Efficient Descriptor-Vector Multiplications in Stochastic Automata Networks

Paulo Fernandes, Brigitte Plateau, William J. Stewart

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Efficient Descriptor-Vector Multiplications in Stochastic Automata Networks

Paulo Fernandes, Brigitte Plateau, William J. Stewart

N° 2935

10 juillet 1996

THÈME 1

ISSN 0249-6399

Efficient Descriptor-Vector Multiplications in Stochastic Automata Networks

Paulo Fernandes[*], Brigitte Plateau[†], William J. Stewart[‡]

Thème 1 — Réseaux et systèmes
Projet APACHE

Rapport de recherche n° 2935 — 10 juillet 1996 — 35 pages

Abstract: This paper examines numerical issues in computing solutions to networks of stochastic automata. It is well-known that when the matrices that represent the automata contain only constant values, the cost of performing the operation basic to all iterative solution methods, that of matrix-vector multiply, is given by

$$\rho_N = \prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} n_i,$$

where n_i is the number of states in the i^{th} automaton and N is the number of automata in the network. We introduce the concept of a generalized tensor product and prove a number of lemmas concerning this product. The result of these lemmas allows us to show that this relatively small number of operations is sufficient in many practical cases of interest in which the automata contain functional and not simply constant transitions. Furthermore, we show how the automata should be ordered to achieve this.

Key-words: Markov chains, Stochastic automata networks, Generalized tensor algebra, 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*, and CAPES-COFECUB Agreement (Project 140/93).
[†] IMAG–LMC, 100 rue des Mathématiques, 38041 Grenoble cedex, France. Research supported by the (CNRS – INRIA – INPG – UJF) joint project *Apache*, and CAPES-COFECUB Agreement (Project 140/93).
[‡] 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

Multiplication Vecteur-Descripteur Efficace pour les Réseaux d'Automates Stochastiques

Résumé : Cet article étudie les problèmes numériques posés par la résolution de modèles de réseaux d'automates stochastiques. Il est bien connu que si les matrices qui représentent les automates ne contiennent que des constantes numériques, le coût de l'opération de base de toute méthode itérative, celui du produit matrice-vecteur est donné par la formule

$$\rho_N = \prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} n_i,$$

où n_i est le nombre d'états du i-ième automate et N le nombre d'automates du réseaux. Nous introduisons le concept de produit tensoriel généralisé et prouvons quelques lemmes sur cet opérateur. Les résultats de ces lemmes permettent de montrer que ce petit nombre d'opérations est suffisant dans de nombreux cas pratiques où les automates contiennent des taux de transitions fonctionnels et non simplement constants. De plus, nous donnons l'ordre dans lequel les automates doivent être rangés afin d'obtenir ce résultat.

Mots-clé : Chaîne de Markov, Réseau d'automates stochastiques, Algèbre tensorielle généralisée, Multiplication vecteur-descripteur.

1 Introduction

The use of *Stochastic Automata Networks* (SANs) 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, [16, 17], 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 [17, 2, 6]. 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, [20]. Nevertheless, computational results such as those presented in this paper 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 use of symmetries as well as lumping and various superpositioning of the automata to reduce the computational burden, [1, 8, 24]. Furthermore, in [13], 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 [7, 8, 9, 14, 23, 25], equivalence relations and symmetries are used to decrease the computational burden of obtaining performance indices. In [15], 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 [10], nearly independent subnets are exploited in an iterative procedure in which a global solution is obtained from partial solutions. In [5, 3] product forms are found in Petri nets models, using, in the first, the state space structure and in the second, flow properties. In [12] it is shown that the tensor structure of the transition matrix may be extracted from a stochastic Petri net, and in [18] 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 sophisicated 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. The number of iterations needed to compute the solution to a required accuracy depends on the method chosen. In a previous paper, [28], it was shown how projection methods such as Arnoldi and GMRES could be used to substantially reduce the number of iterations needed when compared with the basic power method. Additionally, some results were also given concerning the development of preconditioning strategies that may be used to speed the iterative process even further, but much work still remains to be done in this particular domain. In this paper we concentrate on procedures that allow us to keep the amount of computation per iteration to a minimum. Specifically, we wish to study the cost of performing a matrix-vector multiplication when the matrix is stored as a compact SAN descriptor, since this is a step that is fundamental to all iterative methods and is by far, the major cost per iteration.

It is well-known that when the matrices that represent the automata contain only constant values, the cost of performing a matrix-vector multiply is given by

$$\rho_N = \prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} n_i$$

where n_i is the number of states in the i^{th} automaton and N is the number of automata in the network. When the automata are not independent, which is always the case, (for

otherwise the model can be completely decomposed and the exact solution computed from the solution of the individual pieces), the number of multiplications can become much larger. We provide a number of lemmas that allow us to show that this relatively small number of operations is sufficient in many cases in which the automata are not independent, and we show how the automata should be arranged to achieve this.

Our paper is set out as follows. In the next section we provide a number of basic properties of tensor algebra which we will need throughout the paper. Section 3 presents the descriptor of a SAN and briefly reviews the effects of synchronizing events and functional transition rates. The basic descriptor-vector multiplication procedure is given in Section 4 and an algorithm provided for the nonfunctional case. In Section 5 we introduce the generalized tensor product concept and prove a number of lemmas concerning this product. The result of these lemmas allows us to develop an algorithm for efficiently computing a descriptor-vector product and this is shown in Section 6. This is followed by some numerical experiments in Section 7 and our conclusions in Section 8.

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 product $C = A \otimes B$ is defined by assigning the element of C that is in the (i_2, j_2) position of block (i_1, j_1), the value $a_{i_1 j_1} b_{i_2 j_2}$. We shall write this as

$$c_{\{(i_1,j_1);(i_2,j_2)\}} = a_{i_1 j_1} b_{i_2 j_2}.$$

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. 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 [21] wherein P_σ is explicitly given. Further information concerning the properties of tensor algebra may be found in Davio [11].

3 Stochastic Automata Networks

We consider a stochastic automata network that consists of N individual stochastic automata that act more or less independently of each other. The number of states in the k^{th} automaton shall be denoted by n_k for $k = 1, 2, \ldots, N$. Most often, one's goal is the computation of the stationary probability distribution, π, or transient distributions at arbitrary times t, $\pi(t)$, of the overall N–dimensional system.

Throughout this paper, we present results on the basis of continuous-time SANs, although the results are equally valid in the context of discrete-time SANs. 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 distribution of the N-dimensional system at time t is

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

Although such systems may exist, the more usual case occurs when the transitions of one automaton may depend on the state of a second. 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 *synchronizing* transitions. Synchronizing 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.

3.1 The Effect of Synchronizing Events

Let us denote by $Q_l^{(k)}$, $k = 1, 2, \ldots, N$, the matrix consisting only of the transitions that are local to $\mathcal{A}^{(k)}$. Then, the part of the global infinitesimal generator that consists uniquely of local transitions may be obtained by forming the tensor sum of the matrices

$Q_l^{(1)}, Q_l^{(2)}, \ldots, Q_l^{(N)}$. As a general rule, it is shown in [19], that stochastic automata networks 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 N terms) 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

$$Q = \sum_{j=1}^{2E+N} \otimes_{i=1}^{N} Q_j^{(i)}. \tag{3}$$

This formula is referred to as the *descriptor* of the stochastic automata network. It is important to note that, although Q is an infinitesimal generator, the individual matrices $Q_j^{(i)}$ need not be. For example, such matrices may consist of a single positive element. Since we shall treat the matrices $Q_j^{(i)}$ as purely algebraic quantities, their relationship as stochastic entities is not needed. Notice also that the state space represented in the matrix Q may contain unreachable states. For more information on the structure and properties of these matrices, the interested reader is invited to consult [28].

The computational burden imposed by synchronizing events is two-fold. First, the number of terms in the *descriptor* is increased, — two for each synchronizing event. A second and even greater burden is that the simple form of the solution, as a tensor product of independent solutions, 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 $Q_j^{(i)}$. This is a direct result of the fact that the automata are not independent. Thus this second computational burden is not unique to synchronizing transitions but also arises with functional transitions.

3.2 The Effect of Functional Transition Rates

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. So, although the effect of the dependent interactions among the individual automata prevents us from writing the solution in the simple form as a tensor product of individual solutions, it may still be possible to profit from the fact that the nonzero structure is unchanged. This is the concept behind the extended tensor algebraic approach, [21]. The descriptor is still written as in equation (3), 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. In this paper we shall show how to efficiently compute the product of a vector with a SAN descriptor using the extended (generalized) tensor algebra concepts.

3.3 Examples

We now introduce two 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.

3.3.1 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.

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

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

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.

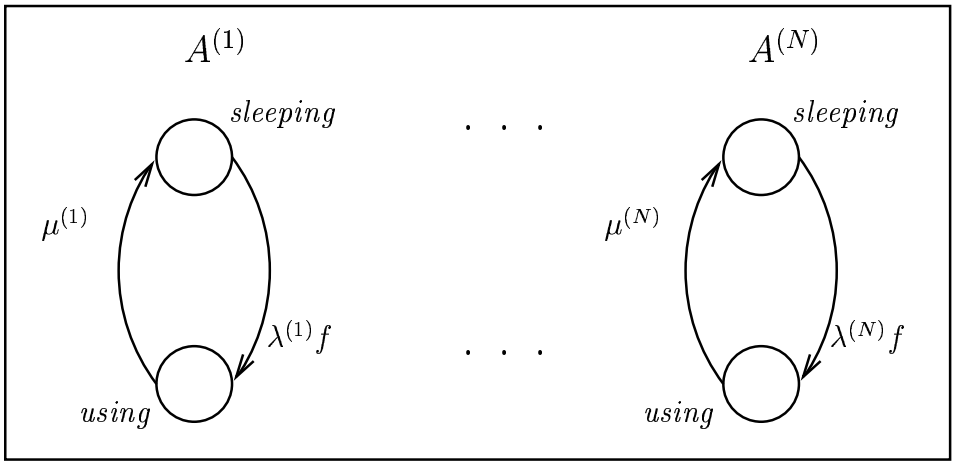

Figure 1: Resource Sharing Model

3.3.2 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

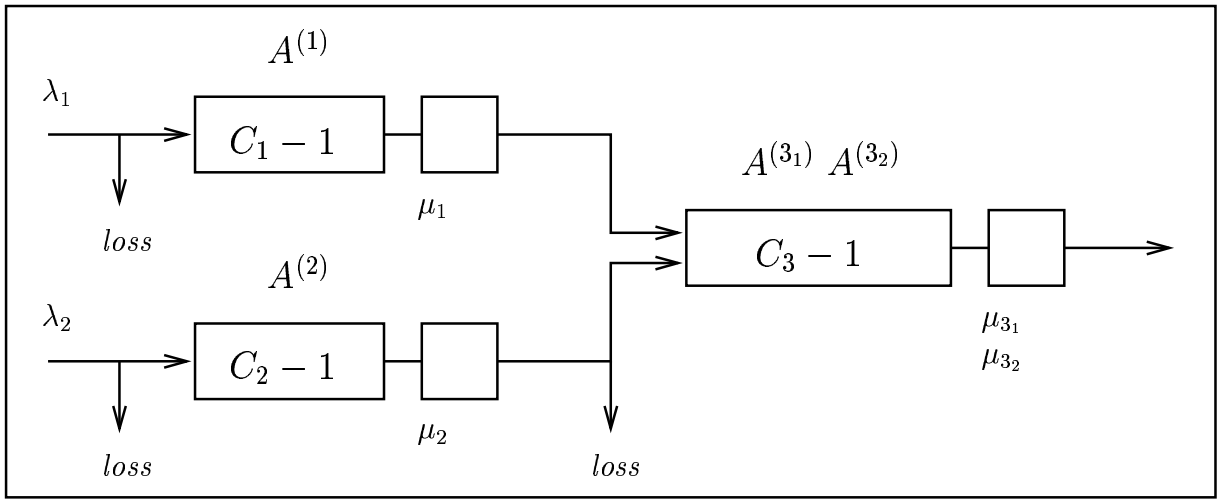

Figure 2: Network of Queues Model

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:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The overall descriptor for this model is given by

$$D = \bigoplus\nolimits_g Q_l^{(i)} + \bigotimes\nolimits_g Q_{s_1}^{(i)} + \bigotimes\nolimits_g \bar{Q}_{s_1}^{(i)} + \bigotimes\nolimits_g Q_{s_2}^{(i)} + \bigotimes\nolimits_g \bar{Q}_{s_2}^{(i)},$$

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.

4 Computing Probability Distributions

When the global infinitesimal generator of a SAN is available only in the form of a SAN descriptor, equation (3), the numerical methods most suitable to obtaining probability distributions are iterative, [28]. 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 $Q_j^{(i)}$.

4.1 Multiplication with a Vector: The Nonfunctional Case

The following theorem holds when the transition rate matrices corresponding to the stochastic automata contain only constant terms.

Theorem 4.1 *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

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

To compare this with the number needed when advantage is not taken of the special structure resulting from the tensor product, observe that to naively expand $\otimes_{i=1}^{N} Q^{(i)}$ requires $\left(\prod_{i=1}^{N} n_i\right)^2$ multiplications and that this same number is required each time the product $x\left(\otimes_{i=1}^{N} Q^{(i)}\right)$ is formed. A proof of the theorem is given in [21] and is based on the following tensor product decomposition.

$$\begin{aligned} y = x \times \otimes_{i=1}^{N} Q^{(i)} &= x \times \prod_{i=1}^{N} I_{n_1} \otimes \cdots \otimes I_{n_{i-1}} \otimes Q^{(i)} \otimes I_{n_{i+1}} \otimes \cdots \otimes I_{n_N} \\ &= x \times \prod_{i=1}^{N} I_{1:i-1} \otimes Q^{(i)} \otimes I_{i+1:N}. \end{aligned} \tag{4}$$

The individual terms within the product, (i.e., the $I_{1:i-1} \otimes Q^{(i)} \otimes I_{i+1:N}$) are referred to as the *normal factors* of the decomposition. It can be shown (see Equation 9), that these normal factors commute. Notice that we have used $I_{1:i-1}$ to represent an identity matrix of size $\prod_{k=1}^{i-1} n_k$ and $I_{i+1:N}$ to represent one of size $\prod_{k=i+1}^{m} n_k$. In general, we shall let $I_{i:j} \equiv I_{j:i}$ denote an identity matrix of size $\prod_{k=i}^{j} n_k$. The following algorithm implements this multiplication.

Algorithm: Multiplication with a Tensor Product, $x \otimes_{i=1}^{N} Q^{(i)}$

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 $j = 1, 2, \ldots, nright$ do
       - $index = base + j$
       - For $l = 1, 2, \ldots, n_i$ do
         - $z_l = x_{index}$; $index = index + nright$
       - Multiply: $z' = z \times Q^{(i)}$
       - $index = base + j$
       - 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. $x = x'$

During the execution of this algorithm, the variables $nleft$ and $nright$ assume respectively, the size of all left and right identity matrices in the normal factors of equation (4). As the normal factors commute, we have chosen, for reasons that will become clear later, to perform the multiplications in the order from N to 1. Furthermore, if we let $I_{\tilde{n}_i}$ denote the identity matrix of size $\prod_{k=1,k\neq i}^{N} n_k$, then it may be observed that this algorithm operates by implicitly incorporating permutations so that each term in the product becomes

$$I_{\tilde{n}_i} \otimes Q^{(i)},$$

i.e., a block diagonal matrix whose $\prod_{k=1,k\neq i}^{N} n_k$ diagonal blocks are identically equal to $Q^{(i)}$. In fact, it is actually the permuted elements of the vector x that are used, (the value of $index$ in the innermost loops (the l-loops) is $index = (k-1) \times jump + j + (l-1) \times nright$). The manner in which the elements of the $Q^{(i)}$ are stored is not explicitly defined so that any storage scheme, including a compact format suitable for sparse matrices, may be used.

We are now ready to 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 not always possible. It was previously observed in [28] that in the case of two automata $\mathcal{A}$ and $\mathcal{B}$ with matrix representations A and $B[\mathcal{A}]$ respectively, where to anticipate the introduction of some new notation, $B[\mathcal{A}]$ indicates that the matrix B contains elements that may be a function of the state of $\mathcal{A}$, the number of multiplications needed to premultiply $A \otimes B[\mathcal{A}]$ with a vector x remains identical to the nonfunctional case and that exactly the same algorithm could be used. Furthermore, it was also observed that when A contains functional transition rates, but not B, the computation could be rearranged to compute $x(A[\mathcal{B}] \otimes B)$ in the same small number of multiplications as the nonfunctional case. When both contained functional transition rates no computational savings appeared to be possible. Furthermore, it was speculated that in SANs with many functional automata, there exists optimal permutations of the automata for minimizing the computational task of forming the product of a vector and a SAN descriptor. However, this presupposes that it is in fact possible to permute the terms in a generalized tensor products.

5 Generalized Tensor Products

We now turn our attention to some properties relating to *Generalized Tensor Products (GTPs)* as opposed to *Ordinary Tensor Products (OTP)*. We assume throughout that all matrices are square. As indicated in the previous section, $B[\mathcal{A}]$ will indicate that the matrix B may contain transitions that are a function of the state of the automaton $\mathcal{A}$, or more generally, $A^{(m)}[\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(m-1)}]$ will indicate 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

$$A \otimes_g B[\mathcal{A}] = \begin{pmatrix} a_{11}B(a_1) & a_{12}B(a_1) & \cdots & a_{1n_a}B(a_1) \\ a_{21}B(a_2) & a_{22}B(a_2) & \cdots & a_{2n_a}B(a_2) \\ \vdots & \vdots & \ddots & \vdots \\ a_{n_a1}B(a_{n_a}) & a_{n_a2}B(a_{n_a}) & \cdots & a_{n_an_a}B(a_{n_a}) \end{pmatrix}, \tag{5}$$

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

$$A[\mathcal{B}] \otimes_g B[\mathcal{A}] = \begin{pmatrix} a_{11}[\mathcal{B}]I_{n_b} \times B(a_1) & a_{12}[\mathcal{B}]I_{n_b} \times B(a_1) & \cdots & a_{1n_a}[\mathcal{B}]I_{n_b} \times B(a_1) \\ a_{21}[\mathcal{B}]I_{n_b} \times B(a_2) & a_{22}[\mathcal{B}]I_{n_b} \times B(a_2) & \cdots & a_{2n_a}[\mathcal{B}]I_{n_b} \times B(a_2) \\ \vdots & \vdots & \ddots & \vdots \\ a_{n_a1}[\mathcal{B}]I_{n_b} \times B(a_{n_a}) & a_{n_a2}[\mathcal{B}]I_{n_b} \times B(a_{n_a}) & \cdots & a_{n_an_a}[\mathcal{B}]I_{n_b} \times B(a_{n_a}) \end{pmatrix}. \tag{6}$$

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

We are now in a position to determine whether the basic properties of tensor products and additions described in Section 2 also hold in the gereralized case.

Lemma 5.1 (GTP: Associativity)

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

Proof: We shall evaluate the left-hand side first. Referring back to equation (6), it is evident that $D[\mathcal{C}] \equiv A[\mathcal{B}, \mathcal{C}] \otimes_g B[\mathcal{A}, \mathcal{C}]$ is a block matrix whose ij^{th} block is equal to

$$D_{ij}[\mathcal{C}] = diag\{a_{ij}(b_1, \mathcal{C}), a_{ij}(b_2, \mathcal{C}), \ldots, a_{ij}(b_{n_b}, \mathcal{C})\} \times B(a_i, \mathcal{C}).$$

Each element on the k^{th} row of $B(a_i, \mathcal{C})$ is thus multiplied by $a_{ij}(b_k, \mathcal{C})$ so that the kl^{th} element of $D_{ij}[\mathcal{C}]$ is

$$a_{ij}(b_k, \mathcal{C}) b_{kl}(a_i, \mathcal{C}). \tag{7}$$

Now consider the effect of incorporating the third term into the generalized tensor product. In computing $D[\mathcal{C}] \otimes_g C[\mathcal{A}, \mathcal{B}]$, each of the elements in each of the blocks $D_{ij}[\mathcal{C}]$ multiplies the matrix $C[\mathcal{A}, \mathcal{B}]$; in particular the kl^{th} element (given in equation (7)) multiplies the matrix $C(a_i, b_k)$ giving the $(n_c \times n_c)$ block $a_{ij}(b_k, \mathcal{C}) b_{kl}(a_i, \mathcal{C}) C(a_i, b_k)$. The mn^{th} element of this final block is

$$a_{ij}(b_k, c_m) b_{kl}(a_i, c_m) c_{mn}(a_i, b_k).$$

To show that we get the same result from the right-hand side, we proceed in a similar fashion. Let $Z[\mathcal{A}] \equiv B[\mathcal{A}, \mathcal{C}] \otimes C[\mathcal{A}, \mathcal{B}]$. Then $Z[\mathcal{A}]$ is block matrix whose kl^{th} block is equal to

$$Z_{kl}[\mathcal{A}] = diag\{b_{kl}(\mathcal{A}, c_1), b_{kl}(\mathcal{A}, c_2), \ldots, b_{kl}(\mathcal{A}, c_{n_c})\} \times C(\mathcal{A}, b_i).$$

The mn^{th} element of $Z_{kl}[\mathcal{A}]$ is

$$b_{kl}(\mathcal{A}, c_m) c_{mn}(\mathcal{A}, b_k).$$

When forming $A[\mathcal{B}, \mathcal{C}] \otimes_g Z[\mathcal{A}]$, the product of each of the elements of $A[\mathcal{B}, \mathcal{C}]$ and the matrix $Z[\mathcal{A}]$ is formed. In particular, the ij^{th} element is multiplied with $Z[\mathcal{A}]$ to give the $(n_b n_c \times n_b n_c)$ block

$$a_{ij}(\mathcal{B}, \mathcal{C}) Z[\mathcal{A}]$$

When the mn^{th} element of the kl^{th} block of $Z[\mathcal{A}]$ is considered, this gives

$$a_{ij}(b_k, c_m)b_{kl}(a_i, c_m)c_{mn}(a_i, b_k)$$

as before. □

Lemma 5.2 (GTP: Distributivity over Addition)

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

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

Proof: As for the previous lemma, the proof of Lemma 5.2 is by simple algebraic identification. □

As was the case in Section 2 for ordinary tensor products, compatibility with multiplication generally does not hold for the generalized tensor product 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. These are given next.

Lemma 5.3 (GTP: Compatibility over Multiplication: I — Two Factors)

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

Proof: Since we assume that $A[\mathcal{C}]$ and $B[\mathcal{C}]$ are square matrices, their dimensions must be identical for the product $A[\mathcal{C}] \times B[\mathcal{C}]$ to exist. Thus, $(A[\mathcal{C}] \times B[\mathcal{C}]) \otimes_g I_{n_c}$, $A[\mathcal{C}] \otimes_g I_{n_c}$ and $B[\mathcal{C}] \otimes_g I_{n_c}$ all have the same (square) dimensions. Observe that all three of these matrices have block elements that are in fact diagonal matrices of order I_{n_c}. The ij^{th} block element of the matrix on the left-hand side is given by $(\sum_{k=1}^{n_a} a_{ik}[\mathcal{C}]b_{kj}[\mathcal{C}])\, I_{n_c}$. The corresponding block on the right-hand side is formed from the product of the i^{th} block-row of $A[\mathcal{C}] \otimes_g I_{n_c}$ with the j^{th} block-column of $B[\mathcal{C}] \otimes_g I_{n_c}$. This is given by $(\sum_{k=1}^{n_a} a_{ik}[\mathcal{C}]I_{n_c} \times b_{kj}[\mathcal{C}]I_{n_c}) = (\sum_{k=1}^{n_a} a_{ik}[\mathcal{C}]b_{kj}[\mathcal{C}])\, I_{n_c}$ as required. □

Remark: The same property holds analogously for the case

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

Lemma 5.4 (GTP: Compatibility over Multiplication: II — Two Factors)

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

Proof: The proof follows immediately on observing that the right-hand side

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

may be written as

$$\begin{pmatrix} B(a_1) & & & \\ & B(a_2) & & \\ & & \ddots & \\ & & & B(a_{n_a}) \end{pmatrix} \begin{pmatrix} a_{11}I_{n_b} & a_{12}I_{n_b} & \cdots & a_{1n_a}I_{n_b} \\ a_{21}I_{n_b} & a_{22}I_{n_b} & \cdots & a_{2n_a}I_{n_b} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n_a 1}I_{n_b} & a_{n_a 2}I_{n_b} & \cdots & a_{n_a n_a}I_{n_b} \end{pmatrix}$$

□

It is worth noticing that factors containing A and $B[\mathcal{A}]$ are reversed when comparing the right-hand side and left-hand side.

Lemma 5.5 (GTP: Compatibility over Multiplication: III — Two Factors)

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

Proof: The proof of this lemma follows immediately from the fact that

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

$$= \begin{pmatrix} a_{11}[\mathcal{B}]I_{n_b} & a_{12}[\mathcal{B}]I_{n_b} & \cdots & a_{1n_a}[\mathcal{B}]I_{n_b} \\ a_{21}[\mathcal{B}]I_{n_b} & a_{22}[\mathcal{B}]I_{n_b} & \cdots & a_{2n_a}[\mathcal{B}]I_{n_b} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n_a 1}[\mathcal{B}]I_{n_b} & a_{n_a 2}[\mathcal{B}]I_{n_b} & \cdots & a_{n_a n_a}[\mathcal{B}]I_{n_b} \end{pmatrix} \begin{pmatrix} B & & & \\ & B & & \\ & & \ddots & \\ & & & B \end{pmatrix}. \qquad (8)$$

Notice from equation (8) that the right-hand matrix is a block diagonal matrix whose diagonal blocks are all equal to B and the left-hand matrix is formed from blocks that are diagonal and the same size as B. It therefore follows that

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

□

We wish to point out that this lemma still holds if I_{n_a} is replaced by any constant matrix.

Remark: A direct consequence of these two lemmas is that for matrix with constant entries, the following product is commutative, (See also [11]). This property does not hold for matrices with functional entries.

$$A \otimes B = (A \otimes I_{n_b}) \times (I_{n_a} \otimes B) = (I_{n_a} \otimes B) \times (A \otimes I_{n_b}) \tag{9}$$

The next lemma is a generalization of Lemma 5.3 and is particularly useful in treating SANs.

Lemma 5.6 (GTP: Compatibility over Multiplication — Many Factors)

$$\begin{aligned} A^{(1)} \otimes_g A^{(2)}[\mathcal{A}^{(1)}] \otimes_g A^{(3)}[\mathcal{A}^{(1)}, \mathcal{A}^{(2)}] \otimes_g \cdots \otimes_g A^{(m)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(m-1)}] = \\ & I_{1:m-1} \otimes_g A^{(m)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(m-1)}] \\ \times \quad & I_{1:m-2} \otimes_g A^{(m-1)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(m-2)}] \otimes_g I_{m:m} \\ \times \quad & \cdots \\ \times \quad & I_{1:1} \otimes_g A^{(2)}[\mathcal{A}^{(1)}] \otimes_g I_{3:m} \\ \times \quad & A^{(1)} \otimes_g I_{2:m} \end{aligned} \tag{10}$$

Proof: We shall prove this lemma by induction.

Define $H^{(1)} = A^{(1)}$ and $H^{(2)} = H^{(1)} \otimes_g A^{(2)}[\mathcal{A}^{(1)}]$. Then, from Lemma 5.4,

$$H^{(2)} = \left(I_{1:1} \otimes_g A^{(2)}[\mathcal{A}^{(1)}]\right) \times \left(H^{(1)} \otimes_g I_{2:2}\right) = \left(I_{1:1} \otimes_g A^{(2)}[\mathcal{A}^{(1)}]\right) \times \left(A^{(1)} \otimes_g I_{2:2}\right) \tag{11}$$

where $I_{1:1}$ is the identity matrix of size n_1 and $I_{2:2}$, the identity matrix of size n_2 with n_1 and n_2 denoting the number of states in automata $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ respectively. $H^{(2)}$ is a matrix of size $n_1 n_2$. Before proceeding to the general induction step, let us consider $H^{(3)}$. We have

$$H^{(3)} = H^{(2)} \otimes_g A^{(3)}[\mathcal{A}^{(1)}, \mathcal{A}^{(2)}]. \tag{12}$$

Applying Lemma 5.4 to the right-hand side of equation (12) gives

$$H^{(3)} = \left(I_{1:2} \otimes_g A^{(3)}[\mathcal{A}^{(1)}, \mathcal{A}^{(2)}]\right) \times \left(H^{(2)} \otimes_g I_{3:3}\right).$$

We would like to substitute for $H^{(2)}$ from equation (11), but we must be careful for $I_{2:2}$ in equation (11) is the identity matrix of order n_2. Substituting we find

$$H^{(3)} = \left(I_{1:2} \otimes_g A^{(3)}[\mathcal{A}^{(1)}, \mathcal{A}^{(2)}]\right) \times \left(\left[\left(I_{1:1} \otimes_g A^{(2)}[\mathcal{A}^{(1)}]\right) \times \left(A^{(1)} \otimes_g I_{n_2}\right)\right] \otimes_g I_{3:3}\right).$$

Now applying Lemmas 5.1 and 5.3 we obtain

$$H^{(3)} = \left(I_{1:2} \otimes_g A^{(3)}[\mathcal{A}^{(1)}, \mathcal{A}^{(2)}]\right) \times \left(\left[\left(I_{1:1} \otimes_g A^{(2)}[\mathcal{A}^{(1)}] \otimes_g I_{3:3}\right) \times \left(A^{(1)} \otimes_g I_{n_2}\right) \otimes_g I_{3:3}\right]\right),$$

which we can write in the form of the lemma statement as

$$H^{(3)} = \left(I_{1:2} \otimes_g A^{(3)}[\mathcal{A}^{(1)}, \mathcal{A}^{(2)}]\right) \times \left(I_{1:1} \otimes_g A^{(2)}[\mathcal{A}^{(1)}] \otimes_g I_{3:3}\right) \times \left(A^{(1)} \otimes_g I_{2:3}\right).$$

Our inductive approach to proving this lemma should now be clear. The basis step has already been established. For our inductive assumption, we define

$$H^{(m)} \equiv A^{(1)} \otimes_g A^{(2)}[\mathcal{A}^{(1)}] \otimes_g A^{(3)}[\mathcal{A}^{(1)}, \mathcal{A}^{(2)}] \otimes_g \cdots \otimes_g A^{(m)}[\mathcal{A}^{(1)}, \ldots \mathcal{A}^{(m-1)}]$$

and assume that the relation given in the statement of the lemma is true. We now prove that the relation is true for $H^{(m+1)}$. We have

$$H^{(m+1)} = H^{(m)} \otimes_g A^{(m+1)}[\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(m)}].$$

Applying Lemma 5.4 gives

$$H^{(m+1)} = \left(I_{1:m} \otimes_g A^{(m+1)}[\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(m)}]\right) \times \left(H^{(m)} \otimes_g I_{m+1:m+1}\right).$$

Substituting for $H^{(m)}$ from equation (10) and repeatedly applying Lemmas 5.3 and 5.1 gives

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

The proof of the lemma follows immediately given that the terms of the form $I_{i+1:m}$ for $i = 1, 2, \ldots, m-1$ are identity matrices of size $\prod_{k=i+1}^{m} n_k$. □

Remark: Due to the existence of Lemma 5.5 the same property holds for

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

In Lemma 5.6, 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

right-hand side of equation (10) *must* be ranked; i.e., according to the automata on which they *may* depend. A given automaton may actually depend on a subset of the automata in its parameter list.

Now consider an arbitrary permutation of the factors on the left-hand side of (10). We have on the left-hand side,

$$A^{(\sigma_1)}[\mathcal{A}^{(1)},\ldots,\mathcal{A}^{(\sigma_1-1)}]\otimes_g\cdots\otimes_g A^{(\sigma_m)}[\mathcal{A}^{(1)},\ldots,\mathcal{A}^{(\sigma_m-1)}].$$

Indeed equation (13) is just such a permutation. No matter which permutation is chosen, the right-hand side remains essentially identical in the sense that the terms are always ordered from largest to smallest set of *possible* dependencies. The only change results from the manner in which each normal factor is written. Each must have the form

$$I_{\sigma_1:\sigma_{i-1}}\otimes_g A^{(\sigma_i)}[\mathcal{A}^{(1)},\ldots,\mathcal{A}^{(\sigma_i-1)}]\otimes_g I_{\sigma_{i+1}:\sigma_m},$$

with $I_{\sigma_l:\sigma_k}=\prod_{i=l}^{k}n_{\sigma_i}$. Consider, as an example, the following three automata, $\mathcal{A}^{(1)}$ which is completely independent, $\mathcal{A}^{(2)}$ which depends on $\mathcal{A}^{(3)}$ and $\mathcal{A}^{(3)}$ which depends on $\mathcal{A}^{(1)}$. Notice that the possible dependency set of the automaton $\mathcal{A}^{(2)}$ is not only $\mathcal{A}^{(3)}$, but $\mathcal{A}^{(3)}$ and $\mathcal{A}^{(1)}$. We have for example the following left-hand side arrangement:

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

while a different rearrangement gives:

$$\begin{aligned}A^{(3)}[\mathcal{A}^{(1)}]\otimes_g A^{(2)}[\mathcal{A}^{(3)}]\otimes_g A^{(1)} = & \\ & I_{3:3}\otimes_g A^{(2)}[\mathcal{A}^{(3)}]\otimes_g I_{1:1}\\ \times\quad & A^{(3)}[\mathcal{A}^{(1)}]\otimes_g I_{2:1}\\ \times\quad & I_{3:2}\otimes_g A^{(1)}.\end{aligned}$$

On the right-hand side, the normal factor containing $A^{(2)}[\mathcal{A}^{(3)}]\equiv A^{(2)}[\mathcal{A}^{(1)},\mathcal{A}^{(3)}]$ is always first while that containing $A^{(1)}$, the independent term, is always last.

Lemma 5.7 (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*

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

Proof:

To prove this lemma we need to address the entries of a matrix X of order $\prod_{i=1}^{N} n_i$ in different ways. X may be considered as a matrix formed from n_1 square blocks each of size $\prod_{i=2}^{N} n_i$; each block may in turn be viewed as formed from n_2 square blocks each of size $\prod_{i=3}^{N} n_i$ and so on. To refer to an individual element of X we use the notation

$$x_{\{(i_1,\ldots,i_N);(j_1,\ldots,j_N)\}}$$

where $i_m, j_m \in [1,\ldots,n_m]$ and (i_m, j_m) indicates the position of the element in the m-th embedded block. Let σ be a permutation of the integer $[1,2,\ldots,N]$. Then the individual elements of X may also be referenced with respect to this permutation and its corresponding embedded block structure. We have

$$x_{\{(i_1,\ldots,i_N)_\sigma;(j_1,\ldots,j_N)_\sigma\}} \quad where \quad i_m, j_m \in [1,\ldots,n_{\sigma(m)}]$$

Nothing prevents us from mixing these two possibilities. Thus we may refer to the element that belongs to row $(i_1,\ldots,i_N)$ and to column $(j_1,\ldots,j_N)_\sigma$, or vice versa. Let P_σ be the permutation matrix whose elements are given by

$$p_{\{(i_1,\ldots,i_N);(\zeta_1,\ldots,\zeta_N)_\sigma\}} = \begin{cases} 1, & if\ for\ all\ m,\ \zeta_m = i_{\sigma(m)} \\ 0, & otherwise \end{cases}$$

The elements of its transpose, P^T, are given by

$$p^T_{\{(\xi_1,\ldots,\xi_N)_\sigma;(j_1,\ldots,j_N)\}} = \begin{cases} 1, & if\ for\ all\ m,\ \xi_m = j_{\sigma(m)} \\ 0, & otherwise \end{cases}$$

It follows that XP_σ^T permutes the columns of X and $P_\sigma X$ permutes the rows of X. Let $Y = P_\sigma X P_\sigma^T$. Then the elements of Y are given by

$$y_{\{(i_1,\ldots,i_N);(j_1,\ldots,j_N)\}} =$$

$$\sum_{(\zeta_1,\ldots,\zeta_N)=(1,\ldots,1)}^{(n_{\sigma(1)},\ldots,n_{\sigma(N)})} p_{\{(i_1,\ldots,i_N);(\zeta_1,\ldots,\zeta_N)_\sigma\}} \sum_{(\xi_1,\ldots,\xi_2)=(1,\ldots,1)}^{(n_{\sigma(1)},\ldots,n_{\sigma(N)})} x_{\{(\zeta_1,\ldots,\zeta_N)_\sigma;(\xi_1,\ldots,\xi_N)_\sigma\}} p^T_{\{(\xi_1,\ldots,\xi_N)_\sigma;(j_1,\ldots,j_N)\}}$$

which leads directly to

$$y_{\{(i_1,\ldots,i_N);(j_1,\ldots,j_N)\}} = x_{\{(i_{\sigma(1)},\ldots,i_{\sigma(N)})_\sigma;(j_{\sigma(1)},\ldots,j_{\sigma(N)})_\sigma\}}. \tag{14}$$

Notice that, from the definition of a generalized tensor product, the elements of the matrix

$$Y \equiv \bigotimes_{g\ k=1}^{N} A^{(k)}[\mathcal{A}^{(1)},\ldots,\mathcal{A}^{(N)}]$$

are given by

$$y_{\{(i_1,\ldots,i_N);(j_1,\ldots,j_N)\}} = \prod_{k=1}^{N} a^{(k)}_{i_k j_k}(a^{(1)}_{i_1},\ldots,a^{(N)}_{i_n}). \tag{15}$$

Similarly, we may write the elements of

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

as

$$x_{\{(i_{\sigma(1)},\ldots,i_{\sigma(N)})_\sigma;(j_{\sigma(1)},\ldots,j_{\sigma(N)})_\sigma\}} = \prod_{k=1}^{N} a^{(\sigma(k))}_{i_{\sigma(k)} j_{\sigma(k)}}(a^{(1)}_{i_1},\ldots,a^{(N)}_{i_n}).$$

Since $\sigma(k)$ is a bijection of $[1,2,\ldots,N]$, a simple change of variable allows us to write the elements of X as

$$\prod_{k=1}^{N} a^{(k)}_{i_k j_k}(a^{(1)}_{i_1},\ldots,a^{(N)}_{i_n}). \tag{16}$$

The proof of the lemma follows directly from equations (14), (15), and (16). □

Theorem 5.1 (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

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

by the algorithm described below.

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. $x = x'$

Proof: Observe that the inner-most loop (which is in fact implicitly rather than explicitly defined in the algorithm) involves computing the product of a vector z, of length n_i, with a matrix of size $n_i \times n_i$. This requires $(n_i)^2$ multiplications, assuming that the matrices are full. This operation lies within consecutively nested i, k and j loops, Since the k and j loops together involve $\prod_{l=1}^{N} n_l/n_i$ iterations, and the outermost i loop is executed N times, the total operation count is given by

$$\sum_{i=1}^{N} \left(\frac{\prod_{l=1}^{N} n_l}{n_i}\right) n_i^2 \quad = \quad \prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} n_i.$$

□

This complexity result improves upon previous results given in the paper by Fourneau et al., [21]. For example, given a tensor product with two terms, we have $A^{(1)} \otimes_g A^{(2)}[\mathcal{A}^{(1)}]$

with a complexity of $n_1 n_2(n_1 + n_2)$ given by Theorem 5.1, whereas previously using the procedures given in [21] the complexity is $n_1 n_2(n_1 + n_1 n_2)$. Naturally, this generalizes to the case of m terms. Furthermore, it is immediately apparent that when the matrices are not full, but possess a special structure such as tridiagonal, contain only one nonzero row or column, etc., or are sparse, then this may be taken into account and the complexity reduced even further.

This algorithm differs from the previous (nonfunctional) algorithm (Theorem 4.1) in the computation of the state indices at the beginning of the k-loop and in the evaluation of functions. However, this will not affect the order of the complexity of the algorithm. Indeed, once incorporated into an iterative solution method, it is possible that most of the computation involved in generating the state indices could be performed only once during an initialization phase.

What is not immediately apparent from the algorithm as described is the fact that in OTP the normal factors commute and the order in which the innermost multiplication is carried out may be arbitrary. In GTP however this freedom of choice does not exist; the order is prescribed. As indicated by the right hand side of equation (10), each automata $\mathcal{A}^{(i)}$ is represented by a factor of the form

$$I_{1:i-1} \otimes_g A^{(i)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(i-1)}] \otimes_g I_{i+1:m}$$

which *must* be processed before any factor of its arguments $[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(i-1)}]$ is processed.

We would like to point out that although Theorem 5.1 allows us to reorganize the terms of the left-hand side of equation (10), in any way we wish, the advantage of leaving them in the form given by equation (10) is precisely that the computation of the state indices can be moved outside the innermost j-loop of the algorithm. A different rearrangement would require more extensive index computation *within* the innermost j-loop. Note that the order given by the left-hand side of equation (10) is precisely the opposite of that given by its right-hand side.

We now introduce one final lemma that allows us to prove a theorem (Theorem 5.2) 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 and in particular, when the dependencies are cyclic.

Lemma 5.8 (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.

Proof: From equation (5)

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

from which it follows that

$$\ell_k(A) \otimes B[a_k] = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \\ a_{k1}B(a_k) & a_{k2}B(a_k) & \cdots & a_{kn_a}B(a_k) \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0v \end{pmatrix}.$$

Then, algebraically,

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

□

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. Thus $\mathcal{G}$ contains an arc from node i to node j if and only if automaton $\mathcal{A}^{(i)}$ depends on automaton $\mathcal{A}^{(j)}$. Let $\mathcal{T}$ be a *cutset* of the cycles of $\mathcal{G}$, [4]. 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 5.2 (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

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

Proof: Using Lemma 5.7, let us assume, without loss of generality, that the automata are ordered so that the cutset $\mathcal{T}$ contains the automata $\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(t)}$. We have

$$\bigotimes_{g\ i=1}^{N} A^{(i)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(N)}] = \bigotimes_{g\ i=1}^{t} A^{(i)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(N)}] \bigotimes_{g\ i=t+1}^{N} A^{(i)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(N)}]$$

$$= \bigotimes_{g\ i=1}^{t} \left(\sum_{k=1}^{n_i} \ell_k(A^{(i)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(N)}])\right) \bigotimes_{g\ i=t+1}^{N} A^{(i)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(N)}].$$

Applying Lemma 5.2, followed by Lemma 5.8, yields

$$\bigotimes_{g\ i=1}^{N} A^{(i)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(t)}, \mathcal{A}^{(t+1)}, \ldots, \mathcal{A}^{(N)}] =$$

$$\sum_{k_1=1}^{n_1} \cdots \sum_{k_t=1}^{n_t} \bigotimes_{g\ i=1}^{t} \ell_k(A^{(i)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(t)}, \mathcal{A}^{(t+1)}, \ldots, \mathcal{A}^{(N)}]) \bigotimes_{g\ i=t+1}^{N} A^{(i)}[\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(t)}, \mathcal{A}^{(t+1)}, \ldots, \mathcal{A}^{(N)}]$$

$$= \sum_{k_1=1}^{n_1} \cdots \sum_{k_t=1}^{n_t} \bigotimes_{g\ i=1}^{t} \ell_{k_i}(A^{(i)}[a_{k_1}^{(1)}, \ldots, a_{k_t}^{(t)}, \mathcal{A}^{(t+1)}, \ldots, \mathcal{A}^{(N)}]) \bigotimes_{g\ i=t+1}^{N} A^{(i)}[a_{k_1}^{(1)}, \ldots, a_{k_t}^{(t)}, \mathcal{A}^{(t+1)}, \ldots, \mathcal{A}^{(N)}]$$

As the cycles have been removed, applying Theorem 5.1 to each term leads us to conclude that the number of multiplications involved is given by

$$\left(\prod_{i=1}^{t} n_i\right)\left(\prod_{i=1}^{N} n_i\right)\left(\sum_{i=1}^{N} n_i\right).$$

However, because of the row matrices in the set $\mathcal{T}$, this complexity reduces to

$$\left(\prod_{i=1}^{t} n_i\right)\left(\prod_{i=1}^{N} n_i\right)\left(\sum_{i=t+1}^{N} n_i\right)$$

as indicated in the statement of the theorem. □

6 Using the Lemmas

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 5.1 or 5.2. In the

case of SANs in continuous-time, and with no synchronizing transitions, the descriptor is given by

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

and we can apply Theorem 5.1 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 5.1, advantage can be taken of this to reduce the number of multiplications even further.

Consider now what happens when we add synchronizing transitions. The part of the SAN that corresponds to *local* transitions has the same minimal cost as above. As far as the remainder of the SAN is concerned, recall that in Section 3.1, it was mentioned 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, and the second corresponds to an updating of the diagonal elements in the infinitesimal generator to reflect these transitions. More detailed information may be found in [21]. The first of the additional terms may be written as

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

For 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. We have

- The matrices $Q_j^{(i)}$ corresponding to automata that do not participate in the synchronizing event are represented by identity matrices of appropriate dimension.
- We shall use E to denote the matrix belonging to the automaton selected as the one with which the synchronizing event is associated. In a certain sense, this automaton may be considered to be the *owner* of the synchronizing event. E is therefore a matrix of transition rates. For example, if the 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 only the rates of the synchronizing transitions are functional, 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 5.1. This only 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 5.1; otherwise we must resort to Theorem 5.2. Note that if the functional dependency graph is fully connected, Theorem 5.2 offers no savings compared with ordinary multiplication. This shows that Theorem 5.2 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, [20], 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.

7 Numerical Experiments

Our purpose in this section is to provide a measure of the cost of using the SAN methodology when applied to realistic examples. The complexity results presented in Sections 4 and 5 implicitly assume that the matrices are dense. Their purpose is to assist in developing an optimal multiplication procedure for SANs. When the matrices are sparse, the actual operation count will be reduced. In this section we present numerical experiments in which advantage is taken of sparsity, both in the SAN approach and in a competing numerical approach in which the global matrix is generated once and stored in a compact format. We refer to this latter as the *sparse matrix approach* and use the software package MARCA, [27] to develop the models and perform the multiplications. As we have mentioned before, no one approach will work best in all modelling situations, but each has its niche, and we wish to use these experiments to identify and to a certain extent quantify, the advantages of the SAN approach.

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 [26]. 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. We shall see that this can becomes excessive for large models. One of the points that we wish to demonstrate in the experiments of this section is 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, we shall see that 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 the results in Section 5 that allow us to keep multiplication cost to a minimum.

One final point concerning the results that follow: The software package MARCA was used to generate the results concerning the sparse matrix approach while those concerning SAN were obtained from PEPS. The first of these is written in Fortran while the second is in C. Since Fortran generally generates faster executable code, this should be kept in mind when examining the results in the tables below. Except where noted below, all experiments were run on an IBM RS6000-370 under AIX version 3.2.5.

7.1 Results from the Resource Sharing Model

We first consider the resource sharing model. In this model the number of terms in the SAN descriptor is equal to the number of processes, N. Furthermore, 50% of the transitions are functional, which is quite high. Table 1 presents information concerning the size of the state space (both the complete product state space and the reachable state space, n); the number of nonzero elements, nz, in the sparse matrix representation; and various timing results under a variety of model parameter values.

It may be observed that the reachable state space varies according to P (the number of resources that may simultaneously access the resource), from almost nothing to one less than the size of the product state space. Also, the large number of nonzero elements in the last four examples brings us to the limit of what we are able to generate using the sparse matrix approach on the IBM machine. For the sake of completeness, we ran the last four sparse examples on a SUN SPARCstation 4 and tried to compute a multiplicative factor so we could relate the times on one machine to the next. Our experiments with floating point

Model Parameters		State Space		Sparse Matrix	Computation Time (seconds)		
				Number of	SAN	Sparse Matrix	
N	P	Product	Reachable	Nonzeros (nz)	10 iters	Generation	10 iters
12	1	4 096	13	37	2.72	0.30	0.00
12	4	4 096	794	6 362	2.78	0.56	0.03
12	8	4 096	3 797	47 381	2.86	1.94	0.27
12	11	4 096	4 095	53 223	2.90	2.07	0.30
16	1	65 536	17	49	65.54	0.23	0.00
16	4	65 536	2 517	20 949	65.72	1.55	0.12
16	8	65 536	39 203	563 491	66.02	25.93	3.14
16	12	65 536	64 839	1 094 983	66.98	46.42	6.02
16	15	65 536	65 535	1 114 079	67.12	46.74	6.07
20	1	1 048 576	21	61	1 440.29	0.20	0.00
20	4	1 048 576	6 196	52 596	1 446.34	4.65	0.31
20	8	1 048 576	263 950	4 031 310	1 454.77	**238.65**	**71.46**
20	12	1 048 576	910 596	18 114 756	1 459.45	**1 199.71**	**340.52**
20	16	1 048 576	1 047 225	21 972 345	1 461.04	**1 508.20**	**414.58**
20	19	1 048 576	1 048 575	22 020 055	1 462.25	**1 534.38**	**414.72**

Table 1: Resource Sharing Model

multiplications and array indexing together with some overhead seem to indicate that the IBM machine requires 0.76 of the time required by the SUN. The numbers in bold type in Table 1 are 0.76 times the number of seconds obtained on the SUN. At the very least, this allows us to observe trends in the results which is really what we are most concerned with. Memory limitations prevented us from going to larger examples with the sparse matrix approach on either machine. For example, to run a similar set of experiments with the parameter N set to 24 was impossible on either machine.

As for the computation times, it will be observed that, unlike the sparse matrix approach, the time taken by the SAN approach to perform 10 iterations does not vary with P. This is due to the fact that the current implementation in PEPS operates with the entire product state space rather than the reachable state space. We believe that it is possible to improve this situation by detecting in advance when elements will compute to zero and halting their computation early. Although the times for 10 iterations using the sparse approach is always less than that taken by the SAN, the time to generate the matrix must also be included. Thus when we factor in the difference due to programming languages, it appears that the computation time for the SAN approach is not prohibitive, especially given its minimal memory requirements. Finally, we point out that no effort was made to optimize the SAN model. Possibilities that immediately come to mind include taking advantage of symmetry and of grouping many of the small automata into a few large ones. For example, if instead of associating an automaton with each process, we choose to model this system using only four

automata with 32 states each, then the computation time per 10 iterations in the largest example ($N = 20,\ P = 19$), reduces to only 481 seconds.

7.2 The Queueing Network Model

Whereas the previous model contained only functional transitions, the queueing network model contains both functional transitions and synchronizing events. The number of transitions that are functional varies from 37% in the case $C_1 = C_2 = C_3 = 10$ to 62% in the case $C_1 = C_2 = 5,\ C_3 = 20$. The number of functional transitions varies from 44% in all cases when $C_1 = 15$ to 49% when $C_1 = 10$. The numerical results for this model are described in Table 2. All results were obtained on the IBM machine, although the difference in implementation languages still remains.

Model Parameters			State Space		Sparse Matrix	Computation Time (seconds)		
					Number of	SAN	Sparse Matrix	
C_1	C_2	C_3	Product	Reachable	Nonzeros (nz)	10 iters	Generation	10 iters
5	5	10	2 500	1 375	6 905	0.68	0.32	0.05
10	10	10	10 000	5 500	29 710	2.74	0.77	0.20
5	5	20	10 000	5 250	26 855	2.67	0.74	0.18
10	10	20	40 000	21 000	115 610	10.74	2.49	0.77
15	15	20	90 000	47 250	266 265	24.34	5.66	1.76
10	10	30	90 000	46 500	257 510	24.07	5.53	1.69
15	15	30	202 500	104 625	593 115	54.21	12.70	3.86
30	30	30	810 000	418 500	2 427 330	214.99	62.11	16.02
10	10	50	250 000	127 500	709 310	66.90	15.58	4.62
20	20	50	1 000 000	510 000	2 938 220	262.89	84.30	18.94

Table 2: Queueing Network Model

The results are somewhat similar to those obtained with the first example, althought the effect of the synchronizing events is an increased computation time. Nevertheless, even here, they are not so prohibitive as to invalidate the SAN approach.

8 Conclusion

Our purpose in this paper has been to provide a series of theoretical results that permit a reduction in the time required to compute the product of a vector and a sparse SAN descriptor. The SAN methodology requires a minimum of memory for representing a Markovian model of a complex system, but loses somewhat on the computational side of the scale. The results in this paper help to readdress this imbalance. However, more work needs to be done. We have already mentioned that it is likely that elements that compute to zero

can be detected early so that unnecessary computation may be avoided. In addition, many small automata may be grouped together to form larger automata. This will increase the memory requirements of the SAN, but it will still be insignificant compared to the requirements of the sparse matrix approach. As indicated in Section 7, this can lead to substantial reductions in computation time. Furthermore, if it is possible to group automata in such a way as to minimize functional transitions and synchronizing events, this helps even further. As our theorems show, it is especially important to compress the set of automata for which the functional dependency graph is cyclic. Properties like symmetry and lumpability could also be used for this purpose. Our final comment must be that SANs are a viable modelling concept, that they offer great promise for the future and that current difficulties with computation time are being overcome. Our results help in this effort.

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

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

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

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

[28] 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.

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