From SAN to PEPA: A Technology Transfer *

Jane Hillston*, Leïla Kloul*!

*LFCS
University of Edinburgh
Kings Buildings
Edinburgh EH9 3JZ, Scotland

!PRiSM
Université de Versailles
45, Av. des Etats-Unis
78035 Versailles Cedex, France

Abstract

In this paper, we compare two performance modelling techniques: the SAN and PEPA. The SAN models a system using a graphical representation of its behaviour. Based on a different approach, PEPA is a formalism which models a system explicitly, describing its behaviour step by step. We point out their differences and their similitudes, and we show how to go from one formalism to the other.

1 Introduction

Performance investigation of modern computer and communication systems requires the development of relevant and efficient modelling techniques. The rich synchronisation constraints and the size of these systems lead to complex models with exponential growth of the number of states. Traditional performance models, based on queueing networks, cannot readily capture these constraints; thus several new performance modelling techniques have been developed, e.g. Stochastic Petri Nets (SPN), Stochastic Automata Networks (SAN) and Stochastic Process Algebras (SPA).

Petri nets were designed to represent synchronisation constraints within concurrent systems and protocols; SPN associate random variables with timed transitions within the net [1, 2, 3]. However, although the graphical representation of Petri Nets presents the dynamic behaviour of the model, it provides little insight into the structure of the system being modelled.

The new generation of performance modelling techniques try to take into account both factors, providing mechanisms to represent the increasing complexity of the synchronization constraints of modern systems whilst retaining the compositional structure explicitly within the model. Based on these structured modelling techniques solutions are proposed to overcome the state space explosition problem: a compact representation of the generator matrix

*This work is supported by CNRS/RS project (UIIVV 78171) and EPSRC project COMPA (G/L10215).

of the underlying Markov process with solving techniques adapted to this representation, or simplification techniques adapted to the formalism itself.

Stochastic Automata Networks (SAN) is the first new generation modelling technique which provides a compact representation of the generator matrix to overcome the state space explosition problem. Moreover, solving techniques have been adapted to this representation.

The SAN, like stochastic Petri nets, are based on graphical models characterized by an underlying stochastic hypothesis. They have been introduced by Plateau [4] as an efficient method to represent complex systems with interacting components such as parallel systems or distributed systems. In the SAN formalism, the dynamic behaviour of a component is represented by a transition from one state to another. A label on the transition specifies the instant and the occurrence probability of the action. This label specifies also the type of the action. Under adequate probabilistic hypothesis, the behaviour of the automata set is then represented by a multi-dimensional Markov chain whose states are those of the space product. Using a technique based on tensorial decomposition, it has been proved [5, 6] that this method automatically provides an analytic derivation of the generator matrix of the Markov chain called the *descriptor*. Compared to an extensive description of the generator, the structure of this descriptor leads to a considerable benefit for the state space.

Based on different concepts, another new generation modelling technique, called Performance Evaluation Process Algebra (PEPA) [7], proposes a different solution to overcome the state space explosition problem. This solution consists of applying on the PEPA models, before building the Markov chain, a simplification technique adapted to the formalism itself.

Introduced in 1994 by Hillston [7], the formalism PEPA extends classical process algebra by associating a random variable, representing duration, with every action. These random variables are assumed to be exponentially distributed and this leads to a clear relationship between the process algebra model and a continuous time Markov process. Via this underlying Markov process performance measures can be extracted from the model.

In this paper, we investigate the introduction of new concepts in PEPA formalism. Inspired from the SAN, these concepts are the functional dependency and the compact representation of the generator matrix or descriptor. The former will introduce a certain modelling flexibility in PEPA and sometimes a reduction of the models size. The latter will allow the modeller to reduce the number of entries to store, specially if it is applyied after the simplification technique of PEPA.

The paper is structured as follows. In Section 2, we give a short formal presentation of SAN. In Section 3 we present formaly the formalism PEPA and its simplification technique. In Section 4, using small examples, we make a high-level description of both formalisms PEPA and SAN. Section 5 is devoted to a case study in which we show the interest of the functional dependencies. In Section 6 we show how to introduce these dependencies in PEPA. Section 7 is dedicated to the tensor representation of a Markov chain. First, we present the tensor representation as introduced in SAN and finally we show how we derive this type of representation for PEPA models. We finally conclude with some remarks and possible future works.

2 SAN: A formal Presentation

In the SAN approach, the dynamic behaviour of each component of the system is modelled by an automaton and the interactions between the different components by labels on the directed edges which may represent synchronization events and transition rates. There are 2 types of transitions: local and synchronized. A local transition occurs only in the automaton whereas the synchronized transition occurs in several automata at the same time.

A Stochastic Automata Network is a set of N automata where each automaton A_i is defined by the tuple (S_i, L, Q_i) where

- S_i is the set of states of the automaton,
- L is the set of labels. A label l is a list that may contain either a function τ, or a list of tuples (e, τ_e, p_e) with different symbols e, or both of them such that:
  - e is the name of the synchronizing event or synchronization,
  - τ and τ_e are the transition rates, functions defined from $\Pi_{i=1}^{N} S_i$ to R^+,
  - p_e is the probability transition function defining a conditional routing probability between local states given that the synchronizing event occurs.
- Q_i is the transition function which associates a label from L with every arc of A_i.

In this standard definition of SAN, we need to give some more explanation about the label set L. A label on an edge allows us to specify the type and the rate of the transition as follows:

- If $Q_i(x_i, y_i)$ contains a function τ, then we have a transition local to automaton A_i between states x_i and y_i. If τ is not a constant, the transition is still local to automaton A_i, but its rate depends on the state of the other automata of the network.
- If $(e, \tau_e, p_e(x_i, y_i)) \in Q_i(x_i, y_i)$, then the transition between states x_i and y_i is a synchronized transition, e and τ_e being the name and the rate of the transition of the synchronizing event. $p_e(x_i, y_i)$ is the routing probability between local states x_i and y_i. The name of the event allows us to define the automata and the transitions concerned by this event. The distinction between the rate and the probability is required because the first must be unique for a given synchronizing event, thus the same on all concerned automata, whilst the probabilities may (and generally will) differ.

3 The PEPA Formalism

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 $\top$. The combinators, together with their names and interpretations, are presented below.

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

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

Cooperation: $\boldsymbol{P \underset{L}{\bowtie} Q}$ The components proceed independently with any activities whose types do not occur in the *cooperation set* L (*individual activities*). However, activities with action types in the set L require the simultaneous involvement of both components (*shared activities*). These activities are only enabled in $P \underset{L}{\bowtie} Q$ when they are enabled in both P and Q. Thus one component may become blocked, waiting for the other component to be ready to participate. The shared activity occurs at the rate of the slowest participant. If an activity has an unspecified rate in a component, the component is *passive* with respect to that action type. This means that the component does not influence the rate at which any shared activity occurs. 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 $P \underset{\emptyset}{\bowtie} Q$.

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

The semantics of PEPA, presented in the structured operational semantics style, are given in [7]. The underlying transition system also characterises the Markov process represented by the model. PEPA is the labelled *multi*-transition system $(\mathcal{C}, \mathcal{A}ct, \{\!|\xrightarrow{(\alpha,r)} \mid (\alpha, r) \in \mathcal{A}ct\, |\!\})$ where $\mathcal{C}$ is the set of components, $\mathcal{A}ct$ is the set of activities and the multi-relation $\xrightarrow{(\alpha,r)}$ is given by the rules in [7].

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. 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.

4 High-level model abstractions for representation

Both SAN and PEPA aim to provide a compositional framework for model construction, allowing the modeller to focus on the behaviour of individual components within a system and the interactions between them rather than tackling the complexity of the whole system all at once.

4.1 Events vs Activities

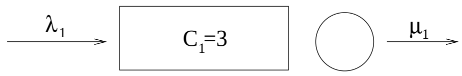
In the SAN, a system is represented by a number of *automata*. These elements are assumed to work more or less independently of one another, with only limited interaction. Within an individual automaton, the behaviour of a component or aspect of the system is captured as a set of *states* and rules that govern the *events* which determine movement from one state to another. Automata are defined in terms of state transition diagrams in which each node represents a state and each arc or transition denotes the parameter of a negative exponential distribution which governs the rate at which the event may occur.

For example, consider a buffer, called $queue_1$, with a finite capacity $C_1 = 3$, an arrival rate λ_1 and a departure rate μ_1 (Figure 1). The behaviour of this system is modelled using an automaton $\mathcal{A}_1$ composed of four states $(0, .., 3)$, each one corresponding to the number of customers that may be present in the buffer after the occurrence of an event. In this single buffer system, the events that may occur are the arrival of a new customer and the service completion. On automaton $\mathcal{A}_1$ (Figure 1), the effect of the first event is represented by the transitions with rate λ_1 and the effect of the second by the transitions with rate μ_1.

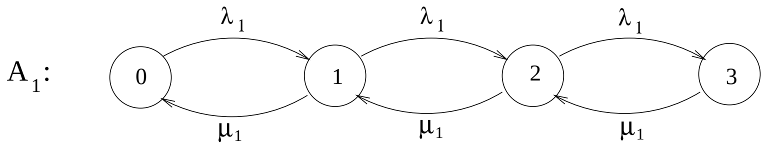
In PEPA, a system is modelled as the interaction of a number of *components* or *agents*. The behaviour of each component is built up using the *combinators* of the language from the primitives of the formalism, *activities*. Generally applied syntactic restrictions mean that the "components" considered in the system are so-called *sequential components*. Informally, this means that each is a finite state machine, whose states are syntactic terms and whose arcs are labelled by activities.

For example, consider the simple three-place buffer represented in Figure 1. In PEPA, it would be represented by a model composed component $Buffer_0$ with three derivatives $Buffer_i$, $i = 1..3$, and two activities which types are *in* and *service*. The first one is used to describe the arrival of a new customer in the buffer and the second the service completion. As we can see, only the prefix and choice combinators are used in the construction of sequential components (Figure 1).

Through this simple example, we can see that PEPA is a more explicit formalism than



queue$_1$



The SAN model

$$\begin{aligned}
Buffer_0 &\stackrel{def}{=} (in, \lambda_1).Buffer_1 \\
Buffer_1 &\stackrel{def}{=} (in, \lambda_1).Buffer_2 + (service, \mu_1).Buffer_0 \\
Buffer_2 &\stackrel{def}{=} (in, \lambda_1).Buffer_3 + (service, \mu_1).Buffer_1 \\
Buffer_3 &\stackrel{def}{=} (service, \mu_1).Buffer_2
\end{aligned}$$

The PEPA model

Figure 1: A single buffer system model

SAN. In PEPA, each part of the behaviour of a system is explicitly modelled. In SAN models, the system behaviour is implicitly represented since the transitions are the results of the implicit occurrence of events. The following will confirm this observation.

4.2 Components interaction

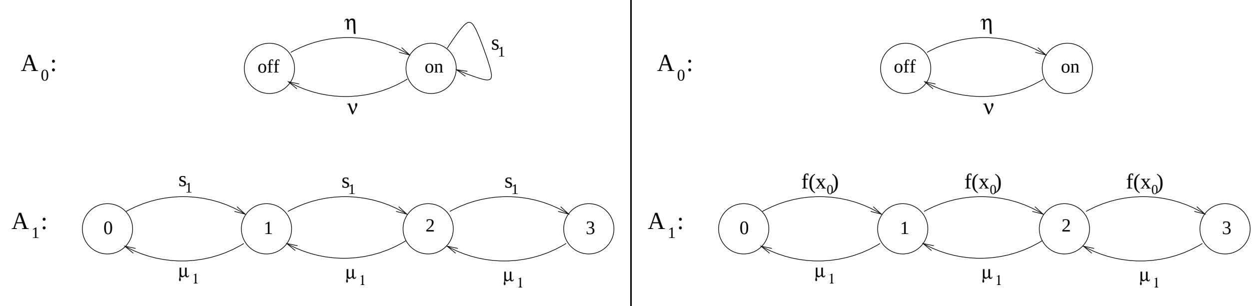
Once we consider the system as a whole, instead of focussing on its constituent parts, we wish to capture the ways in which components interact. In the SAN setting, automata are able to influence one another in two ways, both related to events. Direct interaction between automata is modelled by *synchronizing transitions*. These transitions are made up of events from two or more automata which must be carried out synchronously. The rate of such transitions are determined at the global level, and have no concrete realisation in the individual automata.

For example, consider the single buffer system described above and assume now that it accepts arrivals from a distinct arrival process with a simple *on/off* source. The system must be represented as the interaction of two components, the source and the buffer. For that we use two automata $\mathcal{A}_0$ and $\mathcal{A}_1$, the first to model the source behaviour and the second to model $queue_1$ (Figure 2 (a)). The events that may occur in such a system are the following:

1. The source switches to state *on*. This event is represented by the transition with rate η in automaton $\mathcal{A}_0$.
2. The generation of a customer by the source and its arrival in $queue_1$. As this event concerns both the source and $queue_1$, this is represented by synchronisation transitions labelled s_1 on both automata. This synchronisation is at rate λ_1.
3. The source switches to state *off*. This event is represented by the transition with rate ν in automaton $\mathcal{A}_0$.

4. The service completion of a customer in $queue_1$. It is represented by the transitions with rate μ_1 on automata $\mathcal{A}_1$.

Events 1, 3 and 4 concern only one automaton. These events are called *local events* whereas event 2 is called a *synchronizing event.*



(a) *Interaction by synchronization*

(b) *Interaction by function*

Figure 2: Automata Interactions

The other form of interaction is less direct. It is possible for the rate of a local transition within one automata to be influenced by the local states of one or more other automata. This can be illustrated by the same example as above by taking a different representation of the arrival process, capturing only whether the process is currently "*on*" or not. So, the transitions in automaton $\mathcal{A}_1$ which capture the arrival of a new customer are no longer synchronizing transitions, but *functional transitions*(Figure 2 (b)). If variable x_0 denotes the local state of automaton $\mathcal{A}_0$ then we can define a function f as

$$f(x_0) = \begin{cases} 1 & \text{if } x_0 = on \\ 0 & \text{otherwise} \end{cases}$$

The rate of a transition labelled $f(x_0)$ depends on the state of $\mathcal{A}_0$. There is a *functional dependency* between the two automata.

Note that in SAN there is little sense of identity within the model. Individual states and events are not *named* in any explicit sense. Identities are only explicitly associated with automata and synchronizations. However, when a functional rate is used, as in the example above, there is an implicit assumption that the local states of the influencing automaton are labelled, although this labelling need not be unique. This is necessary for the definition of the function which determines the transition rate in the influenced automaton.

In PEPA, direct interaction between sequential components is modelled by synchronization, known as *cooperation.* Similarly to synchronizing events in SAN, activities carried out in cooperation require the synchronous participation of two or more components. Thus, if we wish to model the arrival process to the buffer separately, with the *in* activity shared, the components are defined as shown below.

$$\begin{aligned}
Buffer_0 &\stackrel{def}{=} (in, \top).Buffer_1 \\
Buffer_1 &\stackrel{def}{=} (in, \top).Buffer_2 + (service, \mu).Buffer_0 \\
Buffer_2 &\stackrel{def}{=} (in, \top).Buffer_3 + (service, \mu).Buffer_1 \\
Buffer_3 &\stackrel{def}{=} (service, \mu).Buffer_2
\end{aligned}$$

$$\begin{aligned}
Arrival_{on} &\stackrel{def}{=} (in, \lambda).Arrival_{on} + (off, \nu).Arrival_{off} \\
Arrival_{off} &\stackrel{def}{=} (on, \eta).Arrival_{on}
\end{aligned}$$

$$System \stackrel{def}{=} Buffer_0 \underset{\{in\}}{\bowtie} Arrival_{on}$$

Note that now, in addition to the mutually recursive sets of equations defining the behaviour of each sequential component, we also have a *system equation* ($System \stackrel{def}{=} \ldots$) which defines the cooperation between the two components. The set decorating the cooperation combinator $\bowtie$, in this case $\{in\}$, specifies which activities require the synchronous participation of the two arguments of the combinator, in this case $Buffer_0$ and $Arrival_{on}$. Other activities of the two components, such as *service* and *off* can be carried out independently.

There is no mechanism for setting *functional rates* for PEPA activities, analogous to that in SAN. However the final PEPA combinator, hiding, offers *abstraction* facilities to the modeller. When a component is within the scope of a hiding operator $/\mathcal{L}$, all activities whose types are within the set $\mathcal{L}$ become private or *hidden* within the component. This means that their type is no longer visible, instead being replaced by the distinguished type τ. The timing behaviour of the activity is unaffected. Once a type is hidden it cannot be used within a cooperation and so will be carried out independently, whatever context the component is placed into.

5 Case Study: A finite capacity queueing network

Consider an open queueing network composed of three finite capacity queues called $queue_1$, $queue_2$ and $queue_3$ (Figure 3). We denote by C_i the finite buffer capacity at $queue_i$, $i = 1..3$. The customers of this network are of two classes. Class 1 customers arrive from outside the network to $queue_1$ according to a Poisson process with rate λ_1. An arriving customer is considered lost if $queue_1$ is full when it arrives. Similarly, class 2 customers arrive from outside the network to $queue_2$ according to a Poisson process with rate λ_2. If $queue_2$ is full when a class 2 customer arrives, it is considered lost too. The servers at $queue_1$ and $queue_2$ provide exponential services at rate μ_1 and μ_2 respectively.

After its service in $queue_1$, a class 1 customer tries to join $queue_3$. If this buffer is full, the customer is blocked in the server until a slot becomes available in $queue_3$. Similarly, at its service completion in $queue_2$, a class 2 customer tries to join $queue_3$. However if the buffer is full, the class 2 customer is lost.

The server at $queue_3$ provides an exponential service at rate μ_{31} to class 1 customers and at rate μ_{32} to class 2 customers. We assume that class 1 customers have a preemptive priority over class 2 customers for the service. Customers receiving service at $queue_3$ leave the network.

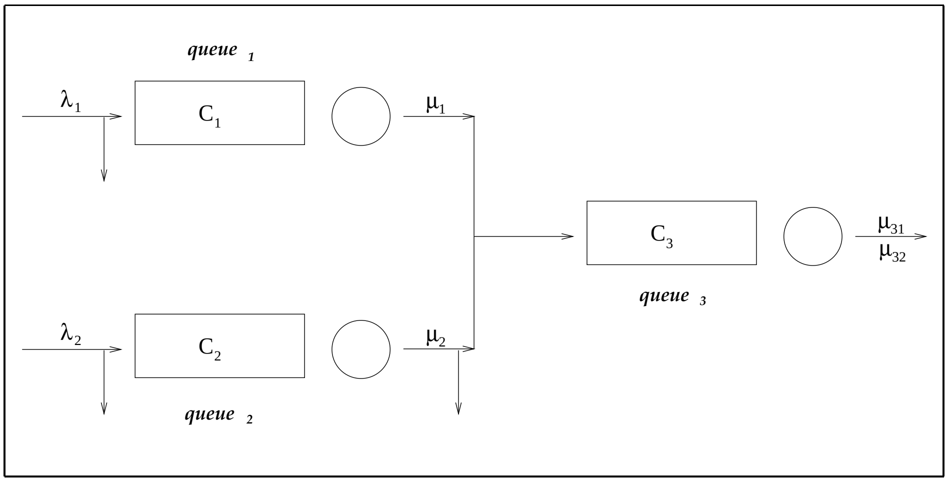


Figure 3: A finite capacity queueing network

In the following we show how to model this system using both our chosen formalisms. When presenting the SAN model we will show that there are alternatives, made possible by the use of functional rates, which do not exist for the PEPA model.

5.1 The SAN Model

To capture the behaviour of the whole network, we need to capture the behaviour of each queue and the interactions between all queues. For that we model $queue_1$ and $queue_2$ using automaton $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. We consider a one-to-one correspondence between the number of customers in the buffer and the state of the associated automaton. Representing $queue_3$ requires two automata because the customers in its buffer are of two classes and one class has a preemptive priority service over the other. The first automaton ($\mathcal{A}_{31}$) provides the number of class 1 customers in $queue_3$, whereas the second ($\mathcal{A}_3$) provides the total number of customers of both classes (Figure 4).

To understand the different automata of the SAN model presented in Figure 4 let us see what are the events that may occur in the corresponding queueing network of Figure 3.

1. The arrival of a new customer in $queue_1$. This event is a local event since it has no effect on the other queues of the network. It is represented by a local transition in automaton $\mathcal{A}_1$ with rate λ_1.

2. The arrival of a new customer in $queue_2$. As previously this event is a local event. It is represented by a local transition in automaton $\mathcal{A}_2$ with rate λ_2.

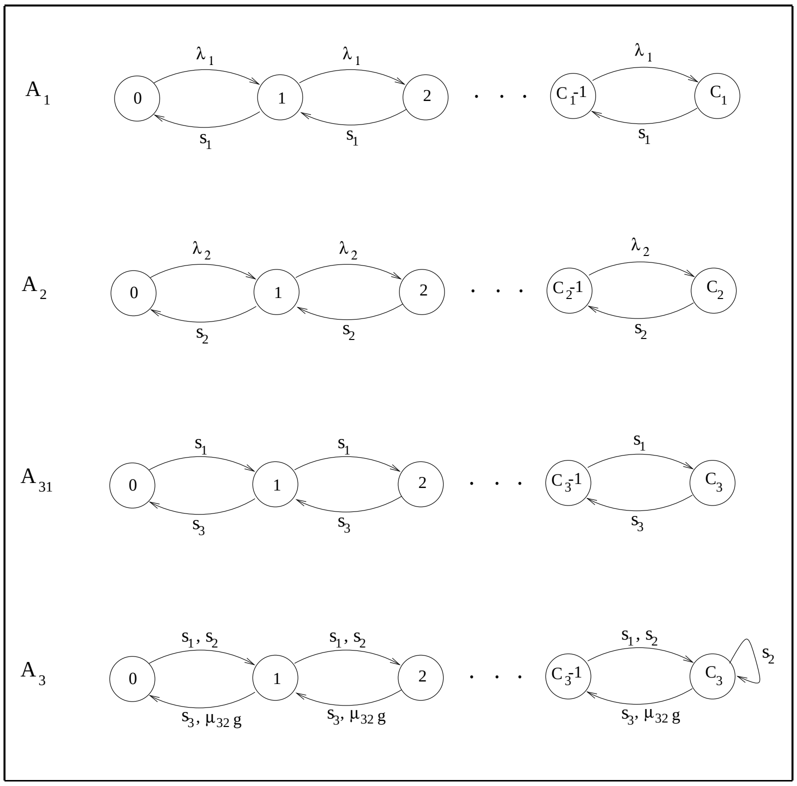


Figure 4: The SAN Model 1

3. The service completion in $queue_1$. Either there is an available slot in $queue_3$ and the customer joins it, or the buffer is full and therefore the customer is blocked in the server of $queue_1$. In both cases, both $queue_1$ and $queue_3$ are involved by this event. This is modelled using synchronization s_1 which appears in automata $\mathcal{A}_1$ and $\mathcal{A}_3$ with rate μ_1, and in $\mathcal{A}_{31}$ with a functional rate $\mu_1 f(x_3)$.

4. The service completion in $queue_2$. In this case, either there is an available slot in $queue_3$ and the customer joins it, or the buffer is full and then the customer is lost. This event is represented using synchronization s_2 with rate μ_2 in automata $\mathcal{A}_2$ and $\mathcal{A}_3$. The case of the loss of the customer is represented by the loop on the last state of $\mathcal{A}_3$.

5. The service completion in $queue_3$. If the customer is of class 1 this event is represented using synchronization s_3 with rate μ_{31} in automata $\mathcal{A}_{31}$ and $\mathcal{A}_3$. The service completion of a class 2 customer is represented by the transitions in $\mathcal{A}_3$ with the functional rate $\mu_3 g(x_{31})$.

Because of the mechanism provided by the SAN to set functional rates, this is not the unique way to model the queueing network of Figure 3. In the model described above (Figure 4), $queue_3$ is modelled considering two aspects of this buffer: the total number of class 1 customers and the total number of customers. Another way to model $queue_3$ is to consider the total number of class 1 customers and the total number of class 2 customers inside this buffer. The corresponding SAN model is given in Figure 5.

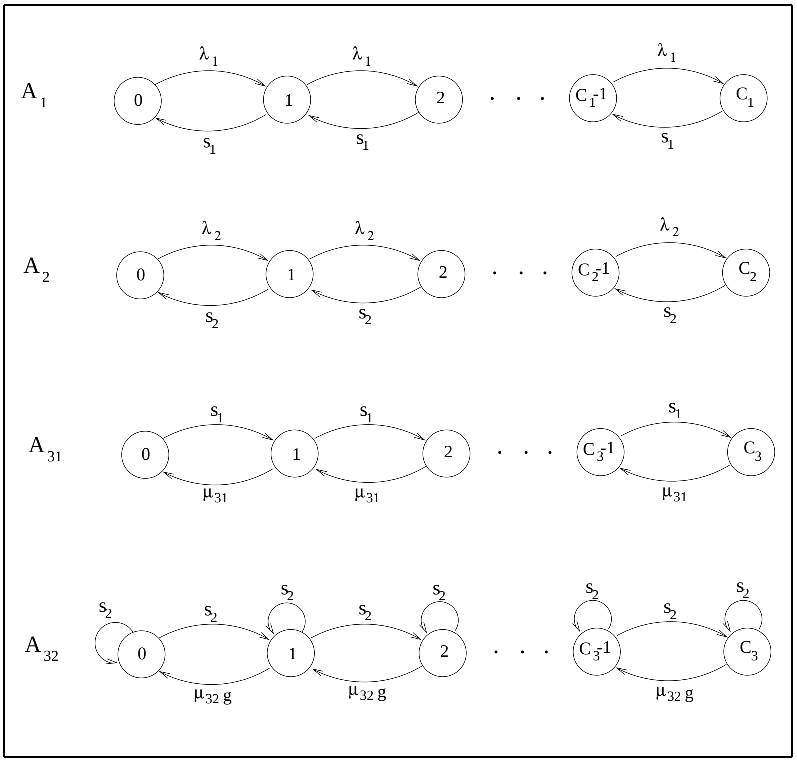


Figure 5: The SAN Model 2

As in the first model, automata $\mathcal{A}_1$ and $\mathcal{A}_2$ model the total number of customers in $queue_1$ and $queue_2$ respectively. $\mathcal{A}_{31}$ provides the total number of class 1 customers in $queue_3$ and $\mathcal{A}_{32}$ provides the total number of class 2 in the same queue.

The events that are taken into account in this model are the same as the different events enumerated in the first model. The first three events are represented are previously in the automata concerned. The others events are represented as follows:

- The service completion in $queue_2$. This event is represented using synchronization s_2 with rate μ_2 in automaton $\mathcal{A}_2$. In automaton $\mathcal{A}_{32}$, the rate is $\mu_2 f(x_{31})$ for the transitions from state i to $i+1$. Here f is a function defined as $f(x_{31}) = 1$ if the number of class 1 customers in $queue_3$ is less than C_3. Otherwise this function is null. The service completion of a class 2 customer and its loss is represented by the loops on the states with rate $\mu_2(1 - f(x_{31}))$ for all states except for state C_3 where the rate is μ_2.

- The service completion in $queue_3$. In this model, as we represent the customer number of each class in $queue_3$, we have to make a distinction between the service completion of both classes. The service completion of a class 1 customer is represented by a transition with rate μ_{31} in automaton $\mathcal{A}_1$. The service completion of class 2 customer is represented by a transition with rate $\mu_{32}g(x_{31})$. g is the same function as in the first model.

5.2 The PEPA Model

The PEPA model is composed of four components $\mathit{Buffer}_0^{(1)}$, $\mathit{Buffer}_0^{(2)}$, $\mathit{Buffer}_0^{(3)}$ and $Class1_0^{(3)}$. $\mathit{Buffer}_0^{(1)}$ and $\mathit{Buffer}_0^{(2)}$ describe the behaviour of $queue_1$ and $queue_2$ respectively. As in the SAN model, two components are required to describe the behaviour of $queue_3$; $\mathit{Buffer}_0^{(3)}$ describes the total number of customers of both classes and $Class1_0^{(3)}$ the number of class 1 customers in $queue_3$.

Component $\mathit{Buffer}_0^{(1)}$: To capture the behaviour of $queue_1$, we need to use four activities types: in_1 describes the arrival of a new customer in the buffer, $service_1$ models a customer's service, *blocking* represents the blocking after service possibility and finally $loss_1$ models the case where the buffer is full and therefore an arriving customer is lost. The complete behaviour of component $\mathit{Buffer}_0^{(1)}$ is as follows:

$$
\begin{array}{lcl}
\mathit{Buffer}_0^{(1)} & \stackrel{def}{=} & (in_1, \lambda_1).\mathit{Buffer}_1^{(1)} \\
\mathit{Buffer}_1^{(1)} & \stackrel{def}{=} & (in_1, \lambda_1).\mathit{Buffer}_2^{(1)} + (service_1, \mu_1).\mathit{Buffer}_0^{(1)} + (blocking, \mu_1).\mathit{Buffer}_1^{(1)} \\
\vdots & & \vdots \\
\mathit{Buffer}_{C_1}^{(1)} & \stackrel{def}{=} & (loss_1, \lambda_1).\mathit{Buffer}_{C_1}^{(1)} + (service_1, \mu_1).\mathit{Buffer}_{C_1-1}^{(1)} + (blocking, \mu_1).\mathit{Buffer}_{C_1}^{(1)}
\end{array}
$$

Component $\mathit{Buffer}_0^{(2)}$: This component describes the behaviour of $queue_2$. As for $queue_1$, we use four activities: in_2 describes the arrival of a new customer in the buffer, $service_2$ models a customer's service, $loss_2$ represents the loss of a customer because it arrives when $queue_2$ is full. Similarly $loss_2'$ represents the loss of a class 2 customer, when this one tried to join $queue_3$ which buffer is full. The complete behaviour of component $\mathit{Buffer}_0^{(2)}$ is as follows:

$$
\begin{array}{lcl}
\mathit{Buffer}_0^{(2)} & \stackrel{def}{=} & (in_2, \lambda_2).\mathit{Buffer}_1^{(2)} \\
\mathit{Buffer}_1^{(2)} & \stackrel{def}{=} & (in_2, \lambda_2).\mathit{Buffer}_2^{(2)} + (service_2, \mu_2).\mathit{Buffer}_0^{(2)} + (loss_2', \mu_2).\mathit{Buffer}_0^{(2)} \\
\vdots & & \vdots \\
\mathit{Buffer}_{C_2}^{(2)} & \stackrel{def}{=} & (loss_2, \lambda_2).\mathit{Buffer}_{C_2}^{(2)} + (service_2, \mu_2).\mathit{Buffer}_{C_2-1}^{(2)} + (loss_2', \mu_2).\mathit{Buffer}_{C_2-1}^{(2)}
\end{array}
$$

Component $\mathit{Buffer}_0^{(3)}$: This component describes the behaviour of $queue_3$ for both customers classes. The arrival of a customer to this queue means the end of service either in $queue_1$ (activity $service_1$) or in $queue_2$ (activity $service_2$). These activities are synchronizing activities whose rate in this component is unspecified. The service completion of a customer in $queue_3$ is modelled using activity $service_{31}$ or $service_{32}$ according to the class of this customer. Component $Buffer_0^{(3)}$ must synchronize with $Buffer_0^{(1)}$ on the activity $blocking$ when $queue_3$ is full and with $Buffer_0^{(2)}$ on activity $loss'_2$. The complete behaviour of component $Buffer_0^{(3)}$ is as follows:

$$
\begin{array}{lcl}
Buffer_0^{(3)} & \stackrel{def}{=} & (service_1, \top).Buffer_1^{(3)} + (service_2, \top).Buffer_1^{(3)} \\
Buffer_1^{(3)} & \stackrel{def}{=} & (service_1, \top).Buffer_2^{(3)} + (service_2, \top).Buffer_2^{(3)} + (service_{31}, \mu_{31}).Buffer_0^{(3)} \\
 & + & (service_{32}, \mu_{32}).Buffer_0^{(3)} \\
Buffer_2^{(3)} & \stackrel{def}{=} & (service_1, \top).Buffer_3^{(3)} + (service_2, \top).Buffer_3^{(3)} + (service_{31}, \mu_{31}).Buffer_1^{(3)} \\
 & + & (service_{32}, \mu_{32}).Buffer_1^{(3)} \\
\vdots & & \vdots \\
Buffer_{C_3}^{(3)} & \stackrel{def}{=} & (service_{31}, \mu_{31}).Buffer_{C_3-1}^{(3)} + (service_{32}, \mu_{32}).Buffer_{C_3-1}^{(3)} + (loss'_2, \top).Buffer_{C_3}^{(3)} \\
 & + & (blocking, \top).Buffer_{C_3}^{(3)}
\end{array}
$$

Component $Class1_0^{(3)}$: This component models the total number of class 1 customers in $queue_3$. The arrival of a class 1 customer to $queue_3$, means the service completion in $queue_1$. This is modelled here by activity $service_1$ on which this component must synchronize with components $Buffer_0^{(3)}$ and $Buffer_0^{(1)}$. When a class 1 customer is served in $queue_3$, activity $service_{31}$ is fired. A class 2 customer is served in this queue only if no class 1 customers are in the buffer. This is modelled by activity $service_{32}$ in $Class1_0^{(3)}$. Below is the complete definition of component $Class1_0^{(3)}$.

$$
\begin{array}{lcl}
Class1_0^{(3)} & \stackrel{def}{=} & (service_1, \top).Class1_1^{(3)} + (service_{32}, \top).Class1_0^{(3)} \\
Class1_1^{(3)} & \stackrel{def}{=} & (service_1, \top).Class1_2^{(3)} + (service_{31}, \top).Class1_0^{(3)} \\
\vdots & & \vdots \\
Class1_{C_3}^{(3)} & \stackrel{def}{=} & (service_{31}, \top).Class1_{C_3-1}^{(3)}
\end{array}
$$

The complete system: The complete behaviour of the system is given by the following equation:

$$System \stackrel{def}{=} (Buffer_0^{(1)} \underset{\mathcal{K}}{\bowtie} Buffer_0^{(3)} \underset{\mathcal{L}}{\bowtie} Buffer_0^{(2)}) \underset{\mathcal{M}}{\bowtie} Class1_0^{(3)}$$

where $\mathcal{K}$ is the cooperation set on which $Buffer_0^{(1)}$ and $Buffer_0^{(3)}$ must synchronize, $\mathcal{L}$ the cooperation set on which $Buffer_0^{(3)}$ and $Buffer_0^{(2)}$ must synchronize and finally $\mathcal{M}$ is the set on which $Class1_0^{(3)}$, $Buffer_0^{(1)}$ and $Buffer_0^{(3)}$ must synchronize.

$$
\begin{array}{lcl}
\mathcal{K} & = & \{service_1, blocking\} \\
\mathcal{L} & = & \{service_2, loss'_2\} \\
\mathcal{M} & = & \{service_1, service_{31}, service_{32}\}
\end{array}
$$

5.3 Summary of Case Study

The functional rates are a means to avoid an explicit modelling of all parts of a system's bahaviour. For example, a server is a part of a system that's usually modeled implicitly using the functional rates. In a SAN, such rates imply that some transitions rates of an automaton will depend on the state of one or several automata of the network.

The advantage of using such rates is the reduction of the model size. Indeed modelling implicitly a system part means using less automata for modelling the complete system. This benefits is more appreciated when building/solving the underlying Markov chain.

Moreover, the use of functional rates within SAN appears to offer the modeller greater flexibility in how to construct a model. We saw this when we considered the two alternative representations of $queue_3$. In the second representation we were able to implicitly capture the local state of one component ($\mathcal{A}_{31}$) within another ($\mathcal{A}_{32}$) using the functional dependency. In the PEPA model a component representing the local state class 1 customers only needs access to the total number of customers. This can only be achieved by giving the complementary component, $Class2_0^{(3)}$ say, intended to represent class 2 customers, the ability to witness all state changes within the queue, i.e. the representation of $Class2_0^{(3)}$ is forced to become $Buffer_0^{(3)}$.

6 Functional dependencies in PEPA

The flexibility provided by the functional dependencies may be added to PEPA models. For that we have to introduce the notion of functional rates in PEPA formalism which may be straitforward.

In PEPA the set of activities $\mathcal{A}ct$ is defined as $\mathcal{A}ct \subseteq \mathcal{A} \times \mathbb{R}^+$ where $\mathbb{R}^+$ is the set of positive real numbers defined as follows:

$$\mathbb{R}^+ = \{r | r > 0; r \in \mathbb{R}\} \cup \{\top\}$$

In the context of PEPA, a functional dependency may involve one or several components. In a functional dependency involving a single component, the rate value of one or several activities of the component depends on the current state of the component itself. This translates the presence of several apparent rates for an activity in a component. In this type of functional dependency, the rate value expressed as a function of the current component state is still a positive real number and can never be zero. However, this may not always be the case if the functional dependency involves two or several components. For example, a functional dependency between two components means that the behaviour of one component depends on the current state of the other one. This implies that either the activity to be performed by the first component and/or its rate value will be determined by the current state of the second component. The rate value may then be any positive real number of $\mathbb{R}^+$ including zero, particularly when the choice of the activity to be performed is done according to another component state.

The introduction of functional dependencies in PEPA requires now to release the constraint on the definition domain of an activity rate. Thus, the set of activities $\mathcal{A}ct$ is now defined as $\mathcal{A}ct \subseteq \mathcal{A} \times \mathbb{R}^*$ where $\mathbb{R}^*$ is the set of positive real numbers defined as follows:

$$\mathbb{R}^* = \{r | r \geq 0; r \in \mathbb{R}\} \cup \{\top\}$$

A typical example where the functional rate may be nul is the queueing network we have modeled in Section 5. If now we opte for functional rates, the system may then be modeled differently. Let f be a function of i, the number of class 1 customers and j the number of class 2 customers in $queue_3$ as follows:

$$f(i,j) = \begin{cases} 1 & if \;\; i+j < C_3 \\ 0 & otherwise \end{cases}$$

and let $g(i,j) = 1 - f(i,j)$. To simplify, in the following we use f and g instead of $f(i,j)$ and $g(i,j)$. The PEPA model is then composed of four components: $\mathit{Buffer}_0^{(1)}$, $\mathit{Buffer}_0^{(2)}$, $Class1_0^{(3)}$ and $Class2_0^{(3)}$. As in the previous PEPA model, $queue_1$ and $queue_2$ are modelled using respectivelly $\mathit{Buffer}_0^{(1)}$ and $\mathit{Buffer}_0^{(2)}$ as follows.

Component $\mathit{Buffer}_0^{(1)}$:

$$\begin{array}{lcl}
\mathit{Buffer}_0^{(1)} & \stackrel{def}{=} & (in_1, \lambda_1).\mathit{Buffer}_1^{(1)} \\
\mathit{Buffer}_1^{(1)} & \stackrel{def}{=} & (in_1, \lambda_1).\mathit{Buffer}_2^{(1)} + (service_1, \mu_1 \times f).\mathit{Buffer}_0^{(1)} + (blocking, \mu_1 \times g).\mathit{Buffer}_1^{(1)} \\
\vdots & & \vdots \\
\mathit{Buffer}_{C_1}^{(1)} & \stackrel{def}{=} & (loss_1, \lambda_1).\mathit{Buffer}_{C_1}^{(1)} + (service_1, \mu_1 \times f).\mathit{Buffer}_{C_1-1}^{(1)} + (blocking, \mu_1 \times g).\mathit{Buffer}_{C_1}^{(1)}
\end{array}$$

Component $\mathit{Buffer}_0^{(2)}$:

$$\begin{array}{lcl}
\mathit{Buffer}_0^{(2)} & \stackrel{def}{=} & (in_2, \lambda_2).\mathit{Buffer}_1^{(2)} \\
\mathit{Buffer}_1^{(2)} & \stackrel{def}{=} & (in_2, \lambda_2).\mathit{Buffer}_2^{(2)} + (service_2, \mu_2 \times f).\mathit{Buffer}_0^{(2)} + (loss_2', \mu_2 \times g).\mathit{Buffer}_0^{(2)} \\
\vdots & & \vdots \\
\mathit{Buffer}_{C_2}^{(2)} & \stackrel{def}{=} & (loss_2, \lambda_2).\mathit{Buffer}_{C_2}^{(2)} + (service_2, \mu_2 \times f).\mathit{Buffer}_{C_2-1}^{(2)} + (loss_2', \mu_2 \times g).\mathit{Buffer}_{C_2-1}^{(2)}
\end{array}$$

Opting for functional dependencies, $queue_3$ may then be represented by components $Class1_0^{(3)}$ and $Class2_0^{(3)}$. The first one models the total number of class 1 customers in $queue_3$ and the second the total number of class 2 customers in this queue. It is not necessary to explicitly represent the information about the total number of customers in this queue. The information provided by components $Class1_0^{(3)}$ and $Class2_0^{(3)}$ is now sufficient.

Component $Class1_0^{(3)}$:

$$\begin{array}{lcl}
Class1_0^{(3)} & \stackrel{def}{=} & (service_1, \top).Class1_1^{(3)} + (blocking, \top).Class1_0^{(3)} \\
Class1_1^{(3)} & \stackrel{def}{=} & (service_1, \top).Class1_2^{(3)} + (service_{31}, \mu_{31}).Class1_0^{(3)} + (blocking, \top).Class1_1^{(3)} \\
\vdots & & \vdots \\
Class1_i^{(3)} & \stackrel{def}{=} & (service_1, \top).Class1_{i+1}^{(3)} + (service_{31}, \mu_{31}).Class1_{i-1}^{(3)} + (blocking, \top).Class1_i^{(3)} \\
\vdots & & \vdots \\
Class1_{C_3}^{(3)} & \stackrel{def}{=} & (service_{31}, \mu_{31}).Class1_{C_3-1}^{(3)} + (blocking, \top).Class1_{C_3}^{(3)}
\end{array}$$

Component $Class2_0^{(3)}$:

$$
\begin{array}{lcl}
Class2_0^{(3)} & \stackrel{def}{=} & (service_2, \top).Class2_1^{(3)} + (loss'_2, \top).Class2_0^{(3)} \\
Class2_1^{(3)} & \stackrel{def}{=} & (service_2, \top).Class2_2^{(3)} + (service_{32}, \mu_{32} \times h).Class2_0^{(3)} + (loss'_2, \top).Class2_1^{(3)} \\
\vdots & & \vdots \\
Class2_j^{(3)} & \stackrel{def}{=} & (service_2, \top).Class2_{j+1}^{(3)} + (service_{32}, \mu_{32} \times h).Class2_{j-1}^{(3)} + (loss'_2, \top).Class2_j^{(3)} \\
\vdots & & \vdots \\
Class2_{C_3}^{(3)} & \stackrel{def}{=} & (service_{32}, \mu_{32} \times h).Class2_{C_3-1}^{(3)} + (loss'_2, \top).Class2_{C_3}^{(3)}
\end{array}
$$

Here h is a function which depends on the number of class 1 customers i in $queue_3$ as follows:

$$h(i) = \begin{cases} 1 & if \;\; i = 0 \\ 0 & otherwise \end{cases}$$

The complete system:

$$System \stackrel{def}{=} (Buffer_0^{(1)} \underset{\mathcal{K}'}{\bowtie} Class1_0^{(3)}) \parallel (Buffer_0^{(2)} \underset{\mathcal{L}'}{\bowtie} Class2_0^{(3)})$$

where

$$\mathcal{K}' = \{service_1, blocking\} \qquad\qquad \mathcal{L}' = \{service_2, loss'_2\}$$

Besides the flexibility provided by the functional dependency, we may now be able to reduce the PEPA model size in cases where some parts of a system may be implicitly modelled. However, using functional rates means also that the transitions values will be known only when solving the underlying Markov chain. This may imply a certain complexity in computing these rates. This issue will be discussed later in the paper.

7 Tensor Representation

The modelling formalisms SAN and PEPA are prone to state space explosion. Therefore, both formalisms incorporate techniques for overcoming this problem.

To tackle the state space explosion problem, Plateau [4] has proved that the generator matrix of the Markov process underlying a SAN model can be analytically represented using Kronecker algebra. The solution of the model can then be achieved via this tensor expression of submatrices—the complete matrix does not need to be generated.

Similarly, to overcome the state space problem for a PEPA model, one of the solutions proposed is a tensor representation of the corresponding Markov chain. In [8], we have showed that a PEPA model can be represented analytically using Kronecker algebra and solved without constructing the complete generator matrix. Just as for SAN models, we have showed that the translation from the model to the compact representation is automatic.

The ability to represent the Markov chain associated with a SAN or a PEPA model by a compact formula is particularly important since it may allow the modeller to deal with systems which may have large state-spaces.

7.1 The SAN descriptor

In a stochastic automata network, an automaton itself is not Markovian. The Markovian assumption holds only for the global SAN behaviour if we assume exponential distribution and independence for the firing of the transitions. Note that the states of the automata remain distinct and the *global state* of the system is simply the collection of *local states* corresponding to each constituent automaton.

The generator matrix Q is automatically translated from the SAN description . This translation is based on tensor algebra of matrices. It has been proved in [4] that, if the states are in a lexicographic order, then the generator matrix Q of the Markov chain associated with a continuous-time SAN is given by:

$$Q = \bigoplus_{i=1}^{N} F_i + \sum_{e \in \varepsilon} \tau_e \left(\bigotimes_{i=1}^{N} P_{i,e} + \bigotimes_{i=1}^{N} \overline{P}_{i,e} \right) \tag{7.1}$$

where

- N is the total number of automata in the network and ε is the set of synchronizations.
- F_i is the transition matrix of automaton $\mathcal{A}_i$ without synchronizations.
- $P_{i,e}$ is the transition matrix of automaton $\mathcal{A}_i$ due to synchronization e which rate is τ_e.
- $\overline{P}_{i,e}$ is a matrix representing the normalization associated with the synchronization e on automaton $\mathcal{A}_i$.
- $\bigoplus$ and $\bigotimes$ denote tensor sum and product, respectively.

Unlike the local transition matrices F_i, the synchronizing matrices $P_{i,e}$ are not generators. The diagonal corrector matrices $\overline{P}_{i,e}$ have been introduced to normalize the synchronizing matrices.

Note that the transition matrix of the Markov chain associated with a discrete-time SAN exists and is given by an equivalent formula where the tensor sum is replaced by the tensor product. This is due to the fact that in the discrete-time basis, several events may happen at once. That phenomenon is caught by the tensor product.

7.2 The PEPA descriptor

In [8], we have showed that PEPA models can be represented analytically using Kronecker algebra and solved without constructing the complete generator matrix. We proceed similarly to SANs: firstly, we capture local transitions for each constituent component in an appropriate generator matrix, then we represent each type of interaction by the tensor product of matrices capturing each component's capacity to participate in this shared activity. In order to ensure that the rate of the shared activity is represented correctly we need to exercise a little care in constructing this set of tensor products. Finally, and analogously to the SAN case, we need to introduce diagonal corrector matrices which normalize the cooperation matrices.

The generator matrix Q of the Markov chain associated with a PEPA model is

$$Q = \bigoplus_{i=1}^{N} R_i + \sum_{\alpha \in \mathcal{Z}} r_\alpha \left(\bigotimes_{i=1}^{N} P_{i,\alpha} - \bigotimes_{i=1}^{N} \overline{P}_{i,\alpha} \right) \tag{7.2}$$

where

- N is the total number of components in the PEPA model and $\mathcal{Z}$ the set of cooperating action types, determined syntactically.
- r_α is the minimum of the functional rates of action type α over all components C_i, $i = 1 \ldots N$.
- R_i is the transition matrix of component C_i relating solely to its individual activities.
- $P_{i,\alpha}$ is the probability transition matrix of component C_i due to activity of type α. Its elements' values are between 0 and 1.
- $\overline{P}_{i,\alpha}$ is a matrix representing the normalization associated with the shared activity α in component C_i.

Unlike the local transition matrices R_i, the cooperation matrices $P_{i,\alpha}$ are not generators. So we need to introduce diagonal corrector matrices $\overline{P}_{i,\alpha}$ to normalize the cooperation matrices, i.e. ensure that row sums are zero. This is shown in (7.2).

In order to apply the equation above we must place a restriction on the use of types within cooperation sets, to ensure that each action type uniquely defines a synchronisation event. To see the need for this restriction, consider the model $M \stackrel{def}{=} (V \underset{\{\alpha\}}{\bowtie} S) \parallel (T \underset{\{\alpha\}}{\bowtie} U)$. Assuming that each component enables α with just one apparent rate $r_\alpha(V)$, $r_\alpha(S)$, etc., applying the equation above, we write the generator matrix of this model as follows:

$$\begin{aligned} Q \;\; = \;\; & \bigoplus_{i=1}^{4} R_i + \min(r_\alpha(V), r_\alpha(S), r_\alpha(T), r_\alpha(U)) \, (P_{V,\alpha} \otimes P_{S,\alpha} \otimes P_{T,\alpha} \otimes P_{U,\alpha} \\ & \qquad - \; \overline{P}_{V,\alpha} \otimes \overline{P}_{S,\alpha} \otimes \overline{P}_{T,\alpha} \otimes \overline{P}_{U,\alpha}) \end{aligned}$$

It is clear that in this representation all the components are forced to make the *same* α cooperation. However, applying the semantics, there are two potential shared α activities: one involving V and S and one involving T and U. These can proceed concurrently with each other.

Thus we preprocess the model: when the same action type appears in distinct cooperation sets we rename the action type in the appropriate components and cooperation sets so that they are distinguished in $\mathcal{Z}$. For example, in the model above, we might distinguish α_ℓ (affecting V and S) and $\alpha_\mathfrak{r}$ (affecting T and U) and rename all α activities in V, S, T and U appropriately.

8 Conclusions

In this paper, we have investigated the introduction of functional dependencies in PEPA and their impact on the PEPA model and the generator matrix. In PEPA, these functions capture the exact timing behaviour of cooperating PEPA activities. Through examples, we have shown the advantages of using these dependencies in terms of flexibility in constructing the model and in some cases in terms of reduction of the model size. Moreover the use of functional dependencies allow us to have a direct tensorial representation of the Markov process underlying a PEPA model similar to the one used in SAN formalism.

References

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

[2] M. Ajmone Marsan, G. Conte, and G. Balbo. A Class of Generalised Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems*, 2(2):93–122, May 1984.

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

[4] B. Plateau. *De l'Evolution du Parallélisme et de la Synchronisation.* PhD Thesis, Université de Paris-Sud, Orsay, 1984.

[5] 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.

[6] B. Plateau and J.M. Fourneau. A methodology for solving markov models of parallel systems. *Journal of Parallel and Distributed Computing*, August 1991.

[7] J. Hillston. *A Compositional Approach to Performance Modelling.* Phd. Thesis, University of Edinburgh, 1994.

[8] J. Hillston and L. Kloul. An efficient kronecker representation for pepa models. *Lecture Notes in Computer Science, Proceedings of the Joint International Workshop, PAPM-PROBMIV 2001, Aachen*, (2165):120–135, 2001.