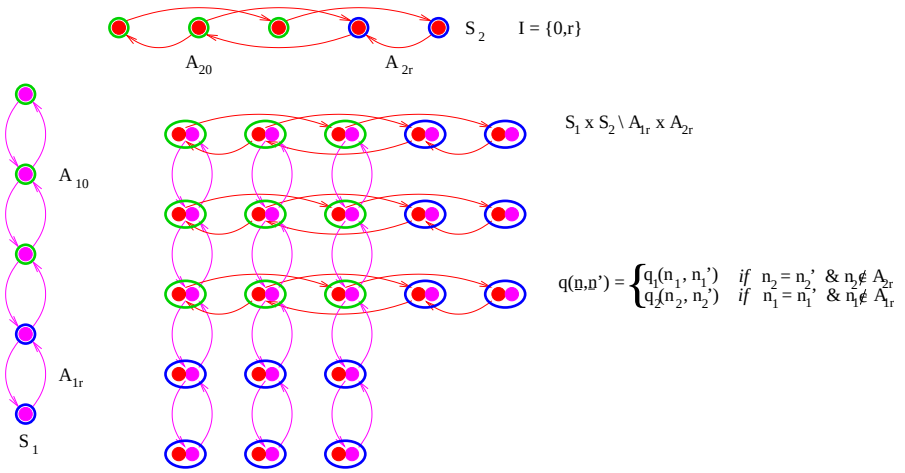


Fig. 7. Simple example of two-dimensional resource contention Markov process

in equation (1), the general form of these process algebra terms and the resulting product form is, schematically:

$$\pi\left((P \parallel Q) \underset{L}{\bowtie} R\right) = B \times \pi_P(P \underset{L}{\bowtie} R) \times \pi_Q(Q \underset{L}{\bowtie} R) \tag{4}$$

where the component R represents the resource, π_P and π_Q are the steady state distributions over the derivatives of $P \underset{L}{\bowtie} R$ and $Q \underset{L}{\bowtie} R$ respectively, and B is the normalising constant. The decomposition is formed by considering each of the model terms (P and Q in this case) acting in synchronisation with the resource (R) in isolation.

In PEPA, a component is termed a *resource* if it is never free to act independently; all its activities must be carried out in synchronisation with the rest of the model. All components are assumed to have cyclic behaviour. In this context a component is considered to be using or holding the resource if it has carried out the first action of the resource's behaviour in synchronisation with the resource. The semantics of PEPA ensure that the state space of the product process is suitably modified, i.e. that two competing components cannot hold the resource simultaneously. In order to ensure that the correct condition is also satisfied by the transition rates of the product process, Hillston and Thomas place a further restriction on the cooperation set in operation between the resource component and the rest of the model. If a model component wishes to use the resource during one of its possible behavioural cycles, it must gain control of the resource at the start of the cycle and release it only at the end. This guarantees that other competing components will be blocked when the component holds the resource. Although presented here informally, these conditions are defined

as formal syntactical conditions which can be checked on the model specification [39].

Any resource component is redundant within the state vector representation of a model, in the sense given in Definition 6. This means that the resource can be eliminated from the state vector and the process will then satisfy the condition that only one constituent component changes its local state for each global transition.

The product form PEPA models discussed in previous subsections have been based on components of a particular structure which interact in a restricted way, preserving a form of independence between the components. Here the characterised models represent the competition of otherwise independent components over resources. The form of these components is not restricted; however they do place a relative restriction on the form of the resource and on the form of the cooperation set in operation between the resource and the rest of the model. As suggested by equation (4) above, these components, together with the resource, are solved in isolation, these partial solutions subsequently being combined to give a solution of the complete model. The outstanding problem of this approach, however, is the calculation of the normalising constant.

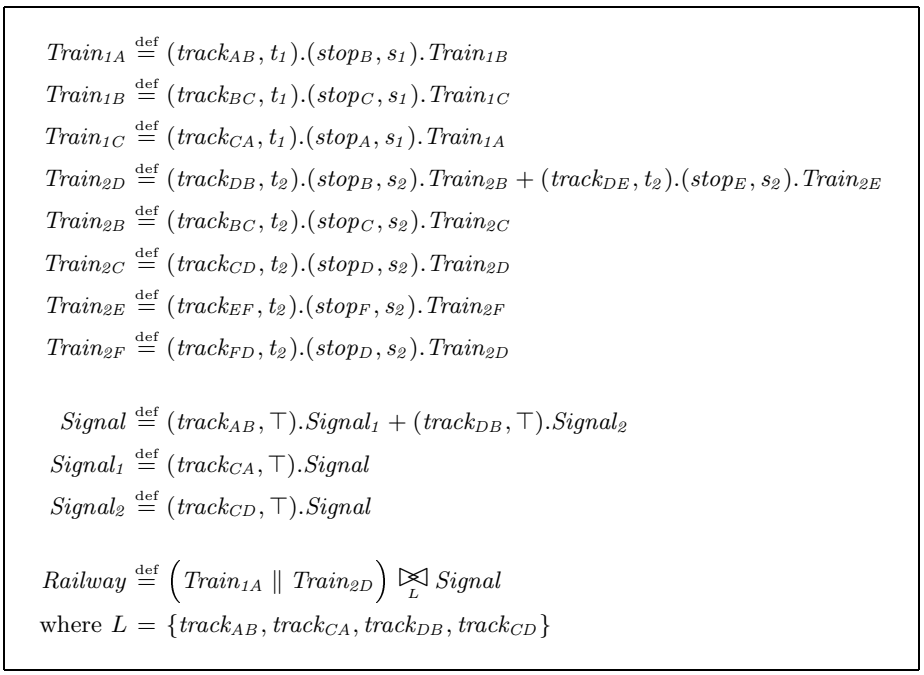


Fig. 8. A small railway system

Figure 8 shows a PEPA model which satisfies the criteria for this form of product form. 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 9).

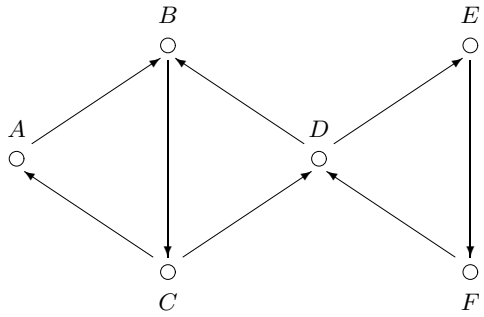


Fig. 9. Schematic representation of the small railway system

The trains are independent of each other but must compete for access to the shared piece of track, as controlled by the signal. In this model the signal is the resource component. It is straightforward to see that it satisfies the condition that it never acts independently, and that its local state can always be deduced from the local states of the two trains. Moreover, for each train, any cycle of behaviour involving the signal can only begin when cooperation with the signal is possible, and ends by *releasing* the signal again.

The models presented in [10], including those presented as SPN, relied on the insight of the modeller to detect the product form structure. Moreover, in the case of the SPN 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.

5.5 Queueing discipline models

Queueing discipline models were introduced by Clark in [15] and presented in more detail in his recent thesis [16]. The initial motivation for Clark's work was the investigation of when a PEPA model could be shown to be *insensitive*[3] to the distribution governing activity duration. In this way he intended to increase the expressiveness of PEPA by removing the restriction to exponential distributions

[3] When a Markov process is *insensitive* its steady state distribution is unaffected by the form of distribution used to determine state sojourn times, and affected only by the mean of such distributions.

for some activities within appropriately structured models. It is well-known from queueing theory that there is a close link between insensitivity and product forms, so it is perhaps unsurprising that Clark's insensitive structures also give rise to models which have a product form steady state solution.

Queueing discipline models are constructed using a new PEPA combinator $Q_{A,\xi}(\cdot)$, the *queue discipline combinator*. This is a *derived* combinator meaning that it can be defined in terms of the existing combinators of the language and so does not necessitate any modification of the semantics of the language. In the combinator syntax the set A denotes a set of *queueing actions* and ξ denotes a set of rates; the argument to the combinator is a set of independent, complete sequential PEPA components, $S_1, \ldots, S_n$. For example, in a model in which the set of rates is left unspecified, the derived form is:

$$Q_A(S_1, \ldots, S_n) \stackrel{\text{def}}{=} (S_1 \parallel \cdots \parallel S_n) \underset{M_A}{\bowtie} R_A$$

for suitably chosen M_A (the *arbiter synchronisation set*) and R_A (the *arbiter*).

Intuitively, the components S_i proceed in parallel except when they wish to carry out an activity whose type appears in A. When this occurs the component must *queue* to carry out the queue action, possibly waiting for other components queueing on the same action to complete it first. The arbiter imposes the necessary FCFS queueing discipline. This is shown schematically in Figure 10.

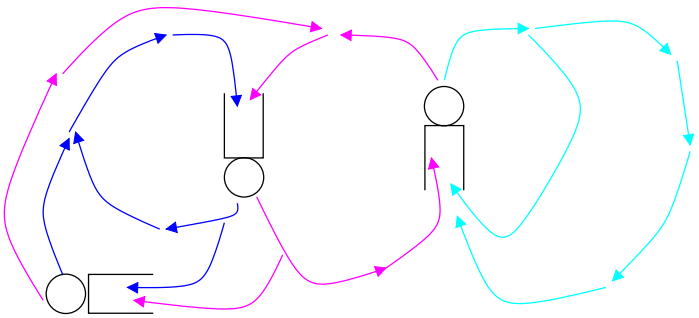


Fig. 10. Schematic view of queueing discipline models

If a model is constructed in this way, Clark has shown that it will have a product form steady state solution, as stated in the following theorem (see [16] for more details).

Theorem 4 (Solution of queueing discipline PEPA models). *Consider* $P \equiv Q_\chi(S_1, \ldots, S_n)$*, assuming that there are* N *queues and that the rate of leaving queue* j *when there are* i *processes waiting is* ξ_{ij}*. Then the steady state*

solution of P is given by

$$\pi(P) = B \prod_{i=1}^{n} \pi_i(S_i) \cdot \prod_{i=1}^{N} \prod_{j=q_i+1}^{d_j} \xi_{ij} \cdot \prod_{\substack{i=1 \\ S_i \in \Theta}}^{n} \sum_{(\alpha,r) \in \mathcal{A}ct(S_i)} r$$

where B is a normalising constant.

One of the major differences between this class of product form models and those considered earlier in the section is that the product form of the solution of the models is guaranteed by construction, and does not have to checked against a syntactic characterisation. In other words the queueing discipline combinator can be regarded as a product form combinator, i.e. it specifies interaction between components in such a form that their independence, in terms of the steady state distribution, is preserved.

Figure 11 shows a PEPA model of a database locking system, such as might be found in a transaction processing system. This model is constructed using the new combinator. It consists of components to represent a collection of transaction classes. A manager, which enforces locking rules on database objects is represented implicitly by the queueing discipline and the queueing activities. A transaction in the model may either attempt to

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

This behaviour can be seen in the representation of the transactions.

The model has a product form steady state distribution as described by Theorem 4. Moreover, Clark's insensitivity result means that the *think* activity associated with each transaction class is not restricted to have an exponentially distribution. For example, the duration of this activity can be set to be deterministic.

The product form solution of the queueing discipline structure suggests a link to the well-known class of BCMP queueing networks [3]. These PEPA models appear to capture infinite server and FCFS service stations. However, note that the PEPA model views the model *from the customer's perspective*, i.e. the behaviour of the system is captured as a set of states recording the point in its cycle round the system that each customer has reached. Using this view of the system, intuitively speaking, each non-queueing activity corresponds to an infinite server station, whilst each queueing activity corresponds to a FCFS service station. More explanation of this mapping can be found in [17].

5.6 Quasi-separability

A quasi-separable Markov process does not have a product form solution in the sense of the other classes of models considered in this section. However we

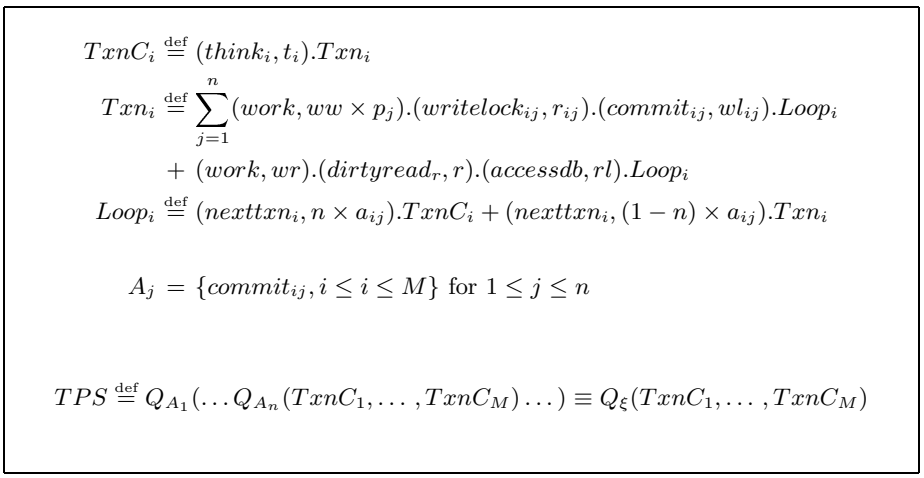


Fig. 11. Transaction processing system model with queueing discipline structure

discuss this approach here because it has more similarities with the product form cases than with the aggregated decomposed solutions considered in the following section. Unlike the case with product form processes it is not possible to find the exact solution of the steady state distribution of a quasi-separable process as a product of the local steady state distributions. Nevertheless decomposed solution can lead to exact results for the local steady state distributions and many performance measures, and no aggregated model needs to be formed.

As with reversibility, quasi-reversibility and queueing discipline, the notion of quasi-separability is one which has been developed in relation to queueing networks, in particular queueing networks in which breakdowns occur [51, 50]. It is assumed that the Markov process is comprised of a number of components and that there are two pertinent pieces of information for each component. A representation of the whole process can then be formed as a pair of vectors, each vector capturing one piece of information for each component. For a process to be *quasi-separable* it must be possible to analyse the behaviour of a component, say component i, given the ith element of the first vector and all elements of the second, or vice versa. This allows the complete process to be reduced to a number of sub-models, each of which contains all the information about exactly one component. In the queueing network context the two pieces of information about each queue are typically its operational state and the number of customers present.

In [60], Thomas and Gilmore present a characterisation of PEPA models which are quasi-separable. It is assumed in this characterisation that the information which must be included in each decomposed submodel is not distributed between the components but maintained by a single *scheduler* component. There are several conditions on the way in which this component may interact with

the other components of the model, which do correspond to the components of the system. When the scheduler changes its state it must do so by an individual action, or by a shared action in which the other participant is passive. Furthermore the individual components have no direct interaction between them—they must be in parallel composition with no synchronisation. In other words, each of these components interacts only with the scheduler. This bears some similarities both with the scenario for product form over submodels and for the queueing discipline structure. However, note that the behaviour of the scheduler and the resource, in relation to the rest of the model, are quite different. The model is decomposed into a set of models, each comprising of a single component considered with the scheduler, in isolation. More details can be found in [60, 59].

6 Aggregated decomposed solutions of MPA models

6.1 Time scale decomposition

The work on time scale decomposition in MPA is based on the notion of *near completely decomposable* Markov processes [18], and inspired by previous work on time scale decomposition of SPN models [6, 2]. A characterisation of a near completely decomposable Markov process at the matrix level is that the matrix is block structured with elements in the diagonal blocks being at least an order of magnitude larger than elements in the off-diagonal blocks. This implies that the model is made up of subsystems whose internal interactions are much more frequent than the interactions between subsystems. As a consequence it can be assumed that the subsystems reach an internal equilibrium between external interactions. Thus a steady state for each Markov process corresponding to a diagonal block of the original process is found; the interactions are modelled by an aggregated model capturing the interactions between subsystems as represented by the off-diagonal blocks. The aggregated model has one state for each subsystem/diagonal block. There are known error bounds for the technique based on the magnitude of the largest element in an off-diagonal block.

The initial classification of MPA models susceptible to time scale decomposition [37], relied on a classification of the sequential components within a model into *fast* or *slow*; this in turn was based on a classification of all actions relative to some threshold rate. A component is considered to be fast if it enables fast or passive actions; a component is considered to be slow if it enables only slow actions. Only models consisting solely of fast and slow components could be analysed. The state of such a process was represented by its state vector, ordered in such a way as to emphasise the distinction between fast and slow components. Thus a model P which is comprised of k fast and ℓ slow components may be represented as: $P \equiv (F_1, \ldots, F_k, S_1, \ldots, S_\ell)$. Then each decomposed component corresponds to a set of state vectors which exhibit the same derivatives in all the slow components:

$$A_{[S_1,\ldots,S_\ell]} \equiv \{(F'_1, \ldots, F'_k, S'_1, \ldots, S'_\ell) \mid S'_1 \equiv S_1, \ldots, S'_\ell \equiv S_\ell\}$$

The elements of this set are found using the semantics of the language when the original model is considered in composition over a synchronisation set which blocks all the slow actions. Finding other decomposed components, and constructing the aggregated model, is achieved by removing this blocking synchronisation and allowing the current submodel to evolve just one step by a slow action. Note that the aggregated model does not have a representation at the MPA level, only as a Markov process.

Later work by Mertsiotakis [46, 47], extends the class of models to which the technique can be applied by tackling the problem of *hybrid* components. These are sequential components which cannot be classified as either fast or slow since they enable both fast and slow actions. Mertsiotakis' solution is to extract the slow behaviour of the hybrid into a separate *shadow* component, making the original component passive with respect to these actions. In [47] it is shown that such a transformation preserves the behaviour of the hybrid component, and, since the equivalence relation is a congruence, the behaviour of the complete model. Once the transformation has been completed the original procedure of [37] can be applied.

6.2 Decision free processes

Another decomposed solution technique presented in Mertsiotakis' thesis allows throughput approximation for a class of MPA models termed *decision free processes*. This technique, developed with Silva in [48, 49], is based on earlier work on throughput approximation in a class of SPN called *marked graphs* [43]. Essentially, the idea is to partition the model into components, typically two in the marked graph case. Decomposed models are then formed in which one component is fully represented while the other is reduced to a minimal form, usually consisting of a single transition. In addition to these two decomposed models there is also an extremely simple aggregated model, consisting of the two minimal forms linked appropriately. An iterative scheme is then used to find a solution to the model, the influence of one component on the other being represented in the decomposed model by the rate of the transition in the minimal form.

The MPA decision free process approach to throughput approximation [48, 49, 47] relies on the decomposition of a decision free process into three components, one of which acts as an intermediary between the other two. This component is distinguished as the *interface*. Note that the components do not necessarily correspond to sequential components (c.f. time scale decomposition). The decomposed Markov processes are generated from the consideration of the two possible (component, interface) pairs. In each case a reduced representation of the interface is used, corresponding to this component's view. In addition a basic skeleton is formed which corresponds to a greatly reduced version of the complete model, in which only the interface actions are carried out. Once this decomposition has been made, the algorithm follows the same general form as outlined above for the marked graph case.

Rather than a characterisation to recognise models of this form, this work relies on models having been constructed in the specified way. The class of deci-

sion free processes is defined via a reduced syntax, disallowing the choice combinator, +, and placing restrictions on where action types may appear within components. In particular each action type may be performed only once within any cycle of behaviour. This condition removes the possibility of implicit choice, when the action is to be carried out in synchronisation with another component. It is recognised that even working within this class the necessary structure, with an interface component acting as intermediary between two other components, may not be immediately apparent within the model. Therefore a series of possible transformations are defined, each of which is shown to be based on an equivalence relation which preserves some aspects of the model's behaviour. However, note that for two of the transformations the equivalence considers only functional, not temporal, behaviour. Thus it guarantees, for example, that no deadlock is introduced into the model but tells us nothing about the timing characteristics of the new model in relation to the old one.

6.3 Near-independence

In [13], Ciardo and Trivedi present a decomposition technique for stochastic reward nets (a version of SPN with immediate transitions, inhibitor arcs and rewards) based on the notion of *near-independence*. Components are considered to be near-independent if they operate in parallel and only rarely interact. The basic idea is that near-independent components can be solved independently in conjunction with a graph model which represents the (limited) dependencies that do still exist between them. Several examples of canonical near-independent net structures are given in the paper, but in general recognising such structures in any given model, and whether necessary conditions on the graph model are met, rely on the expertise of the modeller.

Components are solved in isolation, as if they were independent, but their influence upon each other is approximated by the rate at which synchronisation can take place. This is estimated using the dependency graph. In general, fixed point iteration may be necessary in order to achieve a satisfactory solution of the complete model, depending on the structure of the graph.

In [7], Bohnenkamp and Haverkort suggest that this technique could be adapted for MPA models. This paper does not progress this directly in terms of an MPA language but does report some interesting experiments which investigate the feasibility of the approach. In the proposed approach the dependence between components is recognised as the actions on which they synchronise (synchronisation between delays is not allowed). In effect the behaviours of the near-independent components are serialised, first capturing the work which can be done until blocking occurs due to a synchronisation point and then the work necessary to achieve the synchronisation.

6.4 Decomposition via synchronisation points

In [8], Bohnenkamp and Haverkort develop the ideas from [7] in a slightly different direction. The approach still centres on the role of synchronisations between

parallel components, but now the aim is to reformulate the underlying Markov process in terms of a set of semi-Markov processes. These semi-Markov processes are solved via their embedded Markov chains and evaluations of the distribution of the times between synchronisation points. Working within a MPA framework, they consider a class of models in which there is a fixed number of sequential processes composed in parallel, assuming each composition is subject to the same set of global synchronisation actions. Within this class of models their solution technique is exact with respect to throughputs and local steady state probabilities.

In the original MPA languages a delay is associated with each action representing its duration, resulting in a labelled transition system in which arcs are labelled by two types of information: action type and rate information. All the work already described in this chapter is based on such languages. However, Bohnenkamp and Haverkort use a language in which actions and time delays are treated separately. Several recent MPA languages [28, 19] take this approach, which is also found in the timed process algebras. Here the transition system has two distinct transition relations: one representing instantaneous actions and the other representing the passing of (stochastic) time.

From the point of view of the work reported in [8], this simplifies somewhat as only actions are allowed to synchronise, and the authors do not need to be concerned about the meaning of synchronisation between two delays. The sequential components of the model are treated as the decomposed processes of the underlying Markov process. The aggregated model, the embedded Markov chain of a semi-Markov process, is constructed compositionally: a tensor expression is formed from the embedded Markov chain of the semi-Markov process corresponding to the synchronisation process of each sequential process algebra component. This EMC may have several disjoint components but the initial state of the process is used to ensure that only the "live" component is considered. The reader is referred to [8] and [9] for more details.

7 Future prospects

Research over the last five years has clearly demonstrated that decomposed solution of MPA models is possible. Furthermore, it can be achieved by exploiting structures which are introduced at the process algebra level, to elucidate structures in the underlying Markov process. Of course, the class of models which can be recognised and handled by each of the current techniques, reported in this chapter, is somewhat limited. However the diversity of these techniques means that together they represent a substantial class of models.

In most of the work undertaken so far, the primary focus has been on characterising MPA models which, when the semantics of the language are applied and the labelled transition system formed, generate Markov processes within the given class. The aim is to develop the characterisation as a set of syntactic conditions which can be tested without having to apply the semantics to the whole model, although investigation of the state space of individual components may

be necessary. Moreover, these conditions should be sufficiently formal that they can be incorporated into one of the MPA tools, such as the PEPA Workbench [22], allowing the recognition of the structure to be automated. Note that in each case, the current characterisation is known to be incomplete in the sense that there are MPA models which give rise to Markov processes of the appropriate class which would not be recognised by the current conditions. Extending the characterisations is on-going work.

One way to extend a characterisation is to look for ways in which the syntactic conditions imposed on models can be relaxed whilst still generating an underlying Markov process that still satisfies the necessary conditions. Another is to look for cases when a given MPA model, which does not apparently satisfy the characterisation, can be transformed into one which does. As with most process algebras, PEPA allows the same model to be expressed in alternative forms; for example, a pair of interacting components may be expressed as a single sequential component or a PEPA model $(P \underset{K}{\bowtie} R) \underset{L}{\bowtie} Q$ may be equivalent to $(P \parallel Q) \underset{K \cup L}{\bowtie} R$. In the case of the latter model, in the first form it is not a candidate for the product form over submodels while in the second form it is. Recent work at Edinburgh has been developing rewriting rules which allow a model to be systematically re-expressed towards the target of eliciting a quasi-reversible structure, if the underlying model has one.

When using term rewriting rules we are not altering the structure of the model (as represented by its labelled transition system), only the syntactic representation of it. In general, however, such non-intrusive manipulations will not be sufficient. The model will not belong to a class with efficient solution techniques. Thus another use of the known characterisations is as the target for model simplifications, in which a model that does not have the correct structure is modified until it does have one of the product form structures, and thus is more amenable to solution. The basic idea is that once a model has been constructed, with tool assistance, the modeller will be able to massage her model into one of the classes of models corresponding to a decomposed solution technique. This approach has been investigated in recent work by Hillston and Tomasik [61, 40].

In [61] the authors study several variations on a PEPA model whose structure is close to that for the reversible class of product form solution. In each case some manipulation of the model is necessary in order to transform the model into a reversible one. The impact of the freedom afforded the modeller during these modifications (to choose activity rates etc.) are investigated together with the effect of the modifications on the performance measures of interest. In [40], a more formal approach to model simplification is taken. Whilst the motivation of this work is still transformation of a model to one which satisfies the criteria for reversibility, the focus is on defining a model simplification technique and assessing its impact on both the steady state distribution and associated performance measures.

In general, interaction between components is the major barrier to exact decomposition (i.e. product form solution), although it holds the key to some of the aggregated decompositional techniques. There is scope to review the com-

binators for interaction which are offered by the MPA languages. As Clark's work on insensitive structures shows, it is possible to define a combinator which *guarantees* the independence of components from the point of view of the steady state distribution. Similarly *flow cooperation*, as defined in the characterisation of quasi-reversibility, captures an acceptable form of interaction. Further study needs to be made to see when models can be shown to be product form by construction, rather than by syntactic checking after construction.

The work that has been completed so far demonstrates that *automated* decomposed solution of MPA models is feasible. As such, it offers a solution, albeit partial, to the state space explosion problem. Moreover, in several cases this solution can already be applied transparently to the modeller [47]. This represents a significant step forwards for performance analysis using process algebras.

Acknowledgements

Jane Hillston's work is supported by the Engineering and Physical Sciences Research Council via the COMPA project (G/L10215). I would like to thank my colleagues who have contributed the interesting work reported throughout this chapter. I would like to apologise to those whose work I did not include: this was due to the limited space available.

References

1. A. Aldini, M. Bernardo, and R. Gorrieri. An algebraic model for evaluating the performance of an ATM switch with explicit rate marking. In J. Hillston and M. Silva, editors, *Proc. of 7th Int. Workshop on Process Algebra and Performance Modelling (PAPM'99)*, pages 119–138. Prensas Universitarias de Zaragoza, September 1999.
2. H.H. Ammar and S.M. Rezaul Islam. Time Scale Decomposition of a Class of Generalized Stochastic Petri Net Models. *IEEE Transactions on Software Engineering*, 15(6):809–820, June 1989.
3. 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.
4. M. Bernardo and R. Gorrieri. A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time. *Theoretical Computer Science*, 202(1–2):1–54, July 1998.
5. M. Bhabuta, P.G. Harrison, and K. Kanani. Detecting reversibility in Markovian Process Algebra. In *Performance Engineering of Computer and Telecommunications Systems*, Liverpool John Moores University, September 1995. Springer-Verlag.
6. A. Blakemore and S. Tripathi. Automated Time Scale Decomposition of SPNs. In *Proc. of 5th International Workshop on Petri Nets and Performance Models (PNPM '93)*, Toulouse, 1993.
7. H. Bohnenkamp and B. Haverkort. Decomposition Methods for the Solution of Stochastic Process Algebra Models: a Proposal. In E. Brinksma and A. Nymeyer, editors, *Proc. of 5th Process Algebra and Performance Modelling Workshop*, 1997.

8. H. Bohnenkamp and B. Haverkort. Semi-Numerical Solution of Stochastic Process Algebra Models. In C. Priami, editor, *Proc. of 6th Process Algebra and Performance Modelling Workshop*, 1998.
9. H. Bohnenkamp and B. Haverkort. Stochastic event structures for the decomposition of stochastic process algebra models. In J. Hillston and M. Silva, editors, *Proc. of 7th Int. Workshop on Process Algebra and Performance Modelling (PAPM'99)*, pages 119–138. Prensas Universitarias de Zaragoza, September 1999.
10. R.J. Boucherie. A Characterisation of Independence for Competing Markov Chains with Applications to Stochastic Petri Nets. *IEEE Transactions on Software Engineering*, 20(7):536–544, July 1994.
11. R.J. Boucherie and M. Sereno. On the Traffic Equations of Batch Routing Queueing Networks and Stochastic Petri Nets. Technical report, European Research Consortium for Informatics and Mathematics, 1994.
12. 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.
13. G. Ciardo and K.S. Trivedi. A Decomposition Approach for Stochastic Petri Net Models. *Performance Evaluation*, 1992.
14. G. Clark. An Extended Weak Isomorphism for Model Simplification. In E. Brinksma and A. Nymeyer, editors, *Proc. of 5th Process Algebra and Performance Modelling Workshop*, 1997.
15. G. Clark. Stochastic Process Algebra Structure for Insensitivity. In J. Hillston and M. Silva, editors, *Proc. of 7th Int. Workshop on Process Algebra and Performance Modelling (PAPM'99)*, pages 63–82. Prensas Universitarias de Zaragoza, September 1999.
16. G. Clark. *Techniques for the Construction and Analysis of Algebraic Performance Models*. PhD thesis, LFCS, University of Edinburgh, 2000.
17. G. Clark and J. Hillston. Product form solution for an insensitive stochastic process algebra structure. *Performance Evaluation*, 2001. To appear.
18. P.J. Courtois. *Decomposability: Queueing and Computer System Applications*. ACM Series. Academic Press, New York, 1977.
19. P. D'Argenio, J-P. Katoen, and E. Brinksma. General Purpose Discrete Event Simulation Using Spades. In C. Priami, editor, *Proc. of 6th Process Algebra and Performance Modelling Workshop*, 1998.
20. 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.
21. J-M. Fourneau, L. Kloul, and F. Valois. Performance Modelling of Hierarchical Cellular Networks using PEPA. In J. Hillston and M. Silva, editors, *Proc. of 7th Int. Workshop on Process Algebra and Performance Modelling (PAPM'99)*, pages 139–154. Prensas Universitarias de Zaragoza, September 1999.
22. S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In G. Haring and G. Kotsis, 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.
23. 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.

24. S. Gilmore, J. Hillston, and L. Recalde. Elementary structural analysis for PEPA. Technical Report ECS-LFCS-97-377, Laboratory for Foundations of Computer Science, Department of Computer Science, The University of Edinburgh, 1997.
25. 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.
26. W. Henderson, D. Lucic, and P.G. Taylor. A Net level Performance Analysis of Stochastic Petri Nets. *Journal of the Australian Mathematical Society, Series B*, 31:176–187, 1989.
27. W. Henderson and P.G. Taylor. Embedded Processes in Stochastic Petri Nets. *IEEE Transactions on Software Engineering*, 17(2):108 – 116, February 1991.
28. H. Hermanns and M. Rettelbach. Towards a Superset of Basic LOTOS for Performance Prediction. In M. Ribaudo, editor, *Proc. of 6th Process Algebra and Performance Modelling Workshop*, 1996.
29. 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.
30. U. Herzog. Formal description, time and performance analysis: A framework. Technical Report 15/90, IMMD VII, Friedrich-Alexander-Universität, Erlangen-Nürnberg, Germany, September 1990.
31. J. Hillston. The Nature of Synchronisation. In U. Herzog and M. Rettelbach, editors, *Proc. of 2nd Process Algebra and Performance Modelling Workshop*, 1994.
32. J. Hillston. Compositional Markovian Modelling Using a Process Algebra. In W.J. Stewart, editor, *Numerical Solution of Markov Chains*. Kluwer, 1995.
33. J. Hillston. *A Compositional Approach to Performance Modelling*. Distinguished Dissertations in Computer Science. Cambridge University Press, 1996.
34. J. Hillston. A Class of PEPA Models Exhibiting Product Form Solution. Technical Report ECS-LFCS-98-382, LFCS, Dept. of Computer Science, University of Edinburgh, February 1998.
35. J. Hillston, H. Hermanns, U. Herzog, V. Mertsiotakis, and M. Rettelbach. Integrating Qualitative and Quantitative Modelling with Stochastic Process Algebras. Technical report, IMMD VII, Universität Erlangen-Nürnberg, May 1994.
36. J. Hillston and L. Kloul. An Efficient Kronecker Representation for PEPA Models. Submitted for publication, 2000.
37. 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.
38. J. Hillston and N. Thomas. A Syntactical Analysis of Reversible PEPA Models. In *Proc. of 6th Process Algebra and Performance Modelling Workshop*, Nice, France, September 1998. University of Verona.
39. J. Hillston and N. Thomas. Product Form Solution for a class of PEPA Models. *Performance Evaluation*, 35:171–192, 1999.
40. J. Hillston and J. Tomasik. Amalgamation of transition sequences in the PEPA formalism. In *Proc. of ICALP Workshops 2000 (PAPM)*. Carleton Scientific Press, 2000.
41. D.R.W. Holton. A PEPA Specification of an Industrial Production Cell. *The Computer Journal*, 38(6), 1995. Special Issue: Proc. of 3rd Process Algebra and Performance Modelling Workshop.
42. J.R. Jackson. Jobshop-like Queueing Systems. *Management Science*, 10:131–142, 1963.

43. H. Jungnitz. *Approximation Methods for Stochastic Petri Nets*. PhD thesis, Rensselaer Polytechnic Institute, May 1992.
44. F. Kelly. *Reversibility and Stochastic Processes*. Wiley, 1979.
45. A.A. Lazar and T.G. Robertazzi. Markovian Petri Net Protocols with Product Form Solution. *Performance Evaluation*, 12(1):67–77, January 1991.
46. V. Mertsiotakis. Time Scale Decomposition of Stochastic Process Algebra Models. In E. Brinksma and A. Nymeyer, editors, *Proc. of 5th Process Algebra and Performance Modelling Workshop*, 1997.
47. V. Mertsiotakis. *Approximate Analysis Methods for Stochastic Process Algebras*. PhD thesis, Universität Erlangen–Nürnberg, Martensstraße 3, 91058 Erlangen, September 1998.
48. V. Mertsiotakis and M. Silva. A Throughput Approximation Algorithm for Decision Free Processes. In M. Ribaudo, editor, *Proc. of 6th Process Algebra and Performance Modelling Workshop*, 1996.
49. V. Mertsiotakis and M. Silva. A Throughput Approximation Algorithm for Decision Free Processes. In M. Ribaudo, editor, *Proc. of 7th International Workshop on Petri Nets and Performance Models*, 1996.
50. I. Mitrani and N. Thomas. Routing Among Different Nodes Where Servers Break Down Without Losing Jobs. In F. Baccelli, A. Jean-Marie, and I. Mitrani, editors, *Quantitative Methods in Parallel Systems*, pages 248–261. Springer, 1995.
51. I. Mitrani and P.E. Wright. Routing in the Presence of Breakdowns. *Performance Evaluation*, 20:151–164, 1994.
52. M.K. Molloy. Performance Analysis using Stochastic Petri Nets. *IEEE Transactions on Computers*, 31(9):913–917, September 1982.
53. B. Plateau. On the Stochastic Structure of Parallelism and Synchronisation Models for Distributed Algorithms. In *Proc. ACM Sigmetrics Conference on Measurement and Modelling of Computer Systems*, 1985.
54. B. Plateau, J.M. Fourneau, and K.H. Lee. PEPS: A Package for Solving Complex Markov Models of Parallel Systems. In *Proc. of the 4th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, 1988.
55. 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.
56. 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.
57. M. Sereno. *Performance Models for Discrete Event Systems with Synchronisations*, volume II, chapter Product form and Petri nets, pages 637–660. Kronos, 1998.
58. W.J. Stewart, K. Arif, and B. Plateau. The numerical solution of Stochastic Automata Networks. *European Journal of Operation Research*, 86(3):503–525, 1995.
59. N. Thomas and J. Bradley. Approximating variance in non-product form decomposed models. In *Proc. of ICALP Workshops 2000 (PAPM)*, pages 607–619. Carleton Scientific Press, 2000.
60. N. Thomas and S. Gilmore. Applying Quasi-Separability to Markovian Process Algebra. In *Proc. of 6th Process Algebra and Performance Modelling Workshop*, Nice, France, September 1998. University of Verona.
61. J. Tomasik and J. Hillston. Transforming PEPA Models to Obtain Product Form Bounds. Technical report, Division of Informatics, University of Edinburgh, 2000.

A Structured Operational Semantics for PEPA

Before stating the inference rules we must define the notion of *apparent rate*, which is used to define the rate of a shared activity. The apparent rate of an action type within a component represents the totally capacity of that component to carry out activities of that type when it is in its current state.

Definition 7 (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 [31]. In the case of a passive action it is assumed that the corresponding component does not contribute 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 [33] 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

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

Fig. 12. Operational semantics of PEPA