Behavioural independence and control in PEPA

Nigel Thomas

nigel.thomas@durham.ac.uk
Research Institute in Software Evolution
University of Durham, UK

Abstract

Models of complex systems are inherently difficult to solve due principally to the size of their underlying state space. In Markovian process algebra a significant amount of work has been done to tackle this problem by exploiting the compositional nature of models and identifying structures that are amenable to different approaches to decomposition. In this paper a slightly different approach is presented which concentrates on interaction between components and its impact on behaviour. Two notions are introduced, behavioural independence and control. Informally, behavioural independence exists when one group of components behaves in a way that is not directly influenced by the evolution of other components in the model. Conversely control exists when one or more components dictate the behaviour of another component by the occurrence of internal actions. In many cases these properties can be identified without exploring the underlying state space and their existence can be used to derive a number of different model decompositions and approximations. In particular well known approaches such as reversibility and quasi-separability are explored using this method to complement earlier results. A simple iterative approximation has also been applied to models exhibiting control to derive a numerical solution to marginal component distributions. This paper includes a number of worked examples to highlight the potential of this approach.

1 Introduction

In recent years some effort has been made to identify efficient methods for analysing and solving stochastic process algebra models by decomposition (see [5, 6]). A significant element of this work has been the identification and characterisation of product form solutions, using properties such as reversibility [8] and quasi-reversibility [3] common in queueing network models [9]. Such solutions are derived on the basis that the components in the model are statistically independent in their steady state behaviour, therefore the steady state solutions for components may be found in isolation without the need to generate the entire state space of the model. Clearly product form solutions are an extremely efficient mechanism in deriving important numerical solutions, however the presence of a product form adds little to our understanding of the transient behaviour of models. Indeed, the components themselves may require large amounts of interaction and exhibit highly dependent transient behaviour. Given that stochastic process algebra are used for both performance and behavioural analysis, it seems natural that we should also wish to understand more general notions of independence than are present in product form solutions.

The aim of this paper is to address the problem of identifying what is meant by behavioural independence, to characterise some forms of behavioural independence and to introduce the notion of control between components that forms a behavioural dependency. In addressing this issue an attempt is made to provide a theoretical link between several areas of work that have been progressing in recent years, principally in non-product form and near product form decompositions. A key element has been work on *quasi-separability*, a non-product form decomposition method used to derive marginal probabilities. In characterising quasi-separability [10, 11, 12] the authors identified conditions by which certain combinations of components could be observed to behave as if the other components were not present, i.e. they could be considered to be behaviourally independent of the remainder of the model.

The paper assumes knowledge Hillston's Markovian process algebra, PEPA [4]. Section 2 presents the notions of behavioural independence and control preceded by some necessary definitions. In Sections 4 and 5 the exploitation of the concepts of behavioural independence and control is made in relation to earlier work on non-product form and simple product form decomposition. Finally some conclusions and future work directions are presented.

2 Definitions

In the following sections a number of properties of PEPA models will be referred to. Since the definitions of these properties are presented in detail elsewhere [1, 4], only a general description of them is included here. Informally, it can be said that, two PEPA expressions are *isomorphic* if they give rise to Markov chains which are equivalent such that for every state in each Markov chain there is a corresponding state in the other with the same one-step transition rates to states which are similarly equivalent. A *derivative* is the "state" of a component defining its current behaviour. For example, if $P \stackrel{def}{=} (\alpha, r).Q$ then Q is a derivative of P and if $Q \stackrel{def}{=} (\beta, r_2).R$ then R is a derivative of both Q and P, and so on. The *derivative set*, $ds(P)$, is the set of all the possible derivatives of a component, P. The *current action type set* of a component P, $\mathcal{A}(P)$, contains all the action types (but not the rates) that are enabled in the current derivative of the component P. The *complete action type set* of a component P, $\vec{\mathcal{A}}(P)$, contains all the action types (but not the rates) that are enabled in any of the derivatives $P' \in ds(P)$, hence $\vec{\mathcal{A}}(P) = \bigcup_{P' \in ds(P)} \mathcal{A}(P')$. The *current action set* of a component P, $\mathcal{A}ct(P)$, contains all the actions (type and rate pairs) that are enabled in the current derivative of the component P. In addition it is necessary to construct a number of additional definitions.

Definition 2.1 (Maximum rate)
The maximum rate of an action l in component P, denoted $Mr(l, P)$, is defined as the maximum of all the rates at which action l is specified across all derivatives of P. Thus, if $P \stackrel{\mathrm{def}}{=} (\alpha, r_1).(\alpha, r_2).(\alpha, r_3).P$ then $Mr(\alpha, P) = Max(r_1, r_2, r_3)$. The unspecified rate of passive actions, $\top$, is read as ∞ for the purposes of this function.

Definition 2.2 (Fertile action)
An action γ is said to be fertile in derivative P_i if $P_i \xrightarrow{\gamma} P_j$ and $i \neq j$.

Definition 2.3 (Current fertile action type set)
The current fertile action type set of a component P, denoted $\mathcal{A}_f(P)$, is the set of all action types of actions that are fertile in the current derivative of P.

Definition 2.4 (Complete fertile action type set)
The complete fertile action type set of a component P, denoted $\vec{\mathcal{A}}_f(P)$, is the set of all action types of actions that are fertile in at least one derivative of P.

2.1 Behavioural independence and control

Put simply the notion of *behavioural independence* is simply that a component in a model behaves identically regardless of the current behaviour of the other components in the model. This property can be defined more formally thus:

Definition 2.5 (Behavioural Independence)
The component P is said to be behaviourally independent in the model $P \underset{L}{\bowtie} Q$ if for every $P_i \in ds(P)$

$$\begin{aligned} &\mathcal{A}ct\left((P_i \underset{L}{\bowtie} Q_j)/\{\mathcal{A}(P_i \underset{L}{\bowtie} Q_j)/\{\mathcal{A}_f(P_i) \cap L\}\}\right) = \\ &\mathcal{A}ct\left((P_i \underset{L}{\bowtie} Q_k)/\{\mathcal{A}(P_i \underset{L}{\bowtie} Q_k)/\{\mathcal{A}_f(P_i) \cap L\}\}\right) \end{aligned}$$

$\forall\, Q_j, Q_k \in ds(Q)\ s.t.\ (P_i \underset{L}{\bowtie} Q_j), (P_i \underset{L}{\bowtie} Q_k) \in ds(P \underset{L}{\bowtie} Q)$

Obviously the trivial case for behavioural independence is where there are no shared actions, i.e. $P||Q$, however this is not the only case where components may be considered to be behaviourally independent. Furthermore, the fact that no actions are shared between two components does not mean they will always be behaviourally independent in the presence of other components. For example, in $(P||Q) \underset{L}{\bowtie} R$ the interaction between P and R may influence the interaction between Q and R, causing P and Q to be behaviourally *dependent*. If a component is not behaviourally independent then it must be dependent on some other component to perform one of more actions during its evolution. This property is referred to as *control*, and is more formally defined thus:

Definition 2.6 (Control)
The component P is said to be subject to control in the model $P \underset{L}{\bowtie} Q$ if for any $P_i \in ds(P)$

$$\mathcal{A}ct\left((P_i \underset{L}{\bowtie} Q_j)/\{\mathcal{A}(P_i \underset{L}{\bowtie} Q_j)/\{\mathcal{A}_f(P_i) \cap L\}\}\right) \neq$$
$$\mathcal{A}ct\left((P_i \underset{L}{\bowtie} Q_k)/\{\mathcal{A}(P_i \underset{L}{\bowtie} Q_k)/\{\mathcal{A}_f(P_i) \cap L\}\}\right)$$

$\forall\, Q_j, Q_k \in ds(Q)\ s.t.\ (P_i \underset{L}{\bowtie} Q_j), (P_i \underset{L}{\bowtie} Q_k) \in ds(P \underset{L}{\bowtie} Q)$

This definition states that the fertile actions in any derivative of P (or the rates at which those actions occur) can change as the current derivative of Q evolves. This dependence is expressed by saying that component Q *controls* component P over actions $K \subset L$ in $P \underset{L}{\bowtie} Q$ if the rate at which an action of type $k \in K$ can happen in $P_i \in ds(P)$ depends on the current derivative of Q. Clearly, if Q controls P over K then P cannot be behaviourally independent, but the independence, or otherwise, of Q is not given by this statement. The *control set* of P, $\mathcal{K}(P)$ is referred to as all the actions by which it is controlled. For example, if in the model $(P \underset{L}{\bowtie} Q) \underset{M}{\bowtie} R$, Q controls P over K_1 and R controls P over K_2 then $\mathcal{K}(P) = K_1 \cup K_2$. Clearly P is behaviourally independent iff $\mathcal{K}(P) = \emptyset$.

3 Exploiting behavioural independence

In any given model specification each of the components will be behaviourally independent, controlled, independent and controlling, or controlled and controlling. Each of these categories can be exploited in different ways to improve the efficiency of solution.

It is clear that if a component is behaviourally independent (even if it is also controlling) then it may be studied in isolation without affecting its behaviour, subject to the rates of shared actions being set. If a shared action is not enabled by the partner in the cooperation then the rate of that action will be zero when that component is considered in isolation, otherwise the rate of the shared action will be determined by the rates specified in each participating component. To illustrate such a situation, consider a number of queues in sequence. If all the queues are basic $M/M/1/\infty$ then the system is clearly a simple Jackson network and has a product form solution. However, even if this is not the case then the first queue will still be behaviourally independent (unless there is blocking at the server) and so may be studied in isolation.

3.1 Example: non-product form decomposition

If all the components are behaviourally independent then the system is trivially separable and a simple product form exists. Clearly this situation is extremely unlikely to take place in practice as there would be no meaningful interaction within such a system. A much more useful way of exploiting behavioural independence across an entire system is through the property known as quasi-separability [10, 11].

Informally, a system that is amenable to a quasi-separable solution can be considered in the following way. The entire system operates within a single environment, which may be made up of several sub-environments. Several components operate within this environment such that their behaviour is affected by the state of the environment, but the state of each component does not alter the state transitions of either the environment or the other components. In such a system the only component that is behaviourally independent is the environment. However, the compound component formed by the environment with any one of the other components is also behaviourally independent. Thus it is possible to form a separation for any component in the system so that it may be studied in isolation of all other components except the environment component.

Consider an example of a system where communication between remotely situated sensing devices and a centrally located server is unreliable. These sensors may become overloaded if the data that they collect cannot be downloaded. In such a situation they will not be able to collect data and will need to be reset by the central server for normal operation to resume. There are many reasons for interrupting communication between such distributed units, these include, weather conditions, electrical interference, geological disruptions and equipment failure.

The structure of this model is as follows. In normal operation sensing devices are continually collecting data from their local environment. Each sensor is periodically polled in turn to download data to a central server, whereupon the sensor is reset. In addition there are randomly occurring interruptions to the communication between the central server and the sensors. These interruptions are local, affecting communication with just a single sensor. These interruptions do not affect the ability of the sensors to collect data. However the sensors have limited ability to store data, so if the central server does not contact the sensor for some time it may become overloaded. In such situations it will need to be reset by the central server for normal data collection to resume.

There are several possible polling policies the central server could adopt in the event of a communication interruption. For a local interruption these include:

- simply miss the sensor out of the normal polling sequence until it is available;
- contact the affected sensor immediately it becomes available.

Clearly there are several further possibilities and the situation is greatly complicated if there are simultaneous local and global interruptions or if the central server is unaware of the nature of the interruption. In a performance study of such a system several policies would be compared to deduce the optimum performance over a given set of criteria. Since it is only of interest here to consider how such a model can be decomposed, only the situation where no remedial action will be taken by the central server in the event of a local interruption will be considered. Such a model can be specified thus:

$$\begin{aligned}
Sensor_{j,0} &\stackrel{def}{=} (collect, \gamma_j).Sensor_{j,1} + (downld_j, \top).Sensor_{j,0} \\
Sensor_{j,k} &\stackrel{def}{=} (collect, \gamma_j).Sensor_{j,k+1} + (downld_j, \top).Sensor_{j,0} \\
&\qquad 1 \le k \le M-1 \\
Sensor_{j,M} &\stackrel{def}{=} (collect, \gamma_j).Overload_j + (downld_j, \top).Sensor_{j,0} \\
Overload_j &\stackrel{def}{=} (downld_j, \top).Sensor_{j,0}
\end{aligned}$$

$$\begin{aligned}
NoIntrpt_j &\stackrel{def}{=} (downld_j, \top).NoIntrpt_j + (breakdn, \xi_j).Interrupt_j \\
Interrupt_j &\stackrel{def}{=} (block_j, \top).Interrupt_j + (end, \eta_j).NoIntrpt_j
\end{aligned}$$

$$Interrupt \stackrel{def}{=} (NoIntrpt_1 \parallel \cdots \parallel NoIntrpt_N)$$

$$\begin{aligned}
Server_j &\stackrel{def}{=} (downld_j, \mu).Server_{j-1} + (block_j, \lambda).Server_{j-1} \\
&\qquad 1 \le j \le N \\
Server_0 &\stackrel{def}{=} (wait, \alpha).Server_N
\end{aligned}$$

$$(Sensor_{1,0} \parallel \cdots \parallel Sensor_{N,0}) \underset{\substack{\{downld_1,\ldots,\\ downld_N\}}}{\bowtie} (Server_0 \underset{\substack{\{downld_1,\ldots,downld_N,\\ block_1,\ldots,block_N\}}}{\bowtie} Interrupt)$$

As with most languages, there are many ways to describe this model in PEPA. However, the aim here is not only to produce a concise specification, but also one that captures a specific view of this model and facilitates simplification and solution in a clear manner. In this example there are three categories of components, the sensors, the central sever and the interruptions in communication between the server and the sensors. Also, the action $download_j$ may cause a change in the behaviour of $Server_j$, where $j = 1, \ldots, N$, and is enabled passively in every derivative of $Sensor_{j,k}$, $k = 0, \ldots, M$. The behaviour of each sensor is dependent on the behaviour of the server and the possibility of any interruption between sensor and server. Furthermore it is apparent that the presence of any interruption will affect what actions are carried out by the server, hence each submodel will need to contain the jth sensor, all the interrupts and the server. Clearly this means that the sensors are all behaviourally independent of one another, but are controlled by the server and the interrupts. In addition, the interrupts are all behaviourally independent, as is the $Server_0 \underset{\{\ldots\}}{\bowtie} Interrupt$ combination.

Hence, following the argument presented above, the following submodels can be extracted,

$$Sensor_{j,0} \underset{\{downld_j\}}{\bowtie} (Server_0 \underset{\substack{\{downld_1,\ldots,downld_N,\\ block_1,\ldots,block_N\}}}{\bowtie} Interrupt) \tag{3.1}$$

From these expressions performance measures can be calculated, such as the probability of sensor overload and the average amount of valid data transferred to the central server, without the need to explore the entire state space of the full model. Clearly it is possible to derive an exact solution to the full model. However, the full model, as given above, has $2^N(M+2)^N(N+1)$ states in the underlying CTMC, whereas each submodel, given by (3.1), has $2^N(M+2)(N+1)$ states. Clearly therefore there is a considerable saving in the state space to be made and this saving is dependent on the number of sensors in the system and the amount of storage space each sensor has available.

Interestingly, if $\mu = \lambda$ then a product form solution exists as in that case the behaviour of an individual sensor is only dependent on its own interrupt and the server. The server behaviour is still dependent on all the other interrupts, but the rate for each $block_k$ or $downld_k$ $(k \neq j)$ is identical, so the action type can be ignored (made τ). Hence submodels are obtained of the form,

$$Sensor_{j,0} \underset{\{downld_j\}}{\bowtie} (Server_0 \underset{\{downld_j,\, block_j\}}{\bowtie} NoInterpt_j) \tag{3.2}$$

3.2 Example: product form over components

As well as being used to identify independent behaviour leading to decomposition, behavioural independence and control can also be used to identify cases where product form solutions exist. Such a case is the queueing model with breakdowns specified below.

$$\begin{aligned}
Queue_0 &\stackrel{def}{=} (arrival, \top).Queue_1 \\
Queue_i &\stackrel{def}{=} (arrival, \top).Queue_{i+1} + (service, \top).Queue_{i-1} \\
&\quad\; 1 \leq j \leq N-1 \\
Queue_N &\stackrel{def}{=} (service, \top).Queue_{N-1}
\end{aligned}$$

$$\begin{aligned}
Server_{on} &\stackrel{def}{=} (fail, \xi).Server_{off} + (arrival, \lambda).Server_{on} + (service, \mu).Server_{on} \\
Server_{off} &\stackrel{def}{=} (repair, \eta).Server_{on}
\end{aligned}$$

$$Queue_0 \underset{\{service,arrival\}}{\bowtie} Server_{on}$$

It is clear that the *Server* component is behaviourally independent in this model as neither of the shared actions affects its evolution. Similarly it is clear that the *Server* component controls the *Queue* component over the actions *service* and *arrival*. A number of other important factors are also apparent:

- All the actions of *Queue* are shared actions.
- All shared actions are enabled in $Server_{on}$.
- No shared actions are enabled in $Server_{off}$.
- No shared action causes both *Queue* and *Server* to change state.

These six factors mean that a product form solution exists over the *Server* and *Queue* components such that the joint steady state probabilities are given as:

$$p(Server_j|Queue_i) = p(Server_j).p(Queue_i)$$

where $j \in \{on, off\}$ and $0 \leq i \leq N$.

The model specified here is reversible and it is possible to derive a product form solution using the characterisation derived by Hillston and Thomas [8]. In fact an example with the same structure, referred to as *the drinking gambler* appeared in [8]. That approach required the identification of reversible components and the application of restrictions on the cooperation between them. This requires a detailed study of both the components and the interface, whereas the approach described here only requires a simple inspection of the components and adherence to the following six criteria:

1. One component, A, of a pair $A \underset{L}{\bowtie} B$ is behaviourally independent.
2. The other component, B, is controlled by A over all the actions in the cooperation set, $\mathcal{K}(B) = L$.
3. The complete action type set of B, $\vec{\mathcal{A}}(B)$ is contained within its interface, $\vec{\mathcal{A}}(B) = L$.
4. All actions in the cooperation set, L, are enabled in exactly one derivative of A.
5. No actions in the cooperation set, L, are enabled in any other derivative of A.
6. $\vec{\mathcal{A}}_f(A) \cap \vec{\mathcal{A}}_f(B) = \emptyset$

As long as these six stated conditions are not broken then this model can be easily adapted to incorporate additional (non-reversible) features, such as batch service, without compromising the product form solution. Such a model is specified as follows:

$$\begin{aligned}
Queue_0 &\stackrel{def}{=} (arrival, \top).Queue_1 \\
Queue_i &\stackrel{def}{=} (arrival, \top).Queue_{i+1} + (service, \top).Queue_0 \\
&\quad\; 1 \leq j \leq N-1 \\
Queue_N &\stackrel{def}{=} (service, \top).Queue_0 \\
\\
Server_{on} &\stackrel{def}{=} (fail, \xi).Server_{off} + (arrival, \lambda).Server_{on} \\
&\quad\; +(service, \mu).Server_{on} \\
Server_{off} &\stackrel{def}{=} (repair1, \eta_1).Server_{standby} \\
Server_{standby} &\stackrel{def}{=} (repair2, \eta_2).Server_{on} + (fail, \xi).Server_{off}
\end{aligned}$$

$$Queue_0 \underset{\{service, arrival\}}{\bowtie} Server_{on}$$

3.3 Example: excluded shared actions

In [8] Hillston and Thomas identified the limitations of their approach as being that they could not handle cases where the components were not reversible, but the resultant model was. They illustrated this with the following example:

$$
\begin{array}{rcll}
P_0 & \stackrel{def}{=} & (in_P, r).P_1 + (a, r).P'_1 & \\
P_i & \stackrel{def}{=} & (in_P, r).P_{i+1} + (out_P, s).P_{i-1} & 1 \leq i \leq N-1 \\
P_N & \stackrel{def}{=} & (out_P, s).P_{N-1} & \\
P'_i & \stackrel{def}{=} & (a, r).P'_{i+1} & 1 \leq i \leq M-1 \\
P'_M & \stackrel{def}{=} & (a, r).P_0 & \\
 & & & \\
Q_0 & \stackrel{def}{=} & (in_Q, r).Q_1 + (b, r).Q'_1 & \\
Q_i & \stackrel{def}{=} & (in_Q, r).Q_{i+1} + (out_Q, s).Q_{i-1} & 1 \leq i \leq N-1 \\
Q_N & \stackrel{def}{=} & (out_Q, s).Q_{N-1} & \\
Q'_i & \stackrel{def}{=} & (b, r).Q'_{i+1} & 1 \leq i \leq M-1 \\
Q'_M & \stackrel{def}{=} & (b, r).Q_0 & \\
 & & & \\
Sys & \stackrel{def}{=} & P \underset{\{a,b\}}{\bowtie} Q &
\end{array}
$$

In this model it might appear that the P component controls the Q component over the action b and the Q component controls the P component over the action a. However, $b \notin \vec{\mathcal{A}}(P_0)$ and $a \notin \vec{\mathcal{A}}(Q_0)$, hence both these shared actions are permanently blocked. Therefore it is clear that no control is exerted between these two components, both are behaviourally independent and the model has a trivial product form solution when the excluded behaviours are removed from the components (the rates of the actions a and b are always zero). However, this is not universally true for excluded behaviours. For example, consider the case where P'_M is redefined as follows.

$$P'_M \stackrel{def}{=} (b, r).P_0$$

Now $b \in \vec{\mathcal{A}}(P_0)$, but in this model b will never happen since action a is required to reach P'_M and action a is always blocked. Both components are still behaviourally independent by definition 2.5, however it is not a simple matter to know that an action will not occur unless the states of the underlying CTMC are explored. Therefore cases of excluded actions such as this are, in general, extremely problematical when working at the component level.

4 Approximation using reduced components

Given a model of two components, $X \underset{L}{\bowtie} Y$, such that X controls Y it is possible to generate X' such that the actions of Y are unaltered with respect to L in $X' \underset{L}{\bowtie} Y$ if for any derivative of $X \underset{L}{\bowtie} Y$ there is a corresponding derivative of $X' \underset{L}{\bowtie} Y$ such that

$$\mathcal{A}ct(X \underset{L}{\bowtie} Y) \cap L = \mathcal{A}ct(X' \underset{L}{\bowtie} Y) \cap L$$

In many situations, such as modelling resources, there will be little or no difference between X and the smallest possible X'. However, when X contains many derivatives, X_i where $\mathcal{A}ct(X_i) \cap L$ does not change, then there is potential for substantial state space reduction. The number of derivatives in X' is dependent on the number of actions in L and the number of distinct rates at which they are enabled. If the rates of each shared action is the same wherever it is enabled then the size of the reduced component X' is $2^{|L|}$. Thus the smaller the cooperation set the better the reduction in state space in the system $X' \underset{L}{\bowtie} Y$. Consider the case where $L = \{a\}$, if the rate of this action is the same in every derivative of $X \underset{\{a\}}{\bowtie} Y$ in which it is enabled then the reduced form of X, X' is given as follows.

$$
\begin{aligned}
X'_1 &\stackrel{def}{=} (a, r1).X'_1 + (a, r2).X'_2 + (\tau, r3).X'_2 \\
X'_2 &\stackrel{def}{=} (\tau, r4).X'_1
\end{aligned}
$$

The unknown action type τ is used to denote all those actions internal to X such that the derivative of X changes from X_i to X_j and $(\mathcal{A}ct(X_i) \cap L) \neq (\mathcal{A}ct(X_j) \cap L)$ (for all possible i and j. At this stage it is not possible to know the rates $r1$, $r2$, $r3$ and $r4$ in general, although the sum of the rates $r1$ and $r2$ is clearly the rate of a in every derivative of $X \underset{\{a\}}{\bowtie} Y$ in which it is enabled. If X is behaviourally independent in $X \underset{L}{\bowtie} Y$ then X may be solved in isolation and hence the rates found from the steady state probabilities. However, if Y controls X in $X \underset{L}{\bowtie} Y$ then it is not a simple matter to solve X. Instead an iterative approach is adopted (from [2]) to tackle this problem as follows.

1. Generate the smallest X' such that for any derivative of $X \underset{L}{\bowtie} Y$ there is a corresponding derivative of $X' \underset{L}{\bowtie} Y$ such that $\mathcal{A}ct(X \underset{L}{\bowtie} Y) \cap L = \mathcal{A}ct(X' \underset{L}{\bowtie} Y) \cap L$.
2. Similarly generate the smallest Y' such that for any derivative of $X \underset{L}{\bowtie} Y$ there is a corresponding derivative of $X \underset{L}{\bowtie} Y'$ such that $\mathcal{A}ct(X \underset{L}{\bowtie} Y) \cap L = \mathcal{A}ct(X \underset{L}{\bowtie} Y') \cap L$.
3. Solve $X' \underset{L}{\bowtie} Y$ using estimates of the unknown rates in X'.
4. Use the solution of $X' \underset{L}{\bowtie} Y$ to calculate the unknown rates in Y'.
5. Solve $X \underset{L}{\bowtie} Y'$ to find new estimates of the unknown rates in X'.
6. Solve $X' \underset{L}{\bowtie} Y$ to find new estimates of the unknown rates in Y'.
7. Repeat steps 5 and 6 until convergence (or abandon).

4.1 Example: tandem queues with blocking and feedback

Consider the following definition of a simple two queue network with feedback and blocking.

$$
\begin{aligned}
Queue1_0 &\stackrel{def}{=} (arrival, \lambda).Queue1_1 + (feedback, \top).Queue1_1 \\
Queue1_j &\stackrel{def}{=} (arrival, \lambda).Queue1_{j+1} + (service1, \mu_1).Queue1_{j-1} \\
&\quad + (feedback, \top).Queue1_{j+1} \ , \ \ 1 \leq j \leq N-1 \\
Queue1_N &\stackrel{def}{=} (service1, \mu_1).Queue1_{N-1}
\end{aligned}
$$

$$
\begin{aligned}
Queue2_0 &\stackrel{def}{=} (service1, \top).Queue2_1 \\
Queue2_j &\stackrel{def}{=} (service1, \top).Queue2_{j+1} + (feedback, q\mu_2).Queue2_{j-1} \\
&\quad + (departure, (1-q)\mu_2).Queue2_{j-1} \ , \ \ 1 \leq j \leq N-1 \\
Queue2_N &\stackrel{def}{=} (feedback, q\mu_2).Queue2_{N-1} \\
&\quad + (departure, (1-q)\mu_2).Queue2_{N-1}
\end{aligned}
$$

$$
Queue1_0 \underset{\{service1,\ feedback\}}{\bowtie} Queue2_0
$$

Each queue has an input action which is controlled by the other queue, i.e. it is blocked when the other queue is empty, $Queue1$ has an additional independent *arrival* process. Similarly each queue output action which is also controlled by the other queue, i.e. it is blocked when the other queue is full, $Queue2$ has an additional *departure* process. Hence, both components in this model ($Queue1$ and $Queue2$) are controlling and controlled over the co-operation set $\{service1, feedback\}$.

It is a simple matter to construct the reduced versions of $Queue1$ and $Queue2$, $Queue1'$ and $Queue2'$ respectively.

$$
\begin{aligned}
Queue1'_a &\stackrel{def}{=} (arrival, \lambda).Queue1'_b + (feedback, \top).Queue1'_b \\
Queue1'_b &\stackrel{def}{=} (arrival, p_1\lambda).Queue1'_c + (service1, (1-p_2)\mu_1).Queue1'_b \\
&\quad + (service1, p_2\mu_1).Queue1'_a + (feedback, p_1 q\mu_2).Queue1'_c \\
&\quad + (feedback, (1-p_1)q\mu_2).Queue1'_b \\
Queue1'_c &\stackrel{def}{=} (service1, \mu_1).Queue1'_b
\end{aligned}
$$

$$
\begin{aligned}
Queue2'_a &\stackrel{def}{=} (service1, \top).Queue2'_b \\
Queue2'_b &\stackrel{def}{=} (service1, (1-p_3)\mu_1).Queue2'_b \\
&\quad + (service1, p_3\mu_1).Queue2'_c + (feedback, p_4 q\mu_2).Queue2'_a \\
&\quad + (feedback, (1-p_4)q\mu_2).Queue2'_b \\
&\quad + (departure, p_4(1-q)\mu_2).Queue2'_a \\
Queue2'_c &\stackrel{def}{=} (feedback, q\mu_2).Queue2'_b \\
&\quad + (departure, (1-q)\mu_2).Queue2'_b
\end{aligned}
$$

In this example the reduced form of the components have only three derivatives as there is no possible state where both the input and output actions are blocked. Clearly the derivatives $Queue1_a$, $Queue1_c$, $Queue2_a$ and $Queue2_c$, are analogous to each queue being either empty or full and the derivatives $Queue1_b$ and $Queue2_b$ correspond to all the non-empty and non-full conditions of each queue respectively. This approximation has introduced four new probabilities:

- p_1 is the probability that an additional job entering $Queue1$ will cause it to become full, $p_1 = Pr[Queue1_{N-1}]/(1 - Pr[Queue1_N] - Pr[Queue1_0])$.
- p_2 is the probability that a $service1$ action causes $Queue1$ to become empty, $p_2 = Pr[Queue1_1]/(1 - Pr[Queue1_N] - Pr[Queue1_0])$.
- p_3 is the probability that a $service1$ action causes $Queue2$ to become full, $p_3 = Pr[Queue2_{N-1}]/(1 - Pr[Queue2_N] - Pr[Queue2_0])$.
- p_4 is the probability that a job leaving $Queue2$ will cause it to become empty, $p_4 = Pr[Queue2_1]/(1 - Pr[Queue2_N] - Pr[Queue2_0])$.

The optimal values for these probabilities must be found by iteratively solving the two reduced models:

$$Queue1_0 \underset{\{service1,\, feedback\}}{\bowtie} Queue2'_a$$

and

$$Queue1'_a \underset{\{service1,\, feedback\}}{\bowtie} Queue2_0$$

Each of these reduced models has a CTMC with $3(N+1)$ states whereas the original model has a CTMC with $(N+1)^2$ states. Clearly there is a potentially significant saving on computation in each iteration, however the overall computational efficiency of this approach is dependent not only on the value of N, but the number of iterations required to achieve convergence over the introduced probabilities. Numerical results based on this example given in [13] show this method to be a potentially extremely accurate means of deriving marginal distributions and a useful approach for approximating certain global measures.

5 Conclusions and further work

In this paper a preliminary discussion of the notions of behavioural independence and control has been presented. These notions have been related to earlier work on quasi-separability and product form solutions with three examples. It is probable that many additional classes of product form solution will have subclasses that can be defined using behavioural independence and control, although this remains to be proved. By exploring these subclasses of solutions it will be possible to gain greater understanding of the links between different product form solutions (and non-product form solutions such as quasi-separability) and where they may overlap. In addition, many useful approximate solutions can be derived by exploiting the notion of control, an example of which is also given here. Defining these approximations, their applicability and potential accuracy is a matter of continuing interesting investigation.

Acknowledgements

I would like to thank Dr Jeremy Bradley and Dr David Thornley of Imperial College, London for helpful discussions during the development this work and Dr Jane Hillston (University of Edinburgh) for introducing me to stochastic process algebra and model decomposition.

References

[1] G. Clark, S. Gilmore, J. Hillston and N. Thomas, Experiences with the PEPA performance modelling tools, *IEE Proceedings - Software*, **146**(1), 1999.

[2] M. Gribaudo and M. Sereno, Approximation technique of finite queueing networks exploiting Petri net analysis, in: D. Kouvatsos (ed.), *Proceedings of the Fourth International Workshop on Queueing Networks with Finite Capacity*, Ilkley, UK, 2000.

[3] P.G. Harrison and J. Hillston, Exploiting Quasi-reversible Structures in Markovian Process Algebra Models, *The Computer Journal*, **38**(7), pp. 510–520, 1995.

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

[5] J. Hillston, Exploiting Structure in Solution: Decomposing Composed Models, in: C. Priami (ed.), *Proceedings of 6th International Workshop on Process Algebra and Performance Modelling*, 1998.

[6] J. Hillston, Exploiting Structure in Solution: Decomposing Composed Models, in: *FMPA Lecture Notes*, Springer-Verlag, 2001.

[7] J. Hillston and N. Thomas, Product Form Solution for a Class of PEPA Models, *Performance Evaluation*, **35**(3-4), pp. 171-192, June 1999.

[8] J. Hillston and N. Thomas, A Syntactic Analysis of Reversible PEPA Models, in: C. Priami (ed.), *Proceedings of the 6th International Workshop on Process Algebra and Performance Modelling*, 1998.

[9] F.P. Kelly, *Reversibility and Stochastic Networks*, Wiley, 1979.

[10] N. Thomas and S. Gilmore, Applying Quasi-Separability to Markovian Process Algebra, in: C. Priami (ed.), *Proceedings of 6th International Workshop on Process Algebra and Performance Modelling*, 1998.

[11] N. Thomas, Extending Quasi-separability, in: J. Bradley and N. Davies (eds.), *Proceedings of 15th UK Performance Engineering Workshop*, pp. 131-140, University of Bristol, 1999.

[12] N. Thomas and J. Bradley, Approximating variance in non-product form decomposed models, in: *Proceedings of 8th International Workshop on Process Algebra and Performance Modelling*, ICALP Workshops 2000, Proceedings in Informatics 8, Carleton Scientific Publishers, 2000.

[13] N. Thomas, J. Bradley and D. Thonley, An approximate solution of PEPA models using component substitution, in: M. Ould-Khaoua and L. MacKenzie (eds.), , *Proceedings of 18th UK Performance Engineering Workshop*, University of Glasgow, 2002.