# The Need for and the Advantages of Generalized Tensor Algebra for Kronecker Structured Representations*

Leonardo Brenner, Paulo Fernandes**, and Afonso Sales

PUCRS, Av. Ipiranga, 6681 - 90619-900 - Porto Alegre, Brazil
{lbrenner, paulof, asales}@inf.pucrs.br

**Abstract.** This paper presents the advantages in extending Classical Tensor Algebra (CTA), also known as Kronecker Algebra, to allow the definition of functions, *i.e.*, functional dependencies among its operands. Such extended tensor algebra have been called Generalized Tensor Algebra (GTA). Stochastic Automata Networks (SAN) and Superposed Generalized Stochastic Petri Nets (SGSPN) formalisms use such Kronecker representations. The advantages of GTA do not imply in a reduction or augmentation of application scope, since there is a representation equivalence between SAN, which uses GTA, and SGSPN, which uses only CTA. Two modeling examples are presented in order to draw comparisons between the memory needs and CPU time required for the generation and solution using both formalisms, showing the computational advantages in using GTA.

## 1 Introduction

In the middle of the $19^{th}$ century, the german mathematician Leopold Kronecker proposed a new operation based on *tensors*, a generalization of the matrices in which more than two dimensions could be represented.

However, as far as the authors know, only in the 1970s did computer scientists pay some attention to the Kronecker extension to Linear Algebra. Davio [8] made one of the first studies regarding the *Kronecker* (*tensor*) *product* operation applied to Computer Science. Nearly at the same time, a new operation over tensors made its appearance: *Kronecker* (*tensor*) *sum*. Tensor product and tensor sum operations, as well as their properties compose the *Classical Tensor Algebra* (CTA).

SAN [14] and SGSPN [9] are formalisms that use Tensor Algebra to represent the infinitesimal generator. Comparing these Kronecker representations to the classical approach (a huge sparse matrix), it is obvious that the memory needs are dramatically reduced. This memory savings are quite important to reduce the impact of the classical *state space explosion* problem. Unfortunately, a similar reduction of the time spent to compute solutions is yet to come out. To

* This work was developed in collaboration with HP Brazil R&D.

** Corresponding author. The order of authors is merely alphabetical.

cope with this problem, the research community has been working on numerical techniques to reduce the computational cost in computing exact solutions using mostly iterative methods [10, 5]. Such works achieve some important gains, but the reduction is not as big as the reduction in memory needs.

One of the main goals of our work is to stress some advantages of using *Generalized Tensor Algebra* (GTA), which is an extension to the CTA. In fact, we do believe that GTA can provide a more compact and more manageable representation from a numerical point of view, *i.e.*, models using GTA need less memory and can be solved more rapidly. Our first goal is to show that GTA operators can be viewed as a compact form of describing complex CTA formulas. Therefore, models described with classical Kronecker representations can often be described by GTA operators in a more compact (and efficient) form. This paper provides the intuition that SAN, which uses GTA, has the same application scope of SGSPN, which uses CTA. In fact, such affirmation is based on the formal equivalence proved in [3]. We also show that any SAN model with functions has at least one equivalent representation without functions. Because of that, the use of GTA is not necessary, but it represents some irrefutable computational cost reduction.

Some segments of the research community made a considerable, and fruitful, effort to provide solutions for models with large and complex state spaces [4, 12]. However, such efforts could provide satisfactory solutions for models with a relatively small number of reachable states. We do not want to contest such gains achieved by state space analysis performed using *Multi-Decision Diagrams* (MDD) and *Matrix Diagrams* (MxD) techniques [13]. Nevertheless, we want to show that for models with a large number of reachable states the GTA approach has irrefutable gains.

The next section briefly describes SAN and SGSPN formalisms. Section 3 presents the proof idea for the equivalence between SAN models with functions (needing GTA operators) and SAN models without functions (described only with CTA operators), as well as the representation equivalence between SAN and SGSPN formalisms. In Section 4, we present two examples modeled using SAN and SGSPN. Section 5 draws comparisons between the memory needs and CPU time required for the generation and solution of SAN and SGSPN models. Finally, the conclusion stresses the need for and the advantages of GTA operators.

## 2 Modeling Formalisms

Many formalisms can benefit from a structured tensor (Kronecker) representation, but in this work we focus in two formalisms that explicitly define their tensor representations: Stochastic Automata Networks (SAN) and Superposed Generalized Stochastic Petri Nets (SGSPN). It is important to notice that the use of Kronecker representations is not limited to SAN and SGSPN, but can also be employed in other structured formalisms, *e.g.*, Process Algebras [11] and Stochastic Activity Networks [15].

### 2.1 SAN - Stochastic Automata Networks

The basic idea of SAN is to represent a whole system by a collection of subsystems with an independent behavior (*local transitions*) and occasional interdependencies (*functional rates* and *synchronizing events*). Each subsystem is described as a stochastic automaton, *i.e.*, an automaton in which the transitions are labeled with probabilistic and timing information. Hence, one can build a continuous-time stochastic process[1] related to SAN. The state of a SAN model is called the *global state* and it is defined by the cartesian product of the *local states* of all automata.

There are two types of events that change the global state of a SAN model: *local events* and *synchronizing events*. Local events change the global state passing from a global state to another that differs only by one local state. On the other hand, synchronizing events can change simultaneously more than one local state, *i.e.*, two or more automata can change their local states simultaneously. In other words, the occurrence of a synchronizing event *forces* all concerned automata to fire a transition corresponding to this event. Thus, local events can be viewed as a particular case of synchronizing events that concerns only one automaton.

Each event is represented by an *identifier* and a *rate* of occurrence, which can be a constant value or a function of the state of other automata. Each transition may be fired as result of the occurrence of any number of events. In general, non-determinism among possible different events is dealt according to Markovian behavior, *i.e.*, any of the events may occur and their occurrence rates define how often each on of them will occur. However, from a given local state, if the occurrence of a given event can lead to more than one state, then an additional *routing probability* must be informed[2].

Fig. 1 presents an example of a SAN model with two automata, four local events, one synchronizing event, and one functional rate.

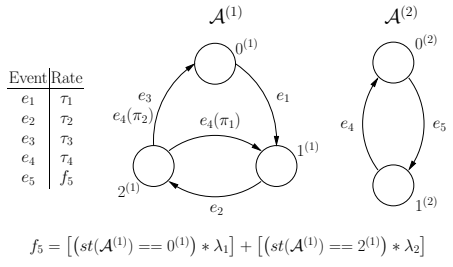



**Fig. 1.** An example of a SAN model

[1] In the context of this paper, only continuous-time SAN will be considered, although discrete-time SAN can also be employed without loss of generality.

[2] The absence of routing probability is tolerated if only one transition can be fired by an event from a given local state.

In this model, event $e_5$ has a functional rate defined by function $f_5$. The firing of the transition from state $0^{(2)}$ to $1^{(2)}$ occurs with rate $\lambda_1$, if $\mathcal{A}^{(1)}$ is in state $0^{(1)}$, or $\lambda_2$ if $\mathcal{A}^{(1)}$ is in state $2^{(1)}$. If $\mathcal{A}^{(1)}$ is in state $1^{(1)}$, the transition from state $0^{(2)}$ to $1^{(2)}$ does not occur (rate equal to 0).

The use of functional rates is not limited to event rates. In fact, routing probabilities also may be expressed as functions. The use of functions is a powerful primitive of the SAN formalism, since it allows to describe very complex behaviors in a very compact format. The computational costs to handle functional rates has decreased significantly with the developments of numerical solutions for SAN models, *e.g.*, the algorithms for generalized tensor products [1, 10].

### 2.2 SGSPN - Superposed Generalized Stochastic Petri Nets

Stochastic Petri Nets (SPN) formalism represents complex state machines (automata) in a compact form, using places connected to transitions and transitions connected to places. Places (represented as circles) may contain tokens. The firing of a transition removes tokens from input places connected to this transition and adds tokens to output places to which this transition is connected.

An exponentially distributed random firing time is associated with each timed transition (represented as rectangles). A transition may have associated a necessary, but not sufficient, condition (called *guard*) to its firing. A guard is a function which depends on the number of tokens in places. Generalized Stochastic Petri Nets (GSPN) formalism is a generalization from SPN formalism, in which immediate transitions (represented as bars), fired in zero time, may appear.

Superposed Generalized Stochastic Petri Nets (SGSPN) is a superposition of GSPN, *i.e.*, a SGSPN is defined as a set of GSPN synchronized by a common subset of transitions. In this paper, we refer to SGSPN, due to its Kronecker explicit representation. However a more general concept of partitions (choices of component $\mathcal{GSPN}$) can be considered without any loss of generality [4, 7].

Fig. 2 presents a SGSPN model equivalent to Fig. 1. This SGSPN model has two $\mathcal{GSPN}$ components, five local transitions, two synchronized transitions, and two guards.

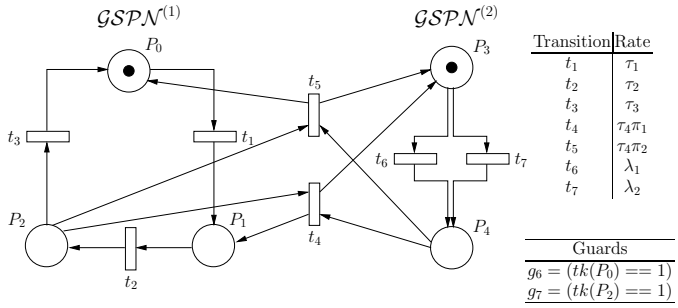



**Fig. 2.** An example of a SGSPN model

Component $\mathcal{GSPN}^{(1)}$ has three places ($P_0$, $P_1$ and $P_2$), whereas component $\mathcal{GSPN}^{(2)}$ has two places ($P_3$ and $P_4$). Both components are synchronized by transitions $t_4$ and $t_5$, and transitions $t_6$ and $t_7$ have guards associated with their firings.

## 3 Representation Equivalence

In this section, we show informally the representation equivalence between SAN and SGSPN formalisms. We understand by representation equivalence between two models, the verification of the same stochastic behavior in both descriptions. For example, if two models have the same underlying Markov chain, we can say they are equivalents. A reader interested in a formal proof of such equivalence between SAN and SGSPN can consult [3]. Note that in this paper, we just indicate the proof idea to this equivalence showing: the equivalence between SAN (GTA) and SAN (CTA) models (Section 3.1); and the equivalence between SAN (CTA) and SGSPN formalisms (Section 3.2). Such equivalence is a shortcut to the quite obvious indirect equivalence, since all formalisms have an equivalence with CTMC. Fig. 3 presents a diagram showing the equivalence levels between the formalisms, detailing the more elaborated step (the equivalence between SAN (CTA) and SGSPN).

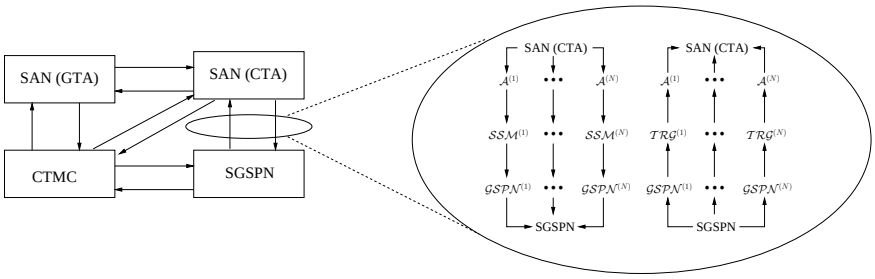



**Fig. 3.** Equivalence diagram between SAN and SGSPN formalisms

### 3.1 SAN (GTA) ↔ SAN (CTA)

In terms of infinitesimal generator description, the use of GTA operations is the main difference between SAN and SGSPN formalisms. In fact, GTA operators in the Markovian descriptor are used to represent the functional rates (or functional probabilities)[3].

[3] Synchronizing events include new terms in the Markovian descriptor, but as long as there is no functional elements, they can be described using classical tensor products.

Our first step is to show that any SAN model needing the use of GTA operators has at least one equivalent model that can be described only using CTA operators, *i.e.*, any SAN model with functional rates (or functional probabilities) can be represented by a SAN model with only constant rates, and vice-versa.

The equivalence between SAN (GTA) and SAN (CTA) models is based on the replacement of functions by synchronizing events. In fact, each event with a functional rate (or functional probability) may be replaced by as many synchronizing events as its possible evaluations.

The equivalence between SAN (CTA) and SAN (GTA) models is trivial, since a SAN (CTA) model may be viewed as a SAN (GTA) model, where all functional rates and functional probabilities are constants.

### 3.2 SAN $\leftrightarrow$ SGSPN

Since any SAN (GTA) model has at least one equivalent SAN (CTA) model, it is possible to show the equivalence between SAN and SGSPN formalisms. The detail in Fig. 3 represents the steps to go from a SAN model not using functions to a SGSPN model, and vice-versa. A SAN model has $N$ automata $\mathcal{A}^{(i)}$, where $i \in [1..N]$. Each automaton $\mathcal{A}^{(i)}$ has an equivalent *Stochastic State Machine* $\mathcal{SSM}^{(i)}$ in which the places and transitions ($\mathcal{SSM}^{(i)}$) correspond to the states and events (automaton $\mathcal{A}^{(i)}$) respectively. Each $\mathcal{SSM}^{(i)}$ may be viewed as a component $\mathcal{GSPN}^{(i)}$ which has only timed transitions. Thus, the SGSPN model equivalent to the SAN model is composed of all those $\mathcal{GSPN}^{(i)}$ components ($i \in [1..N]$).

Analogously, each component $\mathcal{GSPN}^{(i)}$ ($i \in [1..N]$) composing a SGSPN has a *Tangible Reachability Graph* $\mathcal{TRG}^{(i)}$. Each $\mathcal{TRG}^{(i)}$ has an equivalent automaton $\mathcal{A}^{(i)}$ in which the states and events ($\mathcal{A}^{(i)}$) correspond to the nodes and arcs ($\mathcal{TRG}^{(i)}$) respectively.

## 4 Modeling Examples

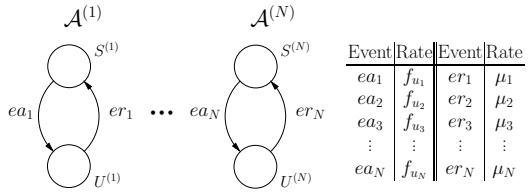
We now present two modeling examples. The first one, modeled by SAN (CTA and GTA) and SGSPN, describes a Resource Sharing model. The second one shows a SAN (CTA and GTA) and SGSPN model of an Alternate Service Pattern.

It is important to notice that SAN (CTA and GTA) and SGSPN models presented in this section, even though equivalents, were not obtained using the conversion steps presented in Fig. 3. In fact, the equivalence diagram is used to prove the existence of a SGSPN model equivalent to each and every SAN model, and vice-versa. However, the models obtained in such way are often not compact or readable ones. An human-made conversion may provide better (more compact) models. The models presented in this section were converted using our best knowledge on the formalisms. Therefore, a more skilled modeler may develop even better models.

### 4.1 RS - Resource Sharing Example

Fig. 4 shows SAN (GTA) model of a *Resource Sharing* system. In this model, there are $N$ processes sharing $R$ resources. Each process is represented by an automaton $\mathcal{A}^{(i)}$, which has two states: $S^{(i)}$ (*sleeping*) and $U^{(i)}$ (*using*). Each local event $ea_i$ has a functional rate $f_{u_i}$ which controls the access to available resources.

Thus, local events $ea_i$ only happen when there are available resources, *i.e.*, the number of automata using resources is less than $R$ ($nb\ [\mathcal{A}^{(1)}..\mathcal{A}^{(N)}]\ U < R$).

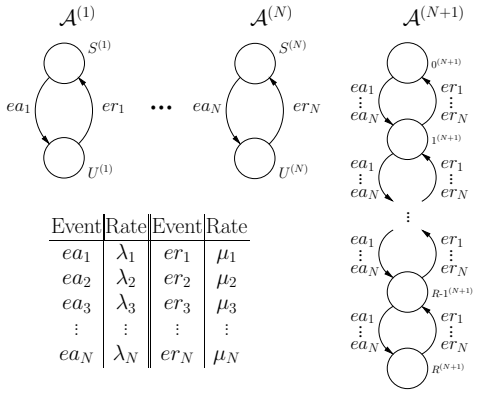



| Event | Rate | Event | Rate |
|---|---|---|---|
| $ea_1$ | $f_{u_1}$ | $er_1$ | $\mu_1$ |
| $ea_2$ | $f_{u_2}$ | $er_2$ | $\mu_2$ |
| $ea_3$ | $f_{u_3}$ | $er_3$ | $\mu_3$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $ea_N$ | $f_{u_N}$ | $er_N$ | $\mu_N$ |

$$f_{u_i} = (nb\ [\mathcal{A}^{(1)}..\mathcal{A}^{(N)}]\ U < R) * \lambda_i$$

**Fig. 4.** RS - SAN (GTA) model

Fig. 5 represents a SAN (CTA) model equivalent to Fig. 4. In this model, a resource pool is represented by automaton $\mathcal{A}^{(N+1)}$ and it has $R+1$ states indicating the number of resources in use. Events $ea_i$ and $er_i$ are synchronizing events between $i^{th}$ process and the resource pool ($\mathcal{A}^{(N+1)}$).




| Event | Rate | Event | Rate |
|---|---|---|---|
| $ea_1$ | $\lambda_1$ | $er_1$ | $\mu_1$ |
| $ea_2$ | $\lambda_2$ | $er_2$ | $\mu_2$ |
| $ea_3$ | $\lambda_3$ | $er_3$ | $\mu_3$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $ea_N$ | $\lambda_N$ | $er_N$ | $\mu_N$ |

**Fig. 5.** RS - SAN (CTA) model

Fig. 6 presents an equivalent SGSPN model. Each process ($\mathcal{GSPN}^{(i=1..N)}$) has two places: $S_i$ (*sleeping*) and $U_i$ (*using*). In component $\mathcal{GSPN}^{(N+1)}$, the tokens in place $RS$ represent the number of available resources, whereas the number of using resources is represented by the tokens in place $RU$. Transition $ta_i$ and $tr_i$ are synchronized transitions between process $i$ and the resource pool. Synchronized transitions $ta_i$ and $tr_i$ have the same rates used by synchronizing events $ea_i$ and $er_i$ of model SAN (CTA) (Fig. 5).

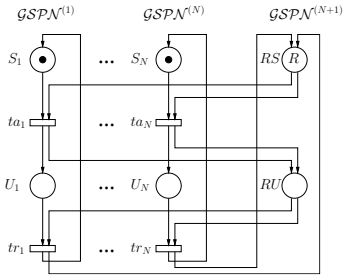



**Fig. 6.** RS - SGSPN model

### 4.2 ASP - Alternate Service Pattern Example

This example describes a small open network composed of four queues ($Q_1$, $Q_2$, $Q_3$ and $Q_4$) with finite capacities $K_1$, $K_2$, $K_3$ and $K_4$ respectively. In the routing pattern of customers for this model, the customers arrive in $Q1$ and $Q_2$ with constant rates $\lambda_1$ and $\lambda_2$ respectively. Customers may leave from $Q_1$ to $Q_3$, if and only if there is room in that queue (blocking behavior), whereas customers may leave from $Q_2$ to $Q_3$ whether there is room or leave the model otherwise (loss behavior). Customers may also leave from $Q_3$ to $Q_4$ with blocking behavior.

While $Q_1$, $Q_2$ and $Q_4$ have standard (single) service behavior, *i.e.*, a same average service rate for all customers ($\mu_1$, $\mu_2$ and $\mu_4$ respectively), queue $Q_3$ has an *Alternate Service Pattern* behavior. The service rate for this queue varies according to $P$ different service patterns ($\mu_{31}, \ldots, \mu_{3P}$). $Q_3$ can exchange its service pattern simultaneously with the end of service of a customer. Therefore, when a customer is served by service pattern $P_i$, $Q_3$ can remain serving the next customer in the same pattern with probability $\pi_{ii}$, or it can alternate to a different service pattern $P_j$, with probability $\pi_{ij}$ (for all service patterns $P_i$: $\sum_{j=1}^{P} \pi_{ij} = 1$).

The SAN model for this example is composed of one automaton to each single-service pattern queue ($Q_1$, $Q_2$ and $Q_4$) and two automata for the alternate service pattern queue ($Q_3$). Let us call $\mathcal{A}^{(i)}$ ($i = 1, \ldots, 4$) the automaton representing the number of customers in $Q_i$, and $\mathcal{A}^{(5)}$ the automaton representing the current service pattern in $Q_3$. Local events $e_1$ and $e_2$ represent the arrival

in queues $Q_1$ and $Q_2$ respectively, and local event $e_4$ represents the departure from $Q_4$. Synchronizing events $e_{13}$ and $e_{34}$ represent the routings between queues $Q_1$ to $Q_3$ and $Q_3$ to $Q_4$ respectively, and synchronizing event $e_{23}$ represents both the routing from $Q_2$ to $Q_3$, and the departure from $Q_2$ due to lack of room in $Q_3$ (loss). All events, besides $e_{34}$, have constant rates according to the model definition. The functional rate of event $e_{34}$ is defined by the function $f_{34}$ and it depends on the state of automaton $\mathcal{A}^{(5)}$.

Fig. 7 presents the SAN (GTA) model for this example with $P = 2$, the extension to a higher number of service patterns will correspond to the addition of more local states to automaton $\mathcal{A}^{(5)}$, which will always have $P$ local states. Note that event $e_{34}$ obviously synchronizes automata $\mathcal{A}^{(3)}$ and $\mathcal{A}^{(4)}$, but it also synchronizes automaton $\mathcal{A}^{(5)}$ due to the (possible) exchange of service pattern. The use of probabilities to event $e_{34}$ represents the stochastic distribution of all the possible exchanges between the service patterns.

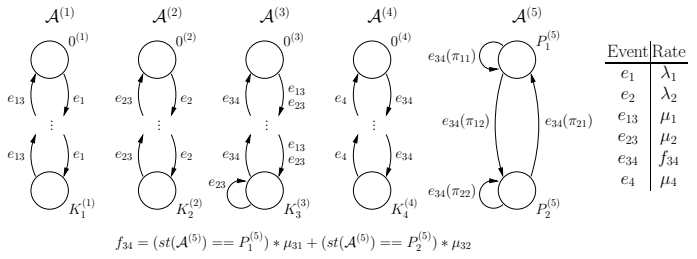



**Fig. 7.** ASP model using SAN (GTA)

Fig. 8 presents an equivalent SAN (CTA) model. Note that there are two synchronizing events ($e_{34(1)}$ and $e_{34(2)}$) to model the exchange between the possible service patterns. Events $e_{34(1)}$ and $e_{34(2)}$ have constant rates equal to $\mu_{31}$ and $\mu_{32}$ respectively. In fact, a SAN (CTA) model for an example with $P$ service patterns will have $P$ synchronizing events $e_{34(1)}, e_{34(2)}, \ldots, e_{34(P)}$ with constant rates $\mu_{31}, \mu_{32}, \ldots, \mu_{3P}$ respectively.

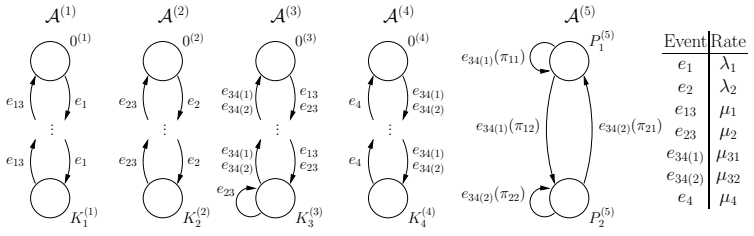
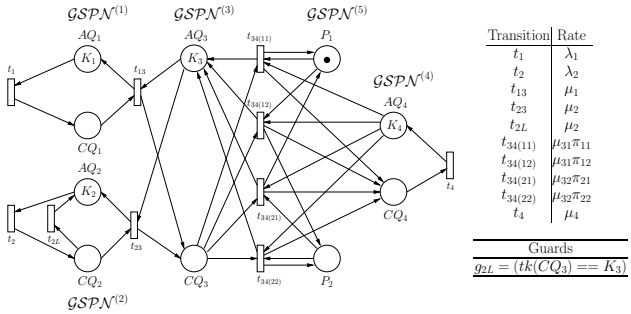



**Fig. 8.** ASP model using SAN (CTA)

The equivalent SGSPN model is presented in Fig. 9 (with $P = 2$). This model defines a pair of places to represent each queue number of customers, one place ($CQ_i$) to represent the number of customers currently in the queue $Q_i$, and another place ($AQ_i$) to represent the available room in this queue. Therefore, the number of tokens in each pair of places $CQ_i$ and $AQ_i$ must always sum the capacity ($K_i$) of queue $Q_i$. Places $P_1$ and $P_2$ indicate which service pattern is being used in the $Q_3$. Component $\mathcal{GSPN}^{(i)}$ $(i = 1, \ldots, 4)$ is composed of places $AQ_i$ and $CQ_i$, which represents the $Q_i$. Component $\mathcal{GSPN}^{(5)}$ is composed of places $P_1$ and $P_2$ (considering $P = 2$), which represents the set of possible service patterns.




| Transition | Rate |
|---|---|
| $t_1$ | $\lambda_1$ |
| $t_2$ | $\lambda_2$ |
| $t_{13}$ | $\mu_1$ |
| $t_{23}$ | $\mu_2$ |
| $t_{2L}$ | $\mu_2$ |
| $t_{34(11)}$ | $\mu_{31}\pi_{11}$ |
| $t_{34(12)}$ | $\mu_{31}\pi_{12}$ |
| $t_{34(21)}$ | $\mu_{32}\pi_{21}$ |
| $t_{34(22)}$ | $\mu_{32}\pi_{22}$ |
| $t_4$ | $\mu_4$ |

| Guards |
|---|
| $g_{2L} = (tk(CQ_3) == K_3)$ |

**Fig. 9.** ASP model using SGSPN

The most significant differences between the events of the SAN model and the transitions of the SGSPN model concern the end of service in $Q_2$ and in $Q_3$. The end of service in $Q_2$ may lead to the routing from $Q_2$ to $Q_3$ ($t_{23}$) which occurs only when $Q_3$ has room for another customer, or to the departure (loss) of $Q_2$ ($t_{2L}$) which occurs when $Q_3$ is full. The end of service in $Q_3$ always represents the routing of one customer from $Q_3$ to $Q_4$, but it may occur simultaneously to the change of service pattern. Therefore, $P^2$ transitions are used to represent this routing considering all possible changes and permaneces in the service pattern. Generically, transition $t_{34(ij)}$ represents the end of service in $Q_3$ with service pattern $P_i$ and changing to service pattern to $P_j$ (when $i \neq j$) or staying in the same service pattern (when $i = j$).

## 5 Numerical Results

Table 1 presents the numerical results for both examples solved using *Power* method. These results were obtained on a 2.8 GHz Pentium IV Xeon under Linux operating system with 2 GBytes of memory. To each example, we solve the SAN (GTA) and SAN (CTA) models using PEPS software tool [2]. The SGSPN model was solved with SMART software tool [6] using MxD technique

with Kronecker storage (**MDK**) and generation of a sparse matrix (**Sparse**). The RS models were solved with 24 processes and 3, 9, 15 and 21 resources. The ASP model were solved with queue capacities equals to 30, 30, 60 and 60 respectively, and 2, 3, 4 and 5 service patterns. The table columns indicate the **size** of infinitesimal generator (in MBytes), CPU time to the generation of the infinitesimal generator (**gen.**), and the CPU time to perform one single power iteration (**iter.**) in seconds.

Note that the number of states and transitions of the underlying Markov chain is not relevant to the size of the models, since the solution is performed only over the tensor representation. In addition, the number of iterations, which depends on required precision and chosen rates, is not relevant to estimate the total solution time.

| RS - Resource Sharing ($N = 24$) | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | | 3 | | | 9 | | | 15 | | | 21 | | |
| | | size | gen. | iter. | size | gen. | iter. | size | gen. | iter. | size | gen. | iter. |
| SAN | CTA | 512.55MB | 8238.8$s$ | 172.2$s$ | - | - | - | - | - | - | - | - | - |
| | GTA | 128.02MB | 140.7$s$ | 214.2$s$ | 128.02MB | 140.3$s$ | 214.8$s$ | 128.02MB | 140.5$s$ | 214.6$s$ | 128.02MB | 140.6$s$ | 214.3$s$ |
| SGSPN | MDK | 0.05MB | 0.2$s$ | $\approx 0.0s$ | 9.92MB | 217.4$s$ | 36.6$s$ | 59.23MB | 1400.0$s$ | 242.0$s$ | 64.05MB | 1679.5$s$ | 278.7$s$ |
| | Sparse | 0.16MB | 0.5$s$ | $\approx 0.0s$ | - | - | - | - | - | - | - | - | - |

| ASP - Alternate Service Pattern ($K_1 = 30$; $K_2 = 30$; $K_3 = 60$; $K_4 = 60$) | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P | | 2 | | | 3 | | | 4 | | | 5 | | |
| | | size | gen. | iter. | size | gen. | iter. | size | gen. | iter. | size | gen. | iter. |
| SAN | CTA | 54.58MB | 38.8$s$ | 2.4$s$ | 81.86MB | 66.1$s$ | 4.4$s$ | 109.14MB | 94.5$s$ | 6.5$s$ | 136.42MB | 130.5$s$ | 9.3$s$ |
| | GTA | 54.54MB | 35.5$s$ | 2.1$s$ | 81.86MB | 53.7$s$ | 3.4$s$ | 109.14MB | 74.9$s$ | 4.7$s$ | 136.42MB | 96.7$s$ | 6.2$s$ |
| SGSPN | MDK | 27.30MB | 69.4$s$ | 19.6$s$ | 40.95MB | 97.4$s$ | 30.4$s$ | 54.60MB | 152.4$s$ | 50.6$s$ | 68.24MB | 196.4$s$ | 66.4$s$ |
| | Sparse | 396.88MB | 677.2$s$ | 1.1$s$ | 674.50MB | 1,299.5$s$ | 1.7$s$ | - | - | - | - | - | - |

**Table 1.** Computational Costs

A clear conclusion from the memory results (size) in both examples is the very optimized storage achieved by SGSPN-MDK technique. Obviously, the SAN formalism could benefit from such efficient technique to compute and store reachable state space [13, 4]. This characteristic of the SGSPN approach explains the quite impressive results that SMART has in model checking applications. However, it is a common mistake in some segments of the research community to generalize the benefits of SGSPN approach to compute stationary solution using power method (or any other iterative method based on vector-generator multiplications).

The results for the RS example shows clearly, how fast is the solution using SGSPN approach for models with a relatively small number of reachable states ($R = 3$ and $R = 9$). The models with few resources can be very efficiently generated by the powerful MxD technique and the resulting generator is a quite efficient Kronecker structure (for SGSPN-MDK) or a quite small sparse matrix (for SGSPN-Sparse). However, for larger models ($R = 15$ and $R = 21$), *i.e.*, models with a large number of reachable states, the memory needs for sparse solution becomes prohibitive, and even the solution (iteration time) is faster for SAN (GTA) than for SGSPN-MDK representation.

The results for the ASP example shows an irrefutable advantage for the SAN approach, which is potentiated by the use of functional rates (GTA). For ASP models, we can observe consistent gains in both generation and solution times when compared to the SGSPN solution using MDK. Obviously, the solution times using SGSPN-Sparse has better results, but it cannot be used for very large models ($P = 4$ and $P = 5$) due to the huge amount of memory needed to handle the generator matrix. Additionally, for models that converge in a small number of iterations (*e.g.* 500) the overall SAN (GTA) solution (generation plus iterations) could be faster than SGSPN sparse solution.

## 6 Conclusion

Our purpose in this paper is not to suggest that a formalism is more suitable than other. As presented in Section 3, SAN and SGSPN formalisms have representation equivalence. Legibility and description facility of a model described by SAN and SGSPN formalisms is personal and extremely dependent on the system. In the ASP example, it is possible to directly observe the interaction relation among queues of SGSPN model (synchronized transitions). On the other hand, this interaction occurs by synchronizing events in SAN model, which has no *visual linking* among them, *i.e.*, there is no *arc* linking the automata to describe the routing of customers from a queue to another. However, in the RS example, the similar interaction among the components may be easily described by SAN formalism, since it allows the concept of functional elements.

In a numerical point of view, the concept of synchronizing events and functional elements in SAN models shows a clear advantage for complex models as ASP. Basically, a SAN model allows to use a small number of synchronizing primitives (synchronizing events in SAN and synchronized transitions in SGSPN). Thus, it is possible to say that the use of functional elements (number of synchronizing primitives reduced) to some SAN models can allow a significantly smaller computational cost than the equivalent SGSPN model. This phenomenon can be observed comparing SAN (GTA) to SAN (CTA) results, but also the SGSPN-MDK results confirm the GTA advantages.

Finally, we may summarise our contribution saying that the use of functions, and consequently the GTA, is not really a "need" since there is an equivalence of formalisms, but in some cases it represents, from a computational cost point of view, some irrefutable "advantages".

## References


1. A. Benoit, L. Brenner, P. Fernandes, and B. Plateau. Aggregation of Stochastic Automata Networks with replicas. *Linear Algebra and its Applications*, 2004 (In Press).
2. A. Benoit, L. Brenner, P. Fernandes, B. Plateau, and W. J. Stewart. The PEPS Software Tool. In Peter Kemper and William H. Sanders, editors, *Computer Performance Evaluation / TOOLS 2003*, volume 2794 of *Lecture Notes in Computer Science*, pages 98–115, Urbana, IL, USA, 2003. Springer-Verlag.

3. L. Brenner, P. Fernandes, and A. Sales. Why you should care about *Generalized Tensor Algebra*. Technical Report TR 037, PUCRS, Porto Alegre, 2003. http://www.inf.pucrs.br/tr/tr037.pdf.
4. P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper. Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models. *INFORMS Journal on Computing*, 13(3):203–222, 2000.
5. P. Buchholz and T. Dayar. Block SOR for Kronecker structured representations. In *Proceedings of the 2003 International Conference on the Numerical Solution of Markov Chains*, pages 121–143, Urbana, IL, USA, 2003.
6. G. Ciardo, R. L. Jones, A. S. Miner, and R. Siminiceanu. Logical and stochastic modeling with SMART. In Peter Kemper and William H. Sanders, editors, *Computer Performance Evaluation / TOOLS 2003*, volume 2794 of *Lecture Notes in Computer Science*, pages 78–97, Urbana, IL, USA, 2003. Springer-Verlag.
7. G. Ciardo and K. S. Trivedi. A Decomposition Approach for Stochastic Petri Nets Models. In *Proceedings of the* $4^{th}$ *International Workshop Petri Nets and Performance Models*, pages 74–83, Melbourne, Australia, December 1991. IEEE Computer Society.
8. M. Davio. Kronecker Products and Shuffle Algebra. *IEEE Transactions on Computers*, C-30(2):116–125, 1981.
9. S. Donatelli. Superposed generalized stochastic Petri nets: definition and efficient solution. In *Proceedings of the* $15^{th}$ *International Conference on Applications and Theory of Petri Nets*, pages 258–277, Springer-Verlag, Berlin Heidelberg, 1994. R. Valette.
10. 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.
11. S. Gilmore, J. Hillston, L. Kloul, and M. Ribaudo. PEPA nets: a structured performance modelling formalism. *Performance Evaluation*, 54(2):79–104, 2003.
12. P. Kemper. Numerical Analysis of Superposed GSPNs. *IEEE Transactions on Software Engineering*, 22(9):615–628, 1996.
13. A. S. Miner, G. Ciardo, and S. Donatelli. Using the exact state space of a Markov model to compute approximate stationary measures. In *Proceedings of the 2000 ACM SIGMETRICS Conference on Measurements and Modeling of Computer Systems*, pages 207–216, Santa Clara, California, United States, June 2000. ACM Press.
14. B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *Proceedings of the 1985 ACM SIGMETRICS conference on Measurements and Modeling of Computer Systems*, pages 147–154, Austin, Texas, United States, 1985. ACM Press.
15. William H. Sanders and John F. Meyer. Reduced Base Model Construction Methods for Stochastic Activity Networks. *IEEE Journal on Selected Areas in Communications*, 9(1):25–36, 1991.