Numerical Analysis of Superposed GSPNs

Peter Kemper
Informatik IV
Universität Dortmund
D-44221 Dortmund, Germany

Abstract

The numerical analysis of various modeling formalisms [4, 10, 14] profits from a structured representation for the generator matrix Q of the underlying continuous time Markov chain, where Q is described by a sum of tensor (kronecker) products of much smaller matrices. In this paper we describe such a representation for the class of superposed generalized stochastic Petri nets (SGSPNs), which is less restrictive than descriptions given before [10]. Furthermore a new iterative analysis algorithm is proposed which pays special attention to a memory efficient representation of iteration vectors as well as to a memory efficient, structured representation of Q. In consequence the new algorithm is able to solve models which have state spaces with several millions of states and where other exact numerical methods become impracticable on a common workstation.

1 Introduction

Generalized stochastic Petri nets (GSPNs) [1] provide a concise and powerful method for the specification and analysis of complex, dynamic systems. Their mapping to a continuous time Markov chain (CTMC) and subsequent analysis for transient and steady state distributions to derive performance measures has been known for long, and a rich variety of software tools exists which employ this (conventional) numerical analysis, among others [3, 5, 6, 11]. Due to the well known state space explosion problem numerical analysis is cumbersome for large models where the underlying CTMC contains more than 10^6 states. In these cases the conventional method, which typically includes three main steps namely state space generation, elimination of vanishing markings, and application of a numerical iteration scheme, frequently collapses for a given computer configuration due to lack of primary memory even if sparse matrix structures are employed for the representation of the stochastic generator matrix Q of the CTMC.

If the CTMC shows certain regularities which are usually imposed by the modeling formalism it is derived from, then Q can be described by a set of much smaller matrices which are combined via tensor operations [8]. This memory efficient representation of Q usually increases the size of solvable models by one order of magnitude. Such structured representations are known for stochastic automata networks (SANs) [14, 15], certain hierarchical colored stochastic Petri nets [4], and superposed generalized stochastic Petri nets (SGSPNs) [10] as an extension of superposed stochastic automata [9]. The idea in SGSPNs is to combine a set of originally independent GSPNs into a single, superposed GSPN by synchronization of timed transitions. This concept is closely related to the concept of SANs and Markovian process algebras which also consider synchronized actions.

In SGSPNs the superposition of component GSPNs can be used for a structured representation of Q under the following restrictions [10]: 1. all synchronized transitions are timed, 2. the tangible reachability graphs (TRGs) of the isolated components are strongly connected, and 3. firing of a synchronized transition leads to a tangible state. In this case Q is described by a sum of tensor products considering matrices derived from the TRGs of the isolated components. The 3rd restriction is rather awkward from a modeler's point of view. In this paper we give a similar structured representation of Q which allows to drop the 3rd restriction, i.e. in our description of Q the firing of synchronized transitions may lead to vanishing or tangible states. This extends the set of SGSPNs for which a concise[1] structured description of Q is known, hence improving efficiency of numerical analysis methods based on it.

The main advantage of a structured representation is that it is a very memory efficient matrix representation, which is essential for the applicability of any numerical analysis. In case of SGSPNs the price paid for this advantage is that the cross product of state spaces of isolated components – in the following denoted by $\mathcal{PS}$ for product space – is considered to be the tangible reachability set ($\mathcal{TRS}$) of the SGSPN, although often $|\mathcal{PS}| >> |\mathcal{TRS}|$ due to the restrictions imposed by synchronization. In the context of structured descriptions the fact that introducing additional restrictions might exclude reachability of certain states in $\mathcal{PS}$ has been mentioned before in [7].

The following simple example demonstrates the impact of synchronization on the relation between $\mathcal{TRS}$ and $\mathcal{PS}$. Consider the net in Fig. 1, which consists of two independent GSPNs A and B, where places a_1,a_2,

[1] in terms of the number of terms in the sum of tensor products of a structured description

Proceedings of the Sixth International Workshop on Petri Nets and Performance Models (PNPM '95)
1063-6714/95 $10.00 © 1995 IEEE

and a_3 belong to A and all other places belong to B. If the transitions in a shaded box are merged into a single transition, the resulting GSPN is a SGSPN with 3 synchronized transitions and components A and B. We assume an initial marking $M_0 = (a_1 = a_2 = 0, a_3 = p, b_1 = b_2 = p + m, b_3 = 0)$, where p and m are nonnegative parameters. By combinatorial arguments it is easily obtained that

$$|\mathcal{PS}| = \binom{p+2}{p}\binom{2(p+m)+2}{2(p+m)} \geq \binom{p+2}{p} = |\mathcal{TRS}| \quad (1)$$

since for the given M_0 in the SGSPN only tokens on a_1, a_2, a_3 distribute freely over places a_1, a_2, a_3 and places b_1, b_2, b_3 are redundant. Due to the additional degree of freedom gained from parameter m, initial markings can be chosen to increase $\mathcal{PS}$ arbitrarily compared to $\mathcal{TRS}$. Obviously synchronization of transitions drastically reduces the independence of components in this example. The example is surely a worst case example, but it clearly indicates that a numerical analysis method based on a structured representation of Q, where Q is a $(|\mathcal{PS}| \times |\mathcal{PS}|)$ matrix, must take care of this problem or otherwise the method becomes hopelessly inefficient for SGSPNs of this type.

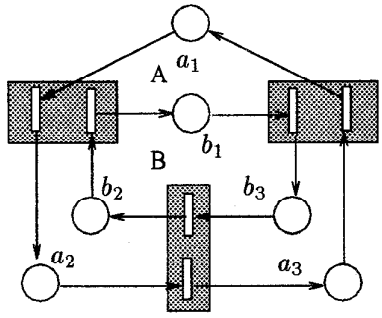

Figure 1: SGSPN superposed from GSPNs A and B by merging shaded transitions

This problem is attacked and solved in this paper in that we give a numerical analysis method which employs a structured representation of Q, restricts the size of iteration vectors to the size of $\mathcal{TRS}$ and considers only states of $\mathcal{TRS}$ in its iteration scheme. For example, the new analysis algorithm is able to analyze a SGSPN with $|\mathcal{PS}| \approx 201 \cdot 10^6$ and $|\mathcal{TRS}| \approx 4 \cdot 10^6$ on a sparc station with 48 MB primary memory, as described in Sec. 6.

The paper is organized as follows. In Sec. 2 we give some basic definitions and describe the structured representation of Q for SGSPNs. The superposition of GSPNs into a SGSPN can impose certain constraints on the state spaces of components, which in turn are useful to reduce the size of $\mathcal{PS}$. These constraints are considered in Sec. 3. Section 4 focuses on an iteration scheme which is able to distinguish states in $\mathcal{TRS}$ and states in $\mathcal{PS}\backslash\mathcal{TRS}$ by means of an appropriate permutation. The new analysis algorithm is given in Sec. 5, and its usefulness and applicability is demonstrated by an example in Sec. 6.

2 Definitions and theoretical basis

The notation for SGSPNs follows mainly [10], thus we only briefly introduce some basic notations and assume that the reader is familiar with GSPNs and their dynamic behavior.

Definition 1 *A GSPN is an eight-tuple*

$$(P, T, \pi, I, O, H, W, M_0)$$

where
P is the set of places,
T is the set of transitions such that $T \cap P = \emptyset$,
$\pi : T \to \{0, 1\}$ is the priority function,
$I, O, H : T \to Bag(P)$, are the input, output, and inhibition functions, respectively, where Bag(P) is the multiset on P,
$W : T \to \mathbb{R}$ is a function that assigns a weight to each transition,
$M_0 : P \to \mathbb{N}$ is the initial marking: a function that assigns a nonnegative integer value to each place.

Based on this definition the reachability set (RS), the reachability graph (RG), the tangible reachability set (TRS) and the tangible reachability graph (TRG) can be defined as usual.

For GSPNs well-known techniques apply to derive a state transition matrix $\bar{Q}$ from the TRG, such that the stochastic generator matrix Q of the underlying CTMC is given by $Q = \bar{Q} - D$ with diagonal matrix D

$$D(i,j) := \begin{cases} \sum_k \bar{Q}(i,k) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Superposed GSPNs are GSPNs, where additionally a partition of the set of places is defined such that SGSPNs can be seen as a set of GSPNs which are synchronized by certain transitions.

Definition 2 *A SGSPN is a ten-tuple*

$$(P, T, \pi, I, O, H, W, M_0, \Pi, TS)$$

where
$(P, T, \pi, I, O, H, W, M_0)$ is a GSPN,
$\Pi = \{P^0, \ldots, P^{N-1}\}$ is a partition of P,
$TS \subseteq \{t \in T | \pi(t) = 0\}$ is the set of synchronized transitions, that are timed by definition.
Moreover Π induces on $T\backslash TS$ a partition of transitions. Such an SGSPN contains N components

$$(P^i, T^i, \pi^i, I^i, O^i, H^i, W^i, M_0^i)$$

for $i \in \{0, 1, \ldots, N-1\}$, where
$T^i := {}^\bullet P^i \cup P^i{}^\bullet$ and
$\pi^i, I^i, O^i, H^i, W^i, M_0^i$ are the functions π, I, O, H, W, M_0 restricted to P^i, resp. T^i.

Proceedings of the Sixth International Workshop on Petri Nets and Performance Models (PNPM '95)
1063-6714/95 $10.00 © 1995 IEEE

Figure 1 shows two component GSPNs A and B in isolation. A SGSPN can be obtained by merging the transitions which are contained in the same shaded box. In the following we assume a partition of a SGSPN into N components with an arbitrary but fixed index and refer to a single component by its index.

Note that synchronized transitions of a SGSPN are timed by definition. This ensures that within an SGSPN components have "borderlines" just built from timed transitions. Consequently the firing of immediate transitions in different components is rather independent in that the firing of an immediate transition in component i cannot enable or disable the firing of an immediate transition in component j within the SGSPN. This is a nice special case of [2]: immediate transitions in different components cannot be in the same equal conflict set (ECS), i.e. they cannot enable or disable each other. This simplifies the elimination of vanishing markings considerably. According to [2] the state transition matrix $\bar{Q}$ of the whole (S)GSPN can be calculated from

$$\bar{Q}(i,j) = \sum_{t:M_i-t\to M_j} w(t) \prod_{n=0}^{N-1} Prob[F^n(t,M_i) \to M_j^n] \quad (2)$$

$w(t)$ denotes the firing rate of timed transition t. M_j^n is the restriction of tangible marking M_j to component n. $F^n(t,M_i)$ denotes the successor marking after firing transition t in marking M_i with restriction to component n. $F^n(t,M_i)$ can be the start of a firing sequence σ of immediate transitions which is local in n and leads to the tangible state M_j, so $Prob[F^n(t_k,M_i) \to M_j^n]$ gives the probability to reach M_j^n over all such sequences σ, which can be empty, under the condition that t is fired in M_i^n. Informally the idea is that the probability of firing sequences of immediate transitions enabled by firing of a timed transition t is given by a product of subsequences, where each subsequence consists of transitions of a single ECS. Since transitions in different ECS do not interfere, their firing probabilities only require a normalization which is local to the ECS. A partition into ECSs is naturally given in SGSPNs by the partition into components, such that the product in (2) considers all N components in an arbitrary, but fixed order, which is equivalent to the introduction of additional priorities on different ECS in [2]. For example the net in Fig. 2 has components A, B, and C and firing of synchronized transition ts_1 enables immediate transitions ti_1, ti_2, and ti_3. Let M_j be a marking reached from $F(ts_1,M_i)$ by firing of ti_1 and ti_3 in an arbitrary order, then (2) states that for $\bar{Q}(i,j)$ it is sufficient to consider probabilities of isolated components, i.e. $\text{Prob}[F^A(ts_1,M_i) \to M_j^A] = 1$, because $F^A(ts_1,M_i) = M_j^A$, $\text{Prob}[F^B(ts_1,M_i) \to M_j^B] = w(ti_1)/(w(ti_1)+w(ti_2))$ for transition ti_1, and $\text{Prob}[F^C(ts_1,M_i) \to M_j^C] = w(ti_3)/w(ti_3) = 1$ for transition ti_3.

We will make use of this property in proving that a representation of Q for the CTMC underlying an SGSPN is correct, which represents Q by a sum of tensor products and allows that firing of a synchronized transition enables immediate transitions. The definition of tensor (kronecker) product is based on a mapping function using mixed radix number representation [8].

Definition 3 *Mapping function mix*
Let $TRS^i := \{0,1,\ldots,k^i-1\}$ be some finite sets with arbitrary but fixed constants k^i for all $i \in \{0,1,\ldots,N-1\}$ and $k = \prod_{i=0}^{N-1} k^i$. A mapping $mix : \times_i TRS^i \longrightarrow \{0,1,\ldots,k-1\}$ is defined by

$$mix(x^{N-1},\ldots,x^1,x^0) := \sum_{i=0}^{N-1} x^i g_i$$

with weights $g_0 := 1, g_i := k^{i-1} g_{i-1}$.

A vector $(x^{N-1},\ldots,x^0) \in \times_i TRS^i$ is the mixed radix number representation of $x = mix(x^{N-1},\ldots,x^0)$ with respect to basis $(k^{N-1},\ldots,k^0)$. In the following definition of tensor product we follow the notation in [8] but regard only the restricted case of square matrices to keep a concise notation, because only square matrices occur in the context of SGSPNs.

Definition 4 *Tensor product and sum for square matrices*
Let $A^0,\ldots,A^{N-1}$ be square matrices of dimension $(k^i \times k^i)$ then their tensor product $A = \bigotimes_{i=0}^{N-1} A^i$ is defined by $a(x,y) := \prod_{i=0}^{N-1} a^i(x^i,y^i)$ where $x = mix(x^{N-1},\ldots,x^0)$ and $y = mix(y^{N-1},\ldots,y^0)$.

The tensor sum $B = \bigoplus_{i=0}^{N-1} A^i$ is then given by $\bigoplus_{i=0}^{N-1} A^i := \sum_{i=0}^{N-1} I_{l^i} \bigotimes A^i \bigotimes I_{r^i}$ where I_{l^i}, I_{r^i} are matrices of dimension $l^i \times l^i$, resp. $r^i \times r^i$ where $r^i = \prod_{j=0}^{i-1} k^j$, $l^i = \prod_{j=i+1}^{N-1} k^j$ and $I(a,b) = 1$ iff $a = b$ and 0 otherwise.

A tensor product formalizes the operation of multiplying every matrix element of one matrix with all matrix elements of the other matrices and these products of matrix elements are arranged in lexicographical order in the resulting matrix, for more details see e.g. [16].

The components of a SGSPN are GSPNs themselves such that they can be analyzed in isolation to obtain corresponding tangible reachability sets TRS^i as for any GSPN. In the following we will regard only SGSPNs such that the TRS^i of their components are strongly connected. This restriction is mentioned also in [10]. Due to our assumption that all states in TRS^i are consecutively numbered from 0 to k_i, mix induces such a numbering on $\times_{i=0}^{N-1} TRS^i$ as well. Since such a numbering allows to identify states, we will not distinguish between a state

Proceedings of the Sixth International Workshop on Petri Nets and Performance Models (PNPM '95)
1063-6714/95 $10.00 © 1995 IEEE

$M_x = (M_x^{N-1}, \ldots, M_x^0)$ and its number, resp. component numbers $x = mix(x^{N-1}, \ldots, x^0)$ in order to preserve readability. Furthermore let $\mathcal{PS} = \times_{i=0}^{N-1} TRS^i$ denote the product space obtained from the TRS^i of isolated components of an SGSPN. The set of reachable tangible states of an SGSPN is denoted by $\mathcal{TRS}$.

Proposition 1 *All $x \in \mathcal{TRS}$ are also element of $\mathcal{PS}$.*

Proof. Since in the cross product of TRS^i obtained from isolated components, enabling conditions of synchronized transitions are less restrictive than in the SGSPN and all TRS^i are strongly connected, any $x \in TRS$ is reachable in $\mathcal{PS}$ as well. □

Together with TRS^i, state transitions matrices $\bar{Q}^i$ can be derived for every isolated component i such that a $\bar{Q}^i$ does not contain vanishing markings any more, i.e. the elimination of vanishing markings has been applied beforehand. Matrix $\bar{Q}^i$ can be seen as a sum of matrices $\bar{Q}^i = \sum_{t \in T^i} w(t)\bar{Q}_t^i$, such that non-zero entries are separated according to the timed transition t which contributes to that entry. The elimination of vanishing markings implies that $\bar{Q}_t^i(x, y)$ gives the conditional probability to reach marking M_y^i if transition t is fired in M_x^i, or more formally $Prob[F^i(t, M_x^i) \rightarrow M_y^i]$ like in (2).

The terms of the sum which correspond to unsynchronized (local) timed transitions shall be denoted by $\bar{Q}_l^i := \sum_{t \in T^i \setminus TS} w(t)\bar{Q}_t^i$.

Theorem 1 *The state-transition matrix $\bar{Q}$ for $\mathcal{PS} = \times_{i=0}^{N-1} TRS^i$ of an SGSPN with strongly connected TRS^i equals*

$$\bigoplus_{i=0}^{N-1} \bar{Q}_l^i + \sum_{t \in TS} w(t) \bigotimes_{i=0}^{N-1} \bar{Q}_t^i$$

where $\bar{Q}_t^i = I^i$ if $t \notin T^i$.

The main ideas underlying the following proof are that firings of local transitions are well described by a tensor sum of local state transition matrices $\bar{Q}_l^i$ – this is well known, for a detailed discussion see e.g. [16] – and that firing of a synchronized transition followed by a firing sequence of immediate transitions in various components profits from the fact that immediate transitions in different components belong to different ECSs, i.e. they cannot disable each other. The latter observation allows to exploit (2) for the calculation of a matrix entry, which happens to coincide with the matrix entries that result from $\sum_{t \in TS} w(t) \bigotimes_{i=0}^{N-1} \bar{Q}_t^i$ as shown below.

Proof. Regard two markings $M_x, M_y \in \mathcal{PS}$, with $x = mix(x^{N-1}, \ldots, x^0)$ and $y = mix(y^{N-1}, \ldots, y^0)$.
Since the value on matrix position $\bar{Q}(x, y)$ can be a sum of values caused by the possible firing of several timed transitions performing the same transformation of marking M_x into marking M_y, let $\bar{Q} = \sum_{t \in TS} \bar{Q}_t + \sum_{e=0}^{N-1} \bar{Q}_{l^e}$ be a representation of $\bar{Q}$, such that $\bar{Q}_t$ is a $(\mathcal{PS} \times \mathcal{PS})$ matrix containing all non-zero entries caused by firing a synchronized transition t and $\bar{Q}_{l^e}$ contains all non-zero entries caused by firing a timed transition of $T^e \setminus TS$ of a component e.
The proposed matrix representation can be transformed according to Def. 4
$\bigoplus_{i=0}^{N-1} \bar{Q}_l^i + \sum_{t \in TS} w(t) \bigotimes_{i=0}^{N-1} \bar{Q}_t^i = \sum_{i=0}^{N-1} I_{l^i} \otimes \bar{Q}_l^i \otimes I_{r^i} + \sum_{t \in TS} w(t) \bigotimes_{i=0}^{N-1} \bar{Q}_t^i$.

In order to prove the theorem we show that
1. for any component e : $\bar{Q}_{l^e} = I_{l^e} \otimes \bar{Q}_l^e \otimes I_{r^e}$
By definition the enabling condition and the firing rate of any local timed transition in e is independent of the marking in other components. Furthermore all synchronized transitions are timed, hence immediate transitions enabled by the firing of a local timed transition in e must be local in e as well, and no immediate transitions in other components can be enabled. In consequence the value of any non-zero entry $\bar{Q}_{l^e}(x, y) = \bar{Q}_l^e(x^e, y^e)$. Since only local transitions of e are involved, the marking in other components remains obviously unchanged, such that in summary we have

$$\bar{Q}_{l^e}(x, y) = \begin{cases} \bar{Q}_l^e(x^e, y^e) & \text{if } M_x^i = M_y^i \text{ for all } i \neq e \\ 0 & \text{otherwise} \end{cases}$$

Since matrices I are 1 for all I(i,j) where i=j and 0 otherwise, and a product is non-zero iff all of its factors are non-zero, $\bar{Q}_{l^e}(x, y) = [\prod_{i=e+1}^{N-1} I^i(x^i, y^i)]\, \bar{Q}_l^e(x^e, y^e)\, \prod_{i=0}^{e-1} I^i(x^i, y^i)$ so due to Def. 4 follows $\bar{Q}_{l^e} = I_{l^e} \otimes \bar{Q}_l^e \otimes I_{r^e}$

It remains to show that
2. for any $t \in TS : \bar{Q}_t = w(t) \bigotimes_{i=0}^{N-1} \bar{Q}_t^i$
A synchronized transition t is enabled in state $M_x \in \mathcal{PS}$ iff t is enabled in all components e with $t \in T^e$. (Note that the initial marking with k=1 in Fig. 2 can serve as an example for M_x and $t = ts_1$ to illustrate the argumentation.) Firing of t can only change the marking of places in components e with $t \in T^e$, and the marking of places in other components remains unchanged. Since M_x is a tangible state, no immediate transition can be enabled at M_x, and all immediate transitions enabled after firing t can again only be transitions in components e with $t \in T^e$. Since the borderline between components is formed by timed transitions, the immediate transitions enabled by firing t which belong to different components belong to different ECSs. So according to (2): $\bar{Q}_t(x, y) = w(t) \prod_{i=0}^{N-1} P[F^i(t, M_x) \rightarrow M_y^i]$. By definition of $\bigotimes_{i=0}^{N-1} \bar{Q}_t^i$, each non-zero element $\bar{Q}_t^i(x^i, y^i)$ gives $P[F^i(t, M_x^i) \rightarrow M_y^i] = P[F^i(t, M_x) \rightarrow M_y^i]$ iff t is enabled in M_x and $t \in T^i$ according to the independency of local immediate transitions. Once again the product $w(t) \prod \bar{Q}_t^i(x^i, y^i)$ ensures that the resulting value is non-zero iff all factors are non-zero,

Proceedings of the Sixth International Workshop on Petri Nets and Performance Models (PNPM '95)
1063-6714/95 $10.00 © 1995 IEEE

i.e. all components i with $t \in T^i$ fulfill $\bar{Q}_t^i(x^i, y^i) = P[F^i(t, M_x^i) \rightarrow M_y^i] \neq 0$ and all components i with $t \notin T^i$ $\bar{Q}_t^i((x^i, y^i) = 1$ which in turn is only the case if $M_x^i = M_y^i$ since for $t \notin T^i$: $\bar{Q}_t^i = I^i$ is 1 for all $I^i(x^i, y^i)$ where $x^i = y^i$ and 0 otherwise. □

Starting from a structured description of state transition matrix $\bar{Q}$, it is straightforward to derive the diagonal matrix D such that the stochastic generator matrix is given by $Q = \bar{Q} - D$. In the following we will not attempt to obtain a structured description of D and rather use a standard representation, i.e. a vector enumerating the diagonal values of D. So far we have given a representation of Q based on a structured description of $\bar{Q}$ which can directly be employed within numerical analysis methods, e.g. a Jacobi iteration can be performed by $x^{(n+1)} = x^{(n)}\bar{Q}D^{-1}$. The main idea is that the vector-matrix multiplication $x^{(n)}\bar{Q}$ can be performed by multiplying appropriate projections of $x^{(n)}$ with matrix elements of $\bar{Q}_t^i$, resp. $\bar{Q}_t^i$, cf. [16, 17]. In the following we come back to the problem stated in Sec. 1, namely that often $|PS| >> |TRS|$, and describe how to reduce $|PS|$ and how to avoid overhead imposed by $|PS| >> |TRS|$.

3 Upper limits derived from SGSPN

In this section we consider the relationship between P-invariants of an SGSPN and P-invariants of its components. Since the components of an SGSPN are superposed by synchronization of transitions, superposition of GSPNs into a SGSPN preserves the P-invariants of involved GSPNs. More formally let $x_i \in \mathbb{N}_0^{P^i}$ be a P-invariant of an isolated component i, then the corresponding vector $x \in \mathbb{N}_0^P$ in the SGSPN is given by

$$x[p] := \begin{cases} x_i[p] & \text{if } p \in P^i \\ 0 & \text{otherwise} \end{cases}$$

Lemma 1 *Let x be the corresponding vector in a SGSPN for a P-invariant x_i of a component i, then $xC = 0$, i.e. x is a P-invariant of the SGSPN.*

Proof. Since the incidence functions concerning places of i remain unchanged by synchronization of transitions and the corresponding vector x is padded with zeros for all places not contained in i, $xC = 0$ is satisfied, hence x is a P-invariant [13]. □

On the other hand not all semi-positive P-invariants of the SGSPN can be derived from the semi-positive P-invariants of its components as the example net in Fig. 1 shows. A generating set of P-invariants in A is given by a unit vector; the same holds for B. Their corresponding vectors x and y are given in the table below. Nevertheless the SGSPN has additional semi-positive P-invariants z_1, z_2, z_3 which cannot be obtained as a linear combination of x and y. The additional P-invariants can be seen as global constraints imposed by superposition.

	a_1	a_2	a_3	b_1	b_2	b_3
x	1	1	1	0	0	0
y	0	0	0	1	1	1
z_1	1	0	0	1	0	0
z_2	0	1	0	0	1	0
z_3	0	0	1	0	0	1

It is well-known that semi-positive P-invariants are useful to calculate upper limits for the number of tokens on places which belong to their support. Consequently we suggest to obtain upper limits in this manner and to obey them during the generation of state spaces for the isolated components. These upper limits can be very effective, e.g. for the net in Fig.1 places (b_1, b_2, b_3) are limited to $(p+m, p+m, p)$ for the initial marking $M_0^B = (p+m, p+m, 0)$ and due to P-invariants z_1, z_2, z_3. Again by combinatorial arguments we have:

$$\begin{aligned} |\mathcal{PS}| &= \binom{3+p-1}{p} \sum_{i=0}^{p} \binom{2+i-1}{i} \\ &= \binom{p+2}{p}^2 \geq \binom{p+2}{p} = |\mathcal{TRS}| \end{aligned}$$

This way $|\mathcal{PS}|$ is significantly reduced compared to (1) but it is still much larger than $\mathcal{TRS}$.

4 A Permutation to distinguish $\mathcal{TRS}$ from $\mathcal{PS} \backslash \mathcal{TRS}$

The P-invariant based approach of the previous section attempts to reduce the number of unreachable states in the representation of Q. In this section we accept that the representation of Q contains such states for the price of a memory efficient matrix representation. Instead we attack the negative consequences of unreachable states for numerical analysis, which are twofold. Firstly, they increase the size of iteration vectors, which is crucial for the applicability of the method. Secondly, they cause useless multiplications of matrix and vector elements, since unreachable states permanently stay at zero probability. Whenever vector-entries corresponding to unreachable states are considered in the iteration it is a waste of time.

The main problem in recognizing unreachable states during iteration is that their vector positions are mixed with positions of reachable states and the set of reachable states is not known. Since the latter can be solved by performing a state space exploration based on the tensor representation of Q as described in the following, let us assume for now the set $\mathcal{TRS}$ is known. Separating $\mathcal{TRS}$ from unreachable states can be formally described by defining an appropriate permutation, which reorders states according to their reachability.

Definition 5 *For a SGSPN with $|\mathcal{PS}| = k$, a bijective function perm : $\{0, 1, \ldots, k-1\} \longrightarrow \{0, 1, \ldots, k-1\}$ is a TRS-permutation if*

Proceedings of the Sixth International Workshop on Petri Nets and Performance Models (PNPM '95)
1063-6714/95 $10.00 © 1995 IEEE

- $\forall M_x \in \mathcal{PS} : perm(x) < |\mathcal{TRS}| \iff M_x \in \mathcal{TRS}$
- $\forall M_x, M_y \in \mathcal{TRS} : perm(x) < perm(y) \iff x < y$

Note that several TRS-permutations exist for the $\mathcal{PS}$ of a given SGSPN, since the definition requires only reachable states to be mapped in an order preserving way into the set $\{0, 1, \ldots, |\mathcal{TRS}|-1\}$, the mapping of unreachable states into the set $\{|\mathcal{TRS}|, \ldots, k-1\}$ is bijective but not necessarily order preserving. Such a TRS-permutation can be described by a $(k \times k)$ matrix $\mathcal{P}$ with

$$p(i,j) := \begin{cases} 1 & \text{if } j = perm(i) \\ 0 & \text{otherwise} \end{cases}$$

Let $x_p = x\mathcal{P}$ denote the permuted vector of an iteration vector x, then the following transformation can be applied to an arbitrary iteration method with iteration matrix H:

$$\begin{aligned} & x^{(n+1)} = x^{(n)} H \\ \iff \quad & x_p^{(n+1)} \mathcal{P}^{-1} = x_p^{(n)} \mathcal{P}^{-1} H \\ \iff \quad & x_p^{(n+1)} = x_p^{(n)} \mathcal{P}^T H \mathcal{P} \end{aligned}$$

Note that *perm* is bijective, which ensures existence of $\mathcal{P}^{-1}$ and that in the special case of permutation matrices $\mathcal{P}^{-1} = \mathcal{P}^T$ holds. In Jacobi iteration $H_J = \bar{Q}D^{-1}$ for $Q = \bar{Q} - D$, where $\bar{Q}$ is the transition matrix and D the matrix with diagonal values of Q.

$$\begin{aligned} & x_p^{(n+1)} = x_p^{(n)} \mathcal{P}^T (\bar{Q} D^{-1}) \mathcal{P} \\ \iff \quad & x_p^{(n+1)} = x_p^{(n)} \mathcal{P}^T \bar{Q} (\mathcal{P}\, \mathcal{P}^T) D^{-1} \mathcal{P} \\ \iff \quad & x_p^{(n+1)} = x_p^{(n)} (\mathcal{P}^T \bar{Q} \mathcal{P})\; D_p^{-1} \end{aligned}$$

where $D_p^{-1} := \mathcal{P}^T D^{-1} \mathcal{P}$ is again a diagonal matrix and diagonal values are permuted according to *perm*. The practical implications of employing a TRS-permutation are that iteration vectors $x_p^{(n)}, x_p^{(n+1)}$ can be represented by arrays of size $|\mathcal{TRS}|$, where $perm(x) = i$ gives the appropriate position i for a state $M_x \in \mathcal{TRS}$, as the probability of unreachable states is known to be zero. Furthermore D_p^{-1} can be represented by an array of size $|\mathcal{TRS}|$ in a similar way, and the same holds for $\mathcal{P}, \mathcal{P}^T$. In fact at the end of this section we show that it is sufficient to use a single integer array of size $|\mathcal{TRS}|$ to represent both $\mathcal{P}$ and $\mathcal{P}^T$.

In this way no component of the modified iteration scheme is of size $|\mathcal{PS}|$ any more and we cured the problem of oversized iteration vectors. The second negative implication of $|\mathcal{PS}| >> |\mathcal{TRS}|$, namely the useless computations for unreachable states during iteration, can be easily avoided by consecutively performing computations only for states in the first part of $x_p^{(n)}$, where reachable states are located.

Of course if $|\mathcal{TRS}|$ is almost of size $|\mathcal{PS}|$ the suggested approach does not pay off, but in this case the algorithm can easily fall back to the standard iteration scheme without permutation, thus avoiding overhead caused by a permutation.

Exploration of $\mathcal{TRS}$ and generation of $\mathcal{P}$ The definition of a $\mathcal{P}$ presupposes that the $\mathcal{TRS}$ is known. The $\mathcal{TRS}$ can be explored efficiently by employing the structured representation of $\bar{Q}$. The basic idea is to make use of the fact that the tensor operations map combinations of component states into the states of the $\mathcal{PS}$ according to function *mix* (cf. Def. 3). *mix* is bijective and hence it gives a perfect hash function, i.e. no collisions are possible. A simple search procedure follows:

1. push(initial state)
2. while stack not empty
   (a) pop(state) and calculate successors of this state from $\bar{Q}$
   (b) for every successor s
      i. if s has not been reached before then push(s)

Data structures for such a procedure can be a stack for still-to-search states and a hash table to answer the question whether a state has been reached before. Obviously it is sufficient to use a bit-vector as a hash table and function *mix* can serve as a hash function.

Since a state M_x is represented on the stack by its corresponding integer value x and the size of the stack can not exceed $|\mathcal{TRS}|$ it is not critical in terms of memory requirements. Furthermore access to the stack shows a high degree of locality by nature, so even a large stack can be handled very well by virtual memory.

The dimension of the hash table is $|\mathcal{PS}|$ and thus more critical, but since a single entry requires only 1 bit of memory, reasonably large state spaces can be explored. In fact if $\frac{|\mathcal{TRS}|}{|\mathcal{PS}|} \geq \frac{1}{64} = 0.015625$, i.e. if at least 1.5 % of $\mathcal{PS}$ is reachable, then the hash table requires less(!) memory than one double-precision iteration vector of dimension $|\mathcal{TRS}|$.

Surely different data structures can be employed at this stage but experimental results show that this kind of state space exploration is not critical in terms of computation time and memory requirements. It takes an amount of time which is less than the time for a single iteration step.

In order to obtain $\mathcal{P}$, the hash table is transformed into an integer-vector of length $|TRS|$ which contains the indices of all reachable states. A single vector is sufficient to represent $\mathcal{P}$ and $\mathcal{P}^T$, since the entry x at vector position i gives $perm^{-1}(i)$ and for $perm(x)$ a binary search with logarithmic time complexity yields position i if $x \in \mathcal{TRS}$ or denotes that $x \notin \mathcal{TRS}$, where the latter tells an iteration method that x is not reachable and thus irrelevant.

Proceedings of the Sixth International Workshop on Petri Nets and Performance Models (PNPM '95)
1063-6714/95 $10.00 © 1995 IEEE

5 A numerical analysis method

In this section we compose the results of the previous section to a new numerical analysis method, which computes the steady state distribution of the CTMC underlying a SGSPN. SGSPNs are restricted in that the tangible reachability sets of their components have to be strongly connected. The algorithm is new in that it uses a memory efficient representation of the iteration matrix and(!) a memory efficient representation of the iteration vector, which in combination allows to analyze SGSPN models with tangible reachability sets of several million states.

Input: SGSPN
Output: steady state distribution if convergence is obtained

1. Calculate minimal P-invariants of the SGSPN and derive upper limits for the number of tokens on places in P.

2. Generate TRS^i and matrices $\bar{Q}_l^i$, $\bar{Q}_t^i$ for all components i in isolation which includes elimination of vanishing markings. Place capacities in the isolated components are set according to the upper limits derived from P-invariants.

3. Explore the $\mathcal{TRS}$ of the SGSPN on state transition matrix $\bar{Q}$ of $\mathcal{PS}$ and generate permutation matrix $\mathcal{P}$ from $\mathcal{TRS}$ as described in Sec. 4, last paragraph.

4. Choose initial distribution on TRS, e.g. uniform distribution.

5. if $|\mathcal{TRS}| << |\mathcal{PS}|$ perform an iterative method employing the permutation matrix $\mathcal{P}$
In case of Jacobi or JOR generate D_p^{-1} from $\bar{Q}$ for all elements of $\mathcal{TRS}$, for power method generate D_p respectively.

Jacobi

$$x_p^{(n+1)} = [x_p^{(n)}\mathcal{P}^T(\bigoplus_{i=0}^{N-1}\bar{Q}_l^i)\mathcal{P} + \sum_{t\in TS} x_p^{(n)}\mathcal{P}^T(w(t)\bigotimes_{i=0}^{N-1}\bar{Q}_t^i)\mathcal{P}]D_p^{-1}$$

Jacobi overrelaxation (JOR)
choose relaxation factor $\omega \in]0,2[$

$$x_p^{(n+1)} = (1-\omega)x_p^{(n)} + \omega[x_p^{(n)}\mathcal{P}^T(\bigoplus_{i=0}^{N-1}\bar{Q}_l^i)\mathcal{P} + \sum_{t\in TS} x_p^{(n)}\mathcal{P}^T(w(t)\bigotimes_{i=0}^{N-1}\bar{Q}_t^i)\mathcal{P}]D_p^{-1}$$

Power method
let $\delta := 0.99/max_j|D_p(j,j)|$ and $D' := \mathcal{P}^T(I-\delta D_p)\mathcal{P}$

$$x_p^{(n+1)} = \delta[x_p^{(n)}\mathcal{P}^T(\bigoplus_{i=0}^{N-1}\bar{Q}_l^i)\mathcal{P} + \sum_{t\in TS} x_p^{(n)}\mathcal{P}^T(w(t)\bigotimes_{i=0}^{N-1}\bar{Q}_t^i)\mathcal{P}] + x_p^{(n)}D'$$

with normalization of $x_p^{(n+1)}$ until convergence is observed.

6. otherwise perform an ordinary iterative method
In case of Jacobi or JOR generate D^{-1} from $\bar{Q}$ for all elements of $\mathcal{PS}$, for power method generate D respectively.

Jacobi

$$x^{(n+1)} = [x^{(n)}\bigoplus_{i=0}^{N-1}\bar{Q}_l^i + \sum_{t\in TS} x^{(n)}(w(t)\bigotimes_{i=0}^{N-1}\bar{Q}_t^i)]D^{-1}$$

Jacobi overrelaxation (JOR)
choose relaxation factor $\omega \in]0,2[$

$$x^{(n+1)} = (1-\omega)x^{(n)} + \omega[x^{(n)}\bigoplus_{i=0}^{N-1}\bar{Q}_l^i + \sum_{t\in TS} x^{(n)}(w(t)\bigotimes_{i=0}^{N-1}\bar{Q}_t^i)]D^{-1}$$

Power method
let $\delta := 0.99/max_j|D(j,j)|$ and $D' := \mathcal{P}^T(I-\delta D)\mathcal{P}$

$$x^{(n+1)} = \delta[x^{(n)}\bigoplus_{i=0}^{N-1}\bar{Q}_l^i + \sum_{t\in TS} x^{(n)}(w(t)\bigotimes_{i=0}^{N-1}\bar{Q}_t^i)] + x_p^{(n)}D'$$

with normalization of $x_p^{(n+1)}$ until convergence is observed.

The algorithm is implemented and tested in a variant of QPN-Tool [3]. For step 1 an algorithm to calculate minimal P-invariants can be found in [12]. The generation of TRS^i for an isolated component i in step 2 follows the conventional algorithms for state space exploration, elimination of vanishing markings and matrix generation, but it additionally obeys place capacities imposed by P-invariants of the SGSPN. These additional restrictions can have a major influence on the size of TRS^i as the example in Sec. 1 shows and can speed up this step significantly.

For step 3 the exploration of $\mathcal{TRS}$ based on $\bar{Q}$ is briefly described in Sec. 4. The implemented search algorithm follows Depth-First-Search and is based on hashing as a search data structure for $\mathcal{TRS}$ and a stack for states which require further investigation. Hashing profits from mapping function *mix*, which gives a perfect hash function, and uses a bit-vector as a hash table for $\mathcal{TRS}$. The hash table requires less memory than an iteration vector of length $|\mathcal{TRS}|$ if at least 1.5% of $\mathcal{PS}$ is reachable (cf. Sec. 4). The exploration of $\mathcal{TRS}$ based on $\bar{Q}$ turns out to be very

Proceedings of the Sixth International Workshop on Petri Nets and Performance Models (PNPM '95)
1063-6714/95 $10.00 © 1995 IEEE

efficient in terms of memory requirements and calculation time, i.e. exploration of $\mathcal{PS}$ with about 200 million states and $\mathcal{TRS}$ with about 4 million states takes less than 15 min elapsed time on a sparc work station with 50 MHz CPU and 48MB primary memory. The resulting representation of $\mathcal{P}$ and $\mathcal{P}^T$ is a single vector containing integer values and has length $\mathcal{TRS}$. The $\mathcal{P}$ vector contains all states of $\mathcal{TRS}$ in increasing order, hence it is a TRS-permutation (cf. Sec. 4). The fact that states in $\mathcal{P}$ are ordered allows to employ binary search on $\mathcal{P}$ in order to get the position $i = perm(x)$ for a state x of $\mathcal{TRS}$ in O(log($\mathcal{TRS}$)) or to recognize that $x \notin \mathcal{TRS}$. In the context of an iteration method in step 5, requests for finding $i = perm(x)$ occur with certain regularities such that a sophisticated implementation reduces the average search effort significantly. Obviously $\mathcal{P}$ provides $x = perm^{-1}(i)$ in O(1) by accessing $\mathcal{P}$ at position i.

Diagonal values for vector D_p^{-1} can be calculated as a by-product of $\mathcal{TRS}$ exploration or in a simplified iteration step which sums up all non-zero row-entries in $\bar{Q}$ considering all states of $\mathcal{TRS}$. Note that D_p^{-1} is a vector of length $|\mathcal{TRS}|$ and since its entries are already permuted according to $\mathcal{P}$, the multiplication with the resulting vector of $x_p^{(n)}\mathcal{P}^T\bar{Q}\mathcal{P}$ does not consider $\mathcal{P}$ any more. If the power method is applied, D_p instead of D_p^{-1} has to be generated analogously.

For step 4 the knowledge of $\mathcal{TRS}$ allows to choose an arbitrary initial distribution on $\mathcal{TRS}$ with respect to the applied iteration method, since some iteration methods are sensitive to zero initial probabilities for states in $\mathcal{TRS}$ [16]. An initial distribution should be chosen carefully due to its impact on convergence. Selection of a "good" initial distribution is surely model dependent, such that the degree of freedom obtained by the knowledge of TRS is valuable here.

The structured representation of $\bar{Q}$ and the knowledge of diagonal values allows to perform the power method as well as Jacobi iteration or Jacobi overrelaxation (JOR).

Furthermore different implementations are possible to perform the basic vector-matrix multiplication if the matrix is represented by a tensor sum or product. For step 6 we suggest to follow the method used in [14, 15, 16], which enumerates the non-zero matrix entries in a specific order. This method makes excellent use of the regular matrix structure imposed by the tensor operation and is efficient if $\mathcal{PS}$ is not much larger than $\mathcal{TRS}$. In this case it is advisable to use iteration vectors of size $\mathcal{PS}$ and perform an ordinary iteration method without permutation. The initial distribution chosen in step 4 on $\mathcal{TRS}$ is then projected on the corresponding subset of $\mathcal{PS}$.

If $|\mathcal{PS}| >> |\mathcal{TRS}|$, which frequently happens due to synchronizations, efficiency requires that an algorithm considers just non-zero matrix entries $\bar{Q}(x, y)$ where $x, y \in TRS$ in step 5. Hence in this case the algorithm for the vector matrix multiplications in $x_p^{(n)}\mathcal{P}^T(\bigoplus \bar{Q}_l^i)\mathcal{P}$ and $x_p^{(n)}\mathcal{P}^T(w(t)\bigotimes_{i=0}^{N-1}\bar{Q}_i^i)\mathcal{P}$ runs through all elements x in $\mathcal{P}$ (= $\mathcal{TRS}$) and performs multiplications only for the corresponding rows in $\bigoplus_{i=0}^{N-1}\bar{Q}_l^i$ and $\bigotimes_{i=0}^{N-1}\bar{Q}_i^i$. Since $\bar{Q}$ is a state transition matrix, for any non-zero $\bar{Q}(x, y)$ holds that $x \in \mathcal{TRS} \Rightarrow y \in \mathcal{TRS}$. In this way the search effort to calculate $i = perm(x)$ can be minimized.

6 Analysis of an example SGSPN

In this section we consider a simple example, which, however, is sufficient to demonstrate the benefits of the new approach. Fig. 2 shows a SGSPN with components A, B, and C. The initial marking is given in the graphical representation. Places without inscription are initially empty; k is an integer parameter which is modified to obtain state spaces of different sizes. Synchronized transitions are ts_1, ts_2, and ts_3. Component B contains two immediate transitions ti_1 and ti_2 which describe an non-deterministic choice for tokens on place b_3. All arc weights are assumed to be 1. Firing of synchronized transition ts_1 enables immediate transitions ti_1 and ti_2 in B and ti_3 in C.

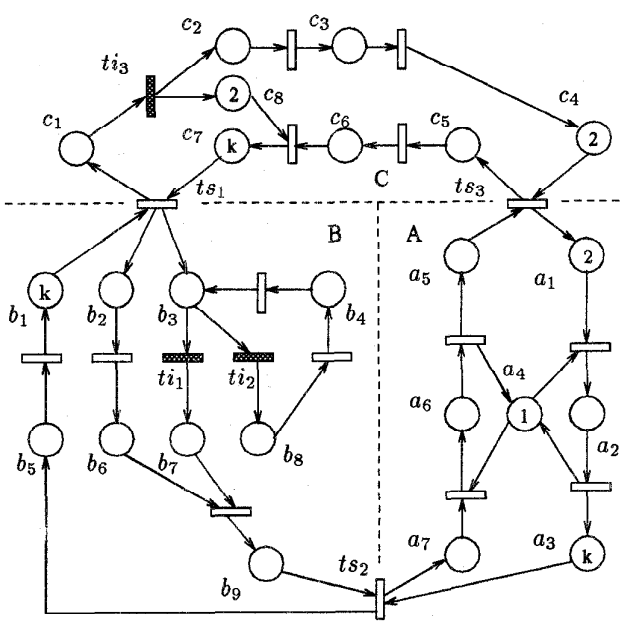

Figure 2: Example SGSPN with components A, B, and C

Note that this example cannot be analyzed as it is by the approach given in [10], since it breaks the restriction "firing of synchronized transitions leads to tangible states". Surely the SGSPN can be transformed into a net which fulfills the restriction, but this transformation enlarges the structured description of Q, as ts_1 would be replaced by a set of synchronized transitions. Due to Theorem 1 this restriction does not apply for our approach, such that the example can be directly analyzed.

Proceedings of the Sixth International Workshop on Petri Nets and Performance Models (PNPM '95)
1063-6714/95 $10.00 © 1995 IEEE

	ignoring limits					considering limits			
k	TRS^A	TRS^B	TRS^C	$\mathcal{PS}$	$\mathcal{TRS}$	TRS^A	TRS^B	TRS^C	$\mathcal{PS}$
1	40	9	56	20160	1002	28	9	38	9576
2	75	42	126	396900	14472	63	42	108	285768
3	126	140	252	4445280	128115	114	140	234	3734640
4	196	378	462	34228656	814968	184	378	444	30881088
5	288	882	792	201180672	4076640	276	882	774	188416368
6	405	1848	1287	963242280	16974198	393	1848	1269	921629016

Table 1: State spaces of example system for different values of k

The SGSPN contains several P-invariants which are not P-invariants of components in isolation. Two P-invariants which are useful to derive upper limits for the number of tokens on places are

$$inv_1(p) := \begin{cases} 1 \text{ if } p \in \{a_5, a_6, a_7, b_2, b_6, b_9, c_5, c_6, c_7\} \\ 0 \text{ otherwise} \end{cases}$$

$$inv_2(p) := \begin{cases} 1 \text{ if } p \in \{a_5, a_6, a_7, b_3, b_4, b_7, b_8, b_9, c_5, c_6, c_7\} \\ 0 \text{ otherwise} \end{cases}$$

Limits obtained from inv_1 and inv_2 in turn prove to be effective place capacities limiting the size of TRS^A and TRS^C of components A and C in isolation. An upper limit for places p in the support of a P-invariant inv are given by $lim(p) = \sum_{x \in P} inv(x)M_0(x)/inv(p)$, e.g $lim(a_5) = k/1$ for inv_1. In this example the relative impact of these limits decreases for increasing values of k, since a limit of k relates to a maximum token number of k+2 without considering limits in components A and C, but this effect is highly model dependent. Table 1 shows the sizes of TRS^A, TRS^B and TRS^C for increasing values of k together with their $|\mathcal{PS}|$ once with and once without consideration of limits derived from the P-invariants of the SGSPN. Note that e.g. consideration of these limits reduces $\mathcal{PS}$ by about 12.8 million states for k=5, which is only about 6.3 % but saves about 1.6 MB memory for the hash table. Naturally the $\mathcal{TRS}$ remains the same if these limits are ignored or taken into consideration.

Although exploiting P-invariants is useful to reduce $\mathcal{PS}$, the results clearly indicate that considering these limits alone is not sufficient. It is quite obvious that an iteration scheme is not applicable if a single iteration vector exceeds the size of available primary memory. This happens for an iteration scheme without using a TRS-permutation on our test configuration with 48 MB primary memory for k=4, where one double precision iteration vector requires about 274 MB.

With our new algorithm, which employs a TRS-permutation if $|\mathcal{TRS}| << |\mathcal{PS}|$, we are able to analyze the example SGSPN for k=5 on the same configuration. For k=6 it is possible to perform the state space exploration (cf. Sec. 4) but during the numerical iteration the algorithm relies massively on virtual memory since one iteration vector of size $\mathcal{TRS}$ requires about 136 MB, which slows down the computation to an unacceptable degree. This clearly indicates that the bottleneck of the algorithm is the size of $\mathcal{TRS}$ for iteration vectors, which on the other hand states that the employed data structures in state space exploration, in particular the bit-vector as a hash table for $\mathcal{PS}$, is perfectly suitable from a practical point of view. For the sake of completeness it should be noted that the conventional method fails if k $\geq$ 4.

7 Conclusions

In this paper we describe a numerical analysis technique for CTMCs derived from SGSPNs. The technique is based on a structured description of the generator matrix Q, which describes Q by a sum of tensor products. Structured descriptions based on tensor operations for Q matrices have been developed and successfully employed for various modeling formalisms including SGSPNs as well [4, 10, 14, 15]. The structured description we propose is similar to the one in [10] but less restrictive in that it only requires that synchronized transitions have to be timed and the tangible reachability graphs of isolated components (subnets) have to be strongly connected. Our description consists of N+TS tensor products, one for each component and for each synchronized transition. We decided to use a direct representation of diagonal values as a vector in the size of the tangible reachability set, which allows us to use other iteration methods than the power method, e.g. the Jacobi and Jacobi overrelaxation methods.

The main advantage of a structured description of Q is that it is very memory efficient, since only a set of relatively small matrices is stored instead of Q. The price for this efficient representation is that in case of SGSPNs the cross product $\mathcal{PS}$ of independent component state spaces is regarded as the relevant state space, which is due to synchronization usually a real superset of the tangible reachability set $\mathcal{TRS}$ of a SGSPN. In fact, the implicit aim of a synchronization is to restrict the behavior of the synchronized systems, e.g. as in mutual exclusion, so frequently $|\mathcal{PS}| >> |\mathcal{TRS}|$.

This effect reduces efficiency and applicability of a structured approach, if not treated adequately. In this paper we propose two means to solve this problem:

1. Constraints imposed by the SGSPN on its components are suitable to restrict the state spaces

Proceedings of the Sixth International Workshop on Petri Nets and Performance Models (PNPM '95)
1063-6714/95 $10.00 © 1995 IEEE

of isolated components for the context they are embedded in. In particular a SGSPN can impose P-invariants, which give effective place capacities for the state space generation of components in isolation. This reduces the size of $\mathcal{PS}$.

2. Orthogonal to this, $\mathcal{PS}$ can be partitioned into the set of reachable states $\mathcal{TRS}$ and unreachable states by an appropriate permutation, such that an iterative numerical method can focus on $\mathcal{TRS}$.

Both ideas are employed in the analysis algorithm, such that SGSPNs can be analyzed on a standard work station where $\mathcal{TRS}$ contains several millions of states and $\mathcal{PS}$ is larger than $\mathcal{TRS}$ by about one order of magnitude. Additionally the analysis algorithm allows to choose an initial distribution, e.g. one derived from an approximate technique, a uniform distribution or P[initial state]=1.0 and 0.0 for all other states. The iteration can be performed according to Jacobi over-relaxation (JOR), Jacobi method or Power method.

The algorithm has been implemented and tested within a modified QPN-Tool [3]. Future work will be dedicated to a parallel implementation and an integration into hierarchical concepts.

References

[1] M. Ajmone Marsan, G. Balbo, and G. Conte. A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(1), May 1984.

[2] G. Balbo, G. Chiola, G. Franceschinis, and G. Molinar-Roet. On the efficient construction of the tangible reachability graph of generalized stochastic Petri nets. In *Int. Workshop on Petri Nets and Performance Models*, pages 136–145. IEEE Computer Society, 1987.

[3] F. Bause and P. Kemper. QPN-Tool for qualitative and quantitative analysis of queueing Petri nets. In G. Haring and K. Kotsis, editors, *Computer Performance Evaluation, Modelling Techniques and Tools, Proc. 7th int. Conf., Vienna, Austria*, LNCS 794, pages 321–334. Springer, 1994.

[4] P. Buchholz. A hierarchical view of GCSPNs and its impact on qualitative and quantitative analysis. *Journal of Parallel and Distributed Computing*, (15):207–224, 1992.

[5] G. Chiola. GreatSPN 1.5 software architecture. In G. Balbo and G. Serazzi, editors, *Computer Performance Evaluation*, pages 121–136. North-Holland, 1992.

[6] G. Ciardo, J. Muppala, and K. Trivedi. SPNP: Stochastic Petri net package. In *Proc. of the 3rd Int. Workshop on Petri Nets and Performance Models*, pages 142–151. IEEE Computer Society, 1989.

[7] G. Ciardo and K.S. Trivedi. A decomposition approach for stochastic Petri net models. In *Proc. 4th Int. Workshop on Petri Nets and Performance Models*, 1991.

[8] M. Davio. Kronecker products and shuffle algebra. *IEEE Transactions on Computers*, C-30(2):116–125, February 1981.

[9] S. Donatelli. Superposed stochastic automata: a class of stochastic Petri nets with parallel solution and distributed state space. *Performance Evaluation*, 18:21–26, 1993.

[10] S. Donatelli. Superposed generalized stochastic Petri nets: definition and efficient solution. In *Application and Theory of Petri nets 1994*, Berlin, 1994. Springer.

[11] C. Lindemann. DSPNexpress: a software package for the efficient solution deterministic and stochastic Petri nets. *Performance Evaluation*, (22), 1995.

[12] J. Martinez and M. Silva. A simple and fast algorithm to obtain all invariants of a generalized Petri net. In C. Girault and W. Reisig, editors, *Application and Theory of Petri Nets*, Informatik Fachberichte 52, 1982.

[13] T. Murata. Petri nets: properties, analysis and application. *Proc. of the IEEE*, (77):541–580, 1989.

[14] B. Plateau and K. Atif. Stochastic automata network for modelling parallel systems. *IEEE Trans. on Software Engineering*, 17(10):1093–1108, 1991.

[15] B. Plateau and J.M. Fourneau. A methodology for solving Markov models of parallel systems. *Journal of Parallel and Distributed Computing*, 12, 1991.

[16] W.J. Stewart. *Introduction to the numerical solution of Markov chains.* Princeton University Press, 1994.

[17] W.J. Stewart, K. Atif, and B. Plateau. The numerical solution of stochastic automata networks. to appear in European Journ. of Oper. Res.

Proceedings of the Sixth International Workshop on Petri Nets and Performance Models (PNPM '95)
1063-6714/95 $10.00 © 1995 IEEE