A comparison of structural formalisms for modeling large Markov models*

M.-Y. Chung[1] G. Ciardo[1] S. Donatelli[2] N. He[1] B. Plateau[3] W. Stewart[4] E. Sulaiman[3] J. Yu[1]

[1]Univ. of California, Riverside [2]Univ. of Torino, Italy [3]ENSIMAG, Grenoble, France [4]North Carolina State University

Abstract

Stochastic automata networks and generalized stochastic Petri nets are the main formalisms used to model complex Markov systems in a structured "Kronecker" approach. We compare them on a suite of examples using two tools, PEPS and SMART.

1 Introduction

Stochastic processes in general, and Markov chains in particular, constitute a modeling paradigm that has broad applicability to many branches of science and engineering. As researchers seek to develop more realistic models it becomes necessary to furnish these models with more detail and this in turn leads to an enormous increase in the size of the state space with its concomitant difficulties in even generating the underlying stochastic matrix.

Currently there are two competing approaches that seek to circumvent these problems by giving researchers the tools to automatically generate, analyze and efficiently solve large scale Markov models. They are the stochastic automata network approach, as implemented in the software PEPS, and an approach based on generalized stochastic Petri nets, as implemented in the SMART software package. Both use Kronecker-inspired data-structures to represent the underlying transition probability matrix of the Markov chain.

Our goal in this paper is to compare and contrast these two approaches in an effort to quantify their strengths and weaknesses, not only from the point of view of computational time and memory requirements, but also and perhaps potentially more importantly, in terms of modelling power and efficiency of representation.

In the following sections, we introduce the two formalisms and their respective software packages. We define a comprehensive suite of test examples taken from the literature of both and conduct a set of numerical experiments. We conclude with some tentative observations on the behaviors of the two approaches, but observe that further and more extensive tests still need to be performed.

*This work was partially supported by the National Science Foundation under grant ACI-0203971.

2 Stochastic Automata Networks

Many systems, whether they be software systems, hardware systems or indeed economic or biological systems, have the characteristics that they may be described in terms of their constituent components. This characteristic provides the rationale for a methodology based on *Stochastic Automata Networks (SANs)* with which to model such systems. SANs have been discussed in the literature for over a decade, [1, 3, 4, 5, 6, 7, 8, 9]. It has been observed that they provide a natural means of describing parallel and distributed systems. Each component of the system is represented by a separate, low-order matrix which describes, probabilistically, how this component makes a transition from one possible state to another. Thus, if a component were completely independent, its representing matrix would define a Markov chain whose evolution in time mimics that of the component. Since components generally do not function independently of each other, information must also be provided which details their interaction. For example, the actions of one component might be synchronized with those of another, or again the behavior of one component might depend on (i.e., be a function of) the internal state of a second component.

2.1 The SAN Formalism

A SAN is a set of automata whose dynamic behavior is governed by a set of *events*. Events are said to be *local* if they provoke a transition in a single automaton, and *synchronizing* if they provoke a transition in more than one automaton. It goes without saying that a single event can generate more than one transition. A transition that results from a synchronizing event is said to be a *synchronized transition*; otherwise it is called a *local transition*. We denote the number of states in automaton i by n_i and by N, the number of automata in the SAN.

The behavior of each automaton, $\mathcal{A}^{(i)}$, for $i = 1, \ldots, N$, is described by a set of square matrices, all of order n_i. The rate at which event transitions occur may be constant or may depend upon the state in which they take place. In this last case they are said to be functional (or state-

IEEE COMPUTER SOCIETY

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

dependent). Synchronized transitions may be functional or non-functional. Functional transitions allow a system to be modeled as a SAN using fewer automata and fewer synchronizing transitions. In other words, if functional transitions cannot be handled by the modelling techniques used, then a given system can be modeled as a SAN only if additional automata are included and these automata are linked to others by means of synchronizing transitions.

In the absence of synchronizing events and functional transitions, the matrices which describe $\mathcal{A}^{(i)}$ reduce to a single infinitesimal generator matrix, $Q^{(i)}$, and the global Markov chain generator may be written as

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

The tensor sum formulation is a direct result of the independence of the automata, and the formulation as a sum of tensor products, a result of the defining property of tensor sums [2]. The probability distribution at any time t of this independent N-dimensional system is known to be

$$\pi(t) = \bigotimes_{i=1}^{N} \pi^{(i)}(t). \tag{2}$$

Now consider the case of SANs which contain synchronizing events but no functional transitions and let us denote by $Q_l^{(i)},\ i = 1, 2, \ldots, N$, the matrix consisting only of the transitions that are local to $\mathcal{A}^{(i)}$. 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 is shown in [7], 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. The first of these two additional tensor products may be thought of as representing the actual synchronizing event and its rates, and the second corresponds to an updating of the diagonal elements in the infinitesimal generator to reflect these transitions. Equation (1) becomes

$$Q = \bigoplus_{i=1}^{N} Q_l^{(i)} + \sum_{e \in \mathcal{E}} \left(\bigotimes_{i=1}^{N} Q_{e^+}^{(i)} + \bigotimes_{i=1}^{N} Q_{e^-}^{(i)} \right). \tag{3}$$

Here $\mathcal{E}$ is the set of synchronizing events. Furthermore, since tensor sums are defined in terms of a 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

$$Q = \sum_{j=1}^{2E+N} \bigotimes_{i=1}^{N} Q_j^{(i)}. \tag{4}$$

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

2.2 Functional Transitions

In real systems, most events do not simply occur of and by themselves. They occur as a result of activities and constraints that, if not global to the system are, at a minimum, dependent on various aspects of it. In other words, they are *functions* of the state of different components in the system. Thus it is natural when building mathematical models of complex systems to include transitions that are functions of the different components. Although this is the most natural manner with which to model functional transitions, it is not the only way. We can always create a model without functional transitions, but this implies a possibly substantial increase in the number of automata and synchronizing transitions needed. Each different function often requires one additional automaton and a synchronizing transition for each automaton associated with the function.

There is yet another reason why we should work with functional transitions. The incorporation of functional transitions into SANs generally leads to a small number of matrices that are relatively full. On the other hand, avoiding functional transitions using additional automata and synchronizing events leads to many very sparse matrices. Given that the SAN approach is especially effective when the matrices are full, (Indeed, for a large number of sparse matrices, the SAN approach is less effective than a general sparse matrix, [4]), it behooves us to work with functional transitions whenever possible.

Now consider the effect of introducing functional transitions into SANs. It should be apparent that the introduction 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, because of possible value changes, the usual tensor operations are no longer valid. Since regular tensor products are unable to handle functional transitions it is necessary to use a *Generalized Tensor Algebra*, (GTA) [4], to overcome this difficulty. In particular, this GTA provides some associativity, commutativity, distributivity and compatibility over multiplication properties that enable the descriptor of a SAN with synchronizing events and functional transitions to be handled with algorithms almost identical to those of SANs with no functional transitions. We use $B[\mathcal{A}]$ to denote a matrix B, associated with the automaton $\mathcal{B}$, which may contain transitions that are a function of the state of the automaton $\mathcal{A}$; $A^{(m)}[\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(m-1)}]$ indicates that the matrix $A^{(m)}$ may contain elements that are a function of one or more of the states of the automata $\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(m-1)}$. The notation $\otimes_g$ denotes a generalized tensor product. Thus

IEEE COMPUTER SOCIETY

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

$A \otimes_g B[\mathcal{A}]$ denotes the generalized tensor product of the matrix A (having only constant entries) with the functional matrix $B[\mathcal{A}]$.

As we just mentioned, the use of functional transitions in SANs implies the need to work with a generalized tensor algebra and this has an associated cost. When the matrices which represent the automata contain only constant values and are full, the cost of performing the operation basic to all iterative solution methods, that of matrix-vector multiply, or in this case, the product of a vector with the SAN descriptor, is given by

$$\rho_N = \prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} n_i, \tag{5}$$

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 matrices are sparse, the cost is even less. In [4] it is shown that for sparse matrices, the complexity is of the order of

$$\sum_{i=1}^{N} z_i \prod_{i=1, i\neq j}^{N} n_i = \prod_{i=1}^{N} n_i \sum_{j=1}^{N} \frac{z_j}{n_j}, \tag{6}$$

where z_i denotes the number of nonzero entries in $Q^{(i)}$. To compare this complexity with a global sparse format, the number of nonzero entries of $\otimes_{i=1}^{N} Q^{(i)}$ is $\prod_{i=1}^{N} z_i$. It is hard in general to compare the two numbers $\prod_{i=1}^{N} z_i$ and $\prod_{i=1}^{N} n_i \sum_{j=1}^{N} \frac{z_j}{n_j}$. Note however that if all $z_i = N^{1/(N-1)} n_i$, both orders of complexity are equal[1]. If all matrices are sparse (e.g., below this bound), the sparse approach is probably better in terms of computation time. This remark is valid for a single tensor product. For a descriptor which is a sum of tensor products and where functions in the matrices $Q^{(i)}$ may evaluate to zero, it is hard to compute, a priori, the order of complexity of each operation.

The savings are due to the fact that once a partial product is formed, it may be used in several places without having to re-do the multiplication, [10]. With functional rates, the elements in the matrices may change according to their context so it is conceivable that this same savings is not always be possible. It was observed in [10] that in the case of two automata $\mathcal{A}$ and $\mathcal{B}$ with matrix representations A and $B[\mathcal{A}]$ respectively, the number of multiplications needed to premultiply $A \otimes B[\mathcal{A}]$ with a vector x remains identical to the nonfunctional case and moreover, 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 (via permutations) 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 (i.e., functions of each other) no computational savings appeared to be possible.

[1]The value $N^{1/(N-1)}$ lies between 1 and 2

2.3 PEPS: a tool to study SANs

PEPS (Performance Evaluation of Parallel Systems) is a software package designed for numerically solving very large Markov chains. Its input language is the SANs we just described. The Kronecker descriptor is exploited internally as a compact storage scheme for the transition matrix of the Markov chain and is manipulated through tensor algebra operators that handle basic vector-matrix multiplication. Among its characteristics, it provides a powerful and modular interface; it generates automatically very large Markov Chain transition matrices; it allows for very compact storage and can generate row-wise compact format (Marca compatible); it offers various numerical strategies to improve the time/space trade-off (grouping and lumping); it includes several numerical iterative methods, with and without preconditioning and it computes performance indices from steady-state probability vectors

More recently, additional upgrades have incorporated a new implementation of the shuffle algorithmn to decrease the cost of evaluating functional transitions and a technique based on automata grouping to decrease the product state space and the complexity of the descriptor formula.

These improvements are all incorporated into the current version, PEPS2003. More information is available from the website: *http://www-id.imag.fr/Logiciels/peps/*.

3 Generalized Stochastic Petri nets

Generalized Stochastic Petri nets (GSPNs) are a stochastic extension of Petri nets, a modelling formalism that was introduced in the early 60s to represent and study concurrency. GSPNs were originally proposed in [11] to support the performance evaluation of concurrent and distributed system. Figure 1(c) shows a very simple case of GSPN: there are two *places* (Use and $Sleep$) and two *transitions* (Get and Put). The *marking* (state) of the system is represented by the number of *tokens* in each place (initially, Use is empty and $Sleep$ has n tokens). The marking is modified by the *firing* of a transition: when a transition t fires, it deletes x tokens from each input place p of t, where x is the annotation on the arc from p to t, and adds y tokens to each output place p' of t, where y is the annotation on the arc from t to p'. In the example considered no annotation is associated with the arcs, meaning that exactly one token is added/removed. To avoid negative markings a transition can fire only if there is a sufficient number of tokens in its input places.

The small net of the example has also a circle-headed arc (inhibitor arc) that allows transition Get to fire only if there are less than r tokens in place Use, but has no effect on the change of state. The particular class of GSPN we use in this paper allows an arc to be annotated with a

IEEE COMPUTER SOCIETY

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

marking-dependent function. This may be very convenient, for example, to remove all the tokens of a place, as is done by transition *Fail* of Figure 1(d).

GSPN were initially defined in a monolithic manner, but, later, classes of compositional GSPNs were introduced that allow a model to be defined in terms of a set of basic component nets and operators, most notably transition superposition and place superposition: interestingly enough, transition superposition corresponds to event synchronization in SAN.

GSPNs assume that there is a delay associated with transitions, and that this delay is defined by an exponentially distributed random variable. Given an initial marking, it is possible to compute the set of states reachable from it (the *reachability set*). since the sojourn time in each state is exponentially distributed, the resulting stochastic process associated with a GSPN is a Continous-Time Markov Chain (CTMC).

The number of states of a GSPN, and therefore of the associated CTMC, can be exponential in the number of net places and number of tokens in places, giving rise to large CTMCs, often too large to be solved with standard solution methods. For these reasons, in the early 90s, the tensor-based approach defined for SAN was "imported" in the GSPN world [12], first to work only for GSPN subclasses and later on to work for any type of finite GSPN model.

The basic idea behind Kronecker-based methods for GSPN is to start with a partition of the set of places into N classes, which in turn leads to the identification of N subnets that interact through transition synchronization. Assuming that it is possible to build the state space of each subnet, which is always possible if the set of reachable states in the original monolithic GSPN is finite [17], it is then possible to build a tensor descriptor of a GSPN that has the same shape as the one defined for SAN by Equation 3, where N is the number of GSPN subnets, $\mathcal{E}$ is the set of synchronizing transitions (transitions that have input places in more than one subnet), $Q_l^{(i)}$ is a matrix that provides the contribution of all transitions of subnet i that are "local" (all input places in subnet i), while $Q_{e^+}^{(i)}$ and $Q_{e^-}^{(i)}$ describe the off-diagonal (positive) and diagonal (negative) contribution due to a transition e belonging to the set of synchronizing transitions $\mathcal{E}$.

The use of marking-dependent arcs and marking-dependent firing rates can greatly simplify the modleing, but it can pose problems in conjunction with the Kronecker encoding. At the moment, we know how to cope with dependencies that satisfy: (i) the marking-dependent expression f labelling an input, output, or inhibitor arc between place p and transition t must refer only to places in the submodel to which p belongs (this includes important cases such as emptying a place or doubling its number of tokens, but, if we want to transfer all the tokens from place p to place q when t fires, this restriction requires that p and q be in the same subnet) and (ii) the marking-dependent expression of the rate of a transition t must be decomposable into the product of N expressions f_i, where f_i can refer only to places in subnet i.

When these conditions are not satisfied, the dependency poses the same problems encountered with functional rates in SAN. Lifting such restrictions without loss of efficiency is a challenging field of research, and the use of the generalized Kronecker operator in SANs could be a starting point.

It should also be noted that research on Kronecker and GSPN has given rise to a number of improvements in the way the state space of a GSPN is generated and stored, to take advantage of symbolic data structures similar to the ones that have made model checking so successful (such as Binary Decision Diagrams), while adapting tensor based numerical solution methods to work efficiently with these compact data structures [19]. These ideas and solution methods are all implemented in the tool SMART, discussed in the following.

3.1 SMART: a tool to study GSPNs

The GSPN experiments in this paper are run on the tool SMART[13]. SMART implements advanced structured approaches for logical [14], numerical [15], and discrete-event simulation [16] studies of discrete-state systems. A system is described as an (extended) GSPN and, by partitioning its places, defines a decomposition into subnets.

Relevant to this work are the following features used in the generation and numerical solution of a Markovian GSPN. *Data structures and algorithms for state space generation*: we employ multi-way decision diagrams (MDDs) paired with the efficient "saturation" algorithm [17]. *Data structures for the infinitesimal generator*: we can employ either a Kronecker descriptor, as in PEPS, or matrix diagrams (MXDs) [18], a Kronecker-inspired data structure analogous to MDDs, which is generally faster but may require more memory. *Numerical solution algorithm*: we employ algorithms traditionally used for the solution of large, sparse linear systems, such as SOR or, as a special case when the relaxation parameter ω equals one, Gauss-Seidel. These have the advantage of requiring the storage of as little as one vector of size $|\mathcal{S}|$ during the iterations.

4 Example suite

In this section we present a suite of examples and we model each of them with both SANs and GSPNs. Our goal is mainly to compare the effectiveness of the two modeling formalisms, although such a comparison necessarily also points out differences between the tools we employ, PEPS and SMART, respectively.

IEEE COMPUTER SOCIETY

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

Resource sharing. Fig. 1(a,b) show the SAN and GSPN of a system where n distinguishable processes access r indistinguishable resources. A processor in $Sleep_i$ can access a resource only if the total number of processors in Use_j, for $j \neq i$, is less than r. Place $UseSum$ is required to keep track of this number and still allow a Kronecker-consistent decomposition of the model into $n = 1$ submodels, one per processor (places $Sleep_i$ and Use_i), and one containing only place $UseSum$. If the identity of the processor does not have to be kept because all processors have the same access rate and the same service rate (rates λ and μ for all transitions Get_i and Put_i, respectively), we can "fold" the n subnets into one, greatly simplifying the model, as shown in Fig. 1(c). To correctly represent the timing behavior, the rate of transitions Get and Put is proportinal to the number of tokens in the input places, $\lambda \cdot \#(Sleep)$ and $\mu \cdot \#(Use)$, respectively. This folded model can only be partitioned into two submodels, one per place. Of course, its state space $\mathcal{S}$ is much smaller, $r + 1$ states (a quantity independent of n), vs. $\sum_{i=0}^{r} \binom{n}{r}$.

Resource sharing with global system failure. Fig. 1(d) shows the GSPN of the system described in the previous section, for the folded case, with the additional assumption that global failures can occur, such that all jobs running are restarted following a repair. The rates for transitions $Fail$ and $Repair$ are the constants ϕ and ρ. The firing of transition $Fail$ must empty places $Sleep$ and Use; this is accomplished using marking-dependent cardinality arcs. Since the function labeling the arc from $Sleep$ (or Use) to $Fail$ depends only on the number of tokens in $Sleep$ (or Use), this does not restrict at all the decomposition into submodels. In our study, we use three submodels, one per place. The state space of this model is quite small, $r + 2$ states.

First available server. Fig. 1(e,f) show the SAN and GSPN of a queueing system where customers arrive with rate λ and each customer polls server 1, 2, ..., n, in order. As soon as the customer finds an idle server, it begins service there. If all servers are busy, the customer leaves. The rates of transitions $Join_i$ are set to the constant λ and those of transitions $Leave_i$ to the constant μ, or μ_i, if each server has its own service rate. The state space $\mathcal{S}$ contains 2^n states, and we decompose the model by assigning each place to a different submodel.

Dining philosophers. Fig. 1(g,h) show the Left-handed and the Right-handed SAN and the Left-handed submodel for the GSPN of the famous "dining philosophers" problem (place $Fork_{i+1 \mod n}$ belongs to the next submodel, and place $ThinkSum$ is explained later). It is well-known that this model has a deadlock state, corresponding to the case where all the philosophers have just picked up their right fork (each subnet has a token in place $HasR_i$). To break the deadlock, we choose to make philosopher 1 behave differently, by picking the left fork first and the right fork second. This modified model has no deadlock and, using arbitrary rates for the various transitions, results in an ergodic Markov chain. If we want to find measures such as the expected number of philosophers that are waiting or eating in steady-state, we can simply find the probability of having a token in place $Wait_i$, for $1 \leq i \leq n$, or $HasR_1$ and $HasL_i$ for $2 \leq i \leq n$, and sum these probabilities. However, if we want to know the probability that all philosophers are thinking, we would need to find the probability of being in a state where $\#(Think_1) = 1$ and $\#(Think_2) = 1$, and so on, up to n. Given the current syntactical limitations in the SMART input language, we can specify this expression only if we know the value of n, but this would require having a different file for each value of n, defeating the purpose of allowing parametric models in SMART. An alternative is to add a single global place $ThinkSum$ which keeps track of the total number of tokens in any $ThinkSum_i$; then, we simply needs to find the probability that this place is empty. The presence of this "implicit" place does not affect the size of the state space $\mathcal{S}$, but it can afffect the efficiency of a structural solution, since each transition $GoEat_i$ and $RelL_i$ (or $RelR_1$) now have a dependency to submodel $n + 1$, which contains only place $ThinkSum$ (submodels 1 through n correspond to one philosopher each).

Queueing network with loss and blocking. Fig. 1(i,j) show the SAN and GSPN of an open queueing system with four bounded queues, 0 through 3, and two classes of customers, A and B. Customers of class A enter queue 0 when they arrive and are served by a dedicated server. When they leave queue 0, they join queue 1, which only serves customers of class A, then they join queue 3, which again serves customers of both classes. Customers of class B follow a similar route, except they join queue 2, also with a dedicated server. instead of queue 1 after leaving queue 0. At queue 3, customers of class A have preemptive priority over those of class B. The bounds on each queue is enforced as follows. For queue 0, we simply disable arrivals when $\#(Q_0) = K_0$. For queue 1, we **block** transition T_{01A} whenever $\#(Q_1) = K_1$, while, for queue 2, an arriving customer is **lost** when $\#(Q_2) = K_2$. For queue 3, we again use blocking of T_{13A} and T_{23B} when $\#(Q_3) = K_3$. The rate of transitions T_{01A} and T_{01B} are $\mu_{0A}/(1 + \#(Q_{3A}))$ and $\mu_{0B}/(1 + \#(Q_{3B}))$, respectively. The rates of all other transitions are constant. The state space contains $(K_0 + 2)(K_0 + 1)K_1K_2(K_3 + 2)(K_3 + 1)/4$ states. In SMART, we use the partition $\{\{Q_0, Q_{0A}, Q_{0B}\}, \{Q_1\}, \{Q_2\}, \{Q_3, Q_{3A}, Q_{3B}\}\}$.

Table 1 reports the results obtained by studying the examples in our suite usinig either PEPS or SMART. For PEPS, we report the time per iteration and the number of iterations using one of the three numerical solution meth-

IEEE COMPUTER SOCIETY

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

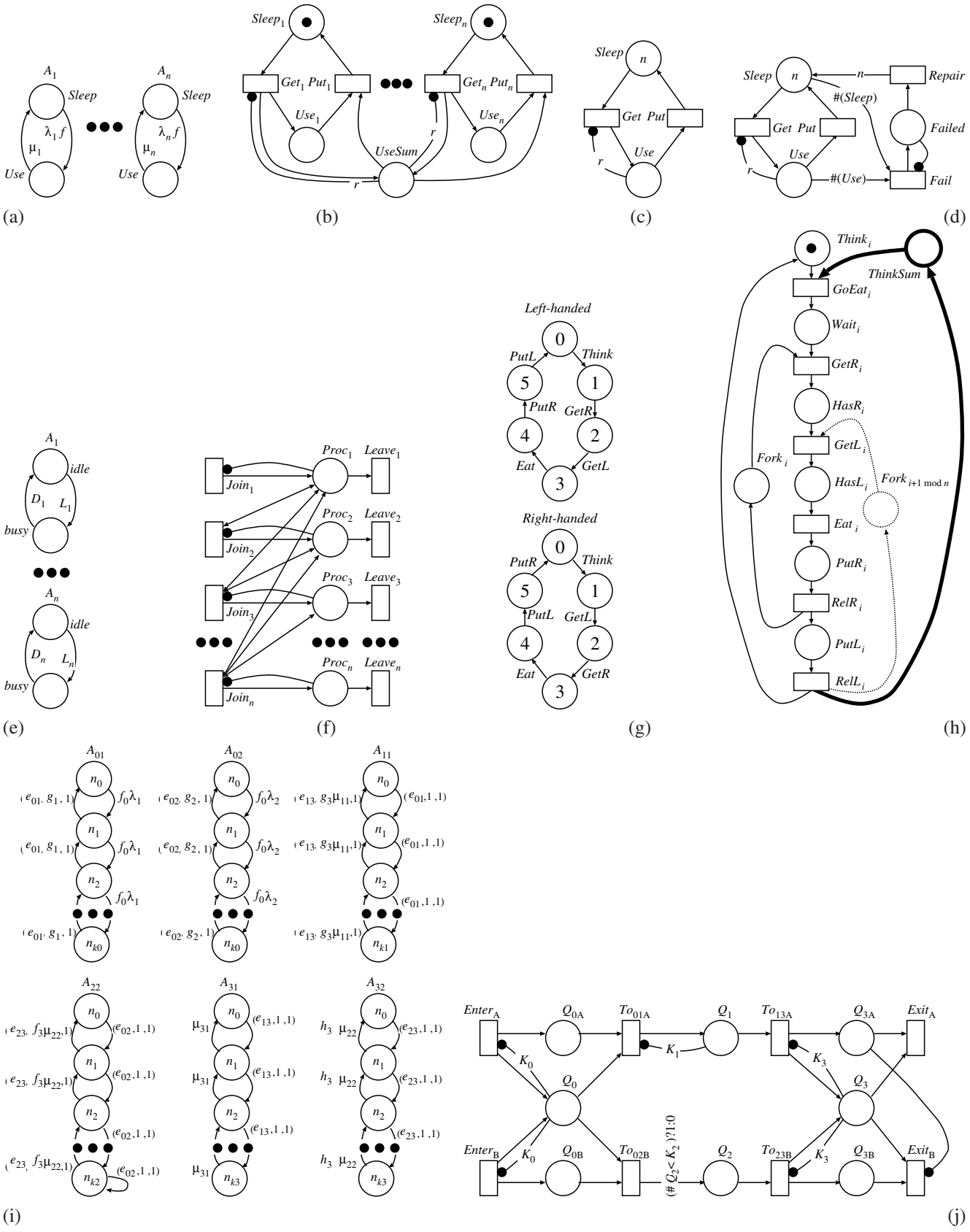

Figure 1: Our models.

IEEE COMPUTER SOCIETY

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

ods available: Power, Arnoldi, GMRES. For SMART, we report the time per iteration and the number of iterations using Gauss-Seidel/SOR in conjunction with either a traditional Kronecker encoding or matrix diagrams; we also report the peak and final memory used to generate the state space using decision diagrams (the runtime is essentially negligible in comparison to that of the numerical solution), and the memory used for Kronecker encoding or matrix diagrams. Finally, we report the number of reachable states ('no model" means that the model has a single automaton, thus PEPS cannot be used; 'n/c' means that the solution does not converge within the default maximum number of iterations).

5 Conclusions

From our study of SAN and GSPN models using PEPS and SMART, it is apparent that the choice of formalism (SAN vs. GSPN), storage representation (Kronecker descriptor vs. Matrix diagram), and numerical solution method (Power, Arnoldi, GMRES, Gauss-Seidel, or SOR) can work better on some system models but not on others.

It is apparent that the GSPN representation tends to be more compact: thanks to the ability of places to contain tokens, a small subnet may correspond to a huge automaton, or even multiple automata. Furthermore, the rules for partitioning a GSPN are quite general: for example, it is possible to partition the examples of Fig. 1(c,d) into two and three subnets, respectively (one per place), while such a decomposition is not natural with SANs, since it does not corresponds to well-defined automata. On the other hand, the power of generalized tensor algebra used in the SAN definition and implemented in PEPS is only at times matched by the restrited marking-dependent behavior allowed in GSPNs. Unless such restrictions are lifted through new research results, one way to make the GSPN formalism as powerful will then be to adopt the same generalized tensor algebra of SANs.

References

[1] 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.

[2] M. Davio. Kronecker products and shuffle algebra. *IEEE Trans. Comp.*, C-30:116–125, Feb. 1981.

[3] S. Donatelli. Superposed Stochastic Automata: a class of stochastic Petri nets with parallel solution and distributed state space. *Perf. Eval.*, 18:21–26, 1993.

[4] P. Fernandes, B. Plateau, and W. J. Stewart. Efficient descriptor-vector multiplication in stochastic automata networks. *Journal of the ACM*, 45(3):381–414, 1998.

[5] 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.

[6] 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.

[7] B. Plateau. On the stochastic structure of parallelism and synchronisation models for distributed algorithms. In *Proc. 1985 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, pages 147–153, Austin, TX, USA, May 1985.

[8] B. Plateau and K. Atif. Stochastic Automata Network for modeling parallel systems. *IEEE Trans. Softw. Eng.*, 17(10):1093–1108, Oct. 1991.

[9] B. Plateau, J.-M. Fourneau, and K. H. Lee. PEPS: a package for solving complex Markov models of parallel systems. In R. Puigjaner, editor, *Proc. 4th Int. Conf. Modelling Techniques and Tools*, pages 341–360, 1988.

[10] W. J. Stewart, K. Atif, and B. Plateau. The numerical solution of stochastic automata networks. *Eur. J. of Oper. Res.*, 86:503–525, 1995.

[11] M. Ajmone Marsan, G. Balbo, and G. Conte, "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems," *ACM Trans. Comp. Syst.*, vol. 2, pp. 93–122, May 1984.

[12] S. Donatelli, "Superposed Stochastic Automata: a class of stochastic Petri nets with parallel solution and distributed state space," *Perf. Eval.*, vol. 18, pp. 21–26, 1993.

[13] G. Ciardo, R. L. Jones, A. S. Miner, and R. Siminiceanu, "Logical and stochastic modeling with SMART," in *Proc. Modelling Techniques and Tools for Computer Performance Evaluation* (P. Kemper

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

PEPS			SMART						State
Time sec iters	Descriptor size KB	PSS/RSS	Kronecker sec iters	MXD sec iters	Peak bytes	Final bytes	Kron bytes	MXD bytes	space states
Resource sharing (unfolded) — processors: $n = 20$ **resources:** $r = 5$									
1.2 6	184	1048576 21700	0.22 11	0.119 11	112560	3932	7396	137832	21700
Resource sharing (unfolded) — processors: $n = 20$ **resources:** $r = 10$									
11.67 12	4833	1048576 616666	8.978 23	3.807 23	202200	6292	9816	2542256	616666
Resource sharing with global system failure (unfolded) — processors: $n = 10$ **resources:** $r = 10$									
0.026 30	19	118098 1025	0.005 23	0.004 23	48092	3528	5272	78144	1025
First available server — servers: $n = 13$									
0.02 46	71	8192 8192	0.025 40	0.023 40	3660	468	2796	45408	8192
Dining philosophers — philosophers: $n = 8$									
2.49 632	1699	1679616 216994	0.886 52	0.577 199	89060	8596	16732	947084	216994
Dining philosophers — philosophers: $n = 9$									
15.98 n/c	7880	10077696 1008100	5.832 58	3.062 235	136508	10952	20948	4137664	1008100
Queueing network with loss and blocking — capacities: $K_0 = 5, K_1 = 3, K_2 = 2, K_3 = 4$									
0.006 230	90	10800 3780	0.002 11033	0.004 40	8376	284	1872	19460	3780
Queueing network with loss and blocking — capacities: $K_0 = K_1 = K_2 = K_3 = 10$									
3.01 n/c	4125	1771561 527076	0.385 n/c	0.679 220	171352	728	7076	2120580	527076

Table 1: Runtime and memory results running PEPS and SMART.

and W. H. Sanders, eds.), LNCS 2794, (Urbana, IL, USA), pp. 78–97, Springer-Verlag, Sept. 2003.

[14] G. Ciardo and R. Siminiceanu, "Structural symbolic CTL model checking of asynchronous systems," in *Computer Aided Verification (CAV'03)* (W. Hunt, Jr. and F. Somenzi, eds.), vol. 2725 of *LNCS*, (Boulder, CO, USA), pp. 40–53, Springer-Verlag, July 2003.

[15] P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper, "Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models," *INFORMS J. Comp.*, vol. 12, no. 3, pp. 203–222, 2000.

[16] G. Ciardo and Y. Lan, "Faster discrete-event simulation through structural caching," in *Proc. Sixth Int. Workshop on Performability Modeling of Computer and Communication Systems (PMCCS-6)*, (Monticello, IL, USA), pp. 11–14, Sept. 2003.

[17] G. Ciardo, R. Marmorstein, and R. Siminiceanu, "Saturation unbound," in *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)* (H. Garavel and J. Hatcliff, eds.), LNCS 2619, (Warsaw, Poland), pp. 379–393, Springer-Verlag, Apr. 2003.

[18] G. Ciardo and A. S. Miner, "A data structure for the efficient Kronecker solution of GSPNs," in *Proc. 8th Int. Workshop on Petri Nets and Performance Models (PNPM'99)* (P. Buchholz, ed.), (Zaragoza, Spain), pp. 22–31, IEEE Comp. Soc. Press, Sept. 1999.

[19] G. Ciardo, "What a structural world," in *Proc. 9th Int. Workshop on Petri Nets and Performance Models (PNPM'01)* (R. German and B. Haverkort, eds.), (Aachen, Germany), pp. 3–16, IEEE Comp. Soc. Press, Sept. 2001. Invited keynote paper.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE