Efficient decomposition algorithm for stationary analysis of complex stochastic Petri net models

Vince Molnár[1], Kristóf Marussy[1], Attila Klenik[1], András Vörös[1], István Majzik[1], and Miklós Telek[2]

[1] Department of Measurement and Information Systems, Budapest University of Technology and Economics, Budapest, Hungary, molnarv@mit.bme.hu,
[2] Department of Networked Systems and Services, Budapest University of Technology and Economics, Budapest, Hungary, telek@hit.bme.hu

Abstract. Stochastic Petri nets are widely used for the modelling and analysis of non-functional properties of critical systems. The state space explosion problem often inhibits the numerical analysis of such models. Symbolic techniques exist to explore the discrete behaviour of even complex models, while block Kronecker decomposition provides memory-efficient representation of the stochastic behaviour. However, the combination of these techniques into a stochastic analysis approach is not straightforward. In this paper we compare various combinations of symbolic techniques and decomposition based analysis methods. Saturation-based exploration is used to build the state space representation and a new algorithm is introduced to efficiently build block Kronecker matrix representation to be used by the stochastic analysis algorithms. Measurements confirm that the novel combination of the two representations can expand the limits of previous approaches.

Keywords: Stochastic Petri nets · stationary analysis · block Kronecker decomposition · numerical algorithms · symbolic methods.

1 Introduction

Stochastic analysis provides information about the quantitative aspects of models and used for the analysis of non-functional properties of critical systems. Stochastic Petri nets are widely used in reliability, availability or performability modelling to capture the stochastic behaviours and the analysis questions are answered with the help of Markovian analysis. However, successful stochastic analysis is often prevented by the state space explosion problem: in addition to the complexity of traditional qualitative analysis, stochastic computations require more involved data structures and numerical algorithms. So efficient algorithms are needed.

Many efficient techniques exist in the literature to the exploration and storage of the state space of Petri net models. One of the most efficient is the so-called

saturation algorithm [13] which uses a special iteration strategy for the state space traversal and stores the state space representation symbolically in decision diagrams. On the other side, numerical algorithms for Markovian analysis need some representation of the stochastic behaviours.

Infinitesimal generator matrix describes the behaviours of stochastic Petri nets and the underlying Markov chains. The size of the matrix is quadratic in the number of reachable states of the system. Thus the storage requirements are proportional to the square of the state space size if every element is stored in a dense matrix form. Sparse matrix formats, such as Compressed Column Storage (CCS) [2, Section 4.3.1], reduce memory requirements to be proportional to the transitions in the system. However, sparse storage may be insufficient even for models of moderate complexity due to the phenomenon of *state space explosion*.

Potential Kronecker methods [7] divide the large matrix representation into smaller matrices using only local information. Computations are then performed with the local matrices and vectors. Unfortunately, using local information leads to storing probabilities for unreachable states. This causes problems in some numerical solution algorithms as well as increases storage requirements.

In contrast, *actual Kronecker methods* [4, 7, 18] apply additional conversions and computations to handle unreachable states in the encoding. This yields higher implementation complexities and computational overhead.

The basis of our work is Block Kronecker decomposition which imposes a hierarchical structure on the reachable state space to solve the issue of unreachable potential states [3, 6, 8].

Several algorithms have been developed that use variations of decision diagrams to represent the infinitesimal generator. Matrix diagrams [12, 26] generalize the Kronecker representation to arbitrary matrices and not necessarily Kronecker consistent model partitions. Multi-Terminal Decision Diagrams (MTDDs) can store both the generator matrix [20] and the vector of state probabilities [19] by extending decision diagrams with terminal nodes corresponding to real numbers. Multiplicative Edge-valued Multilabel Decision Diagrams (EV*MDDs) [31] can provide up to exponential space savings compared to MTDDs by storing matrix and vector entries as edge labels instead of terminal nodes.

While vector-matrix products can be handled with symbolic approaches easily, more elaborate matrix access becomes difficult. For example, efficient accessing a single column of the symbolic descriptor requires the introduction of caching strategies [32].

As it turns out from the literature, the combination of efficient state space traversal and matrix representation techniques into a stochastic analysis approach is not straightforward. In this paper we compare various combinations of symbolic techniques and decomposition based analysis methods. Saturation-based exploration is used to build the state space representation and a new algorithm is introduced to efficiently build block Kronecker matrix representation to be used by the stochastic analysis algorithms. Measurements confirm that the novel combination of the two representations can expand the limits of previous approaches.

2 Background

In this section, we overview the basic formalisms and scope of our work. At first, the stochastic Petri net based formalism is introduced. Kronecker algebra and multivalued decision diagrams are also discussed.

2.1 Stochastic Petri Nets

Stochastic Petri Nets extend Petri nets by assigning exponentially distributed random delays to transitions [22]. After the delay associated with an enabled transition is elapsed the transition fires and transition delays are reset.

Definition 1. *A Stochastic Petri Net is a pair* $SPN = \langle PN, \Lambda \rangle$*, where PN is a Petri net and* $\Lambda \colon T \to \mathbb{R}^+$ *maps the set of transitions to transition rates. PN may contain inhibitor arcs but no priority specifications.*

The stochastic behaviours of a stochastic Petri net are defined by an underlying continuous-time Markov chain (CTMC). The Markov chain associated with an SPN is a stochastic process $X(\tau) \in \mathrm{RS}$, $\tau \geq 0$, where RS is the set of reachable markings of the underlying Petri net.

We will only consider the case when the Petri net is bounded, hence $n = |\mathrm{RS}| < \infty$. In order to establish the transformation between the Petri net and its underlying Markov chain, we have to define a mapping from states to indices. A bijection $\iota \colon \mathrm{RS} \to \{0, 1, \ldots, n-1\}$ exists between the reachable markings and a set of indices. This allows representing the distribution of $X(\tau)$ as a vector

$$\boldsymbol{\pi}(\tau) \in \mathbb{R}^n, \quad \pi(t)[x] = \Pr(X(\tau) = \iota^{-1}(x)),$$

i.e. $\pi(\tau)[\iota(M)]$ is the probability that the SPN is in the marking M at time t.

The time evolution of the $X(\tau)$ is described by the differential equation

$$\frac{\partial \boldsymbol{\pi}(\tau)}{\partial \tau} = \boldsymbol{\pi}(\tau)\, Q, \tag{1}$$

where $Q \in \mathbb{R}^{n \times n}$ is the *infinitesimal generator matrix* of the CTMC associated with the stochastic Petri net.

Off-diagonal elements $q[x, y]$ of Q ($0 \leq x, y < n$ and $x \neq y$) contain the rate of exponentially distributed transitions from the marking $\iota^{-1}(x)$ to $\iota^{-1}(y)$. Diagonal elements $q[x, x]$ are calculated such that $Q\mathbf{1}^{\mathrm{T}} = \mathbf{0}^{\mathrm{T}}$, where $\mathbf{1}$ and $\mathbf{0} \in \mathbb{R}^n$ are vectors with every element equal to 1 and 0, respectively. That is,

$$q[x, y] = \begin{cases} -\sum_{z=0, z \neq x}^{n-1} q[x, z] & \text{if } x = y, \\ \sum_{\iota^{-1}(x)[t\rangle \iota^{-1}(y)}^{t \in T} \Lambda(t) & \text{if } x \neq y. \end{cases} \tag{2}$$

The notation $\iota^{-1}(x)\ [t\rangle\ \iota^{-1}(y)$ indicates that the transition t can be fired in the marking $\iota^{-1}(x)$ to assume the marking $\iota^{-1}(y)$.

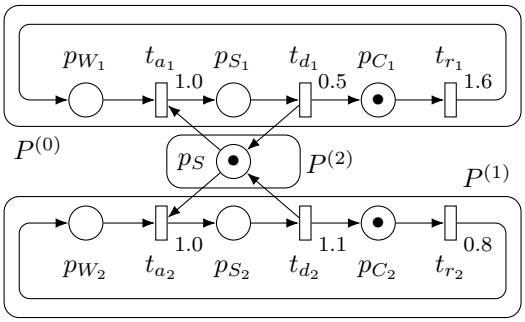

Fig. 1. Example stochastic Petri net with superposed partitions.

$$Q = \begin{pmatrix} * & 1.6 & 0.8 & 0 & 0 & 0 & 0 & 0 \\ 0 & * & 0 & 0.8 & 1 & 0 & 0 & 0 \\ 0 & 0 & * & 1.6 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & * & 0 & 1 & 0 & 1 \\ 0.5 & 0 & 0 & 0 & * & 0.8 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & * & 0 & 0 \\ 1.1 & 0 & 0 & 0 & 0 & 0 & * & 1.6 \\ 0 & 1.1 & 0 & 0 & 0 & 0 & 0 & * \end{pmatrix}$$

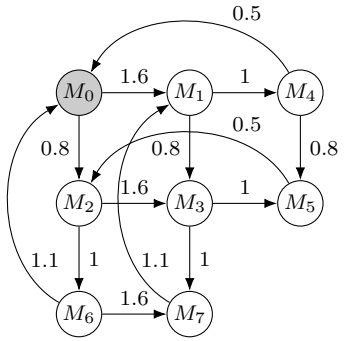

Fig. 2. Continuous-time Markov chain associated with the *SharedResource* SPN.

Example 1. In Fig. 1 we introduce the *SharedResource* model which will serve as a running example throughout this paper.

The model consists of a single shared resource S and two consumers. Each consumer can be in one of the following states: C_i (calculating locally), W_i (waiting for resource) and S_i (using shared resource). The transitions r_i (request resource), a_i (acquire resource) and d_i (done) correspond to behaviors of the consumers. The net has 8 reachable states, which are also shown in Fig. 1. As the net is 1-bounded, only the marked places are listed for each state.

The net is annotated with exponentially distributed transition rates. The clients have different request (1.6 and 0.8) and completion (0.5 and 1.1, respectively) rates, while both clients have an acquire rate of 1.0.

The Markov chain associated with the *SharedResource* SPN is shown in Fig. 2. The rows and columns of the matrix are indexed such that $\iota(M_x) = x$. Elements in the infinitesimal generator matrix Q marked by $*$ are negative numbers selected such that the sum of each row is 0. A graphical representation of the Markov chain is presented on the right side of Fig. 2.

Extensions of stochastic Petri nets include transitions with general or phase-type delay distributions [21,23], Generalized Stochastic Petri Nets (GSPN) with

immediate transitions [24, 29] and Deterministic Stochastic Petri Nets (DSPN) with deterministic firing delays [22]. Stochastic Well-formed Nets (SWN) are a class of colored Petri nets especially amenable to stochastic analysis [10]. Stochastic Activity Networks (SAN) also allow colored places, moreover, they introduce input and output gates for more flexible modeling [25].

2.2 Superposed Stochastic Petri Nets

As the decomposition method in our approach fits the concept of superposed Petri nets, we use it in the rest of the paper.

Definition 2. *A* Superposed Stochastic Petri Net *(SSPN) is a pair* $SSPN = \langle SPN, \mathcal{P} \rangle$*, where* $\mathcal{P} = \{P^{(0)}, P^{(1)}, \ldots, P^{(J-1)}\}$ *is the partitioning of the set of places* P *in the underlying Petri net of SPN such that* $P = P^{(0)} \sqcup P^{(1)} \sqcup \cdots \sqcup P^{(J-1)}$ *[15].*

The partition $P^{(j)}$ is called the jth *local net* or *component* of SSPN. A *local marking* $M^{(j)} \colon P^{(j)} \to \mathbb{N}$ is obtained from a *global marking* $M \colon P \to \mathbb{N}$ by restricting the domain to $P^{(j)}$, i.e. $M^{(j)} = M|_{P^{(j)}}$.

The *local reachable state space* of the jth local net contains the restrictions of the globally reachable markings $\mathrm{RS}^{(j)} = \{M^{(j)} : M \in \mathrm{RS}\}$.

We will assume a bijection $\iota^{(j)} \colon RS^{(j)} \to \{0, 1, \ldots, n_j\}$ from local markings to an index set, where $n_j = |\mathrm{RS}^{(j)}|$. Let the notation $x^{(j)}$ refer to the local marking $\left(\iota^{(j)}\right)^{-1}(x)$.

If $\mathbf{x} = \langle x[0], x[1], \ldots, x[J-1] \rangle$ is a vector of indices, then let the notation

$$M = \langle\!\langle \mathbf{x} \rangle\!\rangle = \langle\!\langle (x[0])^{(0)}, (x[1])^{(j)}, \ldots, (x[J-1])^{(J-1)} \rangle\!\rangle$$

refer to the marking M with the property $M^{(j)} = (x[j])^{(j)}$ for all j, i.e. M is the marking the obtained by joining the local markings indexed by $\mathbf{x}$. We extend this notation to take sets of local states and yield a set of markings.

The *potential state space*

$$\mathrm{PS} = \langle\!\langle \mathrm{RS}^{(0)}, \mathrm{RS}^{(1)}, \ldots, \mathrm{RS}^{(J-1)} \rangle\!\rangle$$

is isomorphic to the Cartesian product of local state spaces. More concretely, for each $M \in \mathrm{PS}$, there is a vector of indices $\mathbf{x}$ such that $M = \langle\!\langle \mathbf{x} \rangle\!\rangle$, i.e. M can be obtained by joining some local states for each component. This vector is the state coding of M expressed by the function $\boldsymbol{\iota}(M) : \mathrm{PS} \to \mathbb{N}^J$, $\boldsymbol{\iota}(\langle\!\langle \mathbf{x} \rangle\!\rangle) = \mathbf{x}$.

Let us write $x^{(j)} \, [t\rangle \, y^{(j)}$ there is a reachable marking $M_x \in \mathrm{RS}$ such that

$$M_x|_{P^{(j)}} = x^{(j)}, \quad M_x \, [t\rangle \, M_y, \quad M_y|_{P^{(j)}} = y^{(j)},$$

i.e. there is a global state transition that takes the jth local net from the state $x^{(j)}$ to $y^{(j)}$. Superposed stochastic Petri nets are *Kronecker consistent*: if $x^{(j)} \, [t\rangle \, y^{(j)}$ and $x^{(j)} \, [t\rangle \, z^{(j)}$, then $y^{(j)} = z^{(j)}$.

Example 2. The *SharedResource* SPN in Fig. 1 is contains three partitions. The local nets $P^{(0)}$ and $P^{(1)}$ correspond to the two clients, while $P^{(2)}$ contains the shared resource. The local state spaces of the components are

$$RS^{(0)} = \begin{cases} 0^{(0)} = \{C_1\} \\ 1^{(0)} = \{W_1\} \\ 2^{(0)} = \{S_1\} \end{cases}, \quad \mathrm{RS}^{(1)} = \begin{cases} 0^{(1)} = \{C_2\} \\ 1^{(1)} = \{W_2\} \\ 2^{(1)} = \{S_2\} \end{cases}, \quad \mathrm{RS}^{(2)} = \begin{cases} 0^{(2)} = \{S\} \\ 1^{(2)} = \emptyset \end{cases}.$$

The reeachable states can be factored over the local state spaces, e.g. the marking $M_4 = \{S_1, C_2\} = \langle\!\langle 2^{(0)}, 0^{(1)}, 1^{(2)} \rangle\!\rangle$. However, the potential state space PS contains $|\mathrm{RS}^{(0)}| \cdot |\mathrm{RS}^{(1)}| \cdot |\mathrm{RS}^{(1)}| = 3 \cdot 3 \cdot 2 = 18$ states, 10 more than the reachable state space RS. For example, the marking $\langle\!\langle 2^{(0)}, 2^{(1)}, 0^{(2)} \rangle\!\rangle = \{S_1, S_2, S\}$, which violates mutual exclusion, is not reachable, although it is in PS.

2.3 Decision Diagrams

Multivalued decision diagrams (MDDs) [13] provide a compact, graph-based representation for boolean functions defined over Cartesian products of domains.

Definition 3. *A quasi-reduced ordered* multivalued decision diagram *(MDD) encoding the function $f(x[0], x[1], \ldots, x[J-1]) \in \{0, 1\}$, where the domain of each variable $x^{(j)}$ is $D^{(j)} = \{0, 1, \ldots, n_j - 1\}$, is a tuple $MDD = \langle V, \underline{r}, \underline{0}, \underline{1}, level, child \rangle$, where*

- *$V = \bigcup_{i=0}^{J} V_i$ is a finite set of* nodes*, where $V_0 = \{\underline{0}, \underline{1}\}$ are the* terminal nodes*, the rest of the nodes $V_N = V \setminus V_0$ are* nonterminal nodes*;*
- *level*: $V \to \{0, 1, \ldots, J\}$ *assigns nonnegative* level numbers *to each node, i.e. $V_i = \{\underline{v} \in V : level(\underline{v}) = i\}$;*
- *$\underline{v} \in V_J$ is the* root node*;*
- *$\underline{0}, \underline{1} \in V_0$ are the* zero *and* one terminal nodes*;*
- *child*: $\left(\bigcup_{i=1}^{J} V_i \times D^{(i-1)}\right) \to V$ *is a function defining edges between nodes labeled by the items of the domains, such that either $child(\underline{v}, x) = \underline{0}$ or $level(child(\underline{v}, x)) = level(\underline{v}) - 1$ for all $\underline{v} \in V$, $x \in D^{(level(\underline{v})-1)}$;*
- *if $\underline{n}, \underline{m} \in V_j, j > 0$ then the subgraphs formed by the nodes reachable from $\underline{n}$ and $\underline{m}$ are either non-isomorphic, or $\underline{n} = \underline{m}$.*

According to the semantics of MDDs, $f(\mathbf{x}) = 1$ if the node $\underline{1}$ is reachable from $\underline{r}$ through the edges labeled with $x[0], x[1], \ldots, x[J-1]$,

$$f(x[0], x[1], \ldots, x[J-1]) = 1 \iff child(child(\ldots child(\underline{r}, x[J-1]) \ldots, x[1]), x[0]) = \underline{1}.$$

Definition 4. *A quasi-reduced ordered* edge-valued multivalued decision diagram *(EDD) [27] encoding the function $g(x^{(0)}, x^{(1)}, \ldots, x^{(J-1)}) \in \mathbb{N}$ is a tuple $EDD = \langle V, \underline{r}, \underline{0}, \underline{1}, level, child, label \rangle$, where*

- *$MDD = (V, \underline{r}, \underline{0}, \underline{1}, level, child)$ is a quasi-reduced ordered MDD;*

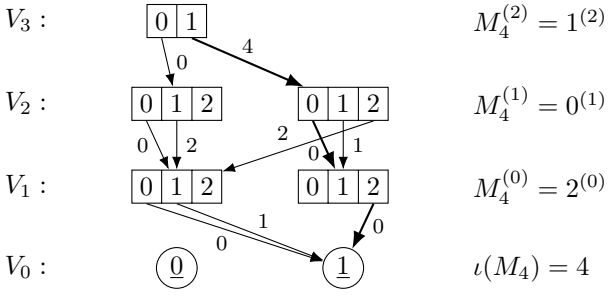

Fig. 3. EDD state space mapping of the *SharedResource* model.

– *label*: $\left(\bigcup_{i=1}^{J} V_i \times D^{(i-1)}\right) \to \mathbb{N}$ *is an edge label function.*

According to the semantics of EDDs, the function g is evaluated as

$$g(\mathbf{x}) = \begin{cases} \text{undefined} & \text{if } f(\mathbf{x}) = 0, \\ \sum_{j=0}^{J-1} \mathit{label}(\underline{n}^{(j)}, x[j]) & \text{if } f(\mathbf{x}) = 1, \end{cases}$$

where f is the function associated with the underlying MDD and $\underline{n}^{(j)}$ are the nodes along the path to $\underline{1}$, i.e. $\underline{n}^{(J-1)} = \underline{r}$, $\underline{n}^{(j)} = \mathit{child}(\underline{n}^{(j+1)}, x[j+1])$.

Symbolic State Spaces. Symbolic techniques involving MDDs can efficiently store large reachable state spaces of superposed Petri nets. Reachable states $M \in \mathrm{RS}$ are associated with state codings $\boldsymbol{\iota}(M) = \mathbf{x}$ The function $f : \mathrm{PS} \to \{0, 1\}$ can be stored as an MDD where $f(\mathbf{x}) = 1$ if and only if $\langle\!\langle \mathbf{x} \rangle\!\rangle \in \mathrm{RS}$. The domains of the MDD are the local state spaces $D^{(j)} = \mathrm{RS}^{(j)}$.

Similarly, EDDs can efficiently store the mapping between symbolic state encodings $\mathbf{x}$ and reachable state indices $x = \iota(\langle\!\langle \mathbf{x} \rangle\!\rangle)$ as the function $g(\mathbf{x}) = x$. This mapping is used to refer to elements of state probability vectors and the sparse generator matrix Q when these objects are created and accessed [12].

Some iteration strategies for MDD state space exploration are *breath-first search* and *saturation* [13].

Example 3. The EDD displayed in Fig. 3 describes the reachable state space of the *SharedResource* model from Example 2. Edges to $\underline{0}$ were omitted for clarity.

The edge labels allow computation of the indexing function ι for a given state. For example, to find the index of the marking $M_4 = \langle\!\langle 2^{(0)}, 0^{(1)}, 1^{(2)} \rangle\!\rangle$, we can follow the edges corresponding to the local states to $\underline{1}$ and sum their labels to find $\iota(M_4) = 4 + 0 + 0 = 4$. In contrast, if we follow the edges corresponding to the unreachable state $\langle\!\langle 2^{(0)}, 2^{(1)}, 0^{(2)} \rangle\!\rangle$, $\underline{0}$ is reached instead of $\underline{1}$.

2.4 Kronecker Algebra

In linear algebra, the Kronecker product operation may be used to build large matrices from smaller ones. It therefore plays an important role in the stochastic

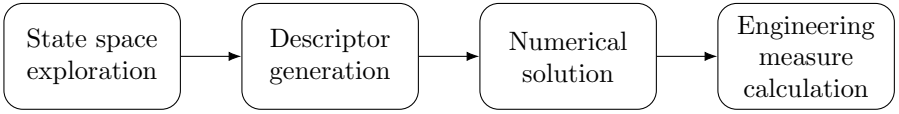

Fig. 4. Stochastic analysis workflow.

analysis of Markovian systems, where Kronecker products of matrices may help reducing the memory requirements of the infinitesimal generator matrix Q.

Definition 5. *The* Kronecker product $A \otimes B$ *of matrices* $A \in \mathbb{R}^{n_1 \times m_1}$ *and* $B \in \mathbb{R}^{n_2 \times m_2}$ *is the matrix* $C = \in \mathbb{R}^{n_1 n_2 \times m_1 m_2}$*, where*

$$c[i_1 n_1 + i_2, j_1 m_1 + j_2] = a[i_1, j_1] b[i_2, j_2].$$

The Shuffle family of algorithms [4] allows efficient evaluation of vector-matrix products of the form

$$\mathbf{v} \cdot (A^{(0)} \otimes A^{(1)} \otimes \ldots \otimes A^{(J-1)}).$$

The factors $A^{(j)} \in \mathbb{R}^{n_j \times m_j}$ together represent an $n_0 n_1 \cdots n_{J-1} \times m_0 m_1 \cdots m_{J-1}$ matrix in the J-way Kronecker product.

Recent developments include the Slice [16] and Split [14] algorithms for vector-descriptor products, which allow parallel implementation while retaining the beneficial properties of the Shuffle algorithm.

3 Stochastic Petri Net Stationary Analysis

In this section the used stochastic analysis approach is introduced.

3.1 Analysis Workflow

The tasks performed by stochastic analysis tools that operate on stochastic petri nets, such as Fig. 4, can be often structured as follows:

1. *State space exploration.* The reachable state space of the Petri net is explored to enumerate the possible behaviors of the model. For superposed stochastic Petri nets, this step includes the exploration of the local state spaces of the component as well as the possible global combinations of states.
2. *Descriptor generation.* The *infinitesimal generator matrix* Q is buildt in order to describe the Markov chain $X(t)$ over the reachable states of the stochastic Petri net.
3. *Numerical solution.* Numerical algorithms obtain probability vectors $\boldsymbol{\pi}$ from the matrix Q.
4. *Engineering measure calculation.* The studied performance measures are calculated from the output of the previous step. The expected values of most measures of interest can be obtained as weighted sums of state probabilities.

In stochastic model checking, where the desired system behaviors are expressed in stochastic temporal logics [1, 5], these analytic steps are called as subrouties to evaluate propositions.

In the steady-state analysis of continuous-time Markovian stochastic systems, the steady state solution

$$\boldsymbol{\pi}(0) = \boldsymbol{\pi}_0, \quad \frac{\partial \boldsymbol{\pi}(\tau)}{\partial \tau} = \boldsymbol{\pi}(\tau)\, Q, \quad \boldsymbol{\pi} = \lim_{\tau \to \infty} \boldsymbol{\pi}(\tau) \tag{3}$$

of Eq. 1 is sought, where $\boldsymbol{\pi}_0$ describes the initial probability distribution and $\boldsymbol{\pi}$ is the stationary solution. If the CMTC is irreducible, i.e. there is a nonzero probability of transitioning from any state to any other, $\boldsymbol{\pi}$ is independent from $\boldsymbol{\pi}_0$ and is the initial solution of the system of linear equations

$$\boldsymbol{\pi}\, Q = \mathbf{0}, \quad \boldsymbol{\pi} \mathbf{1}^{\mathrm{T}} = 1. \tag{4}$$

Example 4. The utilization of the shared resource in the *SharedResource* SPN, presented in Fig. 1, can be calculated as the sum

$$U = \pi[\iota(M_4)] + \pi[\iota(M_5)] + \pi[\iota(M_6)] + \pi[\iota(M_7)]$$

after obtaining $\boldsymbol{\pi}$ from Eq. (4). Notice that the resource is in use in the reachable markings M_4, M_5, M_6 and M_7.

To solve Eq. (4), the matrix Q and the vector $\boldsymbol{\pi}$ must be stored. Additionally, the numerical algorithm may reserve additional vectors for intermediate storage.

3.2 Infinitesimal Generator Matrix Storage

In this section the basic complexity issues behind our developments are summarized. Traditional methods use sparse or dense matrix storage methods.

Dense and Sparse Matrices. The infinitesimal generator matrix Q of a stochastic Petri net is a $n \times n$ matrix of real numbers, where n is the number of reachable states. The storage of Q thus requires memory proportional to the square of the state space size if a dense matrix form is used.

Sparse matrix representation [2, Section 4.3.1], reduces memory requirements to $O(\mathrm{NZ})$, where NZ is the number of nonzero elements in Q.

Kronecker decomposition can reduce the storage requirements by dividing big monolithic matrices into smaller pieces.

Kronecker Decomposition. To alleviate the high memory requirements of Q, the Kronecker decomposition for a superposed SPN expresses the infinitesimal generator matrix as a sum of Kronecker products. Let

$$Q = Q_O + Q_D, \quad Q_D = \mathrm{diag}\{-Q\mathbf{1}^{\mathrm{T}}\}, \tag{5}$$

where Q_O and Q_D are the off-diagonal and diagonal parts of Q, respectively. The off-diagonal part may be written as

$$Q_O = \sum_{t \in T} \Lambda(t) \bigotimes_{j=0}^{J-1} Q_t^{(j)}. \tag{6}$$

The matrix $Q_t^{(j)} \in \mathbb{R}^{n_j \times n_j}$ describes the effects of the transition t on the jth local net. If $t \in p^\bullet \cup {}^\bullet p \cup p^\circ$ for some $p \in P^{(j)}$, i.e. t is connected to a place in the jth local net with an input, output or inhibitor arc,

$$q_t^{(j)}[x, y] = \begin{cases} 1 & \text{if } x^{(j)} \; [t\rangle \; y^{(j)}, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

If t is not adjacent to the jth local net, $Q_t^{(j)}$ is set to an $n_j \times n_j$ identity matrix.

3.3 Block Kronecker Decomposition

In this section, we review the concept of block Kronecker decomposition and hierarchical structuring of the state space from [6].

Let $\widetilde{\mathrm{RS}}^{(j)}$ denote the *local macro states* of the jth local net of an SSPN. Elements of $\widetilde{\mathrm{RS}}^{(j)}$ form a partition of the local state space $\mathrm{RS}^{(j)}$ of the jth component, i.e. $\mathrm{RS}^{(j)} = \bigsqcup \widetilde{\mathrm{RS}}^{(j)}$.

The macro state indexing function $\tilde{\iota}^{(j)} \colon \widetilde{\mathrm{RS}}^{(j)} \to \{0, 1, \ldots, \tilde{n}_j - 1\}$ assigns a unique index to every macro state, where $\tilde{n}_j = |\widetilde{\mathrm{RS}}^{(j)}|$. We use the notation $\widetilde{m}^{(j)}$ to refer to $\left(\tilde{\iota}^{(j)}\right)^{-1}(m) \subseteq \mathrm{RS}^{(j)}$.

A vector $\mathbf{m} = \langle m[0], m[1], \ldots, m[J-1] \rangle$ is a *global macro state index* if

$$\langle\!\langle \widetilde{m[0]}^{(0)}, \widetilde{m[1]}^{(1)}, \ldots, \widetilde{m[J-1]}^{(J-1)} \rangle\!\rangle \subseteq \mathrm{RS},$$

i.e. a subset of reachable markings is isomorphic to a Cartesian product of local macro states. Such subset is called a *global macro state*.

Hierarchical structuring of the reachable state space expresses RS as a disjoint union of global macro states $\widetilde{\mathrm{RS}} = \{\tilde{0}, \tilde{1}, \ldots, \widetilde{\tilde{n}-1}\}$.

The bijection $\tilde{\iota} \colon \widetilde{\mathrm{RS}} \to \{0, 1, \ldots, \tilde{n}-1\}$ assigns indices to global macro states such that $\tilde{\iota}^{-1}(m) = \widetilde{m}$, while the function $\tilde{\boldsymbol{\iota}} \colon \widetilde{\mathrm{RS}} \to \mathbb{N}^J$ assings their respective global macro state index vectors.

Since each global macro state has the structure of a Cartesian product of local macro states, the Kronecker product may be used to construct a matrix of state transitions between two global macro states.

Let $\mathbf{m}$ and $\mathbf{k}$ be two reachable global macro state indices. The matrix $Q_t^{(j)}[m[j], k[j]]$ is obtained from $Q_t^{(j)}$ in Eq. (7) by only keeping the rows that correspond to $m[j]^{(j)}$ and the columns that correspond to $k[j]^{(j)}$. The matrix

$$Q_O[m, k] = \sum_{t \in T} \Lambda(t) \bigotimes_{j=0}^{J-1} Q_t^{(j)}[m[j], k[j]] \tag{8}$$

describes the states transitions from the macro state $\tilde{\iota}^{-1}(m)$ to $\tilde{\iota}^{-1}(k)$, where m and k are the indices of global macro states with index vectors $\mathbf{m}$ and $\mathbf{k}$, respectively.

We can finally express the infinitesimal generator matrix Q as a block matrix with $\tilde{n} \times \tilde{n}$ blocks. Let the $\langle m, k \rangle$th block of the off-diagonal part Q_O be $Q_O[m, k]$ as defined in Eq. (8) above. Then it can be seen that the matrix

$$Q = Q_O + Q_D, \quad Q_D = \operatorname{diag}\{-Q\mathbf{1}^{\mathrm{T}}\} \tag{9}$$

is equivalent to the matrix Q in Eq. (2).

Example 5. The state space of the *SharedResource* model in Fig. 1 may be hierarchically structured as follows.

Recall from Example 2 that the model has three local nets, two corresponding to the clients with three local states each and a local net with two local states corresponding to the shared resource. We can partition the local state spaces into local macro states as

$$\widetilde{RS}^{(0)} = \begin{cases} \tilde{0}^{(0)} = \{0^{(0)}, 1^{(0)}\} \\ \tilde{1}^{(0)} = \{2^{(0)}\} \end{cases}, \quad \widetilde{\mathrm{RS}}^{(1)} = \begin{cases} \tilde{0}^{(1)} = \{0^{(1)}, 1^{(1)}\} \\ \tilde{1}^{(1)} = \{2^{(1)}\} \end{cases}, \quad \widetilde{\mathrm{RS}}^{(2)} = \begin{cases} \tilde{0}^{(2)} = \{0^{(2)}\} \\ \tilde{1}^{(2)} = \{1^{(2)}\} \end{cases}.$$

Observe that for each component j, the macro state $\tilde{0}^{(j)}$ contains local states that are reachable when the shared resource is not in use, while $\tilde{1}^{(j)}$ corresponds to the allocation of the resource.

The global reachable state space is partitioned into global macro states as

$$\widetilde{\mathrm{RS}} = \{\langle\!\langle \tilde{0}^{(0)}, \tilde{0}^{(1)}, \tilde{0}^{(2)} \rangle\!\rangle, \langle\!\langle \tilde{0}^{(0)}, \tilde{1}^{(1)}, \tilde{1}^{(2)} \rangle\!\rangle, \langle\!\langle \tilde{1}^{(0)}, \tilde{0}^{(1)}, \tilde{1}^{(2)} \rangle\!\rangle\},$$

i.e. any state that does not require use of the shared resource is reachable when the shared resource is available, while if one of the clients allocates the resource, the other cannot simultaneously acquire it.

Macro State Construction. Let us introduce the notation

$$\hat{\mathbf{x}}^{(j)} = \langle x[0], x[1], \ldots, x[j-1], x[j+1], \ldots, x[J-1] \rangle.$$

Suppose that the reachable marking M is coded by the vector of local state indices $\mathbf{x} = \boldsymbol{\iota}(M)$, i.e. $M = \langle\!\langle \mathbf{x} \rangle\!\rangle$. Then the vector $\hat{\mathbf{x}}^{(j)}$ contains the local state indices of all components except the jth.

The *environment* of a local state $x^{(j)}$ is the set of vectors

$$\operatorname{env} x^{(j)} = \{\hat{\mathbf{z}}^{(j)} : M \in \mathrm{RS}, \mathbf{z} = \boldsymbol{\iota}(M), z[j] = x\}, \tag{10}$$

i.e. the local state combinations of other local nets that result in reachable markings together with $x^{(j)}$. We define the equivalence relation $\sim^{(j)} \subseteq \mathrm{RS}^{(j)} \times \mathrm{RS}^{(j)}$,

$$x^{(j)} \sim^{(j)} y^{(j)} \quad \Longleftrightarrow \quad \operatorname{env} x^{(j)} = \operatorname{env} y^{(j)}. \tag{11}$$

Algorithm 1: Bit array based macro state construction.

Input: Reachable state space RS, reachable local states $\mathrm{RS}^{(j)}$
Output: Local macro states $\widetilde{\mathrm{RS}}^{(j)}$, global macro states $\widetilde{\mathrm{RS}}$

1 Allocate an array of bits $\mathbf{B} \in \{0,1\}^{n_0 \times n_1 \times \cdots \times n_{J-1}}$
2 **foreach** $M \in RS$ **do** $\mathbf{x} \leftarrow \iota(M)$; $B[x[0], x[1], \ldots, x[J-1]] \leftarrow 1$
3 **for** $j \leftarrow 0$ **to** $J-1$ **do**
4 Reshape $\mathbf{B}$ into a matrix with n_j, where the xth row corresponds to the local state $x^{(j)} \in \mathrm{RS}^{(j)}$ and its environment $\operatorname{env} x^{(j)}$
5 Sort the rows of the matrix lexicographically
6 Partition the rows such that equal rows form local macro states $\widetilde{\mathrm{RS}}^{(j)}$
7 Discard all but one representant row for each local macro state $\widetilde{m}^{(j)}$
8 Nonzero elements of the resulting bit array $\mathbf{B}$ are correspond to the reachable global macro states $\widetilde{\mathrm{RS}}$

The equivalence classes $\mathrm{RS}^{(j)}/{\sim^{(j)}}$ are precisely the local macro states $\widetilde{\mathrm{RS}}^{(j)}$. Now we can construct the equivalence relation $\sim \subseteq \mathrm{RS} \times \mathrm{RS}$,

$$M \sim K \quad \Longleftrightarrow \quad M^{(j)} \sim^{(j)} K^{(j)} \text{ for all } j = 0, 1, \ldots, J-1,$$

i.e. two markings are equivalent if all their local nets belong to the same local macro states. The set of global macro states is the partition $\widetilde{\mathrm{RS}} = \mathrm{RS}/{\sim}$.

Explicit Macro State Algorithm. The original hierarchical structuring algorithm proposed by Buchholz [6] is based on a bit array representation of the potential state space PS.

Algorithm 1 shows local and global macro state generation. The environment $\operatorname{env} x^{(j)}$ is represented explicitly as a row of the reshaped bit array $\mathbf{B}$. Nonzero elements correspond to reachable states.

Lexicographic ordering is used to make local states with equal environments adjacent. After extracting local macro states from the bit array, only a single representant of every local macro state is kept. This both accelerates further iterations of the algorithm and results in reduced bit array at the end that has a nonzero element for every combination of local states that is reachable. Note that the permutations used to reorder local states must be stored in order to recover the global macro state indices.

Storage of the bit array requires $O(|\mathrm{PS}|)$ memory, therefore the bit array based macro state generation is unsuitable for extremely large potential state spaces due to memory requirements.

4 Symbolic Decomposition Algorithm

In this section, we present our symbolic decomposition algorithm that allows the construction of macro state spaces without explicit enumeration and storage of the potential state space PS.

Algorithm 2: Local macro state construction by partition refinement.

Input: Symbolic state space MDD
Output: Local macro states $\widetilde{\mathrm{RS}}^{(j)}$

```
1  for j ← 0 to J − 1 do
2      Initialize the empty queue Q and Done ← {RS^(j)}
3      foreach n ∈ V_{j+1} do
4          foreach S ∈ Done do ENQUEUE(Q, S)
5          Done ← ∅
6          while ¬EMPTY(Q) do
7              S ← DEQUEUE(Q); S_1 ← ∅; S_2 ← ∅
8              Let x_0 be any element of S and m ← child(n, x_0)
9              foreach x ∈ S \ {x_0} do
10                 if m = child(n, x) then S_1 ← S_1 ∪ {x} else S_2 ← S_2 ∪ {x}
11             if S_2 ≠ ∅ then ENQUEUE(Q, S_2)
12             Done ← Done ∪ {S_1}
13     ñ_j ← |Done|; RS̃^(j) ← Done
```

4.1 Description

The memory requirements and run time of bit array based macro state decomposition may be significantly improved by the use of symbolic state space storage instead of a bit vector.

The macro states can be constructed from the paralell edges in the MDD by partition refinement. This process is shown in Algorithm 2.

The key step in partition refinement is in line 10, where the candiate macro state S is split into S_1 and S_2. Edges in S_1 are all parallel and go from $\underline{n}$ to $\underline{m}$, while S_2 is further split. The process is repeated for each node $\underline{n}$ and level V_{j+1} until only parallel macro state candidates remain.

This procedure is based on an idea of Buchholz [9]. We employed partition refinement instead of hashing and proved correctness of the algorithm formally.

Algorithm 2, unlike Algorithm 1, does not output the global macro state space $\widetilde{\mathrm{RS}}$. An MDD representation of $\widetilde{\mathrm{RS}}$ may be obtained by replacing every arc label x with the index m of the local macro state with $x^{(j)} \in \widetilde{m}^{(j)}$.

Example 6. Fig. 5 shows the MDD state space of the *SharedResource* model, its local macro state decomposition and the MDD representation of the global macro states.

The edges corresponding to local states that belong to the same local macro state are parallel for *all* parent nodes on a level. This is represented in the figure with ellipses connecting the edges. In the global macro state MDD, these parallel edge sets collapse to a single edge, as every edge from the original MDD is replaces with its macro state.

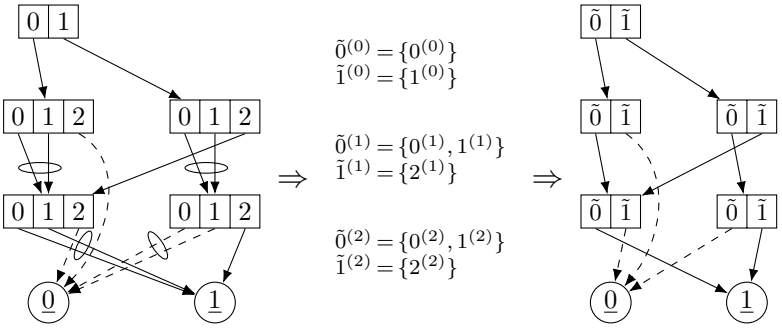

Fig. 5. Symbolic macro state construction for the *SharedResource* model.

4.2 Proof of Correctness

To show the correctness of the partition of the local states $\mathrm{RS}^{(j)}$ into macro states $\widetilde{\mathrm{RS}}^{(j)}$, we will use the notations of above and below substates from [13]:

Definition 6. *The set of* above *substates coded by the node* $\underline{n}$ *is*

$$\mathcal{A}(\underline{n}) \subseteq \{\langle x[j+1], x[j+1], \ldots, x[j-1]\rangle \in D^{(j+1)} \times D^{(j+2)} \times \cdots \times D^{(J-1)}\},$$
$$\mathbf{x} \in \mathcal{A}(\underline{n}) \iff \mathit{child}(\mathit{child}(\ldots \mathit{child}(\underline{r}, x[J-1]) \ldots, x[j+2]), x[j+1]) = \underline{n},$$

where $j = \mathit{level}(\underline{n}) - 1$*, i.e.* $\mathcal{A}(\underline{n})$ *is the set of all paths in the MDD from* $\underline{r}$ *to* $\underline{n}$*.*

Definition 7. *The set of* below *substates coded by the node* $\underline{n}$ *is*

$$\mathcal{B}(\underline{n}) \subseteq \{\langle x[0], x[1], \ldots, x[j]\rangle \in D^{(0)} \times D^{(1)} \times \cdots \times D^{(j)}\},$$
$$\mathbf{x} \in \mathcal{B}(\underline{n}) \iff \mathit{child}(\mathit{child}(\ldots \mathit{child}(\underline{n}, x[j]) \ldots, x[1]), x[0]) = \underline{1},$$

where $j = \mathit{level}(\underline{n}) - 1$*, i.e.* $\mathcal{B}(\underline{n})$ *is the set of all paths in the MDD from* $\underline{n}$ *to* $\underline{1}$*.*

Proposition 1. *If* $\underline{n}$ *and* $\underline{m}$ *are distinct nonterminal nodes of a quasi-reduced ordered MDD,* $\mathcal{A}(\underline{n}) \cap \mathcal{A}(\underline{m}) = \emptyset$ *and* $\mathcal{B}(\underline{n}) \neq \mathcal{B}(\underline{m})$*.*

Proof. We prove the statements by contradiction. Let $\mathbf{a} \in \mathcal{A}(\underline{n}) \cap \mathcal{A}(\underline{m})$. If we follow the path $\mathbf{a}$ from $\underline{r}$ we arrive at $\underline{n}$ because $\mathbf{a} \in \mathcal{A}(\underline{n})$. However, we also arrive at $\underline{m}$, because $\mathbf{a} \in \mathcal{A}(\underline{m})$. Since $\underline{n} \neq \underline{m}$, such $\mathbf{a}$ cannot exist. $\mathcal{A}(\underline{n})$ and $\mathcal{A}(\underline{m})$ must be disjoint.

Now suppose that there are $\underline{n}, \underline{m} \in V_N$ such that $\mathcal{B}(\underline{n}) = \mathcal{B}(\underline{m})$. Because the paths fully $\mathcal{B}(\underline{n})$ describe the subgraph reachable from $\underline{n}$, this means the subgraphs reachable from $\underline{n}$ and $\underline{m}$ are isomorphic. This is impossible since the MDD is quasi-reduced. $\mathcal{B}(\underline{n})$ and $\mathcal{B}(\underline{m})$ must be distinct. □

Proposition 2. *The* environment *of the local state* $x^{(j)}$*, defined in Eq.* (10)*, may be written as*

$$\mathrm{env}\, x^{(j)} = \{\langle\langle \mathbf{b}, \mathbf{a}\rangle\rangle : \underline{n} \in V_{j+1},\, \mathbf{a} \in \mathcal{A}(\underline{n}),\, \mathbf{b} \in \mathcal{B}(\mathit{child}(\underline{n}, x^{(j)}))\}.$$

Proof. Any reachable state $\langle\langle \mathbf{z} \rangle\rangle \in \mathrm{RS}$ that has $z[j] = x$ is represented by a path from $\underline{r}$ to $\underline{1}$ in the MDD that passes through a pair of nodes $\underline{n} \in V_{j+1}$ and $\underline{k} = \mathit{child}(\underline{n}, x^{(j)})$. Therefore, some path $\mathbf{a} \in \mathcal{A}(\underline{n})$ must be followed from $\underline{r}$ to reach $\underline{n}$, then after traversing the edge between $\underline{n}$ and $\underline{k}$, some path $\mathbf{b} \in \mathcal{B}(\underline{k})$ must be followed from $\underline{k}$ to $\underline{1}$.

This means all paths from $\underline{r}$ to $\underline{1}$ containing $x^{(j)}$ are of the form $\mathbf{z} = (\mathbf{b}, x, \mathbf{a})$ and the converse also holds. Thus, $\hat{\mathbf{z}}^{(j)} = \langle \mathbf{a}, \mathbf{b} \rangle$ holds for some $\mathbf{a}$ and $\mathbf{b}$ defined as above for all reachable states $\mathbf{z}$. □

The relation $\sim^{(j)}$ over $\mathrm{RS}^{(j)}$ can be expressed with $\mathcal{A}(\underline{n})$ and $\mathcal{B}(\underline{n})$ is a way that can be handled with symbolic techniques.

Proposition 3. *The relation $x^{(j)} \sim^{(j)} y^{(j)}$ can be formulated as*

$$x^{(j)} \sim^{(j)} y^{(j)} \quad \Longleftrightarrow \quad \mathit{edges}(x, j) = \mathit{edges}(y, j),$$
$$\text{where } \mathit{edges}(z, j) = \{\langle \underline{n}, \mathit{child}(\underline{n}, z) \rangle : \underline{n} \in V_{j+1}\}.$$

Proof. Recall that $x^{(j)} \sim^{(j)} y^{(j)}$ is defined as $\mathrm{env}\, x^{(j)} = \mathrm{env}\, y^{(j)}$ in Eq. (11). Let X and Y be the environments of $x^{(j)}$ and $y^{(j)}$, respectively, such that $x^{(j)} \sim^{(j)} y^{(j)}$ holds if and only if $X = Y$. Define

$$X(\underline{n}) = \{\mathbf{b} : \langle \mathbf{b}, \mathbf{a} \rangle \in X, \mathbf{a} \in \mathcal{A}(\underline{n})\}, \qquad Y(\underline{n}) = \{\mathbf{b} : \langle \mathbf{b}, \mathbf{a} \rangle \in Y, \mathbf{a} \in \mathcal{A}(\underline{n})\}.$$

Observe that $X = Y$ if and only if $X(\underline{n}) = Y(\underline{n})$ for all $\underline{n} \in V_{j+1}$, because the sets $\{X(\underline{n}) \times \mathcal{A}(\underline{n})\}_{\underline{n} \in V_{j+1}}$ and $\{Y(\underline{n}) \times \mathcal{A}(\underline{n})\}_{\underline{n} \in V_{j+1}}$ are partitions of X and Y.

According to Proposition 2,

$$X(\underline{n}) = \mathcal{B}(\mathit{child}(\underline{n}, x)), \qquad Y(\underline{n}) = \mathcal{B}(\mathit{child}(\underline{n}, y)).$$

Thus, $X(\underline{n}) = Y(\underline{n})$ if and only if $\mathit{child}(\underline{n}, x) = \mathit{child}(\underline{n}, y)$, because the $\mathcal{B}$-sets are distinct for each node. Hence $X(\underline{n}) = Y(\underline{n})$ for all $\underline{n} \in V_{j+1}$ is equivalent to the statement $\mathit{edges}(x, j) = \mathit{edges}(y, j)$. □

Proposition 3 can be interpreted as the statement that $x^{(j)} \sim^{(j)} y^{(j)}$ if and only if the MDD edges corresponding to $x^{(j)}$ are always parallel, i.e. from the node $\underline{n}$ they all go to the same node $\underline{m}(\underline{n})$ for all $\underline{n} \in V_{j+1}$.

4.3 Symbolic State Indexing

The last step of the stochastic analysis workflow is the calculation of engineering measures by weighted sums of state probabilities, which were obtained by the numerical solution algorithm. This requires iteration over all states and the corresponding indices of the probability vector.

In this section, we review techniques for state-index mapping in superposed stochastic Petri nets. An implementation of the mapping is presented for macro state decompositions in symbolic form.

The EDD-based state-index mapping is depicted in Fig. 6. A vector of local macro state indices is formed by observing the local states of the components.

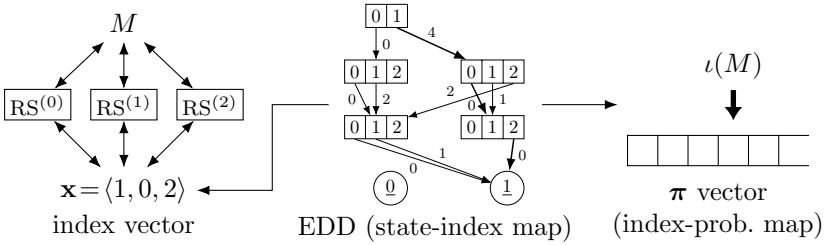

Fig. 6. Index manipulations for EDD-based state spaces.

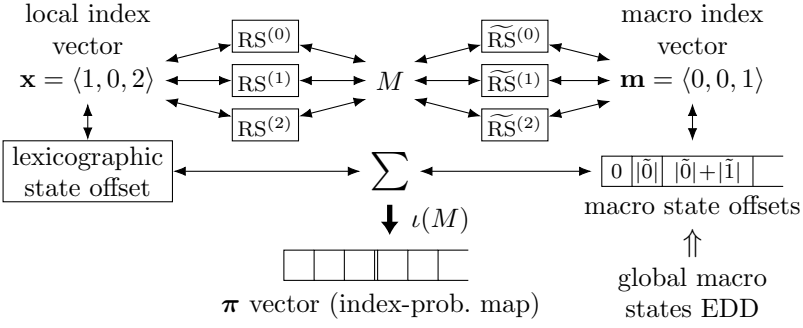

Fig. 7. Index manipulations for EDD-based macro state spaces.

Both forward and reverse mapping between linear indices $\iota(M)$ of the probability vector and index vectors $\mathbf{x}$ is possible [12].

In block Kronecker decompositions, generator matrix, as well as the vector $\boldsymbol{\pi}$ is partitioned according to the global macro states $\widetilde{\mathrm{RS}}$. Within a partition, markings are ordered lexicographicall by their local state indices. Although it is possible to construct an EDD representing this ordering, it can easily grow large, due to the intertwined ordering.

To efficiently store the index mapping (Fig. 7), a macro state offset vector $\mathbf{o}$ of length $\tilde{n}$ is populated with the starting offsets of the global states by enumeration of the global macro state space. The ith marking in the lexicographic ordering within the global macro state $\widetilde{m}$ has the linear index $x = o[m] + i$. Conversion between linear macro state indices m and macro state index vectors $\mathbf{m}$ is performed by an EDD, analogously to the mapping in Fig. 6.

5 Evaluation

The decomposition algorithms were implemented in the PetriDotNet modelling framework, which supports structural analysis of Petri nets, saturation-based CTL and LTL model checking and reachability checking as well as Markovian

Table 1. Measurement results.

		Sparse			Explicit BK				Symbolic BK			
Model	States	t_{gen}	t_{solve}	M_{peak}	t_{macro}	t_{gen}	t_{solve}	M_{peak}	t_{macro}	t_{gen}	t_{solve}	M_{peak}
SR-Sym	26 464	147 ms	**296 ms**	40 M	464 ms	562 ms	303 ms	55 M	**23 ms**	**142 ms**	370 ms	**38 M**
	842 051	2 s	**10 s**	296 M	22 s	11 s	**10 s**	1035 M	**23 ms**	**582 ms**	12 s	**98 M**
	$1.1 \cdot 10^7$	24 s	182 s	3404 M	530 s	183 s	231 s	12 G	**24 ms**	**5 s**	**155 s**	**791 M**
	$8.2 \cdot 10^7$	215 s	1985 s	25 G	Timed out				**26 ms**	**31 s**	**1612 s**	**5563 M**
SR-Asym	26 464	132 ms	**482 ms**	42 M	455 ms	402 ms	580 ms	57 M	**26 ms**	**125 ms**	638 ms	**39 M**
	842 051	2 s	**17 s**	296 M	22 s	10 s	20 s	1035 M	**23 ms**	**519 ms**	21 s	**99 M**
	$1.1 \cdot 10^7$	22 s	327 s	3404 M	540 s	246 s	463 s	12 G	**27 ms**	**6 s**	**285 s**	**793 M**
	$8.2 \cdot 10^7$	217 s	Timed out		Timed out				**31 ms**	**32 s**	Breakdown	
KanBan	58 400	181 ms	**1 s**	51 M	250 ms	847 ms	3 s	77 M	**24 ms**	**130 ms**	2 s	**45 M**
	454 475	1 s	**11 s**	181 M	1 s	4 s	31 s	448 M	**26 ms**	**285 ms**	19 s	**171 M**
	$2.5 \cdot 10^7$	5 s	**81 s**	**388 M**	6 s	28 s	240 s	2276 M	**22 ms**	**1 s**	124 s	780 M
	$1.1 \cdot 10^8$	33 s	**436 s**	3727 M	31 s	138 s	1274 s	10 G	**25 ms**	**5 s**	737 s	**3591 M**
Cloud	$2.7 \cdot 10^6$	38 s	446 s	5846 M	119 s	48 s	913 s	3361 M	**64 ms**	**4 s**	**278 s**	**291 M**
	$2.0 \cdot 10^7$	Out of memory			2021 s	464 s	Timed out		**29 ms**	**9 s**	Timed out	
	$1.3 \cdot 10^8$	Out of memory			Timed out				**32** ms	**75 s**	Timed out	
	$8.2 \cdot 10^8$	Out of memory			Timed out				**42 ms**	**574 s**	Out of memory	

stochastic analysis. The algorithms may be used in Markovian transient, sensitivity and time-to-first-failure analysis in addition to steady state analysis.

Our symbolic block Kroncker decomposition approach is compared with sparse generator matrices and the explicit block Kronecker decomposition algorithm by Buchholz [6]. The explicit decomposition algorithm was executed on an explicit representation of the state space as a hash table, while symbolic algorithms used saturation and MDDs. Steady-state analysis was performed using the BiCGSTAB [30] numerical linear equation solver with a tolerance of 10^{-10}.

Running time was limited to 1 hour on virtual machines with 8 execution threads and 30 GiB memory.

Three scalable families of models were used in the evaluation:

- The *SR* family of models are upscaled variants of the *SharedResource* model that served as a running example throughout this paper. The model was extended with additional clients, moreover, the number of tokens was increased. In *SR-Sym*, each transition rate is set to 1.0, while in *SR-Asym*, different rates were chosen for the transitions.
- The *KanBan* SPN models from [11] describe the kanban manufacturing system with various resource pool sizes.
- Members of the *Cloud* family are SPN performability models of a cloud architecture [17]. Scaling of the state space is achieved by changing the number of physical and virtual machines. Some aspects of the models from [17] were modified, as our tool currently does not support the GPSN formalism.

The models in several SPN formats and the PetriDotNet 1.5 tool are available at `https://inf.mit.bme.hu/en/petridotnet/stochasticanalysis`.

Table 1 shows the measured execution times and memory usages. The execution time of the decomposition algorithm t_{macro}, the construction of the generator matrix t_{gen} and the BiCGSTAB solver t_{solve} is displayed, as well as the

peak memory usage M_{peak}. Minimum values of every measure are emphasised in bold for each model.

The symbolic decomposition algorithm was executed under 100 ms for all the studied models, even in cases when the BiCGSTAB algorithm timed out. Block Kronecker based analysis with symbolic state space also consumed the least amount of memory, except in a single variation of the *KanBan* model.

Matrix construction was also fastest with symbolic decomposition. While the same matrices are constructed after explicit decomposition, the matrices were built slower based on the explicit state space than on MDDs due fast access to the decision diagram data structures.

The state space data structure must be kept in memory throughout the numerical solution, because it is needed in the phase of engineering measure evaluation. Thus, BiCGSTAB ran slower after explicit decomposition due to the pressure on the garbage collector (GC) caused by the larger size of the state space data structures. For smaller models, numerical solution with sparse matrices was faster than with block Kronecker matrices. However, for larger models the decomposed matrix utilized the memory bandwidth and caches more efficiently.

The sole difference between the *Sym* and *Asym* versions of the *SR* models are the transition rates, which only affected the numerical solution time. The structure of the state space remained identical.

Failures of BiCGSTAB included exhaustion of the time limit and the available memory. In the largest *SR-Asym* model, a numerical breakdown condition occurred. This condition may be handled by switching to a more stable solver, such as Jacobi or Gauss–Seidel iteration [28, Section 2.2]. These solvers are also implemented in the PetriDotNet framework.

6 Conclusion and Future Work

In this paper we introduced an efficient stochastic analysis approach using symbolic algorithms to explore the possible behaviours of the system and decomposition based stochastic analysis algorithms to efficiently compute stationary measures of stochastic Petri nets. In our work we established an efficient mapping technique which can bridge the gap between the encoded state space representation and the decomposition-based numerical algorithms. This algorithm supports the analysis of Petri net models having huge state spaces and complex behaviours. Measurements on models with various sizes and characteristics showed the effectiveness of the introduced approach and the efficiency of the new mapping algorithm.

In the future we plan to further extend our stochastic analysis framework with other numerical solution algorithms such as the Split algorithm for Kronecker products. In addition, we will also investigate GPU and distributed implementation of the available algorithms.

References

1. Baier, C., Katoen, J.P., Hermanns, H.: Approximative symbolic model checking of continuous-time Markov chains. In: CONCUR'99 Concurrency Theory, pp. 146–161. Springer (1999)
2. Barrett, R., Berry, M.W., Chan, T.F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., Van der Vorst, H.: Templates for the solution of linear systems: building blocks for iterative methods, vol. 43. Siam (1994)
3. Bause, F., Buchholz, P., Kemper, P.: A toolbox for functional and quantitative analysis of DEDS. In: Puigjaner, R., Savino, N.N., Serra, B. (eds.) Computer Performance Evaluation: Modelling Techniques and Tools, 10th International Conference, Tools '98, Palma de Mallorca, Spain, September 14-18, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1469, pp. 356–359. Springer (1998)
4. Benoit, A., Plateau, B., Stewart, W.J.: Memory-efficient Kronecker algorithms with applications to the modelling of parallel systems. Future Generation Comp. Syst. 22(7), 838–847 (2006)
5. Bianco, A., De Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Foundations of Software Technology and Theoretical Computer Science. pp. 499–513. Springer (1995)
6. Buchholz, P.: Hierarchical structuring of superposed GSPNs. IEEE Trans. Software Eng. 25(2), 166–181 (1999)
7. Buchholz, P., Ciardo, G., Donatelli, S., Kemper, P.: Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models. INFORMS Journal on Computing 12(3), 203–222 (2000)
8. Buchholz, P., Kemper, P.: On generating a hierarchy for GSPN analysis. SIGMETRICS Performance Evaluation Review 26(2), 5–14 (1998)
9. Buchholz, P., Kemper, P.: Kronecker based matrix representations for large Markov models. In: Validation of Stochastic Systems, pp. 256–295. Springer (2004)
10. Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S.: Stochastic well-formed colored nets and symmetric modeling applications. IEEE Trans. Computers 42(11), 1343–1360 (1993)
11. Ciardo, G., Jones, R.L., Miner, A.S., Siminiceanu, R.: Logical and stochastic modeling with SMART. In: Computer Performance Evaluation. Modelling Techniques and Tools, pp. 78–97. Springer (2003)
12. Ciardo, G., Miner, A.S.: A data structure for the efficient Kronecker solution of GSPNs. In: Petri Nets and Performance Models, 1999. Proceedings. The 8th International Workshop on. pp. 22–31. IEEE (1999)
13. Ciardo, G., Zhao, Y., Jin, X.: Ten years of saturation: A petri net perspective. T. Petri Nets and Other Models of Concurrency 5, 51–95 (2012)
14. Czekster, R.M., Rose, C.A.F.D., Fernandes, P.H.L., de Lima, A.M., Webber, T.: Kronecker descriptor partitioning for parallel algorithms. In: McGraw, R.M., Imsand, E.S., Chinni, M.J. (eds.) Proceedings of the 2010 Spring Simulation Multiconference, SpringSim 2010, Orlando, Florida, USA, April 11-15, 2010. p. 242. SCS/ACM (2010)
15. Donatelli, S.: Superposed generalized stochastic Petri nets: Definition and efficient solution. In: Valette, R. (ed.) Application and Theory of Petri Nets 1994, 15th International Conference, Zaragoza, Spain, June 20-24, 1994, Proceedings. Lecture Notes in Computer Science, vol. 815, pp. 258–277. Springer (1994)
16. Fernandes, P., Presotto, R., Sales, A., Webber, T.: An alternative algorithm to multiply a vector by a Kronecker represented descriptor. In: 21st UK Performance Engineering Workshop. pp. 57–67 (2005)

17. Ghosh, R.: Scalable stochastic models for cloud services. Ph.D. thesis, Duke University (2012)
18. Kemper, P.: Numerical analysis of superposed GSPNs. IEEE Trans. Software Eng. 22(9), 615–628 (1996)
19. Kwiatkowska, M., Mehmood, R., Norman, G., Parker, D.: A symbolic out-of-core solution method for Markov models. Electronic Notes in Theoretical Computer Science 68(4), 589–604 (2002)
20. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic symbolic model checking with PRISM: A hybrid approach. International Journal on Software Tools for Technology Transfer 6(2), 128–142 (2004)
21. Longo, F., Scarpa, M.: Two-layer symbolic representation for stochastic models with phase-type distributed events. Intern. J. Syst. Sci. 46(9), 1540–1571 (Jul 2015)
22. Marsan, M.A.: Stochastic Petri nets: an elementary introduction. In: Rozenberg, G. (ed.) Advances in Petri Nets 1989, covers the 9th European Workshop on Applications and Theory in Petri Nets, held in Venice, Italy in June 1988, selected papers. Lecture Notes in Computer Science, vol. 424, pp. 1–29. Springer (1988)
23. Marsan, M.A., Balbo, G., Bobbio, A., Chiola, G., Conte, G., Cumani, A.: The effect of execution policies on the semantics and analysis of stochastic Petri nets. IEEE Trans. Software Eng. 15(7), 832–846 (1989)
24. Marsan, M.A., Conte, G., Balbo, G.: A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. ACM Trans. Comput. Syst. 2(2), 93–122 (1984)
25. Meyer, J.F., Movaghar, A., Sanders, W.H.: Stochastic activity networks: Structure, behavior, and application. In: International Workshop on Timed Petri Nets, Torino, Italy, July 1-3, 1985. pp. 106–115. IEEE Computer Society (1985)
26. Miner, A.S.: Efficient solution of GSPNs using canonical matrix diagrams. In: Petri Nets and Performance Models, 2001. Proceedings. 9th International Workshop on. pp. 101–110. IEEE (2001)
27. Roux, P., Siminiceanu, R.: Model checking with edge-valued decision diagrams. In: Muñoz, C.A. (ed.) Second NASA Formal Methods Symposium - NFM 2010, Washington D.C., USA, April 13-15, 2010. Proceedings. NASA Conference Proceedings, vol. NASA/CP-2010-216215, pp. 222–226 (2010)
28. Stewart, W.J.: Probability, Markov chains, queues, and simulation: the mathematical basis of performance modeling. Princeton University Press (2009)
29. Teruel, E., Franceschinis, G., Pierro, M.D.: Well-defined generalized stochastic Petri nets: A net-level method to specify priorities. IEEE Trans. Software Eng. 29(11), 962–973 (2003)
30. Van der Vorst, H.A.: Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. SIAM Journal on scientific and Statistical Computing 13(2), 631–644 (1992)
31. Wan, M., Ciardo, G., Miner, A.S.: Approximate steady-state analysis of large Markov models based on the structure of their decision diagram encoding. Performance Evaluation 68(5), 463–486 (2011)
32. Zhao, Y., Ciardo, G.: A two-phase Gauss-Seidel algorithm for the stationary solution of EVMDD-encoded CTMCs. In: Ninth International Conference on Quantitative Evaluation of Systems, QEST 2012, London, United Kingdom, September 17-20, 2012. pp. 74–83. IEEE Computer Society (2012)