Stochastic Automata Networks: Product Forms and Iterative Solutions

Brigitte Plateau, William J. Stewart

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Stochastic Automata Networks: Product Forms and Iterative Solutions.

Brigitte Plateau and William J. Stewart

N° 2939

12 juillet 1996

THÈME 1

ISSN 0249-6399

INRIA
RHÔNE-ALPES

Stochastic Automata Networks: Product Forms and Iterative Solutions.

Brigitte Plateau* and William J. Stewart†

Thème 1 — Réseaux et systèmes
Projet APACHE

Rapport de recherche n ° 2939 — 12 juillet 1996 — 38 pages

Abstract: This article presents a global overview of recent results concerning stochastic automata networks. Among the topics considered is the formalism of an extended tensor algebra, the rigorous definition of a Markovian generator in the form of a descriptor, sufficient conditions for product form, the complexity of the vector-descriptor multiplication, the optimization of this product and some numerical results. The whole is illustrated by numerous typical examples.

Key-words: Markov chains, Stochastic automata networks, Product forms, Iterative solutions, Vector-descriptor multiplication.

(Résumé : tsvp)

* IMAG–LMC, 100 rue des Mathématiques, 38041 Grenoble cedex, France. Research supported by the (CNRS – INRIA – INPG – UJF) joint project *Apache*.

† Department of Computer Science, N. Carolina State University, Raleigh, N.C. 27695-8206, USA. Research supported in part by NSF (DDM-8906248 and CCR-9413309).

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN (France)
Téléphone : (33) 76 61 52 00 – Télécopie : (33) 76 61 52 52

Les Réseaux d'Automates Stochastiques: Formes Produits et Solutions Itératives.

Résumé : Cet article présente une vue d'ensemble de résultats récents sur les automates stochastiques: le formalisme de l'algèbre tensorielle étendue, la formalisation du générateur Markovien sous forme de descriptor, des conditions suffisantes de formes produits, la complexité du produit vecteur-descripteur, des optimisations de ce produit et quelques résultats numériques. Ce travail est illustré par de nombreux exemples typiques.

Mots-clé : Chaîne de Markov, Réseau d'automates stochastiques, Formes produits, Produit vecteur-descripteur.

1 Introduction

A *Stochastic Automata Network* (SAN) consists of a number of individual stochastic automata that operate more or less independently of each other. Each individual automaton, $\mathcal{A}$, is represented by a number of states and rules that govern the manner in which it moves from one state to the next. The state of an automaton at any time t is just the state it occupies at time t and the state of the SAN at time t is given by the state of each of its constituent automata.

The use of stochastic automata networks is becoming increasingly important in performance modelling issues related to parallel and distributed computer systems. As such models become increasingly complex, so also does the complexity of the modelling process. Although systems analysts have a number of other modelling strategems at their disposal, it is not unusual to discover that these are inadequate. The use of queueing network modelling is limited by the constraints imposed by assumptions needed to keep the model tractable. The results obtained from the myriad of available approximate solutions are frequently too gross to be meaningful. Simulations can be excessively expensive. This leaves models that are based on Markov chains, but here also, the difficulties are well documented. The size of the state space generated is so large that it effectively prohibits the computation of a solution. This is true whether the Markov chain results from a stochastic Petri net formalism, or from a straightforward Markov chain analyzer.

In many instances, the SAN formalism is an appropriate choice. Parallel and distributed systems are often viewed as collections of components that operate more or less independently, requiring only infrequent interaction such as synchronizing their actions, or operating at different rates depending on the state of parts of the overall system. This is exactly the viewpoint adopted by SANs. The components are modelled as individual stochastic automata that interact with each other. Furthermore, the state space explosion problem associated with Markov chain models is mitigated by the fact that the state transition matrix is not stored, nor even generated. Instead, it is represented by a number of much smaller matrices, one for each of the stochastic automata that constitute the system, and from these all relevent information may be determined without explicitly forming the global matrix. The implication is that a considerable saving in memory is effected by storing the matrix in this fashion. We do not wish to give the impression that we regard SANs as a panacea for all modelling problems, just that there is a niche that it fills among the tools that modellers may use. It is fairly obvious that their memory requirements are minimal; it remains to show that this does not come at the cost of a prohibitive amount of computation time.

Stochastic Automata Networks and the related concept of *Stochastic Process Algebras* have become a hot topic of research in recent years. This research has focused on areas such as the development of languages for specifying SANs and their ilk, [23, 24], and on the development of suitable solution methods that can operate on the transition matrix given as a compact SAN descriptor. The development of languages for specifying stochastic process algebras is mainly concerned with structural properties of the nets (compositionality, equivalence, etc.) and with the mapping of these specifications onto Markov chains for the computation of performance measures [24, 3, 7]. Although a SAN may be viewed as a

stochastic process algebra, its original purpose was to provide an efficient and convenient methodology for the study of quantitive rather than structural properties of complex systems, [32]. Nevertheless, computational results such as those discussed in this article can also be applied in the context of stochastic process algebras.

There are two overriding concerns in the application of any Markovian modelling methodology, viz., memory requirements and computation time. Since these are frequently functions of the number of states, a first approach is to develop techniques that minimize the number of states in the model. In SANs, it is possible to make use of symmetries as well as lumping and various superpositioning of the automata to reduce the computational burden, [1, 9, 39]. Furthermore, in [17], structural properties of the Markov chain graph (specificially the occurrence of cycles) are used to compute steady state solutions. We point out that similar, and even more extensive results have previously been developed in the context of Petri nets and stochastic activity networks. For example, in [8, 9, 10, 18, 38, 40], equivalence relations and symmetries are used to decrease the computational burden of obtaining performance indices. In [21], reduction techniques for Petri nets are used in conjunction with insensitivity results to enable all computations to be performed on a reduced set of markings. In [11], nearly independent subnets are exploited in an iterative procedure in which a global solution is obtained from partial solutions. In [13] it is shown that the tensor structure of the transition matrix may be extracted from a stochastic Petri net, and in [27] that this can be used efficiently to work with the reachable state space in an iterative procedure.

Once the number of states has effectively been fixed, the problem of memory and computation time still must be addressed, for the number of states left may still be large. With SANs, the use of a compact descriptor goes a long way to satisfying the first of these, although with the need to keep a minimum of two vectors of length equal to the global number of states, and considerably more than two for more sophisticated procedures such as the GMRES method, we cannot afford to become complacent about memory requirements. As far as computation time is concerned, since the numerical methods used are iterative, it is important to keep both the number of iterations and the amount of computation per iteration to a minimum.

Our paper is laid out as follows. Sections 2 through 4 present an informal introduction to SANs and tensor algebra. This is followed in Section 5 with the presentation of a number of sufficient conditions for the existence of product forms in SANs, for in some restricted cases, product forms can indeed be found. We note in passing that in [4, 6, 14, 19, 22, 28] product forms have been found in Petri nets models, using either the structure of the state space or flow properties. The numerical issues of computation time and memory requirements in computing stationary distributions by means of iterative methods are discussed in Sections 6 through 10.

2 Basic Properties of Tensor Algebra

Define two matrices A and B as follows:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad and \quad B = \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \end{pmatrix}.$$

The *tensor product* $C = A \otimes B$ is given by

$$C = \begin{pmatrix} a_{11}B & a_{12}B \\ a_{21}B & a_{22}B \end{pmatrix} \tag{1}$$

In general, to define the tensor product of two matrices, A of dimensions $(\rho_1 \times \gamma_1)$ and B of dimensions $(\rho_2 \times \gamma_2)$, it is convenient to observe that the tensor product matrix has dimensions $(\rho_1\rho_2 \times \gamma_1\gamma_2)$ and may be considered as consisting of $\rho_1\gamma_1$ blocks each having dimensions $(\rho_2 \times \gamma_2)$, i.e., the dimensions of B. To specify a particular element, it suffices to specify the block in which the element occurs and the position within that block of the element under consideration. Thus, in the above example, the element c_{47} $(= a_{22}b_{13})$ is in the $(2,2)$ block and at position $(1,3)$ of that block.
The *tensor sum* of two *square* matrices A and B is defined in terms of tensor products as

$$A \oplus B = A \otimes I_{n_2} + I_{n_1} \otimes B \tag{2}$$

where n_1 is the order of A; n_2 the order of B; I_{n_i} the identity matrix of order n_i and "+" represents the usual operation of matrix addition. Since both sides of this operation (matrix addition) must have identical dimensions, it follows that tensor addition is defined for square matrices only. For example, with

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad and \quad B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}, \tag{3}$$

the tensor sum $C = A \oplus B$ is given by

$$C = \left(\begin{array}{ccc|ccc} a_{11}+b_{11} & b_{12} & b_{13} & a_{12} & 0 & 0 \\ b_{21} & a_{11}+b_{22} & b_{23} & 0 & a_{12} & 0 \\ b_{31} & b_{32} & a_{11}+b_{33} & 0 & 0 & a_{12} \\ \hline a_{21} & 0 & 0 & a_{22}+b_{11} & b_{12} & b_{13} \\ 0 & a_{21} & 0 & b_{21} & a_{22}+b_{22} & b_{23} \\ 0 & 0 & a_{21} & b_{31} & b_{32} & a_{22}+b_{33} \end{array}\right).$$

Some important properties of tensor products and additions are

- Associativity:
  $A \otimes (B \otimes C) = (A \otimes B) \otimes C$ and $A \oplus (B \oplus C) = (A \oplus B) \oplus C$.

- Distributivity over (ordinary matrix) addition:
  $(A + B) \otimes (C + D) = A \otimes C + B \otimes C + A \otimes D + B \otimes D$.
- Compatibility with (ordinary matrix) multiplication:
  $(A \times B) \otimes (C \times D) = (A \otimes C) \times (B \otimes D)$.
- Compatibility with (ordinary matrix) inversion:
  $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$.

The associativity property implies that the operations $\bigotimes_{k=1}^{N} A^{(k)}$ and $\bigoplus_{k=1}^{N} A^{(k)}$ are well defined. In particular, observe that the tensor sum of N terms may be written as the (usual) matrix sum of N terms, each term consisting of an N-fold tensor product. We have

$$\bigoplus_{k=1}^{N} A^{(k)} = \sum_{k=1}^{N} I_{n_1} \otimes \cdots \otimes I_{n_{k-1}} \otimes A^{(k)} \otimes I_{n_{k+1}} \otimes \cdots \otimes I_{n_N},$$

where n_k is the order of the matrix $A^{(k)}$ and I_{n_k} is the identity matrix of order n_k.

The operators $\otimes$ and $\oplus$ are not commutative. However, we shall have need of a pseudo-commutativity property that may be formalized as follows. Let σ be a permutation of the set of integer $[1, 2, \ldots, N]$. Then there exists a permutation matrix, P_σ, of order $\prod_{i=1}^{N} n_i$, such that

$$\bigotimes_{k=1}^{N} A^{(k)} \;=\; P_\sigma \bigotimes_{k=1}^{N} A^{(\sigma(k))} P_\sigma^T.$$

A proof of this property may be found in [33] wherein P_σ is explicitly given. Further information concerning the properties of tensor algebra may be found in Davio [12].

3 Stochastic Automata Networks

3.1 Non-Interacting Stochastic Automata

Consider the case of a system that may be modelled by two completely independent stochastic automata, each of which may be represented by a discrete-time Markov chain. Let us assume that the first automaton, denoted $\mathcal{A}^{(1)}$, has n_1 states and stochastic transition probability matrix given by $P^{(1)} \in \mathcal{R}^{n_1 \times n_1}$. Similarly, let $\mathcal{A}^{(2)}$ denote the second automaton; n_2, the number of states in its representation and $P^{(2)} \in \mathcal{R}^{n_2 \times n_2}$, its stochastic transition probability matrix. The state of the overall (two-dimensional) system may be represented by the pair (i, j) where $i \in \{1,\ 2,\ \ldots,\ n_1\}$ and $j \in \{1,\ 2,\ \ldots,\ n_2\}$. Indeed the stochastic transition probability matrix of the two-dimensional system is given by the *tensor product* of the matrices $P^{(1)}$ and $P^{(2)}$. If, instead of being represented by two *discrete-time* Markov chains, the stochastic automata are characterized by *continuous-time* Markov chains with infinitesimal generators, $Q^{(1)}$ and $Q^{(2)}$ respectively, the infinitesimal generator of the two-dimensional system is given by the *tensor sum* of $Q^{(1)}$ and $Q^{(2)}$. Throughout this article,

we present results on the basis of continuous-time SANs, although the results are equally valid in the context of discrete-time SANs.

Continuing with the example, let $\pi_i^{(1)}(t)$ be the probability that the first automaton is in state i at time t and $\pi_j^{(2)}(t)$ the probability that the second is in state j, at time t. Then the probability that, at time t, the first is in state i *and* the second is in state j is simply the product $\pi_i^{(1)}(t) \times \pi_j^{(2)}(t)$. Furthermore, the probability distribution of the overall (two-dimensional) system is given by the tensor product of the two individual probability vectors, $\pi^{(1)} \in \mathcal{R}^{n_1}$ and $\pi^{(2)} \in \mathcal{R}^{n_2}$, viz: $\pi^{(1)} \otimes \pi^{(2)}$.

Now given N *independent* stochastic automata, $\mathcal{A}^{(1)}$, $\mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(N)}$, with associated infinitesimal generators, $Q^{(1)}$, $Q^{(2)}, \ldots, Q^{(N)}$, and probability distributions $\pi^{(1)}(t)$, $\pi^{(2)}(t), \ldots,$ $\pi^{(N)}(t)$ at time t, the infinitesimal generator of the N–dimensional system, which we shall refer to as the *global generator*, is given by

$$Q = \bigoplus_{k=1}^{N} Q^{(k)} = \sum_{k=1}^{N} I_{n_1} \otimes \cdots \otimes I_{n_{k-1}} \otimes Q^{(k)} \otimes I_{n_{k+1}} \otimes \cdots \otimes I_{n_N}.$$

The probability that the system is in state $(i_1,\ i_2,\ \ldots,\ i_N)$ at time t, where i_k is the state of the k^{th} automaton at time t with $1 \leq i_k \leq n_k$ and n_k is the number of states in the k^{th} automaton, is given by $\prod_{k=1}^{N} \pi_{i_k}^{(k)}(t)$ where $\pi_{i_k}^{(k)}(t)$ is the probability that the k^{th} automaton is in state i_k at time t. Furthermore, the probability distribution of the N-dimensional system, $\pi(t)$, is given by the tensor product of the probability vectors of the individual automaton at time t, i.e.,

$$\pi(t) = \bigotimes_{k=1}^{N} \pi^{(k)}(t). \tag{4}$$

To solve N-dimensional systems that are formed from independent stochastic automata is therefore very simple. It suffices to solve for the probability distributions of the individual stochastic automata and to form the tensor product of these distributions. Although such systems may exist, the more usual case occurs when the transitions of one automaton may depend on the state of a second. It is to this topic that we now turn.

3.2 Interacting Stochastic Automata

There are basically two ways in which stochastic automata interact:

1. The rate at which a transition may occur in one automaton may be a *function* of the state of other automata. Such transitions are called *functional* transitions.

2. A transition in one automaton may *force* a transition to occur in one or more other automata. We allow for both the possibility of a *master/slave* relationship, in which an action in one automaton (the master) actually occasions a transition in one or more other automata (the slaves), and for the case of a *rendez-vous* in which the presence (or absence) of two or more automata in designated states causes (or prevents) transitions

to occur. We refer to such transitions collectively under the name of *synchronized* transitions. Synchronized transitions are triggered by a synchronizing event; indeed, a single synchronizing event will generally cause multiple synchronized transitions. Synchronized transitions may also be functional.

The elements in the matrix representation of any single stochastic automaton are either constants, i.e., nonnegative real numbers, or functions from the global state space to the nonnegative reals. Transition rates that depend only on the state of the automaton itself, and not on the state of any other automaton, are to all intents and purposes, constant transition rates. A synchronizing transition may be either functional or constant. In any given automaton, transitions that are not synchronizing transitions are said to be *local* transitions.

Consider as an example, a simple queueing network consisting of two service centers in tandem and an arrival process that is Poisson at rate λ. Each service center consists of an infinite queue and a single server. The service time distribution of the first server is assumed to be exponential at fixed rate μ, while the service time distribution at the second is taken to be exponential with a rate ν that varies with the number and distribution of customers in the network. Since a state of the network is completely described by the pair (n_1, n_2) where n_1 denotes the number of customers at station 1 and n_2 the number at station 2, the service rate at station 2 is more properly written as $\nu(n_1, n_2)$.

We may define two stochastic automata $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ corresponding to the two different service centers. The state space of each is given by the set of nonnegative integers $\{0,\ 1,\ 2,\ \ldots,\ \}$ since any nonnegative number of customers may be in either station. It is apparent that the first automaton $\mathcal{A}^{(1)}$ is completely independent of the second. On the other hand, transitions in $\mathcal{A}^{(2)}$ depend on the first automaton in two ways. Firstly the rate at which customers are served in the second station depends on the number of customers in the network and hence, in particular, on the number at the first station. Thus $\mathcal{A}^{(2)}$ contains functional transition rates, $(\nu(n_1,\ n_2))$. Secondly, when a departure occurs from the first station, a customer enters the second and therefore instantaneously forces a transition to occur within the second automaton. The state of the second automaton is instantaneously changed from n_2 to $n_2 + 1$! This entails transitions of the second type, namely synchronizing transitions.

Let us examine how these two different types of interaction may be specified in a stochastic automata network. Consider first the case of constant and functional transition rates on local transitions; Normally an automaton will contain both. The elements in the infinitesimal generator of any single stochastic automaton are either

- constants, i.e., nonnegative real numbers, or
- functions from the global state space to the nonnegative reals.

Transition rates that depend only on the state of the automaton itself, and not on the state of any other automaton, are to all intents and purposes, constant transition rates. This is

the case if the rate of transition of the second exponential server in the two-station queueing network example is *load-dependent*, i.e., depending only on the number of customers present in station 2. The *load-dependent* value is instantiated at each state and taken to be that constant value in the rest of the analysis. Constant transition rates obviously pose no problem.

If a given state of a stochastic automaton occasions functional transitions, the rate of these transitions must be evaluated according to their defining formulae and the current global state. In certain instances it may be advantageous to evaluate these functional rates only once and to store the values obtained in an array from which they may be retrieved as and when they are needed. In other cases, it may be better to leave them in their original form and to re-evaluate the formula each time.

Whether the transition rates are constant or functional, it is important to note that only the state of the *local* automaton is affected. Therefore all the information concerning constant and functional transition rates within an automaton can be handled within that automaton (assuming only that that automaton has a knowledge of the global system state). *Functional transitions affect the global system only in that they change the state of a single automaton.*

Now consider synchronizing events. The two-station queueing network example given previously clearly shows that the first of the two stochastic automata is independent of the second. However, this does not mean that the complete set of information needed to specify the synchronizing event can be confined to $\mathcal{A}^{(2)}$ — unfortunately. It is not sufficient for $\mathcal{A}^{(2)}$ to realize that it is susceptible to instantaneous transitions forced upon it by another automaton. Additionally $\mathcal{A}^{(1)}$, although completely independent, needs to participate in a mechanism to dissimulate the fact that it executes synchronizing transitions. One way to implement this is to generate a list of all possible synchronizing events that can occur in a SAN. This list needs to provide a unique name for each synchronizing event, the manner in which it occurs and its effect on other stochastic automata. In contrast to functional transitions, *synchronizing events affect the global system by possibly altering the state of several automata.*

3.3 The Effect of Synchronizing Events

We begin with a small example of two[1] interacting stochastic automata, $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$, whose infinitesimal generator matrices are given by

$$Q^{(1)} = \begin{pmatrix} -\lambda_1 & \lambda_1 \\ \lambda_2 & -\lambda_2 \end{pmatrix} \quad \textit{and} \quad Q^{(2)} = \begin{pmatrix} -\mu_1 & \mu_1 & 0 \\ 0 & -\mu_2 & \mu_2 \\ \mu_3 & 0 & -\mu_3 \end{pmatrix}$$

respectively. At the moment, neither contains synchronizing events nor functional transition rates. The infinitesimal generator of the global, two-dimensional system is therefore given as

[1] The extension to more than two is immediate.

$$Q^{(1)} \oplus Q^{(2)} =$$

$$\left(\begin{array}{ccc|ccc} -(\lambda_1+\mu_1) & \mu_1 & 0 & \lambda_1 & 0 & 0 \\ 0 & -(\lambda_1+\mu_2) & \mu_2 & 0 & \lambda_1 & 0 \\ \mu_3 & 0 & -(\lambda_1+\mu_3) & 0 & 0 & \lambda_1 \\ \hline \lambda_2 & 0 & 0 & -(\lambda_2+\mu_1) & \mu_1 & 0 \\ 0 & \lambda_2 & 0 & 0 & -(\lambda_2+\mu_2) & \mu_2 \\ 0 & 0 & \lambda_2 & \mu_3 & 0 & -(\lambda_2+\mu_3) \end{array}\right). \quad (5)$$

Its probability distribution vector (at time t) is obtained by forming the tensor product of the individual probability distribution vectors of each stochastic automata at time t.

Let us now observe the effect of introducing synchronizing events. Suppose that each time automaton $\mathcal{A}^{(1)}$ generates a transition from state 2 to state 1 (at rate λ_2), it forces the second automaton into state 1. It may be readily verified that the global generator matrix is given by

$$\left(\begin{array}{ccc|ccc} -(\lambda_1+\mu_1) & \mu_1 & 0 & \lambda_1 & 0 & 0 \\ 0 & -(\lambda_1+\mu_2) & \mu_2 & 0 & \lambda_1 & 0 \\ \mu_3 & 0 & -(\lambda_1+\mu_3) & 0 & 0 & \lambda_1 \\ \hline \lambda_2 & 0 & 0 & -(\lambda_2+\mu_1) & \mu_1 & 0 \\ \lambda_2 & 0 & 0 & 0 & -(\lambda_2+\mu_2) & \mu_2 \\ \lambda_2 & 0 & 0 & \mu_3 & 0 & -(\lambda_2+\mu_3) \end{array}\right).$$

If, in addition, the second automaton $\mathcal{A}^{(2)}$ initiates a synchronizing event each time it moves from state 3 to state 1 (at rate μ_3), by for example forcing the first automaton into state 1, we obtain the following global generator.

$$\left(\begin{array}{ccc|ccc} -(\lambda_1+\mu_1) & \mu_1 & 0 & \lambda_1 & 0 & 0 \\ 0 & -(\lambda_1+\mu_2) & \mu_2 & 0 & \lambda_1 & 0 \\ \mu_3 & 0 & -(\lambda_1+\mu_3) & 0 & 0 & \lambda_1 \\ \hline \lambda_2 & 0 & 0 & -(\lambda_2+\mu_1) & \mu_1 & 0 \\ \lambda_2 & 0 & 0 & 0 & -(\lambda_2+\mu_2) & \mu_2 \\ \lambda_2+\mu_3 & 0 & 0 & 0 & 0 & -(\lambda_2+\mu_3) \end{array}\right).$$

Our immediate reaction in observing these altered matrices may be to assume that a major disadvantage of incorporating synchronizing transitions is to remove the possibility of representing the global transition rate matrix as a (sum of) tensor products. However, Plateau [31] has shown that, by separating local transitions from synchronizing transitions, this is not necessarily so; that the global transition rate matrix can still be written as a (sum of) tensor products. To observe this we proceed as follows.

The transitions at rates λ_1, μ_1 and μ_2 are not synchronizing events, but rather *local* transitions. The part of the global generator that consists uniquely of local transitions may be obtained by forming the tensor sum of infinitesimal generators $Q_l^{(1)}$ and $Q_l^{(2)}$ that represent only local transitions, viz:.

$$Q_l^{(1)} = \begin{pmatrix} -\lambda_1 & \lambda_1 \\ 0 & 0 \end{pmatrix} \quad and \quad Q_l^{(2)} = \begin{pmatrix} -\mu_1 & \mu_1 & 0 \\ 0 & -\mu_2 & \mu_2 \\ 0 & 0 & 0 \end{pmatrix},$$

with tensor sum

$$Q_l = Q_l^{(1)} \oplus Q_l^{(2)} = \left(\begin{array}{ccc|ccc} -(\lambda_1+\mu_1) & \mu_1 & 0 & \lambda_1 & 0 & 0 \\ 0 & -(\lambda_1+\mu_2) & \mu_2 & 0 & \lambda_1 & 0 \\ 0 & 0 & -\lambda_1 & 0 & 0 & \lambda_1 \\ \hline 0 & 0 & 0 & -\mu_1 & \mu_1 & 0 \\ 0 & 0 & 0 & 0 & -\mu_2 & \mu_2 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

The rates λ_2 and μ_3 are associated with two synchronizing events that we call e_1 and e_2 respectively. The part of the global generator that is due to the first synchronizing event is given by

$$Q_{e_1} = \left(\begin{array}{ccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \lambda_2 & 0 & 0 & -\lambda_2 & 0 & 0 \\ \lambda_2 & 0 & 0 & 0 & -\lambda_2 & 0 \\ \lambda_2 & 0 & 0 & 0 & 0 & -\lambda_2 \end{array}\right)$$

which is the (ordinary) matrix sum of two tensor products, viz:

$$Q_{e_1} = \begin{pmatrix} 0 & 0 \\ \lambda_2 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & -\lambda_2 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Similarly, the part of the global generator due to synchronizing event e_2 is

$$Q_{e_2} = \left(\begin{array}{ccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \mu_3 & 0 & -\mu_3 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \mu_3 & 0 & 0 & 0 & 0 & -\mu_3 \end{array}\right)$$

which may be obtained from a sum of tensor products as

$$Q_{e_2} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \mu_3 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -\mu_3 \end{pmatrix}.$$

Observe that the global infinitesimal generator is now given by

$$Q = Q_l + Q_{e_1} + Q_{e_2}.$$

Although we considered only a simple example, the above approach has been shown to be applicable in general. Stochastic automata networks that contain synchronizing transitions may always be treated by separating out the local transitions, handling these in the usual fashion by means of a tensor sum and then incorporating the sum of two additional tensor products per synchronizing event. Furthermore, since tensor sums are defined in terms of the (usual) matrix sum of tensor products, the infinitesimal generator of a system containing N stochastic automata with E synchronizing events (and no functional transition rates) may be written as

$$\sum_{j=1}^{2E+N} \otimes_{i=1}^{N} Q_j^{(i)}. \tag{6}$$

This quantity is referred to as the *descriptor* of the stochastic automata network.

The computational burden imposed by synchronizing events is now apparent and is two-fold. Firstly, the number of terms in the *descriptor* is increased, — two for each synchronizing event. We may therefore conclude that the SAN approach is not well suited to models in which there are many synchronizing events. On the other hand, it may still be useful for systems that may be modelled with several stochastic automata that operate mostly independently and only infrequently need to synchronize their operations, such as those found in many models of highly parallel machines.

A second and even greater burden is that the simple form of the solution, equation (4), no longer holds. Although we have been successful in writing the descriptor in a compact form as the sum of tensor products, the solution is not simply the sum of the vectors computed as the tensor product of the solutions of the individual $Q_j^{(i)}$. Other methods for computing solutions must be found. The usefulness of the SAN approach will be determined uniquely by our ability to solve this problem. We now turn our attention to functional transition rates, for these may appear not only in local transitions, but also in synchronizing events.

3.4 The Effect of Functional Transition Rates

We return to the two original automata given in equation (5) and consider what happens when one of the transition rates of the second automaton becomes a functional transition rate. Suppose, for example, that the rate of transition from state 2 to state 3 in the second stochastic automaton is $\hat{\mu}_2$ when the first is in state 1 and $\tilde{\mu}_2$ when the first is in state 2. The global infinitesimal generator is now

$$\left(\begin{array}{ccc|ccc} -(\lambda_1+\mu_1) & \mu_1 & 0 & \lambda_1 & 0 & 0 \\ 0 & -(\lambda_1+\hat{\mu}_2) & \hat{\mu}_2 & 0 & \lambda_1 & 0 \\ \mu_3 & 0 & -(\lambda_1+\mu_3) & 0 & 0 & \lambda_1 \\ \hline \lambda_2 & 0 & 0 & -(\lambda_2+\mu_1) & \mu_1 & 0 \\ 0 & \lambda_2 & 0 & 0 & -(\lambda_2+\tilde{\mu}_2) & \tilde{\mu}_2 \\ 0 & 0 & \lambda_2 & \mu_3 & 0 & -(\lambda_2+\mu_3) \end{array}\right).$$

If, in addition, the rate at which the first stochastic automaton produces transition from state 1 to state 2 is $\bar{\lambda}_1$, $\hat{\lambda}_1$ and $\tilde{\lambda}_1$ depending on whether the second automaton is in state 1, 2 or 3, the two-dimensional infinitesimal generator is given by

$$\left(\begin{array}{ccc|ccc} -(\bar{\lambda}_1+\mu_1) & \mu_1 & 0 & \bar{\lambda}_1 & 0 & 0 \\ 0 & -(\hat{\lambda}_1+\hat{\mu}_2) & \hat{\mu}_2 & 0 & \hat{\lambda}_1 & 0 \\ \mu_3 & 0 & -(\tilde{\lambda}_1+\mu_3) & 0 & 0 & \tilde{\lambda}_1 \\ \hline \lambda_2 & 0 & 0 & -(\lambda_2+\mu_1) & \mu_1 & 0 \\ 0 & \lambda_2 & 0 & 0 & -(\lambda_2+\tilde{\mu}_2) & \tilde{\mu}_2 \\ 0 & 0 & \lambda_2 & \mu_3 & 0 & -(\lambda_2+\mu_3) \end{array}\right).$$

However, it is still possible to profit from the fact that the nonzero structure is unchanged. This is essentially what Plateau has done in her extension of the classical tensor algebraic concepts, [33]. The descriptor is still written as in equation (6), but now the elements of $Q_j^{(i)}$ may be functions. This means that it is necessary to track elements that are functions and to substitute the appropriate numerical value each time the functional rate is needed.

A moment's reflection should convince the reader that the introducion of functional transition rates has no effect on the *structure* of the global transition rate matrix other than when functions evaluate to zero in which case a degenerate form of the original structure is obtained. However, even if the structure is preserved, the actual values of the nonzero elements prevents us from writing the solution in the simple form of equation (4). Nevertheless it is still possible to profit from this unaltered nonzero structure. This is the concept behind the extended (generalized) tensor algebraic approach, [33]. The descriptor is still written as in equation (6), but now the elements of $Q_j^{(i)}$ may be functions. This means that it is necessary to track elements that are functions and to substitute (or recompute) the appropriate numerical value each time the functional rate is needed.

4 Examples

We now introduce two fairly large models that we will use for purposes of illustration. The first is a model of resource sharing that includes functional transitions. The second is a finite queueing network model with both functional transitions and synchronizing events.

4.1 A Model of Resource Sharing

In this model, N distinguishable processes share a certain resource. Each of these processes alternates between a *sleeping* state and a resource *using* state. However, the number of processes that may concurrently use the resource is limited to P where $1 \leq P \leq N$ so that when a process wishing to move from the sleeping state to the resource using state finds P processes already using the resource, that process fails to access the resource and returns to the sleeping state. Notice that when $P = 1$ this model reduces to the usual mutual exclusion problem. When $P = N$ all of the the processes are independent. Let $\lambda^{(i)}$ be the rate at which process i awakes from the sleeping state wishing to access the resource, and let $\mu^{(i)}$ be the rate at which this same process releases the resource when it has possession of it.

In our SAN representation, each process is modelled by a two state automaton $\mathcal{A}^{(i)}$, the two states being *sleeping* and *using*. We shall let $s\mathcal{A}^{(i)}$ denote the current state of automaton $\mathcal{A}^{(i)}$. Also, we introduce the function

$$f = \delta\left(\sum_{i=1}^{N} \delta(s\mathcal{A}^{(i)} = using) \; < \; P\right),$$

where $\delta(b)$ is an integer function that has the value 1 if the boolean b is true, and the value 0 otherwise. Thus the function f has the value 1 when access is permitted to the resource and has the value 0 otherwise. Figure 1 provides a graphical illustration of this model.

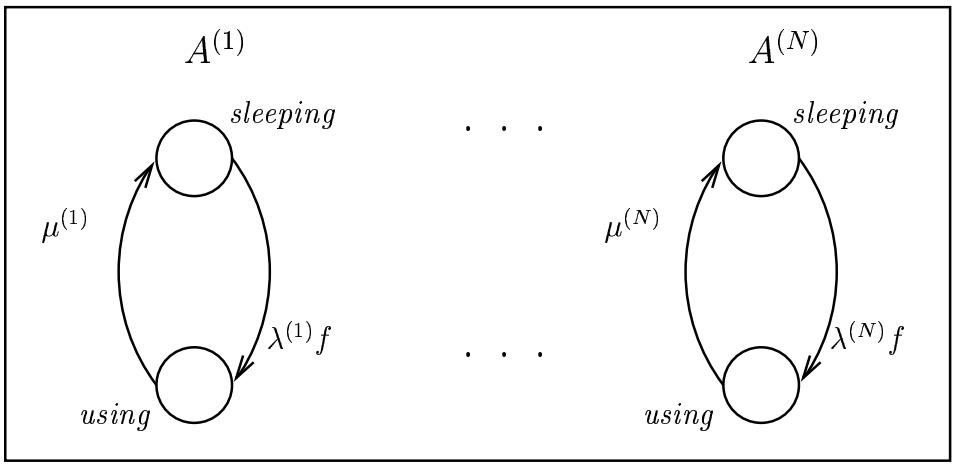

Figure 1: Resource Sharing Model

The local transition matrix for automaton $\mathcal{A}^{(i)}$ is

$$Q_l^{(i)} \; = \; \begin{pmatrix} -\lambda^{(i)} f & \lambda^{(i)} f \\ \mu^{(i)} & -\mu^{(i)} \end{pmatrix},$$

and the overall descriptor for the model is

$$Q = \bigoplus_{g\ i=1}^{N} Q_l^{(i)} = \sum_{i=1}^{N} I_2 \otimes_g \cdots \otimes_g I_2 \otimes_g Q_l^{(i)} \otimes_g I_2 \otimes_g \cdots \otimes_g I_2,$$

where $\otimes_g$ denotes the generalized tensor operator, a precise definition of which is given in Section 5.

The SAN product state space for this model is of size 2^N. Notice that when $P = 1$, the reachable state space is of size $N + 1$, which is considerably smaller than the product state space, while when $P = N$ the reachable state space is the entire product state space. Other values of P give rise to intermediate cases.

4.2 A Queueing Network with Blocking and Priority Service

The second model we shall use is an open queueing network of three finite capacity queues and two customer classes. Class 1 customers arrive from the exterior to queue 1 according to a Poisson process with rate λ_1. Arriving customers are lost if they arrive and find the buffer full. Similarly, class 2 customers arrive from outside the network to queue 2, also according to a Poisson process, but this time at rate λ_2 and they also are lost if the buffer at queue 2 is full. The servers at queues 1 and 2 provide exponential service at rates μ_1 and μ_2 respectively. Customers that have been served at either of these queues try to join queue 3. If queue 3 is full, class 1 customers are blocked (blocking after service) and the server at queue 1 must halt. This server cannot begin to serve another customer until a slot becomes available in the buffer of queue 3 and the blocked customer is transferred. On the other hand, when a (class 2) customer has been served at queue 2 and finds the buffer at queue 3 full, that customer is lost. Queue 3 provides exponential service at rate μ_{3_1} to class 1 customers and at rate μ_{3_2} to class 2 customers. It is the only queue to serve both classes. In this queue, class 1 customers have preemptive priority over class 2 customers. Customers departing after service at queue 3 leave the network. We shall let $C_k - 1$, $k = 1,\ 2,\ 3$ denote the finite buffer capacity at queue k.

Queues 1 and 2 can each be represented by a single automaton ($\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ respectively) with a one-to-one correspondance between the number of customers in the queue and the state of the associated automaton. Queue 3 requires two automata for its representation; the first, $\mathcal{A}^{(3_1)}$, provides the number of class 1 customers and the second, $\mathcal{A}^{(3_2)}$, the number of class 2 customers present in queue 3. Figure 2 illustrates this model.

This SAN has two synchronizing events: the first corresponds to the transfer of a class 1 customer from queue 1 to queue 3 and the second, the transfer of a class 2 customer from queue 2 to queue 3. These are synchronizing events since a change of state in automaton $\mathcal{A}^{(1)}$ or $\mathcal{A}^{(2)}$ occasioned by the departure of a customer, must be synchronized with a corresponding change in automaton $\mathcal{A}^{(3_1)}$ or $\mathcal{A}^{(3_2)}$, representing the arrival of that customer to queue 3. We shall denote these synchronizing events as s_1 and s_2 respectively. In addition to these synchronizing events, this SAN required two functions. They are:

$$f = \delta(s\mathcal{A}^{(3_1)} + s\mathcal{A}^{(3_2)} < C_3 - 1)$$

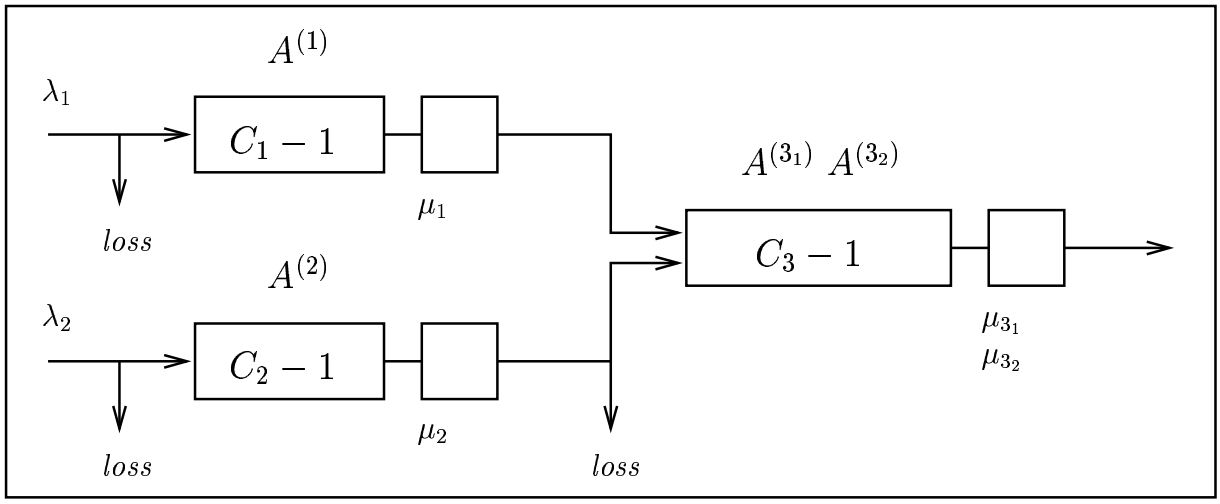

Figure 2: Network of Queues Model

$$g = \delta(s\mathcal{A}^{(3_1)} = 0)$$

The function f has the value 0 when queue 3 is full and the value 1 otherwise, while the function g has the value 0 when a class 1 customer is present in queue 3, thereby preventing a class 2 customer in this queue from receiving service. It has the value 1 otherwise.

Since there are two synchronizing events, each automaton will give rise to *five* separate matrices in our representation. For each automaton k we will have a matrix of local transitions, denoted by $Q_l^{(k)}$; a matrix corresponding to each of the two synchronizing events, $Q_{s_1}^{(k)}$ and $Q_{s_2}^{(k)}$, and a diagonal corrector matrix for each synchronizing event, $\bar{Q}_{s_1}^{(k)}$ and $\bar{Q}_{s_2}^{(k)}$. In these last two matrices, nonzero elements can appear only along the diagonal; they are defined in such a way as to make $\left(\otimes_k Q_{s_j}^{(k)}\right) + \left(\otimes_k \bar{Q}_{s_j}^{(k)}\right)$, $j = 1, 2$, generator matrices (row sums equal to zero). The five matrices for each of the four automata in this SAN are as follows (where we use I_m to denote the identity matrix of order m).

For $\mathcal{A}^{(1)}$:

$$Q_l^{(1)} = \begin{pmatrix} -\lambda_1 & \lambda_1 & 0 & \cdots & 0 \\ 0 & -\lambda_1 & \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -\lambda_1 & \lambda_1 \\ 0 & \cdots & 0 & 0 & 0 \end{pmatrix}, \qquad Q_{s_1}^{(1)} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ \mu_1 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \mu_1 & 0 & 0 \\ 0 & \cdots & 0 & \mu_1 & 0 \end{pmatrix},$$

$$\bar{Q}^{(1)}_{s_1} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & -\mu_1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -\mu_1 & 0 \\ 0 & \cdots & 0 & 0 & -\mu_1 \end{pmatrix}, \qquad Q^{(1)}_{s_2} = I_{C_1} = \bar{Q}^{(1)}_{s_2}.$$

For $\mathcal{A}^{(2)}$:

$$Q^{(2)}_l = \begin{pmatrix} -\lambda_2 & \lambda_2 & 0 & \cdots & 0 \\ 0 & -\lambda_2 & \lambda_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -\lambda_2 & \lambda_2 \\ 0 & \cdots & 0 & 0 & 0 \end{pmatrix}, \qquad Q^{(2)}_{s_2} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ \mu_2 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \mu_2 & 0 & 0 \\ 0 & \cdots & 0 & \mu_2 & 0 \end{pmatrix},$$

$$\bar{Q}^{(2)}_{s_2} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & -\mu_2 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -\mu_2 & 0 \\ 0 & \cdots & 0 & 0 & -\mu_2 \end{pmatrix}, \qquad Q^{(2)}_{s_1} = I_{C_2} = \bar{Q}^{(2)}_{s_1}.$$

For $\mathcal{A}^{(3_1)}$:

$$Q^{(3_1)}_l = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ \mu_{3_1} & -\mu_{3_1} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \mu_{3_1} & -\mu_{3_1} & 0 \\ 0 & \cdots & 0 & \mu_{3_1} & -\mu_{3_1} \end{pmatrix}, \qquad Q^{(3_1)}_{s_1} \begin{pmatrix} 0 & f & 0 & \cdots & 0 \\ 0 & 0 & f & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & f \\ 0 & \cdots & 0 & 0 & 0 \end{pmatrix},$$

$$\bar{Q}^{(3_1)}_{s_1} = \begin{pmatrix} f & 0 & 0 & \cdots & 0 \\ 0 & f & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & f & 0 \\ 0 & \cdots & 0 & 0 & 0 \end{pmatrix}, \qquad Q^{(3_1)}_{s_2} = I_{C_3} = \bar{Q}^{(3_1)}_{s_2}.$$

For $\mathcal{A}^{(3_2)}$:

$$Q^{(3_2)}_l = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ \mu_{3_2} g & -\mu_{3_2} g & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \mu_{3_2} g & -\mu_{3_2} g & 0 \\ 0 & \cdots & 0 & \mu_{3_2} g & -\mu_{3_2} g \end{pmatrix}, \quad Q^{(3_2)}_{s_2} \begin{pmatrix} 1-f & f & 0 & \cdots & 0 \\ 0 & 1-f & f & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1-f & f \\ 0 & \cdots & 0 & 0 & 1 \end{pmatrix},$$

$$\bar{Q}^{(3_2)}_{s_2} = I_{C_3} = Q^{(3_2)}_{s_1} = \bar{Q}^{(3_2)}_{s_1}.$$

The overall descriptor for this model is given by

$$Q = \bigoplus_g Q_l^{(i)} + \bigotimes_g Q_{s_1}^{(i)} + \bigotimes_g \bar{Q}_{s_1}^{(i)} + \bigotimes_g Q_{s_2}^{(i)} + \bigotimes_g \bar{Q}_{s_2}^{(i)},$$

where the generalized tensor sum and the four generalized tensor products are taken over the index set $\{1, 2, 3_1 \text{ and } 3_2\}$. The reachable state space of the SAN is of size $C_1 \times C_2 \times C_3(C_3+1)/2$ whereas the complete SAN product state space has size $C_1 \times C_2 \times {C_3}^2$. Finally, we would like to draw our readers attention to the sparsity of the matrices presented above.

5 Product Forms

Stochastic automata networks constitute a general modelling technique and as such they can sometimes inherit results from those already obtained by other modelling approaches. Jackson networks, for example, may be represented by a SAN; the reversibility results of Kelly, [26], and the competition conditions of Boucherie, [6], can be applied to SANs leading to product forms. In this section we shall present sufficient conditions for a SAN to have a product form solution. These conditions extend those given by Boucherie and in addition may be shown to be applicable to truncated state spaces. They apply only to SANs with no synchronizing events, which means that the transitions of the SAN can only be transitions of one automaton at a time. Thus Jackson networks lie outside their scope of applicability. The way we proceed is to work on global balance equations and search for sufficient conditions on the functional transition rates to obtain a product form solution.

Let us state the problem more formally. Consider a SAN with N automata and local transition matrices $Q_l^{(k)}$, $k = 1, 2, \ldots, N$. The states of $\mathcal{A}^{(k)}$ are denoted $i_k \in S^{(k)}$, and a state of the SAN is denoted $i = (i_1, ..., i_N)$. A state of the SAN without automaton $\mathcal{A}^{(k)}$ is denoted $\bar{i_k} = (i_1, ..., i_{k-1}, i_{k+1}, ..., i_N)$. A state i in which the k^{th} component is replaced by i'_k is denoted by $\bar{i_k}|i'_k$. The element of $Q_l^{(k)}$ are assumed to be of product form type; i.e., for $i_k \neq i'_k$,

$$Q_l^{(k)}(i_k, i'_k) = q^{(k)}(i_k, i'_k) f^{(k)}(i, i'),$$

where $q^{(k)}(i_k, i'_k)$ is a constant transition rate and $f^{(k)}(i, i')$ is any positive function. This formulation does not restrict the class of SAN with functional transition rates. Notice that when the transition of the SAN is occasioned by a transition of its k^{th} automaton, the function $f^{(k)}(i, i')$ actually depends only on i and i'_k.

Assume now that the $q^{(k)}(i_k, i'_k)$ satisfy balance equations in the sense that there exist a set of positive numbers $\pi^{(k)}(i_k)$ which sum to 1 and, for all $i_k \in S^{(k)}$, satisfy

$$\sum_{i'_k \in S^{(k)}} \left(\pi^{(k)}(i_k) q^{(k)}(i_k, i'_k) - \pi^{(k)}(i'_k) q^{(k)}(i'_k, i_k) \right) = 0. \tag{7}$$

The SAN generator is $Q = \bigoplus_{k=1}^{N} Q_l^{(k)}$. Its transition rates, for $i \neq i'$, are given by

$$q(i, i') = \begin{cases} q^{(k)}(i_k, i'_k) f^{(k)}(i, \bar{i_k}|i'_k) & if\ i' = \bar{i_k}|i'_k \\ 0 & otherwise \end{cases}$$

The reachable state space of the SAN is denoted by $\mathcal{R}$ and, because of the effect of functional transitional rates, can be strictly smaller that $S = \prod_1^N S^{(k)}$. The global balance equations for the SAN are, for $i \in \mathcal{R}$,

$$\sum_{i' \in S} (\pi(i)q(i,i') - \pi(i')q(i',i)) = 0. \tag{8}$$

Substituting for $q(i,i')$ and $q(i',i)$ in (8) yields

$$\sum_{k=1}^{N} \sum_{i'_k \in S^{(k)}} \left(\pi(i) q^{(k)}(i_k, i'_k) f^{(k)}(i, \bar{i_k} | i'_k) - \pi(\bar{i_k} | i'_k) q^{(k)}(i'_k, i_k) f^{(k)}(\bar{i_k} | i'_k, i) \right) = 0. \tag{9}$$

This SAN has a product form solution if, for some normalizing constant C, $\pi(i) = C \prod_1^N \pi^{(k)}(i_k)$ is a solution of these balance equations. Substituting this into (9) gives

$$\sum_{k=1}^{N} \prod_{j=1, j \neq k}^{N} \pi^{(j)}(i_j) \sum_{i'_k \in S^{(k)}}^{N} \left(\pi^{(k)}(i_k) q^{(k)}(i_k, i'_k) f^{(k)}(i, \bar{i_k} | i'_k) - \pi^{(k)}(i'_k) q^{(k)}(i'_k, i_k) f^{(k)}(\bar{i_k} | i'_k, i) \right) = 0.$$

Now it only remains to find sufficient conditions on the functions $f^{(k)}$ for which

$$\sum_{i'_k \in S^{(k)}}^{N} \left(\pi^{(k)}(i_k) q^{(k)}(i_k, i'_k) f^{(k)}(i, \bar{i_k} | i'_k) - \pi^{(k)}(i'_k) q^{(k)}(i'_k, i_k) f^{(k)}(\bar{i_k} | i'_k, i) \right) \tag{10}$$

is equal to zero, knowing the local balance equations, (7).

First case: The functions $f^{(k)}$ express a truncation of the state space of the SAN (similar to that described by Kelly). That is to say, the functions are equal to the indicator function of the reachable state space $\mathcal{R}$: $f^{(k)}(i,i') = \delta(i' \in \mathcal{R})$. Thus the expression (10) is trivially equal to zero: either $\bar{i_k} | i'_k \in \mathcal{R}$ (the functions are equal to 1) and we have the local balance equations, or $\bar{i_k} | i'_k \notin \mathcal{R}$ and the functions themselves are zero. The normalizing constant C is the inverse of $\sum_{i \in \mathcal{R}} \prod_{k=1}^N \pi(i)$ and might be difficult to compute if $\mathcal{R}$ is large.

Second case: The functions $f^{(k)}$ depend only on $\bar{i_k}$ and not on the current state of automaton k. This means that the decomposition $Q_l^{(k)}(i_k, i'_k) = q^{(k)}(i_k, i'_k) f^{(k)}(i,i')$ is a real product form. The variable $q^{(k)}(i_k, i'_k)$ is the local transition rate of automaton k and $f^{(k)}(i,i') = f^{(k)}(\bar{i_k})$ expresses the interaction of the rest of the SAN when it is in state $\bar{i_k}$. In essence, the functions $f^{(k)}(\bar{i_k})$ either force the system to halt, if they evaluate to zero, or else permit the automaton $\mathcal{A}^{(k)}$ to execute independently, albeit with modified rates: the function uniformly slows down or speeds up the automata for a given $\bar{i_k}$. When $\bar{i_k}$ changes, the slowing/speeding factor changes. The balance equations are given by

$$f^{(k)}(\bar{i_k}) \prod_{j=1, j \neq k}^{N} \pi^{(j)}(i_j) \sum_{i'_k \in S^{(k)}} \left(\pi^{(k)}(i_k) q^{(k)}(i_k, i'_k) - \pi^{(k)}(i'_k) q^{(k)}(i'_k, i_k) \right) = 0$$

which must hold because the local balance equations themselves, (7), hold. This second case is a generalization of the Boucherie competition conditions. The constant C is equal to 1 when the reachability space is the product space; otherwise it must be chosen so that the individual probabilities sum to one.

Third case: Notice that the two previous cases are not overlapping, and this for two reasons:

- In case 1, $f^{(k)}(i,i') = \delta(i' \in \mathcal{R}) = \delta(\bar{i_k}|i'_k \in \mathcal{R})$. In general, the function $\delta(\bar{i_k}|i'_k \in \mathcal{R})$ depends not only on $\bar{i_k}$, but on i'_k as well.
- In case 2, we have a "uniform" modification of $\mathcal{A}^{(k)}$ rates while in case 1, they are either 0 or unchanged, for a given $\bar{i_k}$.

This presents the possibility of combining cases 1 and 2 to yield a third case:

$$f^{(k)}(i,i') = \delta(i' \in \mathcal{R}) f^{(k)}(\bar{i_k}).$$

Using the notation described above, we may summarize these results in the following theorem:

Theorem 5.1 *Given a SAN having no synchronizing events and in which the elements of* $Q_l^{(k)}$ *are of product form type; i.e., for* $i_k \neq i'_k$,

$$Q_l^{(k)}(i_k, i'_k) = q^{(k)}(i_k, i'_k) f^{(k)}(i,i'),$$

each of the following sufficient conditions leads to a product form solution, $\pi(i) = C \prod_1^N \pi^{(k)}(i_k)$*:*

- *Case 1:* $f^{(k)}(i,i') = \delta(i' \in \mathcal{R})$
- *Case 2:* $f^{(k)}(i,i') = f^{(k)}(\bar{i_k})$
- *Case 3:* $f^{(k)}(i,i') = \delta(i' \in \mathcal{R}) f^{(k)}(\bar{i_k})$

The $\pi(i)$ *satisfy the global balance equations for the SAN,* C *is a normalizing constant and the* $\pi^{(k)}(i_k)$ *are solutions of the local balance equations, (7).*

Examples:

1. The resource sharing example of Section 4.1 falls into case 1 with

$$f^{(k)}(i,i') = \delta\left(\sum_{j=1, j\neq k}^{N} \delta(i_j = using) \; < \; P\right) = \delta\left(\sum_{j=1, j\neq k}^{N} \delta(i'_j = using) \; < \; P\right).$$

2. Consider a number, N, of identical processes each represented by a three state automaton. State 1 represents the state in which the process computes independently; state 2 represents an interacting state (*int*) in which the automaton computes and sends messages to the other automata and state 3 is a state in which the process has exclusive access in order to write (w) to a critical resource. Each process may move from any of the three states to any other according to well defined rates of transition. To provide for mutually exclusive access to the critical resource, the rates of transition to state 3 must be multiplied by a function g defined as

$$g(i,i') = \delta\left(\sum_{j=1}^{N} \delta(i_j = w) = 0\right).$$

To provide for the effect of communication overhead, all transition rates within the automaton $k, k = 1, 2, \ldots, N$ must be multiplied by a function $f^{(k)}$ defined as

$$f^{(k)}(i,i') = \frac{C}{\sum_{j=1,k\neq j}^{N} \delta(i_j = int)}.$$

Such a SAN therefore incorporates a superposition of cases 1 and 2.

3. The examples provided in the paper of Boucherie, [6]; viz: the dining philosophers problem, locking in a database system, and so on, all fall into case 2. In these examples, the functions $f^{(k)}(\bar{i_k})$ express a reachable state space *and* yield a uniform multiplicative factor. Other examples may be found in [14, 19, 22, 28].

4. Our final example explicitly displays the dependence of the function on i' and falls into case 1. Consider a system consisting of P units of resource and N identical processes, each represented by a three-state Markov chain. While in state 0, a process may be considered to be sleeping; in state 1, it uses a single unit of resource; while in state 2, the process uses 2 units of the resource. The transitions among its three states are such that while in the sleeping state (0), it can move directly to either state 1 or 2, (i.e., it may request, and receive, two units of resource, or just a single unit of resource). From state 2, the process can only move to state 1, and from state 1 it can only move to state 0, (i.e., units of the resource are released independently). To cater for the case in which sufficient resources are not available, the rates of transition towards states 1 and 2 must each be multiplied by the function

$$f^{(k)}(i,i'_k) = \delta\left(i'_k + \sum_{j=1,j\neq k}^{N} i_j < P\right).$$

6 Vector-Descriptor Multiplications

In many cases, and perhaps most, product form solutions are not available and the analyst must turn to other solutions procedures. When the global infinitesimal generator of a SAN

is available only in the form of a SAN descriptor, the most general and suitable methods for obtaining probability distributions are numerical iterative methods, [44]. Thus, the underlying operation, whether we wish to compute the stationary distribution, or the transient solution at any time t, is the product of a vector with a matrix. Since

$$xQ = x \sum_{j=1}^{2E+N} \otimes_{i=1}^{N} Q_j^{(i)} = \sum_{j=1}^{2E+N} x \otimes_{i=1}^{N} Q_j^{(i)},$$

the basic operation is

$$x \otimes_{i=1}^{N} Q^{(i)}$$

where, for notational convenience, we have removed the subscripts on the $Q_j^{(i)}$. It is essential that this operation be implemented as efficiently as possible. The following theorem is proven in [33]. It implicitly assumes that the SAN matrices are dense.

Theorem 6.1 *The product*

$$x \otimes_{i=1}^{N} Q^{(i)},$$

where $Q^{(i)}$, of order n_i, contains only constant terms and x is a real vector of length $\prod_{i=1}^{N} n_i$, may be computed in ρ_N multiplications, where

$$\rho_N = n_N \times (\rho_{N-1} + \prod_{i=1}^{N} n_i) \quad = \quad \prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} n_i.$$

Let us now examine the effect of introducing functional rates. The savings made in the computation of $x \otimes_{i=1}^{N} Q^{(i)}$ are due to the fact that once a product is formed, it may be used in several places without having to re-do the multiplication. With functional rates, the elements in the matrices may change according to their context so that this same savings is sometimes possible [44]. This leads to an extension of some of the properties of tensor products and to the concept of *Generalized Tensor Products (GTPs)* as opposed to *Ordinary Tensor Products (OTP)*.

7 Generalized Tensor Products

We assume throughout that all matrices are square. As indicated in the previous section, $B[\mathcal{A}]$ indicates that the matrix B may contain transitions that are a function of the state of the automaton $\mathcal{A}$. More generally, $A^{(m)}[\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(m-1)}]$ indicates that the matrix $A^{(m)}$ may contain elements that are a function of one or more of the states of the automata $\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(m-1)}$. We shall use the notation $\otimes_g$ to denote a generalized tensor product. Thus $A \otimes_g B[\mathcal{A}]$ denotes the generalized tensor product of the matrix A with the functional

matrix $B[\mathcal{A}]$ and we have

$$A \otimes_g B[\mathcal{A}] = \begin{pmatrix} a_{11}B(a_1) & a_{12}B(a_1) & \cdots & a_{1n_a}B(a_1) \\ a_{21}B(a_2) & a_{22}B(a_2) & \cdots & a_{2n_a}B(a_2) \\ \vdots & \vdots & \ddots & \vdots \\ a_{n_a1}B(a_{n_a}) & a_{n_a2}B(a_{n_a}) & \cdots & a_{n_an_a}B(a_{n_a}) \end{pmatrix}, \tag{11}$$

where $B(a_k)$ represents the matrix B when its functional entries are evaluated with the argument a_k, $k = 1, 2, \ldots, n_a$, the a_i being the states of automaton $\mathcal{A}$. Also,

$$A[\mathcal{B}] \otimes_g B = \begin{pmatrix} a_{11}[\mathcal{B}]I_{n_b} \times B & a_{12}[\mathcal{B}]I_{n_b} \times B & \cdots & a_{1n_a}[\mathcal{B}]I_{n_b} \times B \\ a_{21}[\mathcal{B}]I_{n_b} \times B & a_{22}[\mathcal{B}]I_{n_b} \times B & \cdots & a_{2n_a}[\mathcal{B}]I_{n_b} \times B \\ \vdots & \vdots & \ddots & \vdots \\ a_{n_a1}[\mathcal{B}]I_{n_b} \times B & a_{n_a2}[\mathcal{B}]I_{n_b} \times B & \cdots & a_{n_an_a}[\mathcal{B}]I_{n_b} \times B \end{pmatrix},$$

where

$$a_{ij}[\mathcal{B}]I_{n_b} = diag\{a_{ij}(b_1), a_{ij}(b_2), \ldots, a_{ij}(b_{n_b})\}$$

and $a_{ij}(b_k)$ is the value of the ij^{th} element of the matrix A when its functional entries are evaluated with the argument b_k, $k = 1, 2, \ldots, n_b$. Finally, when both automata are functional we have

$$A[\mathcal{B}] \otimes_g B[\mathcal{A}] = \begin{pmatrix} a_{11}[\mathcal{B}]I_{n_b} \times B(a_1) & a_{12}[\mathcal{B}]I_{n_b} \times B(a_1) & \cdots & a_{1n_a}[\mathcal{B}]I_{n_b} \times B(a_1) \\ a_{21}[\mathcal{B}]I_{n_b} \times B(a_2) & a_{22}[\mathcal{B}]I_{n_b} \times B(a_2) & \cdots & a_{2n_a}[\mathcal{B}]I_{n_b} \times B(a_2) \\ \vdots & \vdots & \ddots & \vdots \\ a_{n_a1}[\mathcal{B}]I_{n_b} \times B(a_{n_a}) & a_{n_a2}[\mathcal{B}]I_{n_b} \times B(a_{n_a}) & \cdots & a_{n_an_a}[\mathcal{B}]I_{n_b} \times B(a_{n_a}) \end{pmatrix}.$$

For $A[\mathcal{B}] \otimes_g B[\mathcal{A}]$, the generic entry (l, k) within block (i, j) is $a_{ij}(b_k) \times b_{kl}(a_i)$.

We now present a number of lemmas concerning generalized tensor products. Their proofs may be found in [15]. These lemmas are useful for deriving many important properties of generalized tensor products including Theorems 7.1 and 7.2 which follow and which specify the complexity of forming the product of a vector with a SAN descriptor in the presence of functional transitions.

Lemma 7.1 (GTP: Associativity)

$$(A[\mathcal{B}, \mathcal{C}] \otimes_g B[\mathcal{A}, \mathcal{C}]) \otimes_g C[\mathcal{A}, \mathcal{B}] = A[\mathcal{B}, \mathcal{C}] \otimes_g (B[\mathcal{A}, \mathcal{C}] \otimes_g C[\mathcal{A}, \mathcal{B}])$$

Lemma 7.2 (GTP: Distributivity over Addition)

$$(A_1[\mathcal{B}] + A_2[\mathcal{B}]) \otimes_g (B_1[\mathcal{A}] + B_2[\mathcal{A}]) =$$

$$(A_1[\mathcal{B}] \otimes_g B_1[\mathcal{A}] \; + \; A_1[\mathcal{B}] \otimes_g B_2[\mathcal{A}] \; + \; A_2[\mathcal{B}] \otimes_g B_1[\mathcal{A}] \; + \; A_1[\mathcal{B}] \otimes_g B_2[\mathcal{A}])$$

As for ordinary tensor products, compatibility with multiplication usually does not hold for generalized tensor products either. However, there exists three degenerate compatibility forms when some of the factors are identity matrices and not all of the factors have functional entries. They are

Lemma 7.3 (GTP: Compatibility over Multiplication: I — Two Factors)

$$(A[\mathcal{C}] \times B[\mathcal{C}]) \otimes_g I_{n_c} = (A[\mathcal{C}] \otimes_g I_{n_c}) \times (B[\mathcal{C}] \otimes_g I_{n_c})$$

Similarly,

$$I_{n_c} \otimes_g (A[\mathcal{C}] \times B[\mathcal{C}]) = (I_{n_c} \otimes_g A[\mathcal{C}]) \times (I_{n_c} \otimes_g B[\mathcal{C}]).$$

Lemma 7.4 (GTP: Compatibility over Multiplication: II — Two Factors)

$$A \otimes_g B[\mathcal{A}] = [A \times I_{n_a}] \otimes_g [I_{n_b} \times B[\mathcal{A}]] = (I_{n_a} \otimes_g B[\mathcal{A}]) \times (A \otimes I_{n_b}).$$

Lemma 7.5 (GTP: Compatibility over Multiplication: III — Two Factors)

$$A[\mathcal{B}] \otimes_g B = [A[\mathcal{B}] \times I_{n_a}] \otimes_g [I_{n_b} \times B] = (A[\mathcal{B}] \otimes_g I_{n_b}) \times (I_{n_a} \otimes B).$$

This lemma still holds if I_{n_a} is replaced by any constant matrix.

Lemma 7.6 (GTP: Compatibility over Multiplication — Many Factors)

$$\begin{aligned} A^{(1)} \otimes_g A^{(2)}[\mathcal{A}^{(1)}] \otimes_g A^{(3)}[\mathcal{A}^{(1)}, \mathcal{A}^{(2)}] \otimes_g \cdots \otimes_g A^{(m)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(m-1)}] = \\ & I_{1:m-1} \otimes_g A^{(m)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(m-1)}] \\ \times \quad & I_{1:m-2} \otimes_g A^{(m-1)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(m-2)}] \otimes_g I_{m:m} \\ \times \quad & \cdots \\ \times \quad & I_{1:1} \otimes_g A^{(2)}[\mathcal{A}^{(1)}] \otimes_g I_{3:m} \\ \times \quad & A^{(1)} \otimes_g I_{2:m} \end{aligned} \tag{12}$$

Due to the existence of Lemma 7.5 the same property holds for

$$\begin{aligned} A^{(m)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(m-1)}] \otimes_g A^{(m-1)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(m-2)}] \otimes_g \cdots \otimes_g A^{(2)}[\mathcal{A}^{(1)}] \otimes_g A^{(1)} = \\ & A^{(m)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(m-1)}] \otimes_g I_{m-1:1} \\ \times \quad & I_{m:m} \otimes_g A^{(m-1)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(m-2)}] \otimes_g I_{m-2:1} \\ \times \quad & \cdots \\ \times \quad & I_{m:3} \otimes_g A^{(2)}[\mathcal{A}^{(1)}] \otimes_g I_{1:1} \\ \times \quad & I_{m:2} \otimes_g A^{(1)} \end{aligned} \tag{13}$$

In Lemma 7.6, only one automaton can depend on all the $(m-1)$ other automata, only one can depend on at most $(m-2)$ other automata and so on. One automaton must be independent of all the others. This provides a means by which the individual factors on the left-hand side of equation (12) may be ranked; i.e., according to the number of automata on which they *may* depend. An automaton may actually depend on a subset of the automata in its parameter list.

Lemma 7.7 (GTP: Pseudo-Commutativity) *Let σ be a permutation of the integers $[1, 2, \ldots, N]$, then there exists a permutation matrix, P_σ of order $\prod_{i=1}^{N} n_i$, such that*

$$\bigotimes_{g\ k=1}^{N} A^{(k)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(N)}] = P_\sigma \bigotimes_{g\ k=1}^{N} A^{(\sigma(k))}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(N)}] P_\sigma^T.$$

These lemmas allow the following theorem to be proven. (The proof itself may be found in [15].)

Theorem 7.1 (GTP: Algorithm) *The multiplication*

$$x \times \left(A^{(1)} \otimes_g A^{(2)}[\mathcal{A}^{(1)}] \otimes_g A^{(3)}[\mathcal{A}^{(1)}, \mathcal{A}^{(2)}] \otimes_g \cdots \otimes_g A^{(N)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(N-1)}] \right)$$

where x is a real vector of length $\prod_{i=1}^{N} n_i$ may be computed in $O(\rho_N)$ multiplications, where

$$\rho_N = n_N \times (\rho_{N-1} + \prod_{i=1}^{N} n_i) \quad = \quad \prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} n_i.$$

This complexity result was computed under the assumption that the matrices are full. However, the number of multiplications may be reduced by taking advantage of the fact that the block matrices are generally sparse. It is immediately apparent that when the matrices are not full, but possess a special structure such as tridiagonal, or contain only one nonzero row or column, etc., or are sparse, then this may be taken into account and this number reduced in consequence. The cost of the function evaluations is included in the definition of the big Oh formula. We would like to point out that although Theorem 7.1 allows us to reorganize the terms in the generalized tensor product in any way we wish, the advantage of leaving them in the form given above is precisely that the computation of the state indices can be moved outside the innermost summation of the algorithm.

The following algorithm is based directly on Theorem 7.1 and implements an efficient product of a vector x with a generalized tensor product in which the automata satisfy the functional dependencies described in Lemma 7.6. In this algorithm, the notation $A^{(i)}[a_{k_1}^{(1)}, \ldots, a_{k_{i-1}}^{(i-1)}]$ implies that the matrix is evaluated under the assumption that automaton $\mathcal{A}^{(j)}$ is in state $a_{k_j}^{(j)}$, for $j = 1, 2, \ldots, i-1$.

Algorithm: Vector Multiplication with a Generalized Tensor Product

$$x\left(A^{(1)} \otimes_g A^{(2)}[\mathcal{A}^{(1)}] \otimes_g A^{(3)}[\mathcal{A}^{(1)}, \mathcal{A}^{(2)}] \otimes_g \cdots \otimes_g A^{(N)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(N-1)}]\right)$$

1. Initialize: $nleft = n_1 n_2 \cdots n_{N-1}$; $nright = 1$.
2. For $i = N, \ldots, 2, 1$ do
   - $base = 0$; $jump = n_i \times nright$
   - For $k = 1, 2, \ldots, nleft$ do
     - For $j = 1, 2, \ldots, i-1$ do
       - $k_j = \left(\left[(k-1)/\prod_{l=j+1}^{i-1} n_l\right] mod \left(\prod_{l=j}^{i-1} n_l\right)\right) + 1$
     - For $j = 1, 2, \ldots, nright$ do
       - $index = base + j$
       - For $l = 1, 2, \ldots, n_i$ do
         - $z_l = x_{index}$; $index = index + nright$
       - Multiply: $z' = z \times A^{(i)}[a_{k_1}^{(1)}, \ldots, a_{k_{i-1}}^{(i-1)}]$
       - $index = base + j$
       - For $l = 1, 2, \ldots, n_i$ do
         - $x'_{index} = z'_l$; $index = index + nright$
     - $base = base + jump$
   - $nleft = nleft/n_{i-1}$
   - $nright = nright \times n_i$
3. $x = x'$

We now introduce one final lemma that allows us to prove a theorem (Theorem 7.2) concerning the reduction in the cost of a vector-descriptor multiplication in the case when the functional dependencies among the automata do not satisfy the constraints given above.

Lemma 7.8 (GTP: Decomposability into OTP) *Let $\ell_k(A)$ denote the matrix obtained by setting all elements of A to zero except those that lie on the k^{th} row which are left unchanged. Then*

$$A \otimes_g B[\mathcal{A}] = \sum_{k=1}^{n_a} \ell_k(A) \otimes B[a_k].$$

Thus we may write a generalized tensor product as a sum of ordinary tensor products.

A term $\bigotimes_g A^{(i)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(N)}]$ involved in the descriptor of a SAN is said to contain a *functional dependency cycle* if it contains a subset of automata $\mathcal{A}^{(p)}, \mathcal{A}^{(p+1)}, \ldots, \mathcal{A}^{(p+c)}$,

$c \geq 1$, with the property that the matrix representation of $\mathcal{A}^{(p+i)}$ contains transitions that are a function of $\mathcal{A}^{(p+(i+1) mod_{c+1})}$, for $0 \leq i \leq c$. For example, a SAN with two automata $\mathcal{A}$ and $\mathcal{B}$ contains a term with a functional dependency cycle if and only if the matrix representations are such that A is a function of $\mathcal{B}$, $(A[\mathcal{B}])$, B is a function of $\mathcal{A}$, $(B[\mathcal{A}])$, and $A[\mathcal{B}] \otimes B[\mathcal{A}]$ occurs in the descriptor. Let $\mathcal{G}$ denote a graph whose nodes are the individual automata of a SAN and whose arcs represent dependencies among the automata within a term of the descriptor. Let $\mathcal{T}$ be a *cutset* of the cycles of $\mathcal{G}$, [5]. Then $\mathcal{T}$ is a set of nodes of $\mathcal{G}$ with the property that $\mathcal{G} - \mathcal{T}$ does not contain a cycle where $\mathcal{G} - \mathcal{T}$ is the graph of $\mathcal{G}$ with all arcs that lead into the nodes of $\mathcal{T}$ removed.

Theorem 7.2 (GTP with Cycles: Complexity of Vector-Descriptor Product) *Given a SAN descriptor containing a term with a functional dependency graph $\mathcal{G}$ and cutset $\mathcal{T}$ of size t, the cost of performing the vector-descriptor product*

$$x \times \bigotimes_{g}^{N} \; A^{(i)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(N)}]$$

is

$$\left(\prod_{i=1, i \in \mathcal{T}}^{N} n_i \right) \left(\prod_{i=1}^{N} n_i \right) \left(\sum_{i=1, i \notin \mathcal{T}}^{N} n_i \right).$$

8 Applicability of the Multiplication Theorems

We now return to the context of SANs proper. We have seen that the descriptor of a SAN is a sum of tensor products and we now wish to examine each of the terms of these tensor products in detail to see whether they fall into the categories of Theorem 7.1 or 7.2. In the case of SANs in continuous-time, and with no synchronizing transitions, the descriptor is given by

$$Q = \bigoplus_{g}^{N} \; Q^{(k)} = \sum_{k=1}^{N} I_{n_1} \otimes_g \cdots \otimes_g I_{n_{k-1}} \otimes_g Q^{(k)} \otimes_g I_{n_{k+1}} \otimes_g \cdots \otimes_g I_{n_N},$$

and we can apply Theorem 7.1 directly to each term of the summation. Notice that all but one of the terms in each tensor product is an identity matrix, and as we pointed out in the proof of Theorem 7.1, advantage can be taken of this to reduce the number of multiplications involved.

Consider now what happens when we add synchronizing events. The part of the SAN descriptor that corresponds to *local* transitions has the same minimal cost as above which means that we need only consider that part which is specifically involved with the synchronizing events. Recall from Section 3.3, that each synchronizing event results in an additional two terms in the SAN descriptor. The first of these may be thought of as representing the actual transitions and their rates; the second corresponds to an updating of the diagonal

elements in the infinitesimal generator to reflect these transitions. Since the second is more efficiently handled separately and independently, we need only be concerned with the first. It can be written as

$$\bigotimes_{i=1}^{N} Q_j^{(i)}.$$

We must now analyze the matrices that represent the different automata in this tensor product. There are three possibilities depending upon whether they are unaffected by the transition, are the designated automaton with which the transition rate of the synchronizing event is associated, or are affected in some other manner by the event. These matrices have the following structure.

- Matrices $Q_j^{(i)}$ corresponding to automata that do not participate in the synchronizing event are identity matrices of appropriate dimension.
- With each synchronizing event is associated a particular automaton. In a certain sense, this automaton may be considered to be the *owner* of the synchronizing event. We shall let E denote the matrix of transition rates associated with this automaton. For example, if a synchronizing event e arises as a result of the automaton changing from state i to state j, then the matrix E consists entirely of zeros with a single nonzero element in position ij. This nonzero element gives the rate at which the transition occurs. If several of the elements are nonzero, this indicates that the automaton may cause this same synchronizing event by transitions from and to several states.
- The remaining matrices correspond to automata that are otherwise affected by the synchronizing event. We shall refer to these as Λ matrices. Each of these Λ matrices consists of rows whose nonzero elements are positive and sum to 1 (essentially, they correspond to routing probabilities, not transition rates), or rows whose elements are all equal to zero. The first case corresponds to the automaton being synchronized into different states according to certain fixed probabilities and the state currently occupied by the automaton. In the second case, a row of zeros indicates that this automaton disables the synchronizing transition while it is in the state corresponding to the row of zeros. In many cases, these matrices will have a very distinctive structure, such as a column of ones, the case when the automaton is forced into a particular state, independent of its current state.

When the only functional rates are those of the synchronizing event, only the elements of the matrix E are functional; the other matrices consist of identity matrices or constant Λ matrices. Thus, once again, this case falls into the scope of Theorem 7.1.

This leaves the case in which the Λ matrices contain transition probabilities that are functional. If there is no cycle in the functional dependency graph then we can apply Theorem 7.1; otherwise we must resort to Theorem 7.2. Notice that if the functional dependency graph is fully connected, Theorem 7.2 offers no savings compared with ordinary multiplication. This shows that Theorem 7.2 is only needed when the routing probabilities associated

with a synchronizing event are functional and result in cycles within the functional dependency graph, (which we suspect to be rather rare). For SANs in discrete-time, [32], it seems that we may not be so fortunate since cycles in the functional dependency graph of the tensor product tend to occur rather more often.

Following up on these results, extensive experiments conducted on a set of small examples, and reported in [16], provided a rule of thumb for ordering automata in a network to achieve better performance. More precisely, it is not the automata in the SAN that must be ordered, but rather, within each term of the descriptor, a best ordering should be computed independently.

9 The Memory versus CPU-time Trade-off

An important advantage that the SAN approach has over others that generate and manipulate the entire state space of the underlying Markov chain is that of minimal memory requirements. The infamous *state-space explosion* problem associated with these other approaches is avoided. The price to be paid of course, is that of increased CPU time. The obvious question that should be asked is whether some sort of compromise can be reached. Such a compromise would take the form of reducing a SAN with a certain (possibly large) number of "natural" automata each with only a small number of states, to an "equivalent" SAN with less automata in which many (possibly all) have a larger number of states. A natural way to produce such an equivalent SAN is to collect subsets of the original automata into a small number of groups. The limit of this process is a single automaton containing all the states of the Markov chain. However we do not wish to go to this extreme. Observe that just four automata each of order 100 brings us to the limit of what is currently possible to solve using regular sparse matrix techniques yet memory requirements for four automata of size 100 remain modest. Furthermore, in the process of grouping automata, a number of simplifications may result. For example, automata may be grouped in such a way that some (or all) of the sychronizing events disappear, or grouped so that functional transition rates become constant, or both. Furthermore, it is frequently the case that the reachable state space of the grouped automata is smaller than the product state space of the automata that constitute the group. To pursue this line of thought, we shall need to define our notion of equivalence among SANs and the grouping process.

Consider a SAN containing N stochastic automata $\mathcal{A}_1, \ldots, \mathcal{A}_N$ of size n_i respectively, E synchronizing events $s_1, \ldots, s_E$, and functional transition rates. Its descriptor may be written as

$$Q = \sum_{j=1}^{N+2E} \bigotimes_{g,i=1}^{N} Q_j^{(i)}$$

Let $1, \ldots, N$ be partitioned in B groups called $b_1, \ldots, b_B$, and, without loss of generality, assume that $b_1 = [c_1 = 1, \ldots, c_2]$, $b_2 = [c_2 + 1, \ldots, c_3]$, etc, for some increasing sequence of c_i, $c_{B+1} = N$. The descriptor can be rewritten, using the associativity of the generalized

tensor product, as

$$Q = \sum_{j=1}^{2E+N} \bigotimes_{g,k=1}^{B} \left(\bigotimes_{g,j=c_k+1}^{c_{k+1}} Q_j^{(i)} \right).$$

The matrices $R_j^{(k)} = \bigotimes_{g,j=c_k+1}^{c_{k+1}} Q_j^{(i)}$, for $j \in 1, \ldots, 2E+N$, are, by definition, the transition matrices of a grouped automaton, called $\mathcal{G}_k$ of size $h_k = \prod_{i=c_k+1}^{c_{k+1}} n_i$. The descriptor may be rewritten as

$$Q = \sum_{j=1}^{2E+N} \bigotimes_{g,k=1}^{B} R_j^{(k)}.$$

Separating out the terms resulting from local transitions from those resulting from synchronizing events, we obtain

$$Q = \sum_{j=1}^{N+2E} \bigotimes_{g,i=1}^{N} Q_j^{(i)} = \bigoplus_{g,i=1}^{N} Q_l^{(i)} + \sum_{j=1}^{E} \left(\bigotimes_{g,i=1}^{N} Q_{s_j}^{(i)} + \bigotimes_{g,i=1}^{N} \bar{Q}_{s_j}^{(i)} \right).$$

Grouping by associativity gives

$$Q = \bigoplus_{g,k=1}^{B} R_l^{(k)} + \sum_{j=1}^{E} \left(\bigotimes_{g,k=1}^{B} R_{s_j}^{(k)} + \bigotimes_{g,k=1}^{B} \bar{R}_{s_j}^{(k)} \right),$$

with

$$R_l^{(k)} = \bigoplus_{g,i=c_k+1}^{c_{k+1}} Q_l^{(i)}; \qquad R_{s_j}^{(k)} = \bigotimes_{g,i=c_k+1}^{c_{k+1}} Q_{s_j}^{(i)}; \qquad \bar{R}_{s_j}^{(k)} = \bigotimes_{g,i=c_k+1}^{c_{k+1}} \bar{Q}_{s_j}^{(i)}.$$

First simplification: Removal of synchronizing events.

Assume that one of the synchronizing event, say s_1, is such that it synchronizes automata within a group, say b_1. As a result, this synchronized event becomes internal to group b_1 and may be treated as a transition that is local to $\mathcal{G}_1$. In this case, the value of $R_l^{(1)}$ may be changed in order to simplify the formula for the descriptor. Using

$$R_l^{(1)} \Longleftarrow R_l^{(1)} + R_{s_1}^{(1)} + \bar{R}_{s_1}^{(1)},$$

the descriptor may be rewritten as

$$\bigoplus_{g,k=1}^{B} R_l^{(k)} + \sum_{j=2}^{E} \left(\bigotimes_{g,i=1}^{B} R_{s_j}^{(i)} + \bigotimes_{g,i=1}^{B} \bar{R}_{s_j}^{(i)} \right).$$

The descriptor is thus reduced (two terms having disappeared). This procedure can be applied to all identical situations.

Second simplification: Removal of functional terms.

Assume now that the local transition matrix of $\mathcal{G}_1$ is a tensor sum of matrices whose elements are functions only of the states of the automata that are in the subset b_1. Then the functions in $Q_l^{(i)}$ of

$$R_l^{(1)} = \bigoplus_{g,i=c_1}^{c_2} Q_l^{(i)}$$

may be evaluated when performing the generalized tensor operator and $R_l^{(1)}$ becomes a constant matrix. As for the removal of synchronizing events, this process of replacing functions with constants may be applied in all similar situations.

However, if $R_{s_j}^{(1)}$ is the tensor product of matrices that are functions of the states of automata, some of which are in b_1 and some of which are not in b_1, then performing the generalized tensor product $R_{s_j}^{(1)} = \bigotimes_{g,i=c_1}^{c_2} Q_{s_j}^{(i)}$ allows us to only partially evaluate the functions for the arguments in b_1. Others arguments cannot be evaluated. These must be evaluated later when performing the computation $\bigotimes_{g,i=1}^{B} R_{s_j}^{(i)}$ and may in fact, result in an increased number of function evaluations.

Third simplification: Reduction of the reachable state space.

In the process of grouping, the situation might arise that a grouped automata $\mathcal{G}_i$ has a reachable state space smaller than the product state space, $\prod_{i=c_j+1}^{c_{j+1}} n_i$. This happens after simplifications of type 1 or 2 have been performed. For example, functions may evaluate to zero, or synchronizing events may disable certain transitions. In this case, a reachability analysis may be performed in order to compute the reachable state space. In the SAN methodology, the global reachable state space is known in advance and the reachable state space of a group may be computed by means of a simple projection.

A series of numerical experiments conducted on the two examples presented in Section 4 was reported in [16] and quantity the effect of these simplifications. The goal was to observe the effect on the time required to perform 10 premultiplications of the descriptor by a vector and on the amount of array storage needed.

In the first model, that of resource sharing, the parameters P and N were varied and the automata (recall that all are identical) were grouped in a variety of ways. The results showed a substantial reduction in CPU time as the number of blocks of automata was reduced, — a combined effect of a reduction in the reachable state space, algorithm overhead, and the number of functions that needed to be evaluated, and this with relatively little impact on memory requirements. As concerns memory requirements, two contrasting effects were observed. On the one hand, the reduction in the reachable state space caused a subsequent reduction in the size of the probability vectors and hence an overall reduction in the amount of memory needed. On the other hand, the size of the matrices representing the grouped automata increased thereby increasing the amount of memory needed. This latter effect was observed to become more important as the number of resources (P) approached the number of processes (N) and indeed became the dominant effect.

The queueing network model was also analyzed under a variety of different parameter values and with two different kinds of grouping:

- A grouping of the automata according to customer class ($\mathcal{A}_1$ and $\mathcal{A}_{3_1}$) and ($\mathcal{A}_2$ and $\mathcal{A}_{3_2}$).
- A grouping of the automata according to queue ($\mathcal{A}_1$ and $\mathcal{A}_2$) and ($\mathcal{A}_{3_1}$ and $\mathcal{A}_{3_2}$);

The results showed that the CPU times obtained with the first grouping was *worse* than in the non-grouped case. This is a result of the fact that this model incorporates functions that cannot be removed using simplification 2. Although the first grouping eliminates the synchronizing events, it results in an increase in the number of functions that must be evaluated and increases the overall time needed. The second grouping allowed for the possibility of a reduction in the state space of the joint automata, ($\mathcal{A}_{3_1}$ and $\mathcal{A}_{3_2}$), since the priority queue is now represented by a single automaton. This, along with the elimination of functional elements from the grouped descriptors, lead to a reduction in CPU-time. Additionally, the elimination of non-reachable states reduced the amount of array storage needed so that this grouping lead to a reduction in CPU-time *and* memory needs.

It was concluded from this series of experiments, that the benefits that accrue from grouping are non-negligible, so long as the number of function evaluations do not rise drastically as a result. In fact, it seems that function evaluation should be the main concern in choosing which automata to group together. Indirectly, functions also play an important role in identifying non-reachable states, the elimination of which permit important reductions in CPU time and memory. The number of groups should be kept to a small number. Four or less appeared to be optimal in the set of experiments. Additionally, as we shall see later, a small number of automata may lead to better preconditioning schemes for use with numerical solution methods.

10 Numerical Solution Methods

Consider a stochastic automata network consisting of N automata. Let Q be its descriptor, i.e.,

$$Q = \sum_{j=1}^{2E+N} \otimes_{i=1}^{N} Q_j^{(i)}.$$

Our goal is to find the stationary probability vector, i.e., a vector π such that $\pi Q = 0$ and $\pi e = 1$. Of all the numerical solution methods discussed in [42], only those whose interaction with the infinitesimal generator is its product with a vector, are suitable when the coefficient matrix is given in this form. Thus, the power method and the various projection methods are easy to implement. It is not easy to see how methods such as Gaussian elimination and SOR (which imposes an ordering on the computation of the elements of the solution vector) can be adopted to efficiently solve SANs. Furthermore, as we have already seen in this article, much current research has lead to efficient descriptor-vector multiplication algorithms. In

this final section we shall be content to outline how the SAN descriptor may be imbedded into a suitable numerical algorithm, and provide some information on the choice of suitable preconditioners for these algorithms.

The simplest method that may be used in the context of a SAN is the power method. If P is the stochastic transition probablity matrix of an irreducible Markov chain, the power method is described by the iterative procedure

$$\pi^{(k+1)} = \pi^{(k)} P, \tag{14}$$

where $\pi^{(0)}$ is an arbitrary initial approximation to the solution. When an infinitesimal generator Q, is available, P may be obtained from

$$P = I + \Delta t Q \tag{15}$$

where $\Delta t \leq 1/\max_i |q_{ii}|$. Notice that P may be written as a sum of tensor products, since

$$I + \Delta t Q = \otimes_{i+1}^{N} I_{n_i} + \sum_{j=1}^{2E+N} \Delta t \otimes_{i=1}^{N} Q_j^{(i)}.$$

Thus the power method becomes

$$\pi^{(l+1)} = \pi^{(l)}(I + \Delta t Q) = \pi^{(l)} + \Delta t \pi^{(l)} \left(\sum_{j=1}^{2E+N} \otimes_{i=1}^{N} Q_j^{(i)} \right).$$

This is the form that the power method takes when it is applied to a SAN descriptor.

Projection Methods include the class of methods known as *simultaneous iteration* or *subspace iteration*, [25, 41, 45], which iterate continuously with a fixed number of vectors, as well as methods that begin with a single vector and construct a subspace one vector at a time, [36]. The subspace most often used in Markov chain problems is the *Krylov subspace* given by

$$\mathcal{K}_m(P, v) = span\{v, vP, vP^2, \ldots, vP^{m-1}\}.$$

Notice that this subspace is spanned by consecutive iterates of the power method. Furthermore it is the only part of the projection methods that interacts directly with the coefficient matrix. Thus the same scheme for incorporating a SAN descriptor into the power method is directly applicable in this case also.

It is well known that the projection methods, (and the power method) perform best when accompanied with an effective preconditioner. The objective of preconditioning is to modify the eigenvalue distribution of the iteration matrix so that convergence onto the solution vector may be attained more quickly. In a Markov chain context, this usually implies finding a matrix M^{-1} so that $I - (I - P)M^{-1}$ possesses one unit eigenvalue and $n - 1$ others that are close to zero. The preconditioned power method is written as

$$\pi^{(l+1)} = \pi^{(l)}(I - (I - P)M^{-1}) = \pi^{(l)} - \pi^{(l)}(I - P)M^{-1}. \tag{16}$$

The problem is now one of finding a suitable matrix M^{-1}. One possibility is to approximate the inverse of $I - P$ directly by a polynomial series. It is known that, for any matrix A for which $\|A\| \leq 1$, the inverse of $I - A$ may be written in a Neumann series as, ([20]):

$$(I - A)^{-1} = \sum_{k=0}^{\infty} A^k.$$

Since P is a stochastic matrix, $\|P\| \leq 1$, and so an approximate inverse of $-\Delta t Q = I - P$ is given by

$$M^{-1} = \sum_{k=0}^{K} P^k.$$

When this is substituted into equation (16), it gives

$$\pi^{(l+1)} = \pi^{(l)} + \Delta t \pi^{(l)} Q \left(\sum_{k=0}^{K} P^k \right).$$

Since Q and P may be expressed as a sum of tensor products, all numerical operations may be carried out without having to expand the descriptor of the SAN. When written out in full, the preconditioned power method is given by

$$\pi^{(l+1)} = \pi^{(l)} + \Delta t \pi^{(l)} \left(\sum_{j=1}^{2E+N} \otimes_{i=1}^{N} Q_j^{(i)} \right) \left[\sum_{k=0}^{K} \left(\otimes_{i=1}^{N} I_{n_i} + \sum_{j=1}^{2E+N} \Delta t \otimes_{i=1}^{N} Q_j^{(i)} \right)^k \right]. \quad (17)$$

Similar expressions may be written for preconditioned Arnoldi and GMRES. The advantage of this means of preconditioning is that, by increasing the number of terms in the Neumann expansion, we can obtain an accurate preconditioner and hence expect to converge in relatively few iterations. Unfortunately, the down side is that, as is apparent from equation (17), it is computationally very expensive to compute an accurate preconditioner. The results of a number of experiments in using this preconditioning approach is provided in [44].

Other possibilities for preconditioning involve the computation of incomplete LU factorizations (ILU). These methods essentially follow a Gaussian elimination procedure but drop off elements at various points according to some specific criteria. The result is a lower triangular matrix $\tilde{L}$, an upper triangular matrix $\tilde{U}$ and a matrix $\tilde{E}$ (hopefully small) such that $(I - P) = \tilde{L}\tilde{U} + \tilde{E}$. The preconditioner is taken to be $\tilde{U}^{-1}\tilde{L}^{-1}$. Unfortunately in the context of stochastic automata networks where the infinitesimal generator is in the form of a sum of tensor products, this approach is not economical.

Our recent research has evolved around choosing preconditioners to be the inverses of the individual terms in the descriptor. Since the inverse of a tensor product is just equal to the product of the inverses of its component terms, each individual product in the descriptor is easily invertible. Results seem to indicate that choosing the preconditioner to be the inverse of a single product in the descriptor, even if that product is in some sense dominant, does not

always provide good results. It appears that a better approach is to vary the preconditioners from one iteration to the next, continuously cycling from using the inverse of one product in the descriptor at iteration k to using the inverse of the following product at iteration $k+1$. This is an area of current interest and importance. Much research remains to be done.

11 Conclusion

Any conclusion on a modelling methodology must include a comparison with alternative approaches. In this case the most logical alternative is a sparse matrix approach and the comparison should be based on memory and CPU-time requirements.

In the sparse matrix approach, the infinitesimal generator is stored as a single matrix, albeit in a compact form. This compact form requires a double-precision array for the nonzero elements and two integer arrays to store the positions of the nonzero elements in the matrix. We shall let n denote the order of the infinitesimal generator and nz denote the number of nonzero elements that it contains. Then, the double-precision array must be of length nz and in this array the nonzero elements are arranged by rows; nonzero elements in row k preceed those of row $k+1$ and follow those of row $k-1$. It is not necessary for the nonzero elements within a row to be in order. An integer array, also of length nz holds the column positions of the nonzeros, while a second integer array of length $n+1$ provides pointers to the starting locations of each row in the two other arrays. More information on storing sparse matrices may be found in [42]. We note in passing that the same compact storage scheme may be employed to store the much smaller matrices that arise using the SAN approach. It follows that the sparse matrix approach requires a minimum of nz double precision memory locations and $nz+n+1$ integer locations. This can becomes excessive for large models. Experiments show the feasibility of the SAN approach when memory requirements eliminate the possibility of using a sparse matrix approach.

Our second concern is with computation time. Here we should expect to find cases in which the sparse matrix approach provides much better results, for the contrary would imply that the sparse matrix approach could be eliminated completely, in favor of the SAN approach. Nevertheless, even here, the SAN approach can be competitive in certain cases. A disadvantage of using the sparse matrix approach is that the matrix itself must be generated and the computation cost of this operation increases with the size of the matrix and the percentage of nonzero elements that it contains. The SAN approach does not have this drawback. On the other hand, it does appear that that the cost of a single vector-matrix multiplication is usually more expensive in the SAN approach. This only increases the value of results that allow us to keep multiplication cost to a minimum.

References

[1] K. Atif. Modelisation du Parallelisme et de la Synchronisation. Thèse de Docteur de l'Institut National Polytechnique de Grenoble, 24 September 1992, Grenoble, France.

[2] W.E. Arnoldi. The principle of minimized iteration in the solution of the matrix eigenvalue problem. *Quart. Appl. Math.*, Vol. 9, 1951, pp. 17–29.

[3] F. Baccelli, A. Jean-Marie and I. Mitrani, Editors, Quantitative Methods in Parallel Systems, Part I : Stochastic Process Algebras; *Basic Research Series, Springer*, 1995.

[4] G. Balbo, S.Bruell and M. Sereno. Arrival Theorems for Product-form Stochastic Petri Nets; *Proc. of ACM Sigmetrics Conference 1994*, Nashville, pp. 87-97, 1994.

[5] C. Berge. *Graphes et Hypergraphes.* Dunod, Paris, 1970.

[6] R. Boucherie. A Characterization of Independence for Competing Markov Chains with Applications to Stochastic Petri Nets. *IEEE Transactions on Soft. Eng.*, Vol 20, pp. 536–544, 1994.

[7] P. Buchholz. Equivalence Relations for Stochastic Automata Networks. *Computations with Markov Chains; Proceedings of the 2nd International Meeting on the Numerical Solution of Markov Chains*, W.J. Stewart, Ed., Kluwer International Publishers, Boston, 1995.

[8] P. Buchholz. Aggregation and Reduction Techniques for Hierarchical GCSPNs. *Proceedings of the 5th International Workshop on Petri Nets and Performance Models*, Toulouse, France, IEEE Press, pp. 216–225, October 1993.

[9] P. Buchholz. Hierarchical Markovian Models – Symmetries and Aggregation; *Modelling Techniques and Tools for Computer Performance Evaluation*, Ed. R. Pooley, J.Hillston, Edinburgh, Scotland, pp. 234–246, 1992.

[10] G. Chiola, C. Dutheillet, G. Franceschinis and S. Haddad. Stochastic Well-Formed Colored Nets and Symmetric Modeling Applications. *IEEE Transactions on Computers*, Vol 42, No. 11, pp. 1343–1360, 1993.

[11] G. Ciardo and K. Trivedi. Solution of Large GSPN Models. *Numerical Solution of Markov Chains*, W.J. Stewart, Ed., Marcel Dekker Publisher, New York, pp. 565–595, 1991.

[12] M. Davio. Kronecker Products and Shuffle Algebra. *IEEE Trans. Comput*, Vol. C-30, No. 2, pp. 1099–1109, 1981.

[13] S. Donatelli. Superposed Stochastic Automata: A Class of Stochastic Petri Nets with Parallel Solution and Distributed State Space. *Performance Evaluation*, Vol. 18, pp. 21–36, 1993.

[14] S. Donatelli and M. Serano. On the Product Form Solution for Stochastic Petri Nets. *Proc. of the 13th International Conference on Applications and Theory of Petri Nets*, Sheffield, UK, pp. 154–172, 1992.

[15] P. Fernandes, B. Plateau and W.J. Stewart. Efficient Descriptor–Vector Multiplications in Stochastic Automata Networks. INRIA Report # 2935. Anonymous ftp *ftp ftp.inria.fr/INRIA/Publication/RR.*

[16] P. Fernandes, B. Plateau and W.J. Stewart. Numerical Issues for Stochastic Automata Networks. PAPM 96, Fourth Process Algebras and Performance Modelling Workshop, Torino, Italy, July 1996.

[17] J-M. Fourneau and F. Quessette. Graphs and Stochastic Automata Networks. *Computations with Markov Chains; Proceedings of the 2nd International Meeting on the Numerical Solution of Markov Chains*, W.J. Stewart, Ed., Kluwer International Publishers, Boston, 1995.

[18] G. Franceschinis and R. Muntz. Computing Bounds for the Performance Indices of Quasi-lumpable Stochastic Well-Formed Nets. *Proceedings of the 5th International Workshop on Petri Nets and Performance Models*, Toulouse, France, IEEE Press, pp. 148–157, October 1993.

[19] D. Frosh and K. Natarajan. Product Form Solutions for Closed Synchronized Systems of Stochastic Sequential Processes. *Proc. of 1992 International Computer Symposium*, Taiwan, pp. 392–402, 1991

[20] A. Greenbaum, P.F. Dubois and G.H. Rodrique. Approximating the Inverse of a Matrix for use in Iterative Algorithms on Vector Processors. *Computing*, Vol. 22, 1979, pp. 257-268.

[21] W. Henderson and D. Lucic. Aggregation and Disaggregation through Insensitivity in Stochastic Petri Nets; *Performance Evaluation*, Vol. 17, pp. 91–114, 1993.

[22] W. Henderson and P.G. Taylor. Embedded Processes in Stochastic Petri Nets. *IEEE Trans. in Software Engineering*, Vol. 17, pp 108–116, 1991.

[23] H. Hermanns and M. Rettelbach. Syntax, Semantics, Equivalences, and Axioms for MTIPP. *Proc. of the 2nd Workshop on Process Algebras and Performance Modelling*, U. Herzog, M. Rettelbach, Editors, Arbeitsberichte, Band 27, No. 4, Erlangen, November 1994.

[24] J. Hillston. Computational Markovian Modelling using a Process Algebra. *Computations with Markov Chains; Proceedings of the 2nd International Meeting on the Numerical Solution of Markov Chains*, W.J. Stewart, Ed., Kluwer International Publishers, Boston, 1995.

[25] A. Jennings and W.J. Stewart. Simultaneous iteration for Partial Eigensolution of Real Matrices. *J. Inst. Math. Applics.*, Vol. 15, 1975, pp. 351-361.

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

[27] P. Kemper. Closing the Gap between Classical and Tensor Based Iteration Techniques. *Computations with Markov Chains; Proceedings of the 2nd International Meeting on the Numerical Solution of Markov Chains*, W.J. Stewart, Ed., Kluwer International Publishers, Boston, 1995.

[28] A.A. Lazar and T.G. Robertazzi. Markovian Petri Net Protocols with Product Form Solutions. *Performance Evaluation*, Vol. 12, pp. 67–77, 1991.

[29] C.D. Meyer. The Role of the Group Generalized Inverse in the Theory of Finite Markov Chains. *Siam Review*, Vol. 17, No. 3, July 1975.

[30] B. Philippe, Y. Saad and W.J. Stewart. Numerical Methods in Markov Chain Modelling. *Operations Research*, Vol.40, No. 6, 1992, pp. 1156-1179.

[31] B. Plateau. On the Stochastic Structure of Parallelism and Synchronization Models for Distributed Algorithms. *Proc. ACM Sigmetrics Conference on Measurement and Modelling of Computer Systems*, Austin, Texas, August 1985.

[32] B. Plateau and K. Atif. Stochastic Automata Network for Modelling Parallel Systems. *IEEE Trans. on Software Engineering*, Vol. 17, No. 10, pp. 1093–1108, 1991.

[33] B. Plateau and J.M. Fourneau. A Methodology for Solving Markov Models of Parallel Systems. *Journal of Parallel and Distributed Computing*. Vol. 12, pp. 370–387, 1991.

[34] B. Plateau, J.M. Fourneau and K.H. Lee. PEPS: A Package for Solving Complex Markov Models of Parallel Systems. In R. Puigjaner, D. Potier, Eds., *Modelling Techniques and Tools for Computer Performance Evaluation*, Spain, September 1988.

[35] Y. Saad. Variations on Arnoldi's method for computing eigenelements of large unsymmetric matrices. *Lin. Alg. Appl.*, Vol. 34, 1980, pp. 269–295.

[36] Y. Saad. Krylov Subspace Methods for Solving Unsymmetric Linear Systems. *Mathematics of Computation*, Vol. 37, 1981, pp. 105–126.

[37] Y. Saad and M.H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, Vol. 7, 1986, pp. 856–869.

[38] W.H. Sanders and J.F. Meyer. Reduced Base Model Construction Methods for Stochastic Activity Networks, *IEEE Jour. on Selected Areas in Communication*, Vol. 9, No. 1, pp. 25–36, 1991.

[39] M. Siegle. On Efficient Markov Modelling. *In Proc. QMIPS Workshop on Stochastic Petri Nets*, pp. 213–225, Sophia-Antipolis, France, November 1992.

[40] C. Simone and M.A. Marsan. The Application of the EB-Equivalence Rules to the Structural Reduction of GSPN Models. *Journal of Parallel and Distributed Computing*, Vol. 15, No. 3, pp. 296–302, 1991.

[41] G.W. Stewart. Simultaneous iteration for computing invariant subspaces of non-Hermitian matrices. *Numer. Mat.*, Vol. 25, 1976, pp. 123–136.

[42] W.J. Stewart. *An Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, New Jersey, 1994.

[43] W.J. Stewart. MARCA: Markov Chain Analyzer. *IEEE Computer Repository* No. R76 232, 1976. Also IRISA Publication Interne No. 45, Université de Rennes, France.

[44] W.J. Stewart, K. Atif and B. Plateau. The Numerical Solution of Stochastic Automata Networks. *European Journal of Operations Research*, Vol. 86, No. 3, pp. 503–525, 1995.

[45] W.J. Stewart and A. Jennings. A Simultaneous Iteration Algorithm for Real Matrices. *ACM Transactions on Mathematical Software*, Vol. 7, No. 2, 1981, pp. 184-198.

[46] W.J. Stewart and W. Wu. Numerical Experiments with Iteration and Aggregation for Markov Chains. *ORSA Journal on Computing*, July/August, 1992.

[47] J.H. Wilkinson. *The Algebraic Eigenvalue Problem.* Clarendon Press, New York, 1965.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399