Computation of the normalising constant for product-form models of distributed systems with synchronisation

Simonetta Balsamo[a], Andrea Marin[a], Ivan Stojic[b]

[a] *Università Ca' Foscari Venezia, DAIS, via Torino 155, Venice, Italy*
[b] *Fondazione Bruno Kessler (FBK-irst), via Sommarive 18, Trento, Italy*

Abstract

Performance evaluation of large distributed systems plays a pivotal role in the design of Internet of Things (IoT) applications, or Big Data analysis, where high scalability and low response times are required. However, this performance assessment requires models, methodologies and tools tailored for this type of systems. Product-form queueing networks have been widely adopted for the analysis of sequential computations thanks to the availability of efficient algorithms for the computation of the average performance indices. However, this formalism has strong limitations since it does not allow one to model fork-join constructs or batches of jobs. Product-form stochastic Petri nets partially overcome these limitations but, on the other hand, general algorithms for the computation of the expected performance indices are not known or require strong assumptions on the model structure. In this paper, we define a new algorithm for the analysis of product-form stochastic Petri nets which works under much less restrictive assumptions than those previously proposed. The idea is that, in many cases, the entire state space of the model can be stored in memory thanks to multi-valued decision diagrams and the computation of the net's performance indices can take advantage of the tree structure that characterises this representation. Finally, we present a case study and several examples that will be used to study the performance of the algorithm for realistic scenarios.

1. Introduction

Performance evaluation of large distributed systems is a challenging research topic that has attracted much attention over the last decades. In many practical cases, benchmarking the systems is too complicated and simulations may be time expensive and inappropriate for studies that must compare many possible configurations with the aim of choosing the best according to a certain metric.

Email addresses: balsamo@unive.it (Simonetta Balsamo), marin@unive.it (Andrea Marin), stojic@fbk.eu (Ivan Stojic)

Analytical models, despite their strong assumptions, have shown to be a viable route to understand and tune the system's performance at the early stages of its development (see, e.g., [1]).

Many works rely on queueing systems to study distributed architectures (see, e.g., [2, 3] for some recent references) but due to their very abstract level of representation of resource contention, Markovian queueing networks (QNs) find a wider application (see, e.g., [4, 5, 6]). Under mild conditions, Markovian queueing networks admit a product-form solution, i.e., the expression of the stationary distribution can be derived without resorting to the explicit solution of the Markov chain underlying the model. However, product-forms do not immediately imply the analytical tractability of the models. Indeed, for queueing networks several algorithms have been defined for the computation of the average performance indices in an efficient and numerically stable way (see [7] for a survey of the classical algorithms and [8] for a recent application). The drawback of this formalism consists in its limitations: aside from the assumptions on the distributions of service times and on the scheduling disciplines, product-form queueing networks are not able to model fork-join constructs nor batches of jobs or resources. Clearly, in distributed systems, this is a serious limitation. For example, in MapReduce based computations, the Map and Reduce phases correspond to fork and join constructs, respectively.

One of the most important formalisms that overcome the limitations of QN are the stochastic Petri nets (SPNs) [9, 10, 11]. They are widely used for modeling and performance evaluation of computer systems, computer networks, telecommunication systems as well as in bioinformatics where they are employed in modeling of various biological and biochemical processes. Like QNs [7], their underlying stochastic process is a continuous-time Markov chain (CTMC). Also like QNs, in practical use, SPNs suffer from the problem of state space explosion which consists in an exponential, in model size, growth of the number of distinct states of the underlying CTMC. This limits the scope of performance evaluation methods to models for which the underlying CTMC is small enough to be solved (i.e., to compute its transient or stationary probability distribution), or has a special structure. Because of these difficulties with solving models with large state spaces, a wide range of approximation methods and special solution methods for models with special structure or characteristics have been developed.

Product-form models, firstly identified for QNs [12, 13], have been studied for SPNs [14, 15, 16, 17] and other modeling formalisms; for a model in this class, the stationary probability distribution can be obtained in the form of a product over model components. State probabilities for these models are usually obtained in an unnormalised form (formally, a measure over the state space of the process is obtained), requiring the computation of the normalising constant which can be computed directly as a sum of unnormalised probabilities over the state space of the underlying CTMC or using more efficient convolution algorithms [18, 19]. The normalisation phase is crucial for determining most of the performance indices of the model.

The stationary analysis of product-form models is greatly simplified in com-

parison to other models and more complex models with larger state spaces can be efficiently analysed. However, the convolution algorithm for SPNs requires a special structure of the reachability set and can only be applied to the class of *S-invariant reachable* SPNs. Furthermore, at the current state of the art, it is not known how to automatically check the membership of this class without generating the reachability set, and the convolution is less efficient than in the case of QNs due to the need to solve a large number of systems of linear Diophantine equations. The S-invariant reachability restriction severely limits the utility of the convolution algorithm.

Multi-valued decision diagrams (MDDs) [20], data structures that encode discrete functions of discrete variables, have been used with great success in generation and storage of very large state spaces for bounded SPN models [21, 22]. In the present work, we use MDDs that encode state spaces of product-form models in order to efficiently compute the normalising constant for models with finite state spaces. We also propose related methods for the computation of performance measures of product-form SPNs with finite reachability sets. While we need to generate the reachability set of the SPN, we do not require S-invariant reachability and the proposed algorithm can thus be applied to general SPNs. Also, in contrast to convolution, we do not need to solve systems of Diophantine equations. Hence, the performance of the proposed algorithm on S-invariant reachable SPNs is thus much better than the convolution algorithm if reachability set generation is not taken into account, while being comparable otherwise.

The paper is laid out as follows. First, in Section 2, we propose a review of the literature on this topic. In Section 3, we introduce SPNs and product-forms. Section 4 introduces some notation for state spaces of the models handled in the paper, and defines MDDs. Based on these definitions, we introduce a novel recursive algorithm for computation of the normalising constant for general product-form models. In particular, the algorithm can be applied to SPNs. Section 5 compares the proposed algorithm to the convolution algorithm for a class of product-form SPNs [19] and proposes methods for performance evaluation, comparing them to the mean value algorithm (MVA) [23]. In Section 6 we report results of computational experiments on SPNs in which we test performance of the proposed algorithm and compare it to the performance of the SPN convolution algorithm. Finally, Section 7 concludes the paper. Appendix A contains the proofs of statements from the main part of the paper. While our main contributions are directed to the case of SPNs, for the sake of clarity we include in Appendix B a comparison with the convolution algorithm for a class of product-form QNs [18], which can be considered as a special case of what we propose here. The point of the latter appendix is not to compare the proposed algorithm with the convolution algorithm on QNs, but to drive the intuition of the reader who is expert on the queueing network analysis.

2. Related work

In this section, we review the literature on the solution of SPNs and the theoretical frameworks used for the definition of our algorithm. We leave the discussion on the applications of product-form SPNs for the performance evaluation of distributed systems to Section 3, where the reader will have a clearer view of their potential.

In [24, 25] decision diagrams have been used for the analysis of SPNs. However, unlike in the present paper, 1) these approaches rely on encoding, using decision diagrams, the transition rate matrix of the CTMC underlying the SPN, in addition to encoding the state space, and 2) the approaches are based on iterative computation of successively better approximations of the stationary probabilities, based on solving certain aggregate CTMCs obtained from the structure of the decision diagrams encoding the state space of the SPN under consideration. While for product-form SPNs they do not require knowledge of the product-form formula, these approaches are very different from the present one, and are bound to scale much worse due to the need to encode the transition rate matrix and solve the aggregated CTMCs.

Similarly, there are several tools, such as PRISM [26], for analysis of SPNs which implement symbolic methods based on decision diagrams. However, to the best of our knowledge, none of these tools take advantage of the product-form property of SPNs.

In the domain of queueing networks, several papers have been presented that address the problem of computing the normalising constant for product-form models. This demonstrates the importance that this topic has for the modelling community. In [27], Casale reformulates the problem of computing the normalising constant for closed queueing networks as the computation of an integral. To this aim, a revision of the cubature rules for average size models and a Monte Carlo sampling for larger ones are proposed. This technique cannot be applied for general product-form SPNs because of the complexity of computing the reachability set. The same author, in earlier work [28] proposed the application of the method of moments for solving closed queueing networks in product-form.

The exact or approximate analysis of timed Petri nets is still a very active research area both for non-Markovian [29] and Markovian [30, 31] models. While we leave the deep review of existing results on product-form SPNs to Section 3.2, we mention here the approach proposed in [30] which refines the basic fluidisation approaches previously introduced in the literature to tackle the state space explosion problem. Specifically, the authors study the conditions under which a hybrid model obtained by a partial relaxation of the original Markov model can provide accurate estimates of state probabilities. A completely different approach is used in [32] where the author derives a set of structural inequalities and equalities from the SPN that allow for the computation of bounds on the net performance indices by solving a linear programming problem.

3. Stochastic Petri nets

In this section, we introduce stochastic Petri nets (SPNs) [9, 10, 11] and review their main properties. Then, we introduce a model of a distributed computation that will serve as the running example.

3.1. Stochastic Petri nets

Definition 3.1. *Stochastic Petri net.* An SPN is a 6-tuple $(\mathcal{P}, \mathcal{T}, I, O, W, \boldsymbol{m}_0)$, where $\mathcal{P} = \{P_1, \ldots, P_n\}$ is a nonempty set of n places, $\mathcal{T} = \{T_1, \ldots, T_m\}$ is a nonempty set of m transitions, $I, O : \mathcal{T} \to \mathbb{N}^n$ are functions that define input and output vectors of the transitions, $W : \mathbb{N}^n \times \mathcal{T} \to \mathbb{R}_{>0}$ is a function that defines (possibly state-dependent) transition firing rates, and $\boldsymbol{m}_0 \in \mathbb{N}^n$ is an initial marking of the net.

A *marking* of an SPN is a vector $\boldsymbol{m} = (m_1, \ldots, m_n) \in \mathbb{N}^n$ that associates numbers $m_1, \ldots, m_n$ of *tokens* with the places $P_1, \ldots, P_n$. A transition $T_i \in \mathcal{T}$ is *enabled* to fire in a marking $\boldsymbol{m} \in \mathbb{N}^n$ if $\boldsymbol{m} - I(T_i) \geqslant \boldsymbol{0}$, i.e., all components of the vector on the left-hand side are nonnegative. In an untimed Petri net, the marking process evolves by choosing in a non-deterministic manner one of the enabled transitions and *firing* it. When an enabled transition T_i fires, the marking $\boldsymbol{m}$ is transformed to the marking $\boldsymbol{m} - I(T_i) + O(T_i)$. The *enabling degree* $e_i(\boldsymbol{m})$ of a transition T_i in a marking $\boldsymbol{m}$ is the number of times that the transition T_i can fire in a row without firing any other transitions, and is equal to $e_i(\boldsymbol{m}) = \max\{k \in \mathbb{N} : \boldsymbol{m} - kI(T_i) \geqslant \boldsymbol{0}\}$. A transition T_i that is enabled in a marking $\boldsymbol{m}$ is *singly enabled* in the marking $\boldsymbol{m}$ if $e_i(\boldsymbol{m}) = 1$, and *multiply enabled* if $e_i(\boldsymbol{m}) > 1$.

The marking process of the SPN evolves from the initial marking $\boldsymbol{m}_0$ by the firing of enabled transitions. The set of all markings that are reachable by firing some sequence of (enabled) transitions from the initial marking $\boldsymbol{m}_0$ is called the *reachability set* of the SPN and denoted with $\mathcal{S}$. In this paper it is assumed that the reachability set is finite, which is equivalent to the assumption that there exists a vector of place marking bounds $\boldsymbol{B} \in \mathbb{N}^n$ that is larger than any reachable marking: $\boldsymbol{m} \leqslant \boldsymbol{B}, \forall \boldsymbol{m} \in \mathcal{S}$.

An *S-invariant* or *place invariant* of an SPN is a row-vector $\boldsymbol{U} = (U_1, \ldots, U_n) \in \mathbb{N}^n$ such that $\boldsymbol{U}\boldsymbol{m}_0 = \boldsymbol{U}\boldsymbol{m}$ for all reachable markings $\boldsymbol{m} \in \mathcal{S}$. The product $\boldsymbol{U}\boldsymbol{m}$ can be considered as a weighted sum of the numbers of tokens in places for which the corresponding elements of $\boldsymbol{U}$ are nonzero; existence of an S-invariant thus signifies that the total (weighted) number of tokens in these places is conserved during execution of the SPN. The set of places for which the corresponding elements of an S-invariant $\boldsymbol{U}$ are nonzero is called the *support* of the S-invariant $\boldsymbol{U}$. We define the partial order $\leqslant$ on the set of all S-invariants of an SPN in the following manner: for two S-invariants $\boldsymbol{U}$ and $\boldsymbol{U}'$, $\boldsymbol{U} \leqslant \boldsymbol{U}'$ if and only if $U_k \leqslant U'_k$ for all $k \in \{1, \ldots, n\}$. If an invariant $\boldsymbol{U}$ 1) has a minimal support (in the sense that there is no S-invariant whose support is a proper subset of the support of $\boldsymbol{U}$), and 2) $\boldsymbol{U}$ is a minimal (with respect to the partial order $\leqslant$) S-invariant (in the sense that there is no S-invariant $\boldsymbol{U}'$ such that $\boldsymbol{U}' \leqslant \boldsymbol{U}$ and

$U'_k < U_k$ for some index k), then it is called a *minimal support S-invariant*. For an arbitrary SPN, the set of minimal support S-invariants is finite and forms a basis of all its S-invariants.

So far, timed behaviour has not been considered. In the present paper, it is assumed that the firing delays of singly enabled transitions are exponentially distributed with rates defined by the function W. Various firing semantics for multiply enabled transitions can be defined, two common examples are:

- *single server (SS)* semantics, where the firing delay of an enabled transition T_i in marking $\boldsymbol{m}$ is exponentially distributed with rate that is independent of the enabling degree (we write $W(\boldsymbol{m}, T_i) = W(T_i)$, for all $\boldsymbol{m} \in \mathcal{S}$); T_i can be considered to be processing only a single enabling set of tokens with the rate $W(T_i)$,

- *infinite server (IS)* semantics, where the firing delay of an enabled transition T_i in marking $\boldsymbol{m}$ is exponentially distributed with rate that depends linearly on the enabling degree (we write $W(\boldsymbol{m}, T_i) = e_i(\boldsymbol{m})W(T_i)$, for all $\boldsymbol{m} \in \mathcal{S}$); in this case T_i can be considered to be processing in parallel each of the $e_i(\boldsymbol{m})$ enabling sets of tokens with rates equal to $W(T_i)$.

The resulting stochastic process underlying the SPN is isomorphic to a continuous time Markov chain (CTMC) with the state space equal to the reachability set $\mathcal{S}$ and the transition rates given by the above firing delays [9]. In the following, $P(e)$ denotes the stationary probability, for this CTMC, of an event e. Some common stationary performance measures of interest are:

- average number $n(P_j)$ of tokens in a place P_j, $n(P_j) = \sum_{k=1}^{B_j} kP(m_j = k)$ where B_j is the bound on the number of tokens in P_j;

- utilization $u(P_j)$ of a place P_j (the probability that the place P_j is nonempty), $u(P_j) = 1 - P(m_j = 0)$;

- utilization $u(T_j)$ of a transition T_j (the probability that T_j is enabled), $u(T_j) = P(e_j \geqslant 1)$;

- throughput $x(T_j)$ of a transition T_j (the average number of firings of T_j in a unit of time), $x(T_j) = \sum_{k=1}^{E_j} kW(T_j)P(e_j = k)$, where $E_j = \min_{1 \leqslant i \leqslant n} \{\lfloor B_i / I_i(T_j) \rfloor : I_i(T_j) > 0\}$ is an upper bound on the maximum enabling degree of the transition T_j;

- throughput $x(P_j)$ of a place P_j (the average number of tokens that is removed from P_j in a unit of time), $x(P_j) = \sum_{T \in \mathcal{T}} I_j(T)x(T)$.

3.2. Product-form SPNs

It is assumed throughout the paper that the model under consideration is decomposed into K submodels. The state space of the model is assumed to be finite, and (also finite) local state spaces of the submodels are denoted by $\mathcal{S}_1, \ldots, \mathcal{S}_K$. Each local state space $\mathcal{S}_k$, of cardinality $n_k \in \mathbb{N}$, is identified with

the set $\{0, 1, \ldots, n_k - 1\}$. The cartesian product $\mathcal{S}_K \times \cdots \times \mathcal{S}_1$ of local state spaces is denoted by $\hat{\mathcal{S}}$ and called the *potential state space* and the state space of the model is denoted by $\mathcal{S}$; note that $\mathcal{S} \subseteq \hat{\mathcal{S}}$. We order the local state spaces in the above product from K to 1 to correspond with the usual numbering of MDD levels (introduced later in the paper). A state of the model is denoted by a K-tuple of integers $(s_K, \ldots, s_1) \in \mathcal{S}$, where $s_k \in \mathcal{S}_k$ is the local state of the submodel k.

In the rest of the paper, we consider models with product-form equilibrium state probability distributions. Such models are defined by their stationary state probabilities being products over submodels of the probabilities of submodels' states, i.e. for an arbitrary state $\boldsymbol{s} = (s_K, \ldots, s_1) \in \mathcal{S}$,

$$P(\boldsymbol{s}) = \frac{1}{G} \prod_{k=1}^{K} g_k(s_k) \tag{1}$$

where $P(\boldsymbol{s})$ is the stationary probability of the state $\boldsymbol{s}$, G is a normalising constant needed for the probabilities to sum to 1,

$$G = \sum_{\boldsymbol{s} \in \mathcal{S}} \prod_{k=1}^{K} g_k(s_k) \tag{2}$$

and $g_k : \mathcal{S}_k \to \mathbb{R}_{\geqslant 0}, k = 1, \ldots, K$ are functions defining (not necessarily normalised) probabilities of local states of the submodels, with $\mathcal{S}_k$ being the set of all possible local states.

It is out of the scope of this paper to review the conditions under which an SPN is in product-form, since this would require us to address many cases and possibilities. However, we think it is important to remark that product-form SPNs are quite expressive and include, but strictly extend, the well-known formalisms of Jackson and Gordon-Newell queueing networks. Product-form SPNs can model instantaneous signal propagation as shown in [33], fork-join constructs [34] (including RAID systems, MapReduce), database conflicts [35]. Similar results are being studied for the transmission redundancy, showing that product-form solution makes the analysis of complex systems feasible [36, 37, 38]

Testing the sufficient conditions for product-forms in SPNs is efficient, and can be done in polynomial time in the number of places and transitions. Similarly, the computation of the functions g_k of Equation (1) is very efficient, while obtaining the expression for G is, in general, quite difficult. The brute-force approach induced by Equation (2) is time-expensive and numerically unstable, but, without the value of G, very few considerations can be drawn by the model analysis. The algorithm proposed in this paper provides an efficient computation of the normalising constant G for general product-form SPNs.

3.3. Running example

We consider a system in which computational resources (machines) are used to process two types of jobs, as shown in Figure 1, where we use the standard

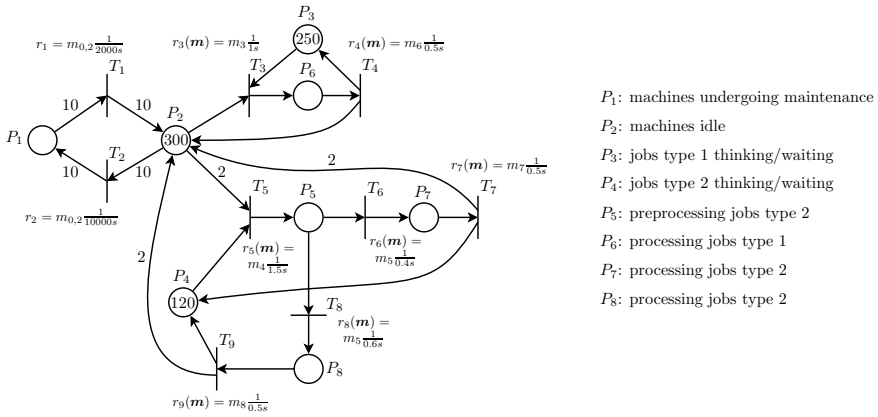

Figure 1: Case study SPN

graphical representation of Petri nets in the form of a directed graph in which circles represent places, numbers in the circles represent tokens (in the figure, the number m_1 of tokens in place P_1 is 0, while the number m_2 of tokens in place P_2 is 300), bars represent transitions, and edges represent the input and output vectors of the transitions (for details, see [11, Section II]). In this SPN, tokens in place P_2 represent idle machines, which may either enter and exit maintenance in batches of 10 machines (transitions T_2 and T_1), or may start to process jobs (transitions T_3 and T_5). Places P_3 and P_4 represent jobs of the two types waiting for the use of the computational resources in the place P_2. Processing of the jobs of type 1 requires a single machine and consists of a single processing phase (transition $T4$), while processing of the jobs of type 2 requires two machines and consists of two phases; the first phase is represented by transitions T_6 and T_8, and the second phase by T_7 and T_9.

The firing delays of all transitions are negative-exponentially distributed, with rates as shown in the Figure 1. Rates of transitions T_1 and T_2 depend on the initial marking $m_{0,2}$ of place P_2 and model the average maintenance frequency $\frac{1}{10000s}$ and maintenance rate $\frac{1}{2000s}$ for a batch of machines, while rates of the rest of the transitions are marking-dependent as shown in the figure—for example, the rate of transition T_4 is equal to $m_6\frac{1}{0.5s}$, modelling the parallel processing of jobs in P_6 with the average processing rate per job of $\lambda_6 = \frac{1}{0.5s}$.

Following the approach described in [19], we define the marking-dependent transition rates for transitions $T_1, \ldots, T_9$ with

$$r_i(\boldsymbol{m}) = \frac{\psi(\boldsymbol{m} - I(T_i))\lambda_i}{\phi(\boldsymbol{m})}, \quad \psi(\boldsymbol{m}) = \phi(\boldsymbol{m}) = \begin{cases} 0 & \text{if } m_j \leqslant 0 \text{ for some } j \\ \prod_{j=3}^{8} \frac{1}{m_j!} & \text{otherwise} \end{cases}$$

(here we define $\lambda_1 = m_{0,2}\frac{1}{2000s}$ and $\lambda_2 = m_{0,2}\frac{1}{10000s}$) in order to obtain the marking-dependent rates as shown in Fig. 1, and we obtain the following invari-

ant measure for the markings $\boldsymbol{m} \in \mathcal{S}$:

$$\pi(\boldsymbol{m}) = \left(\prod_{j=3}^{8} \frac{1}{m_j!}\right) \left(\frac{\lambda_2}{\lambda_1}\right)^{m_1} \left(\frac{\lambda_4}{\lambda_3}\right)^{m_3} \left(\frac{\lambda_6 + \lambda_8}{\lambda_5}\right)^{m_4} \left(\frac{\lambda_6}{\lambda_7}\right)^{m_7} \left(\frac{\lambda_8}{\lambda_9}\right)^{m_8}. \quad (3)$$

The term $\pi(\boldsymbol{m}) \in \mathbb{R}$ is the unnormalised stationary probability of an arbitrary marking $\boldsymbol{m} \in \mathcal{RS}$: there exists a *normalising constant* $G \in \mathbb{R}$ such that the steady-state probabilities of the markings are given by $P(\boldsymbol{m}) = \frac{1}{G}\pi(\boldsymbol{m})$. This formula for $P(\boldsymbol{m})$ is a product form over the places of the SPN. The normalising constant is equal to the sum of the unnormalised probabilities, $G = \sum_{\boldsymbol{m} \in \mathcal{RS}} \pi(\boldsymbol{m})$.

4. Algorithm MDD-rec

As explained in Section 3, the main problem in the analysis of product-form models is efficient computation of the normalising constant G. In this section, we introduce a novel recursive algorithm, named MDD-rec, for its computation. MDD-rec computes the normalising constant by walking over nodes of a multi-valued decision diagram (MDD) that encodes the state space $\mathcal{S}$ of the model. Generation of this MDD is out of the scope of this paper (for further details see [39]) and it is assumed that the MDD encoding of $\mathcal{S}$ is taken as an input to the algorithm.

4.1. Multi-valued decision diagrams

The state space $\mathcal{S}$ of a model that is decomposed into K submodels as above, can be encoded using an MDD [20].

Definition 4.1. *Multi-valued decision diagram.* Let $K \in \mathbb{N}_{>0}$ be a nonzero natural number, and let $\mathcal{S}_k = \{0, 1, \ldots, n_k - 1\}, n_k \in \mathbb{N}_{>0}, k = 1, \ldots, K$ be nonempty sets. A multi-valued decision diagram over $\mathcal{S}_K \times \cdots \times \mathcal{S}_1$ is a directed acyclic multi-graph with labelled arcs, and nodes organized in $K+1$ levels. The nodes on levels $1, \ldots, K$ are called non-terminal nodes while the nodes on level 0 are called terminal nodes. A node in the MDD is denoted by $\langle k.p \rangle$, where $k \in \{0, \ldots, K\}$ is the level of the node, and p is the unique node index in level k. For each non-terminal node $\langle k.p \rangle$, there are exactly n_k outgoing directed arcs labeled with $0, 1, \ldots, n_k - 1$ with the same source node $\langle k.p \rangle$ and destination nodes in level $k-1$. These destination nodes are denoted by $\langle k.p \rangle[0], \ldots, \langle k.p \rangle[n_k - 1]$; for example $\langle k.p \rangle[j] = \langle (k-1).q \rangle$ means that there is an arc labelled with j from node $\langle k.p \rangle$ to node $\langle (k-1).q \rangle$. Readers familiar with MDDs will recognize that here MDDs are assumed to be *ordered* (i.e., their arcs are allowed to point only to nodes in the neighbouring lower level). It is assumed that level K contains only a single root node $\langle K.r \rangle$, and that there are no *duplicate nodes*; two nodes $\langle k.p \rangle$ and $\langle l.q \rangle$ are duplicate if they are on the same level ($k = l$) and all their corresponding arcs point to the same nodes ($\langle k.p \rangle[j] = \langle l.q \rangle[j], \ \forall j \in \{0, 1, \ldots, n_k - 1\}$). MDDs satisfying this requirement are said to be *quasi-reduced*. Level 0 contains exactly two (terminal) nodes, denoted by $\langle 0.0 \rangle$ and $\langle 0.1 \rangle$, that have no outgoing arcs.

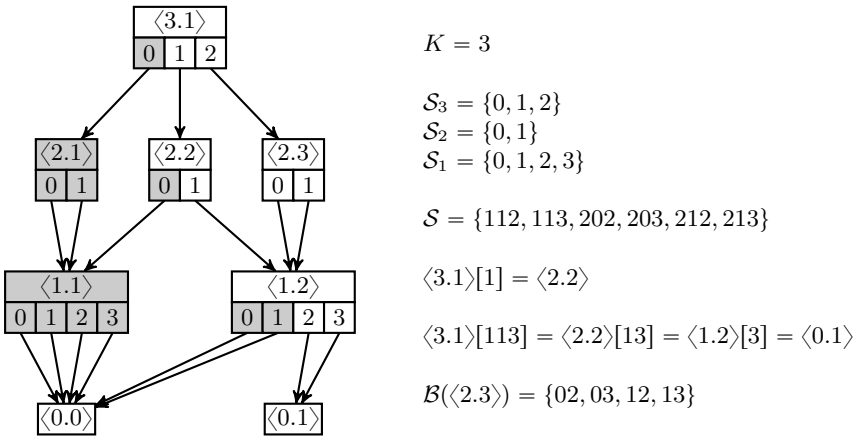

Figure 2: Example of a decision diagram. Arc labels are indicated in the bottom halves of the decision diagram nodes. In an implementation, nodes with gray background do not need to be stored and remaining arcs whose labels have gray backgrounds can be redirected to point directly to node $\langle 0.0\rangle$.

Figure 2 contains an example of a decision diagram with 4 levels. In this paper, levels $1, \ldots, K$ of a decision diagram correspond to the submodels, and labels of arcs exiting level k correspond to local states of the submodel k. Decision diagrams are used to encode state spaces of models. We introduce the notion of the set of substates encoded by an MDD node.

Definition 4.2. *Notation $\cdot[\cdot]$ for nodes.* For a node $\langle k.p\rangle$ in an MDD and for a sequence of integers $\boldsymbol{s} = (s_k, \ldots, s_l) \in \mathcal{S}_k \times \cdots \times \mathcal{S}_l, k \geqslant l$, node $\langle k.p\rangle[\boldsymbol{s}]$ is defined as the unique node that is reached from node $\langle k.p\rangle$ by following arcs labeled with the elements from $\boldsymbol{s}$, and is given recursively with:

$$\langle k.p\rangle[\boldsymbol{s}] = \begin{cases} \langle k.p\rangle[s_k] & \text{if } \boldsymbol{s} = (s_k) \text{ is a singleton,} \\ \langle (k-1).q\rangle[\boldsymbol{t}] & \text{if } \boldsymbol{s} = (s_k)\boldsymbol{t} \text{ and } \langle k.p\rangle[s_k] = \langle (k-1).q\rangle. \end{cases}$$

Here $(s_k)\boldsymbol{t}$ denotes the sequence obtained by concatenation of two sequences (s_k) and $\boldsymbol{t}$. Sequence $\boldsymbol{s}$ is also called substate, or, if $k = K$ and $l = 1$, state.

Definition 4.3. *Set of substates encoded by a node.* For a node $\langle k.p\rangle$ in an MDD, the set $\mathcal{B}(\langle k.p\rangle)$ of substates encoded by the node $\langle k.p\rangle$ is defined as

$$\mathcal{B}(\langle k.p\rangle) = \{\boldsymbol{s} \in \mathcal{S}_k \times \cdots \times \mathcal{S}_1 : \langle k.p\rangle[\boldsymbol{s}] = \langle 0.1\rangle\}.$$

It is said that an MDD with root $\langle K.r\rangle$ encodes a state space $\mathcal{S} \subseteq \mathcal{S}_K \times \cdots \times \mathcal{S}_1$ if the set of states encoded by the root node is equal to $\mathcal{S}$:

$$\mathcal{B}(\langle K.r\rangle) = \mathcal{S}.$$

From the above definitions, it is easy to see that for a potential state $\boldsymbol{s} = (s_K, \ldots, s_1) \in \hat{\mathcal{S}}$, $\langle K.r\rangle[\boldsymbol{s}]$ is equal to one of the terminal nodes, $\langle 0.0\rangle$

or $\langle 0.1 \rangle$. If the terminal nodes $\langle 0.0 \rangle$ and $\langle 0.1 \rangle$ are identified with numbers 0 and 1, respectively, the MDD that encodes a state space $\mathcal{S} \subseteq \hat{\mathcal{S}}$ can be seen to encode the characteristic function $\mathbb{1}_{\mathcal{S}} : \hat{\mathcal{S}} \to \{0, 1\}$ of the set $\mathcal{S}$.

An important property of ordered quasi-reduced MDDs is that they are a canonical representation of the functions $\hat{\mathcal{S}} \to \{0, 1\}$; two functions $f, g : \hat{\mathcal{S}} \to \{0, 1\}$ are equal if and only if the ordered quasi-reduced MDDs encoding f and g are isomorphic. A useful consequence of canonicity is that two non-terminal nodes $\langle k.p \rangle, \langle l.q \rangle$ encode the same set if and only if $k = l$ and $p = q$, that is if they are the same node—otherwise, it can be shown that $\langle k.p \rangle$ and $\langle k.q \rangle$ are either duplicate nodes or that there exist duplicate nodes in one of the lower levels $k - 1, \ldots, 1$, violating the quasi-reduced property from the definition of the MDD.

In the implementation of decision diagrams, nodes of the decision diagram from which all paths lead to node $\langle 0.0 \rangle$ do not need to be stored, since they can be deduced from the rest of the decision diagram. In the rest of the paper, it is assumed that nodes with this property are omitted from the decision diagrams and that the remaining arcs that would point to such nodes point directly to node $\langle 0.0 \rangle$ (these nodes and arcs are depicted in Figure 2 with gray backgrounds). Then the following holds for all remaining non-terminal nodes:

$$\forall \langle k.p \rangle, \forall s_k \in \mathcal{S}_k, \quad \mathcal{B}(\langle k.p \rangle[s_k]) = \varnothing \iff \langle k.p \rangle[s_k] = \langle 0.0 \rangle.$$

4.2. Algorithm MDD-rec

We are now in position to introduce our main contribution. The algorithm MDD-rec strongly relies on the following definition.

Definition 4.4. *Unnormalised mass of an MDD node.* Let D be the MDD encoding of the state space $\mathcal{S}$ and let functions g_k be as defined in Sec. 3.2. We define a function M on the nodes of D, $M : nodes(D) \to \mathbb{R}_{\geqslant 0}$, as follows. For an arbitrary node $\langle l.p \rangle$, its *unnormalised mass* $M(\langle l.p \rangle)$ is defined as

$$M(\langle l.p \rangle) = \begin{cases} \displaystyle\sum_{\boldsymbol{s} \in \mathcal{B}(\langle l.p \rangle)} \prod_{k=1}^{l} g_k(s_k) & \text{if } l > 0, \\ p & \text{otherwise.} \end{cases}$$

Henceforth, for readability, we use simply *mass* in place of *unnormalised mass*.

The mass of a node $\langle l.p \rangle$ can be thought of as a partially computed normalising constant, obtained by restricting the sum in (2) to the substates $\boldsymbol{s} \in \mathcal{B}(\langle l.p \rangle)$, and restricting the product to $k \in \{1, \ldots, l\}$.

From the definitions of M, $\mathcal{B}$ and G it can be easily seen that the mass of the root node $\langle K.r \rangle$ of D is equal to the normalising constant G:

$$M(\langle K.r \rangle) = \sum_{\boldsymbol{s} \in \mathcal{B}(\langle K.r \rangle)} \prod_{k=1}^{K} g_k(s_k) = \sum_{\boldsymbol{s} \in \mathcal{S}} \prod_{k=1}^{K} g_k(s_k) = G.$$

Thus, computing G can be performed by computing the mass of the root node of D. The rest of this section is concerned with efficient computation of the mass $M(\langle K.r\rangle)$.

In general, mass $M(\langle l.p\rangle)$ of an arbitrary non-terminal node can be obtained as a weighted sum of masses of its neighbouring nodes in the level $l-1$:

$$M(\langle l.p\rangle) = \sum_{\boldsymbol{s}\in\mathcal{B}(\langle l.p\rangle)} \prod_{k=1}^{l} g_k(s_k) = \sum_{\substack{s_l\in\mathcal{S}_l \\ \boldsymbol{t}\in\mathcal{B}(\langle l.p\rangle[s_l])}} g_l(s_l) \prod_{k=1}^{l-1} g_k(t_k) =$$

$$= \sum_{s_l\in\mathcal{S}_l} g_l(s_l) \sum_{\boldsymbol{t}\in\mathcal{B}(\langle l.p\rangle[s_l])} \prod_{k=1}^{l-1} g_k(t_k) = \sum_{s_l\in\mathcal{S}_l} g_l(s_l) M(\langle l.p\rangle[s_l]),$$

yielding the recurrence relation

$$M(\langle l.p\rangle) = \sum_{s_l\in\mathcal{S}_l} g_l(s_l) M(\langle l.p\rangle[s_l]). \tag{4}$$

This recurrence relation leads to a recursive algorithm for computation of the mass $M(\langle K.r\rangle)$. The algorithm recursively walks over the nodes of the MDD encoding the set $\mathcal{S}$, computing masses of MDD nodes and caching them in node labels, as depicted in Algorithm 1. The computation of the normalising constant G is performed by invoking MDD-rec($\langle K.r\rangle$, D, NONE, $g_1, \ldots, g_K$) on the root note $\langle K.r\rangle$ of the MDD D encoding the state space $\mathcal{S}$ of the model, with labeling function L set to sentinel value NONE for all nodes, $L \equiv$ NONE.

Algorithm 1: MDD-rec($\langle l.p\rangle, D, L, g_1, \ldots, g_K$)

Data:
node $\langle l.p\rangle$ of the MDD D
MDD D encoding set $\mathcal{S}$
node labels $L : nodes(D) \to \mathbb{R}_{\geqslant 0} \cup \{\text{NONE}\}$
functions $g_k : \mathcal{S}_k \to \mathbb{R}_{\geqslant 0}, k = 1, \ldots, K$ defining the product-form
Result: mass $M(\langle l.p\rangle)$, modified node labels L

```
1 begin
2     if l = 0 then
3         return p
4     if L(⟨l.p⟩) = NONE then
5         L(⟨l.p⟩) ← Σ_{s_l∈S_l} g_l(s_l) MDD-rec(⟨l.p⟩[s_l], D, L, g_1, ..., g_K)
6     return L(⟨l.p⟩)
```

5. Algorithm comparison with convolution on product-form stochastic Petri nets

One of the well-known algorithms for computation of the normalising constant in product-form SPNs is the convolution algorithm presented in [19].

However, this algorithm, in contrast with MDD-rec, is not applicable to any product-form SPN, but only to a subclass of S-invariant reachable SPNs. In this section, we first introduce the conditions required by S-invariant reachable SPNs and then we compare the convolution algorithm of [19] with the proposed algorithm MDD-rec.

5.1. Convolution algorithm for computation of normalising constant for stochastic Petri nets

The convolution algorithm [19] for computation of the normalising constant assumes a particular structure of the reachability set that allows for a recursive decomposition. Let $\boldsymbol{S}$ be the matrix with rows comprised in all minimal support S-invariants of an SPN (as defined in Sec. 3.1), and let $\boldsymbol{V} = \boldsymbol{S}\boldsymbol{m}_0$ be the *load vector* of the SPN. The SPN is *S-invariant reachable* if its reachability set $\mathcal{RS}$ satisfies the following condition:

$$\forall \boldsymbol{m} \in \mathbb{N}^n,\ \boldsymbol{m} \in \mathcal{RS} \iff \boldsymbol{S}\boldsymbol{m} = \boldsymbol{V}. \tag{5}$$

Note that S-invariant reachability is a very strict property and that, to the best of our knowledge, no algorithm is known which, given an SPN, checks whether the SPN satisfies this property, without generating the state space of the SPN. The convolution algorithm is limited to S-invariant reachable SPNs, and in practice can be used only on SPNs for which S-invariant reachability can be easily proved manually, or for which the reachability set can be efficiently generated. For any subset of SPN places $\mathcal{P}' \subseteq \mathcal{P}$ and any vector $\boldsymbol{W} \in \mathbb{R}^n$, the set $\mathcal{E}(\mathcal{P}', \boldsymbol{W}) \subseteq \mathbb{N}^n$ of markings is defined by

$$\mathcal{E}(\mathcal{P}', \boldsymbol{W}) = \{\boldsymbol{m} \in \mathbb{N}^n : \boldsymbol{S}\boldsymbol{m} = \boldsymbol{W} \text{ and } \forall p \in \mathcal{P}\backslash\mathcal{P}', m_p = 0\}. \tag{6}$$

It can be easily seen that $\mathcal{RS} = \mathcal{E}(\mathcal{P}, \boldsymbol{V})$. Given a set of markings $\mathcal{E}(\mathcal{P}', \boldsymbol{W})$, the marking set $M_p(\mathcal{P}', \boldsymbol{W})$ of a place $p \in \mathcal{P}$ is defined as

$$M_p(\mathcal{P}', \boldsymbol{W}) = \{i \in \mathbb{N} : \exists \boldsymbol{m} \in \mathcal{E}(\mathcal{P}', \boldsymbol{W}) \text{ such that } m_p = i\}. \tag{7}$$

This is the set of all different markings of the place p that appear in the set of markings $\mathcal{E}(\mathcal{P}', \boldsymbol{W})$.

The following Lemma appears in [40] but with a different formulation, so we include it here for clarity.

Lemma 5.1. *For any S-invariant reachable SPN $(\mathcal{P}, \mathcal{T}, I, O, W, \boldsymbol{m}_0)$ with load vector $\boldsymbol{V} = \boldsymbol{S}\boldsymbol{m}_0$, the following decomposition of the set $\mathcal{E}(\mathcal{P}', \boldsymbol{W})$ holds for all subsets $\mathcal{P}' \subseteq \mathcal{P}$, arbitrary place $p \in \mathcal{P}'$ and all vectors $\boldsymbol{W}$, $\boldsymbol{0} \leqslant \boldsymbol{W} \leqslant \boldsymbol{V}$:*

$$\mathcal{E}(\mathcal{P}', \boldsymbol{W}) = \bigcup_{i \in M_p(\mathcal{P}', \boldsymbol{W})} \left[i\boldsymbol{e}_p + \mathcal{E}(\mathcal{P}'\backslash\{p\}, \boldsymbol{W} - i\boldsymbol{S}_p) \right],$$

where the union is over mutually disjoint sets (i.e., the sets that appear in the union on the right-hand side define a partition of the set that appears on the left-hand side). Here $\boldsymbol{e}_p$ is a vector of length n with all elements equal to zero except the p-th element which is equal to one, and $\boldsymbol{S}_p$ is the p-th column of the matrix $\boldsymbol{S}$.

The proof can be found in Appendix A.

In particular, note that the reachability set $\mathcal{RS}$ can be partitioned by conditioning on the number of tokens in some place $p \in \mathcal{P}$:

$$\mathcal{RS} = \mathcal{E}(\mathcal{P}, \boldsymbol{V}) = \bigcup_{i \in M_p(\mathcal{P}, \boldsymbol{V})} \left[i\boldsymbol{e}_p + \mathcal{E}(\mathcal{P} \backslash \{p\}, \boldsymbol{V} - i\boldsymbol{S}_p) \right],$$

and all of the sets $\mathcal{E}(\mathcal{P}\backslash\{p\}, \boldsymbol{V} - i\boldsymbol{S}_p), i \in M_p(\mathcal{P}, \boldsymbol{V})$ that appear on the right-hand side can be further partitioned in the same manner. This yields a recursive state space decomposition scheme that is analogous to the decomposition of the state space of queueing networks considered in Appendix B, and is the basis for the convolution algorithm for computation of the normalising constant. Note that this decomposition exists only for a S-invariant reachable SPNs.

For product-form S-invariant reachable nets, a recurrence relation, derived from the above decomposition of the state space, is used as the basis of the convolution algorithm. For any nonempty subset $\mathcal{P}' \subseteq \mathcal{P}$ of the set of places, and any vector $\boldsymbol{W}$, $\boldsymbol{0} \leqslant \boldsymbol{W} \leqslant \boldsymbol{V}$, the following value is defined:

$$G(\mathcal{P}', \boldsymbol{W}) = \sum_{\boldsymbol{m} \in \mathcal{E}(\mathcal{P}', \boldsymbol{W})} \prod_{p \in \mathcal{P}'} g_p(m_p).$$

Obviously, for an S-invariant reachable product-form net with the set of places $\mathcal{P}$ and load vector $\boldsymbol{V} = \boldsymbol{S}\boldsymbol{m}_0$, $G(\mathcal{P}, \boldsymbol{V})$ is equal to the normalising constant G. From the lemma it directly follows that for a nonempty subset $\mathcal{P}' \subseteq \mathcal{P}$ of the set of places, any place $p \in \mathcal{P}'$ and any vector $\boldsymbol{W}$, $\boldsymbol{0} \leqslant \boldsymbol{W} \leqslant \boldsymbol{V}$, the following recurrence relation holds:

$$G(\mathcal{P}', \boldsymbol{W}) = \sum_{i \in M_p(\mathcal{P}', \boldsymbol{W})} g_p(i) G(\mathcal{P}' \backslash \{p\}, \boldsymbol{W} - i\boldsymbol{S}_p), \tag{8}$$

and the boundary conditions can be derived from the definition of $G(\cdot, \cdot)$:

$$\begin{gathered} G(\{p\}, \boldsymbol{W}) = \textstyle\sum_{i \in M_p(\{p\}, \boldsymbol{W})} g_p(i), \forall p \in \mathcal{P}, \forall \boldsymbol{W} \text{ such that } \boldsymbol{0} \leqslant \boldsymbol{W} \leqslant \boldsymbol{V}; \\ G(\mathcal{P}', \boldsymbol{W}) = 0, \forall \mathcal{P}' \subseteq \mathcal{P}, \forall \boldsymbol{W} \text{ such that } \mathcal{E}(\mathcal{P}', \boldsymbol{W}) = \varnothing. \end{gathered} \tag{9}$$

The convolution algorithm is a recursive algorithm that is based on the above recurrence relation; assuming some ordering of the set of places $\mathcal{P}$, it computes the normalising constant $G = G(\mathcal{P}, \boldsymbol{V})$ by conditioning on the number of tokens in the first place p, as in the recurrence relation (8). The values $G(\mathcal{P}\backslash\{p\}, \boldsymbol{V} - i\boldsymbol{S}_p)$ needed in the computation are computed by recursively applying the same recurrence relation, until one of the boundary conditions (9) is reached. The convolution algorithm for SPNs has two drawbacks:

- It can only be applied to S-invariant reachable SPNs, and verification of S-invariant reachability in general requires either generation of the reachability set or a manual proof.

- It computes the marking sets $M_p(\mathcal{P}', \boldsymbol{W})$ for all combinations of places p, subsets $\mathcal{P}'$ of the set of places and vectors $\boldsymbol{W}$ that are encountered during the recursive computation. This in general requires solving systems of linear Diophantine equations—where solutions are sought only in the set of integers—which is a difficult problem, limiting the performance of the convolution algorithm.

Later in this section, we present in Proposition 5.2 a method for computing the marking sets by determining feasibility of certain integer linear programming problems (ILP); this is also a difficult problem (in fact, it is NP-complete [41]) but as shown in Section 6 the obtained ILPs are manageable for the tested models.

5.2. Comparison of MDD-rec with the convolution algorithm for stochastic Petri nets

In the following, MDDs with levels corresponding to SPN places will be used to encode the reachability set of the SPN. Since arc labels with source nodes in levels $l \in \{n, \ldots, 1\}$ of the MDD are confined to sets $\{0, 1, \ldots, k_l\}$ for some numbers $k_l \in \mathbb{N}$, functions $\phi_l : M_l(\mathcal{P}, \boldsymbol{V}) \to \{0, \ldots, |M_l(\mathcal{P}, \boldsymbol{V})|-1\}, l \in \{n, \ldots, 1\}$ are defined that rename SPN place markings into MDD arc labels:

$$\phi_l(i) = |\{j \in M_l(\mathcal{P}, \boldsymbol{V}) : j < i\}|, \forall i \in M_l(\mathcal{P}, \boldsymbol{V}).$$

Function ϕ_l can be seen to map marking i of place l to its ordinal number in the set $M_l(\mathcal{P}, \boldsymbol{V})$ of all possible markings of place l. Since by assumption all places are bounded, these functions are well defined. For $l \in \{1, \ldots, n\}$, where n is the number of SPN places as before, sets of places $\mathcal{P}_l \subseteq \mathcal{P}$ are defined with $\mathcal{P}_l = \{P_1, \ldots, P_l\}$, and sets of renamed submarkings $\mathcal{S}_\phi(\mathcal{P}_l, \boldsymbol{W}) \subseteq \prod_{i=l}^{1} \{0, \ldots, |M_i(\mathcal{P}, \boldsymbol{V})| - 1\}$ for a vector $\boldsymbol{W} \in \mathbb{R}^n$ are defined with

$$\mathcal{S}_\phi(\mathcal{P}_l, \boldsymbol{W}) = \{(\phi_l(m_l), \ldots, \phi_1(m_1)) : \boldsymbol{m} \in \mathcal{E}(\mathcal{P}_l, \boldsymbol{W})\}.$$

With these definitions, from Lemma 5.1 the following formulation of the renamed state space decomposition directly follows:

$$\forall l \in \{1, \ldots, n\}, \forall \boldsymbol{W} \text{ s.t. } \boldsymbol{0} \leqslant \boldsymbol{W} \leqslant \boldsymbol{V},$$

$$\mathcal{S}_\phi(\mathcal{P}_l, \boldsymbol{W}) = \bigcup_{i \in M_l(\mathcal{P}_l, \boldsymbol{W})} (\phi_l(i)) \mathcal{S}_\phi(\mathcal{P}_{l-1}, \boldsymbol{W} - i\boldsymbol{S}_l).$$

Lemma 5.2. *Let an ordered quasi-reduced MDD over the renamed potential reachability set $\hat{\mathcal{S}} = \prod_{i=n}^{1} \{0, \ldots, |M_i(\mathcal{P}, \boldsymbol{V})| - 1\}$ with n levels corresponding to Petri net places encode the renamed reachability set $\mathcal{S}_\phi(\mathcal{P}_n, \boldsymbol{V}) = \{(\phi_n(m_n), \ldots, \phi_1(m_1)) \in \hat{\mathcal{S}} : \boldsymbol{S}\boldsymbol{m} = \boldsymbol{V}\}$ of an S-invariant reachable stochastic Petri net with n places, load vector $\boldsymbol{V} = \boldsymbol{S}\boldsymbol{m}_0$ and matrix of minimal support S-invariants $\boldsymbol{S}$. Assume that a non-terminal node $\langle l.\boldsymbol{W} \rangle$ on some level greater than 1, $l > 1$,*

encodes the set $\mathcal{S}_\phi(\mathcal{P}_l, \boldsymbol{W})$ *for a subset of places* $\mathcal{P}_l \subseteq \mathcal{P}$ *and some vector* $\boldsymbol{W}$, $\boldsymbol{0} \leqslant \boldsymbol{W} \leqslant \boldsymbol{V}$, *that is* $\mathcal{B}(\langle l.\boldsymbol{W}\rangle) = \mathcal{S}_\phi(\mathcal{P}_l, \boldsymbol{W})$.

Then there exist nodes on level $l-1$ *that encode sets from the family* $\{\mathcal{S}_\phi(\mathcal{P}_{l-1}, \boldsymbol{W} - i\boldsymbol{S}_l) : i \in M_l(\mathcal{P}_l, \boldsymbol{W})\}$; *denote these nodes with* $\langle(l-1).(\boldsymbol{W} - i\boldsymbol{S}_l)\rangle, i \in M_l(\mathcal{P}_l, \boldsymbol{W})$, *respectively. The destination nodes of the outgoing arcs of the node* $\langle l.\boldsymbol{W}\rangle$ *are:*

$$\langle l.\boldsymbol{W}\rangle[\phi_l(i)] = \begin{cases} \langle(l-1).(\boldsymbol{W} - i\boldsymbol{S}_l)\rangle & \text{if } i \in M_l(\mathcal{P}_l, \boldsymbol{W}), \\ \langle 0.0\rangle & \text{otherwise.} \end{cases}$$

The proof can be found in Appendix A.

Proposition 5.1. Structure of an MDD encoding state space of an S-invariant reachable SPN. *Let an ordered quasi-reduced MDD over the renamed potential reachability set* $\hat{\mathcal{S}} = \prod_{i=n}^{1}\{0, \ldots, |M_i(\mathcal{P}, \boldsymbol{V})| - 1\}$ *with* n *levels corresponding to Petri net places encode the renamed reachability set* $\mathcal{S}_\phi(\mathcal{P}_n, \boldsymbol{V}) = \{(\phi_n(m_n), \ldots, \phi_1(m_1)) \in \hat{\mathcal{S}} : \boldsymbol{S}\boldsymbol{m} = \boldsymbol{V}\}$ *of an S-invariant reachable stochastic Petri net with* n *places, load vector* $\boldsymbol{V} = \boldsymbol{S}\boldsymbol{m}_0$ *and matrix of minimal support S-invariants* $\boldsymbol{S}$.

Then the structure of the MDD is as follows:

1. *The top level* n *contains a single root node, denoted here with* $\langle n.\boldsymbol{V}\rangle$, *which encodes the renamed reachability set* $\mathcal{S}_\phi(\mathcal{P}_n, \boldsymbol{V})$.
2. *Levels* $l \in \{n-1, \ldots, 1\}$ *each contain exactly the nodes which encode sets from the family* $\{\mathcal{S}_\phi(\mathcal{P}_l, \boldsymbol{U} - i\boldsymbol{S}_{l+1}) : \exists$ *node on level* $l+1$ *that encodes* $\mathcal{S}_\phi(\mathcal{P}_{l+1}, \boldsymbol{U})$ *and* $i \in M_{l+1}(\mathcal{P}_{l+1}, \boldsymbol{U})\}$. *A node that encodes the set* $\mathcal{S}_\phi(\mathcal{P}_l, \boldsymbol{W})$ *from this family is denoted with* $\langle l.\boldsymbol{W}\rangle$.
3. *Each non-terminal node* $\langle l.\boldsymbol{W}\rangle$ *has exactly* $|M_l(\mathcal{P}, \boldsymbol{V})|$ *outgoing arcs with destination nodes as follows. For all* $i \in M_l(\mathcal{P}, \boldsymbol{V})$,

$$\langle l.\boldsymbol{W}\rangle[\phi_l(i)] = \begin{cases} \langle(l-1).(\boldsymbol{W} - i\boldsymbol{S}_l)\rangle & \text{if } l > 1 \text{ and } i \in M_l(\mathcal{P}_l, \boldsymbol{W}), \\ \langle 0.1\rangle & \text{if } l = 1 \text{ and } i \in M_l(\mathcal{P}_l, \boldsymbol{W}), \\ \langle 0.0\rangle & \text{otherwise.} \end{cases}$$

The proof can be found in Appendix A.

From the definition of the mass of MDD node from Section 4, it can be easily seen that the following recurrence relation holds for an MDD that encodes the state space of an S-invariant reachable SPN:

$$M(\langle l.\boldsymbol{W}\rangle) = \sum_{i \in M_l(\mathcal{P}_l, \boldsymbol{W})} g_l(i) M(\langle(l-1).(\boldsymbol{W} - i\boldsymbol{S}_l)\rangle).$$

where $g_1, \ldots, g_n$ are the functions from the product-form (1), as defined in Sec. 3.2. After identification $M(\langle l.\boldsymbol{W}\rangle) = G(\mathcal{P}_l, \boldsymbol{W})$, it is obvious that the above recurrence relation is equivalent to the recurrence relation (8). Further, from the described structure of the MDD, the following boundary conditions can be derived:

$M(\langle 1.\boldsymbol{W}\rangle) = \sum_{i \in M_1(\mathcal{P}_1, \boldsymbol{W})} g_1(i) = G(\mathcal{P}_1, \boldsymbol{W}), \forall \boldsymbol{W}$ such that $\boldsymbol{0} \leqslant \boldsymbol{W} \leqslant \boldsymbol{V}$;
$M(\langle l.\boldsymbol{W}\rangle) = 0 = G(\mathcal{P}_l, \boldsymbol{W}), \forall l \in \{1, \ldots, n\}, \forall \boldsymbol{W}$ such that $\mathcal{E}(\mathcal{P}_l, \boldsymbol{W}) = \varnothing$.

While these boundary conditions include only a subset of the boundary conditions (9), they do include all boundary conditions that can be encountered in the convolution algorithm when computing the normalising constant of an SPN with load vector $\boldsymbol{V}$, assuming the places in the convolution algorithm are taken in the order $P_n, P_{n-1}, \ldots, P_1$. Since $M(\langle n.\boldsymbol{V}\rangle) = G = G(\mathcal{P}, \boldsymbol{V})$, it follows that MDD-rec computes the normalising constant for an S-invariant reachable product-form SPN with load vector $\boldsymbol{V} = \boldsymbol{S}\boldsymbol{m}_0$ and is equivalent to the convolution algorithm in terms of data flow. The following paragraphs discuss the differences between MDD-rec and the convolution algorithm.

Computation of the marking sets. An MDD encoding the reachability set of an S-invariant reachable SPN effectively also encodes the marking sets of places that are needed in the convolution algorithm. For example, if the node $\langle l.\boldsymbol{W}\rangle$ appears in the MDD, the marking set $M_l(\mathcal{P}_l, \boldsymbol{W})$ is encoded by the outgoing arcs of the node $\langle l.\boldsymbol{W}\rangle$:

$$i \in M_l(\mathcal{P}_l, \boldsymbol{W}) \iff \langle l.\boldsymbol{W}\rangle[\phi(i)] \neq \langle 0.0\rangle.$$

Thus the difficult problem of computing the marking sets is here solved by the generation of the reachability set. In contrast, for the convolution algorithm the marking set $M_l(\mathcal{P}_l, \boldsymbol{W})$ can be obtained by considering the feasibility of a certain ILP, as follows.

Proposition 5.2. Computing marking sets. *Let $(\mathcal{P}, \mathcal{T}, I, O, W, \boldsymbol{m}_0)$ be an S-invariant reachable SPN with n places and load vector $\boldsymbol{V} = \boldsymbol{S}\boldsymbol{m}_0$. Consider a nonempty subset $\mathcal{P}'$ of places, $\varnothing \neq \mathcal{P}' \subseteq \mathcal{P}$, place $p \in \mathcal{P}'$ and vector $\boldsymbol{W} \in \mathbb{N}^n$ such that $\boldsymbol{0} \leqslant \boldsymbol{W} \leqslant \boldsymbol{V}$.*

Then for every $i \in \mathbb{N}$, $i \in M_p(\mathcal{P}', \boldsymbol{W})$ if and only if the ILP

$$\begin{array}{ll} \textit{maximize} & 0 \\ \textit{subject to} & \boldsymbol{S}\boldsymbol{x} = \boldsymbol{W} - i\boldsymbol{S}_p \\ & x_q = 0, \forall q \in \mathcal{P}\backslash(\mathcal{P}'\backslash\{p\}) \\ & x_q \geqslant 0, \forall q \in \mathcal{P}'\backslash\{p\} \\ & \boldsymbol{x} \in \mathbb{Z}^n \end{array} \tag{10}$$

is feasible.

The proof can be found in Appendix A.

Therefore, the convolution algorithm can be compared to a recursive walk on an MDD that encodes the reachability set of the SPN where, instead of simply following arcs in the MDD, the feasibility of ILP of the form (10) is solved for each possible marking of the considered place (up to the bound of the marking of the place). Based on these considerations, for S-invariant reachable SPNs with known place marking bounds, it follows that the convolution algorithm can be used to generate an MDD encoding of its reachability set.

Checking the S-invariant reachability condition. As mentioned before, the S-invariant reachability condition on SPNs is needed for the convolution algorithm to be able to decompose the reachability set by conditioning on the number of tokens in a place. In contrast, an MDD that encodes the reachability set of an SPN already defines such decomposition by its very structure; the S-invariant reachability condition is thus not needed for MDD-rec. Thus, for SPNs with finite reachability sets, MDD-rec is more general than the convolution algorithm because it can handle any product-form SPN with a finite state space, in principle. In practice, MDD-rec can be used on the SPNs for which the MDD encoding the reachability set can be generated in reasonable time. Furthermore, to the best of our knowledge there is no known algorithm that can decide S-invariant reachability for a general SPN without generating its reachability set, which further limits the practical scope of the convolution algorithm.

5.3. Efficient computation of the performance measures

We consider efficient computation of the performance measures defined in Sec. 3.1. In order to compute the probabilities occuring in the definitions of the performance measures, we change line 5 of the algorithm MDD-rec by modifying the recurrence relation (4) as follows.

We can efficiently compute the probability $P(m_j = k)$ that the place P_j contains exactly k tokens as

$$P(m_j = k) = \frac{1}{G} M_{m_j=k}(\langle K.r \rangle)$$

where $M_{m_j=k}$ is defined by the modified recurrence relation (4) that sums only over the markings for which $m_j = k$:

$$M_{m_j=k}(\langle l.p \rangle) := \begin{cases} g_l(k) M_{m_j=k}(\langle l.p \rangle[\phi_l(k)]) & \text{if } l = j, \\ \sum_{s_l \in \mathcal{S}_l} g_l(s_l) M_{m_j=k}(\langle l.p \rangle[\phi_l(s_l)]) & \text{otherwise.} \end{cases}$$

The probability $P(m_j = k)$ is used in the computation of the average number of tokens $n(P_j)$ and the place utilization $u(P_j)$.

Similarly, we can efficiently compute the probability $P(e_j \geqslant k)$ that the transition T_j is enabled with the enabling degree equal to or larger than k as

$$P(e_j \geqslant k) = \frac{1}{G} M_{e_j \geqslant k}(\langle K.r \rangle)$$

where $M_{e_j \geqslant k}$ is defined by the modified recurrence relation (4) that sums only over the markings for which $e_j \geqslant k$:

$$M_{e_j \geqslant k}(\langle l.p \rangle) := \sum_{\substack{s_l \in \mathcal{S}_l \\ s_l \geqslant k I_l(T_j)}} g_l(s_l) M_{e_j \geqslant k}(\langle l.p \rangle[\phi_l(s_l)]).$$

The probability $P(e_j = k)$ that the transition T_j is enabled with enabling degree exactly k can then be computed as $P(e_j = k) = P(e_j \geqslant k) - P(e_j \geqslant k+1)$. The

probability $P(e_j \geqslant 1)$ is used in the computation of the transition utilization $u(T_j)$, while $P(e_j = k)$ is used in the computation of the transition throughput $x(T_j)$ and the place throughput $x(P_j)$.

More complex performance measures that cannot be computed by a straightforward modification of MDD-rec could be computed by generating MDDs that encode subsets of the reachability set whose probability masses are needed in the computation of the sought performance measure. The unnormalised probability masses can then be computed by applying the (unmodified) algorithm MDD-rec to the generated MDDs and then normalised using the normalising constant. MDDs as a data structure support many useful set and arithmetic operations that can be used in the generation of the needed MDDs.

MVA [23] is an algorithm for computation of the performance measures of product-form SPNs. Compared with the methods proposed above, however, MVA has several disadvantages: 1) like convolution, it assumes S-invariant reachability, in general requiring manual proof of it, 2) it assumes SS firing semantics (i.e., it only allows marking-independent transition firing rates), and 3) it computes certain measures for all vectors $\boldsymbol{W}$ such that $\boldsymbol{0} \leqslant \boldsymbol{W} \leqslant \boldsymbol{V}$, where $\boldsymbol{V}$ is the load vector of the considered SPN, which makes its performance likely to be much worse than the algorithms proposed here which, due to equivalence of MDD-rec and the convolution algorithm for S-invariant reachable SPNs, always perform a recursive walk over MDD nodes defined using only those vectors that are encountered during execution of the convolution algorithm.

6. Experiments

We have implemented the proposed algorithm MDD-rec 1 for SPNs in C++, using the *Multi-terminal and Edge-valued Decision Diagram LibrarY* (MEDDLY) [42]. In our implementation, the generation of the MDD encoding of the reachability set of the SPN is handled by an efficient saturation algorithm [39] provided by MEDDLY. We have implemented the labeling function L from the algorithm MDD-rec using the hashing-based map implementation `unordered_map` from the C++11 standard.

To compare the performance of MDD-rec with the convolution algorithm for SPNs, we have also implemented the convolution algorithm in C++. First the matrix $\boldsymbol{S}$ of minimal-support place invariants is computed using the standard algorithm [43]. Then the convolution algorithm, based on the recurrence relation (8) (9), is executed as follows. We use the optimisation solver Gurobi Optimizer [44] for computation of the marking sets by solving ILPs from Proposition 5.2. In order to lower the number of ILPs that need to be constructed and solved during the execution of the convolution algorithm, we implemented two optimizations based on place invariants: when computing a marking set $M_p(\mathcal{P}', \boldsymbol{W})$, we 1) search for a place invariant in the matrix $\boldsymbol{S}$ of minimal-support place invariants that uniquely determines the marking of the place p (i.e., an invariant whose support contains p and all other places from the support have already been considered in ancestor recursive calls) and 2) if such invariant is not found, then we consider only the possible markings of the place

p up to a bound that is obtained from the invariants in the matrix $\boldsymbol{S}$ and the vector $\boldsymbol{W}$. These two optimisations result in a reduction in the scale of several orders of magnitude in the number of considered ILPs and in correspondingly much shorter execution time of the convolution algorithm. The caching of values $G(\cdot,\cdot)$ computed during the execution was also implemented using the C++11 `unordered_map`.

6.1. Solution of the running example

Let us return to the SPN presented in Section 3.3. While the considered SPN is in product-form, it is not S-invariant reachable: for example, all markings of the form $(k, 300-k, 250, 120, 0, 0, 0, 0)$ for $0 \leqslant k \leqslant 300$ satisfy the S-invariant reachability condition (5), while among these markings only the markings with k equal to a multiple of 10 are reachable. Therefore, the convolution algorithm cannot be used on this SPN.

We evaluate the proposed method on the SPN shown in Fig. 1, scaling the initial marking by keeping the number of machines in P_2 constant at 300 and increasing the number of jobs in P_4 and P_5. For each tested model size, we compute the average number of tokens in the places and obtain the throughputs of the transitions. Fig. 3 shows the reachability set cardinality for the tested models, reaching nearly 300 millions markings. Fig. 4 shows the computed throughputs of the jobs of the two types. It is interesting to note that the throughputs of the jobs of type 2 first increase and then start decreasing as the network becomes more heavily loaded. Fig. 5 shows the computation time: the proposed method takes less than 10 minutes for each of the tested models.

For the smallest model size, we also performed a simulation-based analysis using the Petri net tool PIPE2 [45]. The results obtained in 10 minutes had relatively large 95% confidence intervals for the average place markings. The exact values (computed by the proposed method) for two places were outside the confidence intervals reported by the simulation, likely due to the sensitivity of the simulation on the initial marking.

6.2. Performance comparison

To test performance of the algorithms, we have adapted models from [46]. All models that follow are S-invariant reachable SPNs with finite reachability sets, with the exception of the Example 3 model (depicted in Fig. 10) which is S-invariant reachable only for $k = 1$. For $k > 1$ this SPN is not S-invariant reachable because place P_3 can be marked only with the numbers of tokens that are multiples of k—so that the reachability set is smaller than the one for $k = 1$—while the matrix of minimal-support S-invariants is the same as in the case $k = 1$; thus in condition 5 only sufficiency (direction $\Rightarrow$) holds. For $k > 1$, the convolution algorithm cannot be used to compute the normalising constant, while MDD-rec can.

Fig. 6 shows an S-invariant reachable SPN parameterized by $n \in \mathbb{N}$ and composed of n identical subnets—the first of which contains places P_1, P_2 and transitions T_1, T_2—connected in a series. Tokens in places $P_{2k}, k = 1, \ldots, n-1$ model

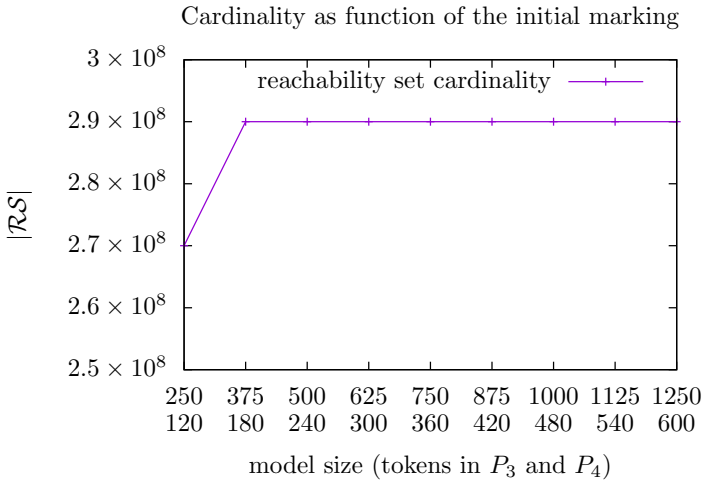

Figure 3: Cardinality of the reachability set

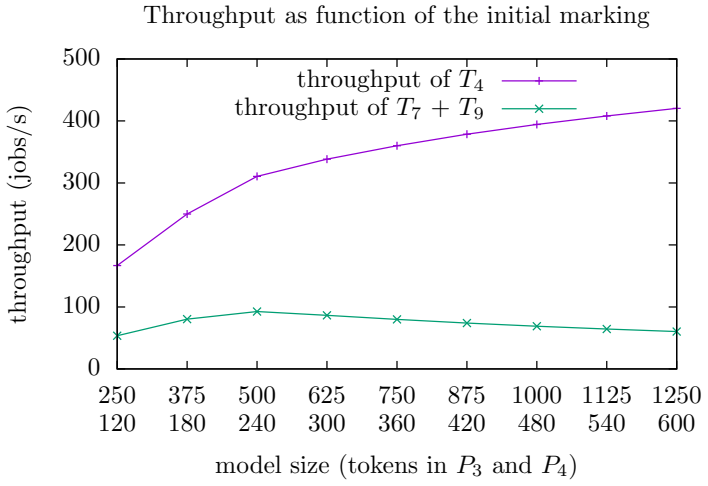

Figure 4: Throughput

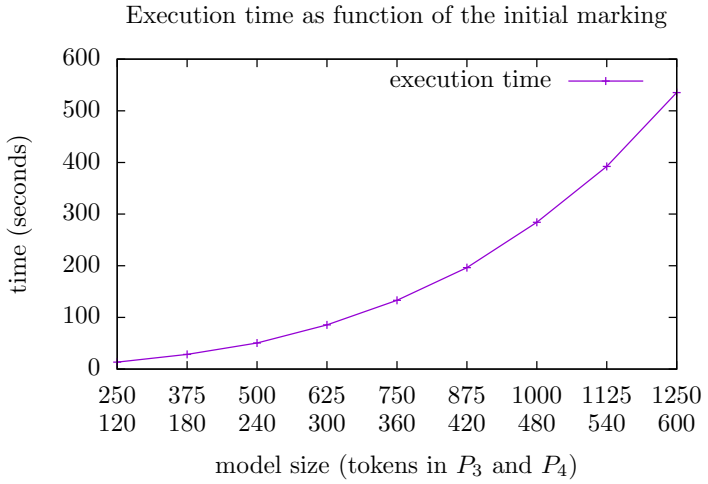

Figure 5: Execution time

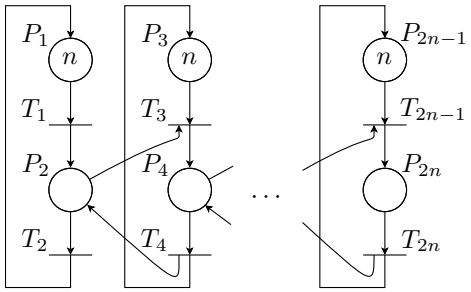

Figure 6: Example 1 SPN.

resources that are reserved by the firing of transitions $T_{2k+1}, k = 1, \ldots, n-1$, respectively, and are released by the firing of transitions $T_{2k+2}, k = 1, \ldots, n-1$, respectively. The resulting net has $2n$ places and the same number of transitions. It was obtained by scaling a variant of the Example 1 net from [46].

Likewise, Fig. 7 shows an S-invariant reachable SPN parameterized by n and consisting of the same n subnets which in this net compete for central resources modeled by tokens in place P_0. This net has $2n + 1$ places and $2n$ transitions. This model was obtained by scaling the Example 2 net from [46].

In order to measure performance of the algorithms, we have run them on the Example 1 and Example 2 SPNs for n ranging from 5 to 100 with step 5. For both SPNs, this results in the range of models with reachability set cardinalities ranging from 252 for $n = 5$ to approximately 9.055×10^{58} for $n = 100$. In the MDDs and the convolution algorithm, the SPN places were taken in the order

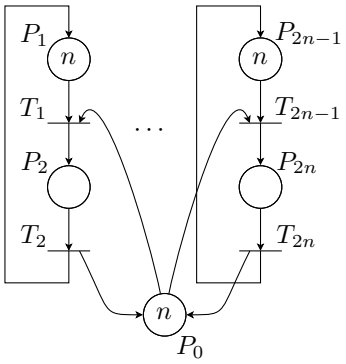

Figure 7: Example 2 SPN.

in which they are numbered in the figures. Fig. 8 and 9 contain log-log plots of the average execution time in seconds (measured over 10 runs) for the MDD generation, the proposed algorithm MDD-rec and the convolution algorithm, as functions of the reachability set size $|\mathcal{RS}|$. The error bars, too tight to see on the plots, represent 95% confidence intervals for the execution time. In both cases, the time taken for MDD-rec is orders of magnitude lower than the time taken for the convolution algorithm; this is due to the convolution algorithm needing to solve many ILPs during execution. The time taken to generate the MDDs is within an order of magnitude of the time taken for convolution. While the convolution algorithm and the combined MDD generation + MDD-rec take a comparable amount of time (note that the y-axis in the plots is logarithmic so that the time needed for MDD generation can be taken as a good approximation of the total time needed for MDD generation and MDD-rec), MDD-rec is fully automatable and applicable to general SPNs while the convolution algorithm works only on S-invariant reachable SPNs and requires a separate proof of S-invariant reachability, which to the best of our knowledge in the general case requires either generating the reachability set or performing a manual proof.

6.3. Sensitivity on the ordering of SPN places

The performance of all three algorithms depends on the order in which the SPN places are taken. We analyse this sensitivity on the ordering of the places on the SPN model adapted from Example 3 in [46] and reproduced here in Fig. 10. This SPN has only 5 places and it is thus possible to systematically test the performance of the algorithms for all $5! = 120$ possible orderings of the places. For $k = 1$, the SPN is S-invariant reachable.

Fig. 11 shows a plot of the average execution time in seconds (over 10 runs) for the Example 3 SPN with $k = 1$ and $n = 50$ with error bars (again too tight to see on the plots) representing 95% confidence intervals for the execution time. On the y-axis is the execution time, and on the x-axis are all 120 possible permutations of places. For clarity, the permutations of places are sorted so

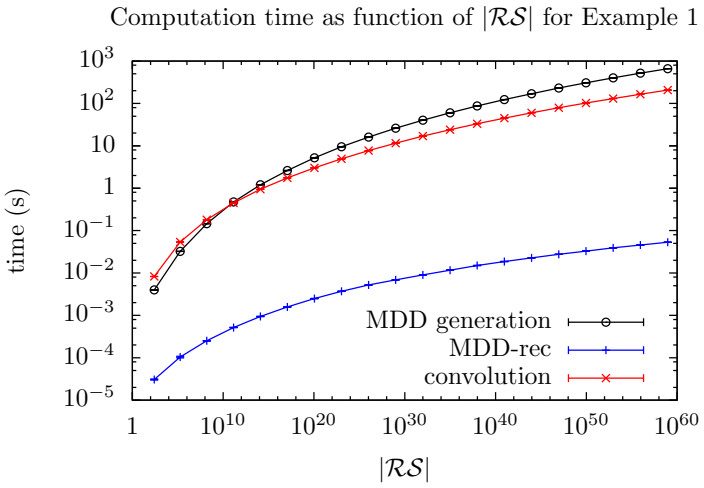

Figure 8: Measured times for Example 1 net.

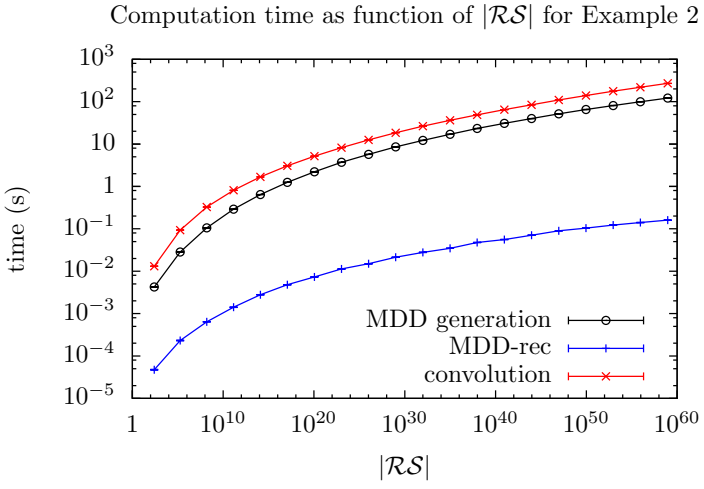

Figure 9: Measured times for Example 2 net.

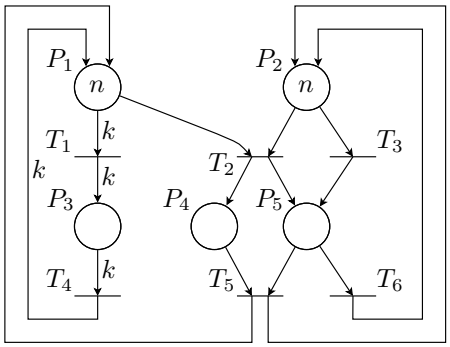

Figure 10: Example 3 SPN which is S-invariant reachable only for $k = 1$.

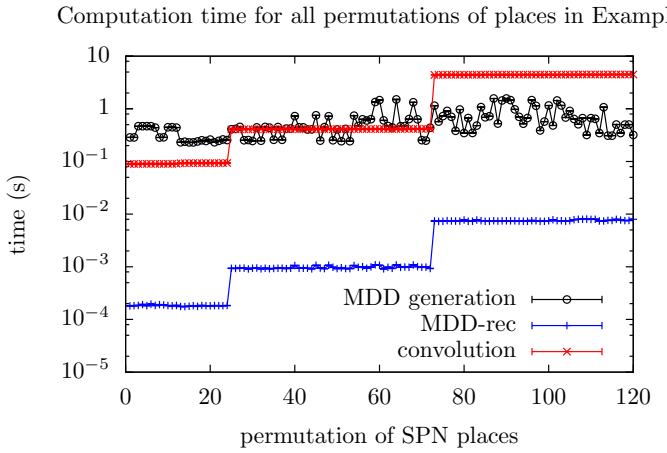

Figure 11: Measured times for Example 3 net ($n = 50$) and all place orderings.

that the time taken by the convolution algorithm is monotonically increasing on the plot.

We note that the set of all permutations of the places can be partitioned into three equivalence classes with regard to the execution time of the convolution algorithm. These are depicted in Table 1 where **class** is the set of indices of the permutations as depicted on the x-axis of Fig. 11, **MDD nodes** is the number of nodes in the MDD that encodes the reachability set of the SPN, and **ILPs solved** is the number of the ILPs solved during the execution of the convolution algorithm. The table shows that the number of nodes in the MDD over which MDD-rec walks (which is equal to the number of values $G(\cdot, \cdot)$ that the convolution algorithm computes) is highly sensitive on the ordering of the SPN places; the same holds for the number of ILPs solved by the convolution algorithm.

class	MDD nodes	ILPs solved
$\{1, \ldots, 24\}$	157	1479
$\{25, \ldots, 72\}$	2757	6579
$\{73, \ldots, 120\}$	5307	72879

Table 1: Classes of place permutations for Example 3 SPN with $n = 50$.

As can be expected, the convolution algorithm and MDD-rec are fastest on the first class which has the lowest number of MDD nodes and are slowest on the third class with the highest number of MDD nodes. The execution times obtained for the MDD generation are more variable due to the saturation algorithm generating many intermediate MDDs whose sizes also depend on the ordering of the places. The MDD generation is in some cases faster and in some cases slower than the convolution algorithm. Additional measurements show that for $n = 10$ the MDD generation is faster than the convolution algorithm for all permutations of the places, and for $n = 100$ it is slower than the convolution algorithm for all permutations. MDD-rec is faster than both by orders of magnitude in all cases.

From the above considerations and other more limited experiments, we expect to find a similar situation for other models as well: performance of all algorithms will likely vary over orders of magnitude with the permutations of places, but MDD-rec will always be much faster than the other two algorithms.

7. Conclusion

In this paper, we have presented the algorithm MDD-rec for computation of the normalising constant for product-form models with finite state spaces, based on efficient generation and encoding of the models' state spaces using MDDs. The algorithm can be applied for the computation of the stationary distribution of general product-form models with finite state spaces, although its main domain of application consists of models expressed in terms of SPNs.

Product-form SPNs are crucial for the performance evaluation of distributed systems since they are much more expressive than queueing networks, and allow the modeler to deal with fork-join constructs and batches of jobs or resources. Therefore, the lack of a general algorithm for their solution has restricted their application or has lead the performance analysis to rely on expensive and sometimes inaccurate simulations (see Section 6). The main contribution of this work is the introduction of a novel algorithm, namely MDD-rec, that aims at covering this gap. In contrast with previous contributions, the application of MDD-rec does not require any manual proof by the analyst and can be applied to the entire domain of bounded SPNs in product-form.

A second contribution consists in the introduction of methods for computation of performance measures for product-form SPNs based on the evaluation of the normalising constant. These methods are more general than the MVA algo-

rithm which shares with convolution the requirement of S-invariant reachability and in addition requires marking-independence of transition firing rates.

Finally, we have implemented the proposed algorithm MDD-rec for SPNs, compared its performance to the convolution algorithm on two example models, and analysed sensitivity of the algorithms to the order in which SPN places are taken during execution on a third model. We find that MDD-rec is much faster due to the need to solve many ILP feasibility problems during execution of the convolution algorithm. When the generation of MDDs is also taken into account, the performance is within an order of magnitude of the convolution algorithm for the considered models; however, for this last comparison to be fair, we think that the time taken for execution of convolution should also include the checking of the S-invariant reachability condition which in general requires generating the reachability set anyway.

The possible avenues for future work include handling SPNs that have a product-form over subnets instead of single places, making a detailed comparison of the performance of MVA with the proposed methods for computing performance measures, testing the proposed methods on a realistic model, and implementing the methods in a tool for analysis of product-form models.

References

[1] C. Smith, L. Williams, Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software, Addison-Wesley Professional, 2001.

[2] P. D. Ezhilchelvan, I. Mitrani, Multi-class resource sharing with batch arrivals and complete blocking, in: QEST 2017, 14th International Conference on Quantitative Evaluation of Systems, Berlin, Germany, September 5-7, 2017, 2017, pp. 157–169.

[3] M. Gribaudo, M. Iacono, M. Kiran, A performance modeling framework for lambda architecture based applications, Future Generation Computer Systems 86 (2018) 1032–1041.

[4] J. Rolia, G. Casale, D. Krishnamurthy, S. Dawson, S. Kraft, Predictive modelling of SAP ERP applications: challenges and solutions, in: 4th International Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS '09, Pisa, Italy, October 20-22, 2009, 2009, p. 9.

[5] E. C. d'Oro, S. Colombo, M. Gribaudo, M. Iacono, D. Manca, P. Piazzolla, Modeling and evaluating performances of complex edge computing based systems: a firefighting support system case study, in: Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS 2017, Venice, Italy, December 05-07, 2017, 2017, pp. 261–262.

[6] A. Marin, S. Balsamo, J. Fourneau, LB-networks: A model for dynamic load balancing in queueing networks, Perform. Eval. 115 (2017) 38–53.

[7] S. Balsamo, A. Marin, Queueing Networks, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 34–82.

[8] V. de Nitto Persone, A. Di Lonardo, Approximating finite resources: An approach based on MVA, Perform. Eval. 131 (2019) 1–21.

[9] M. K. Molloy, Performance analysis using stochastic Petri nets, IEEE Trans. on Comput. 31 (9) (1982) 913–917.

[10] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, G. Franceschinis, Modelling with generalized stochastic Petri nets, Wiley, 1995.

[11] T. Murata, Petri nets: Properties, analysis and applications, Proc. of the IEEE 77 (4) (1989) 541–580.

[12] J. R. Jackson, Jobshop-like queueing systems, Management Science 10 (1963) 131–142.

[13] F. Baskett, K. M. Chandy, R. R. Muntz, F. G. Palacios, Open, closed, and mixed networks of queues with different classes of customers, J. ACM 22 (2) (1975) 248–260.

[14] W. Henderson, D. Lucic, P. G. Taylor, A net level performance analysis of Stochastic Petri Nets, J. Austral. Math. Soc. Ser. B 31 (1989) 176–187.

[15] A. A. Lazar, T. G. Robertazzi, Markovian Petri Net Protocols with Product Form Solution., Perf. Eval. 12 (1) (1991) 67–77.

[16] S. Haddad, J. Mairesse, H. Nguyen, Synthesis and analysis of product-form Petri nets, Fundam. Inform. 122 (1-2) (2013) 147–172.

[17] G. Balbo, S. C. Bruell, M. Sereno, Product form solution for Generalized Stochastic Petri Nets, IEEE Trans. on Software Eng. 28 (10) (2002) 915–932.

[18] J. P. Buzen, Computational algorithms for closed queueing networks with exponential servers, Commun. ACM 16 (9) (1973) 527–531.

[19] J. Coleman, W. Henderson, P. Taylor, Product form equilibrium distributions and a convolution algorithm for stochastic Petri nets, Performance Evaluation 26 (3) (1996) 159 – 180.

[20] T. Kam, T. Villa, R. Brayton, A. Sangiovanni-Vincentelli, Multi-valued decision diagrams: theory and applications, Multiple-Valued Logic 4 (1) (1998) 9–62.

[21] A. S. Miner, G. Ciardo, Efficient reachability set generation and storage using decision diagrams, in: S. Donatelli, J. Kleijn (Eds.), Application and Theory of Petri Nets 1999, Vol. 1639 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1999, pp. 6–25.

[22] G. Ciardo, G. Lüttgen, R. Siminiceanu, Efficient symbolic state-space construction for asynchronous systems, in: M. Nielsen, D. Simpson (Eds.), Application and Theory of Petri Nets 2000, Vol. 1825 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2000, pp. 103–122.

[23] M. Sereno, G. Balbo, Mean value analysis of stochastic Petri nets, Perform. Eval. 29 (1) (1997) 35–62.

[24] A. S. Miner, G. Ciardo, S. Donatelli, Using the exact state space of a Markov model to compute approximate stationary measures, SIGMETRICS Perform. Eval. Rev. 28 (1) (2000) 207–216.

[25] M. Wan, G. Ciardo, A. S. Miner, Approximate steady-state analysis of large Markov models based on the structure of their decision diagram encoding, Performance Evaluation 68 (5) (2011) 463 – 486.

[26] A. Hinton, M. Kwiatkowska, G. Norman, D. Parker, PRISM: A Tool for Automatic Verification of Probabilistic Systems, Springer, Berlin, Heidelberg, 2006, pp. 441–444.

[27] G. Casale, Accelerating performance inference over closed systems by asymptotic methods, POMACS 1 (1) (2017) 8:1–8:25.

[28] G. Casale, A generalized method of moments for closed queueing networks, Perform. Eval. 68 (2) (2011) 180–200.

[29] J. Huls, C. Pilch, P. Schinke, J. Delicaris, A. Remke, State-space construction of HPnGs with multiple general transition firings, in: Proc. of the 16th Int. Conf. on Quantitative Evaluation of Systems (QEST), 2019, p. to appear.

[30] C. R. Vázquez, M. Silva Suárez, Stochastic hybrid approximations of Markovian Petri nets, IEEE Trans. on Systems, Man, and Cybernetics: Systems 45 (2015) 1231–1244.

[31] E. G. Amparore, P. Buchholz, S. Donatelli, Great-Nsolve: a tool integration for (Markov regenerative) stochastic Petri nets, in: Proc. of the 16th Int. Conf. on Quantitative Evaluation of Systems (QEST), 2019, p. to appear.

[32] Z. Liu, Performance analysis of stochastic timed Petri nets using linear programming approach, IEEE Trans. Software Eng. 24 (11) (1998) 1014–1030.

[33] A. Marin, S. Balsamo, P. G. Harrison, Analysis of stochastic Petri nets with signals, Performance Evaluation 69 (11) (2012) 551–572.

[34] R. Osman, P. G. Harrison, Approximating closed fork-join queueing networks using product-form stochastic Petri-nets, Journal of Systems and Software 110 (2015) 264–278.

[35] S. Balsamo, G. Dei Rossi, A. Marin, Modelling retrial-upon-conflict systems with product-form stochastic Petri nets, in: Analytical and Stochastic Modelling Techniques and Applications - 20th International Conference, ASMTA Ghent, Belgium, July 8-10, 2013, 2013, pp. 52–66.

[36] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytia, Reducing latency via redundant requests: Exact analysis, in: Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, 2015, pp. 347–360.

[37] T. Bonald, C. Comte, Balanced fair resource sharing in computer clusters, Performance Evaluation 116 (2017) 70–83.

[38] U. Ayesta, T. Bodas, I. Verloop, On a unifying product form framework for redundancy models, Performance Evaluation 127-128 (2018) 93–119.

[39] A. S. Miner, Saturation for a general class of models, IEEE Transactions on Software Engineering 32 (8) (2006) 559–570.

[40] M. Sereno, G. Balbo, Computational algorithms for product form solution stochastic Petri nets, in: Petri Nets and Performance Models, 1993. Proceedings. 5th International Workshop on, 1993, pp. 98–107.

[41] A. Schrijver, Theory of Linear and Integer Programming, John Wiley & Sons, Inc., New York, NY, USA, 1986.

[42] J. Babar, A. S. Miner, Meddly: Multi-terminal and edge-valued decision diagram library, in: QEST 2010, Seventh International Conference on the Quantitative Evaluation of Systems, Williamsburg, Virginia, USA, 15-18 September 2010, 2010, pp. 195–196.

[43] R. David, H. Alla, Discrete, Continuous, and Hybrid Petri Nets, 2nd Edition, Springer Publishing Company, Incorporated, 2010.

[44] Gurobi Optimization, Inc., Gurobi optimizer, `http://www.gurobi.com/products/gurobi-optimizer`, [online; accessed 22/3/2019] (2019).

[45] N. J. Dingle, W. J. Knottenbelt, T. Suto, Pipe2: A tool for the performance evaluation of generalised stochastic Petri nets, SIGMETRICS Perform. Eval. Rev. 36 (4) (2009) 34–39.

[46] J. L. Coleman, Algorithms for product-form stochastic Petri nets-a new approach, in: Petri Nets and Performance Models, 1993. Proceedings., 5th International Workshop on, 1993, pp. 108–116.

Appendix A. Proofs

This appendix contains the proofs of the statements from Sec. 5.

Lemma 5.1. *For any S-invariant reachable SPN $(\mathcal{P}, \mathcal{T}, I, O, W, \boldsymbol{m}_0)$ with load vector $\boldsymbol{V} = \boldsymbol{S}\boldsymbol{m}_0$, the following decomposition of the set $\mathcal{E}(\mathcal{P}', \boldsymbol{W})$ holds for all subsets $\mathcal{P}' \subseteq \mathcal{P}$, arbitrary place $p \in \mathcal{P}'$ and all vectors $\boldsymbol{W}$, $\boldsymbol{0} \leqslant \boldsymbol{W} \leqslant \boldsymbol{V}$:*

$$\mathcal{E}(\mathcal{P}', \boldsymbol{W}) = \bigcup_{i \in M_p(\mathcal{P}', \boldsymbol{W})} \left[i\boldsymbol{e}_p + \mathcal{E}(\mathcal{P}' \backslash \{p\}, \boldsymbol{W} - i\boldsymbol{S}_p) \right],$$

where the union is over mutually disjoint sets (i.e., the sets that appear in the union on the right-hand side define a partition of the set that appears on the left-hand side). Here $\boldsymbol{e}_p$ is a vector of length n with all elements equal to zero except the p-th element which is equal to one, and $\boldsymbol{S}_p$ is the p-th column of the matrix $\boldsymbol{S}$.

Proof. We successively rewrite the set on the right-hand side of the definition of $\mathcal{E}(\mathcal{P}', \boldsymbol{W})$:

$$\mathcal{E}(\mathcal{P}', \boldsymbol{W}) = \{\boldsymbol{m} : \boldsymbol{S}\boldsymbol{m} = \boldsymbol{W} \text{ and } \forall q \in \mathcal{P} \backslash \mathcal{P}', m_q = 0\}.$$

We split this set into the disjoint union over the possible markings of place p, obtaining

$$\mathcal{E}(\mathcal{P}', \boldsymbol{W}) = \bigcup_{i \in M_p(\mathcal{P}', \boldsymbol{W})} \{\boldsymbol{m} : \boldsymbol{S}\boldsymbol{m} = \boldsymbol{W} \text{ and } m_p = i \text{ and } \forall q \in \mathcal{P} \backslash \mathcal{P}', m_q = 0\}.$$

Rewriting $\boldsymbol{m}$ with $i\boldsymbol{e}_p + \boldsymbol{m}'$ (where $\boldsymbol{m}'$ is obtained from $\boldsymbol{m}$ by replacing its p-th component with 0), we can write the above disjoint union as

$$\bigcup_{i \in M_p(\mathcal{P}', \boldsymbol{W})} \{i\boldsymbol{e}_p + \boldsymbol{m}' : \boldsymbol{S}\boldsymbol{m}' = \boldsymbol{W} - i\boldsymbol{S}_p \text{ and } m'_p = 0 \text{ and } \forall q \in \mathcal{P} \backslash \mathcal{P}', m'_q = 0\}.$$

From this, considering the equivalence (recall that $p \in \mathcal{P}'$)

$$m'_p = 0 \text{ and } \forall q \in \mathcal{P} \backslash \mathcal{P}', m'_q = 0 \iff \forall q \in \mathcal{P} \backslash (\mathcal{P}' \backslash \{p\}), m'_q = 0,$$

we obtain that $\mathcal{E}(\mathcal{P}', \boldsymbol{W})$ is equal to

$$\bigcup_{i \in M_p(\mathcal{P}', \boldsymbol{W})} \{i\boldsymbol{e}_p + \boldsymbol{m}' : \boldsymbol{S}\boldsymbol{m}' = \boldsymbol{W} - i\boldsymbol{S}_p \text{ and } \forall q \in \mathcal{P} \backslash (\mathcal{P}' \backslash \{p\}), m'_q = 0\}.$$

Finally, considering that, by definition (6),

$$\mathcal{E}(\mathcal{P}' \backslash \{p\}, \boldsymbol{W} - i\boldsymbol{S}_p) = \{\boldsymbol{m}' : \boldsymbol{S}\boldsymbol{m}' = \boldsymbol{W} - i\boldsymbol{S}_p \text{ and } \forall q \in \mathcal{P} \backslash (\mathcal{P}' \backslash \{p\}), m'_q = 0\},$$

we obtain the union from the lemma statement

$$\mathcal{E}(\mathcal{P}', \boldsymbol{W}) = \bigcup_{i \in M_p(\mathcal{P}', \boldsymbol{W})} \left[i\boldsymbol{e}_p + \mathcal{E}(\mathcal{P}' \backslash \{p\}, \boldsymbol{W} - i\boldsymbol{S}_p) \right].$$

Clearly this union is over mutually disjoint sets. □

Lemma 5.2. *Let an ordered quasi-reduced MDD over the renamed potential reachability set* $\hat{\mathcal{S}} = \prod_{i=n}^{1}\{0,\ldots,|M_i(\mathcal{P},\boldsymbol{V})|-1\}$ *with* n *levels corresponding to Petri net places encode the renamed reachability set* $\mathcal{S}_\phi(\mathcal{P}_n,\boldsymbol{V}) = \{(\phi_n(m_n),\ldots,\phi_1(m_1)) \in \hat{\mathcal{S}} : \boldsymbol{S}\boldsymbol{m} = \boldsymbol{V}\}$ *of an S-invariant reachable stochastic Petri net with* n *places, load vector* $\boldsymbol{V} = \boldsymbol{S}\boldsymbol{m}_0$ *and matrix of minimal support S-invariants* $\boldsymbol{S}$*. Assume that a non-terminal node* $\langle l.\boldsymbol{W}\rangle$ *on some level greater than* 1*,* $l > 1$*, encodes the set* $\mathcal{S}_\phi(\mathcal{P}_l,\boldsymbol{W})$ *for a subset of places* $\mathcal{P}_l \subseteq \mathcal{P}$ *and some vector* $\boldsymbol{W}$*,* $\boldsymbol{0} \leqslant \boldsymbol{W} \leqslant \boldsymbol{V}$*, that is* $\mathcal{B}(\langle l.\boldsymbol{W}\rangle) = \mathcal{S}_\phi(\mathcal{P}_l,\boldsymbol{W})$*.*

Then there exist nodes on level $l-1$ *that encode sets from the family* $\{\mathcal{S}_\phi(\mathcal{P}_{l-1}, \boldsymbol{W} - i\boldsymbol{S}_l) : i \in M_l(\mathcal{P}_l,\boldsymbol{W})\}$*; denote these nodes with* $\langle(l-1).(\boldsymbol{W}-i\boldsymbol{S}_l)\rangle, i \in M_l(\mathcal{P}_l,\boldsymbol{W})$*, respectively. The destination nodes of the outgoing arcs of the node* $\langle l.\boldsymbol{W}\rangle$ *are:*

$$\langle l.\boldsymbol{W}\rangle[\phi_l(i)] = \begin{cases} \langle(l-1).(\boldsymbol{W}-i\boldsymbol{S}_l)\rangle & \text{if } i \in M_l(\mathcal{P}_l,\boldsymbol{W}), \\ \langle 0.0\rangle & \text{otherwise.} \end{cases}$$

Proof. From the following equalities

$$\begin{array}{c} \mathcal{S}_\phi(\mathcal{P}_l,\boldsymbol{W}) = \bigcup_{i\in M_l(\mathcal{P}_l,\boldsymbol{W})}(\phi_l(i))\mathcal{S}_\phi(\mathcal{P}_{l-1},\boldsymbol{W}-i\boldsymbol{S}_l) \\ \| \\ \mathcal{B}(\langle l.\boldsymbol{W}\rangle) = \bigcup_{i\in M_l(\mathcal{P},\boldsymbol{V})}(\phi_l(i))\mathcal{B}(\langle l.\boldsymbol{W}\rangle[\phi_l(i)]) \end{array}$$

and because ϕ_l is an injective function, it easily follows that

$$\mathcal{B}(\langle l.\boldsymbol{W}\rangle[\phi_l(i)]) = \begin{cases} \mathcal{S}_\phi(\mathcal{P}_{l-1},\boldsymbol{W}-i\boldsymbol{S}_l) & \text{if } i \in M_l(\mathcal{P}_l,\boldsymbol{W}), \\ \varnothing & \text{otherwise.} \end{cases}$$

Therefore there exist nodes on level $l-1$ that encode the sets from the family $\{\mathcal{S}_\phi(\mathcal{P}_{l-1},\boldsymbol{W}-i\boldsymbol{S}_l) : i \in M_l(\mathcal{P}_l,\boldsymbol{W})\}$; denote these nodes as in the statement of the lemma and the configuration of the arcs follows. □

Proposition 5.1. Structure of an MDD encoding state space of an S-invariant reachable SPN. *Let an ordered quasi-reduced MDD over the renamed potential reachability set* $\hat{\mathcal{S}} = \prod_{i=n}^{1}\{0,\ldots,|M_i(\mathcal{P},\boldsymbol{V})|-1\}$ *with* n *levels corresponding to Petri net places encode the renamed reachability set* $\mathcal{S}_\phi(\mathcal{P}_n,\boldsymbol{V}) = \{(\phi_n(m_n),\ldots,\phi_1(m_1)) \in \hat{\mathcal{S}} : \boldsymbol{S}\boldsymbol{m} = \boldsymbol{V}\}$ *of an S-invariant reachable stochastic Petri net with* n *places, load vector* $\boldsymbol{V} = \boldsymbol{S}\boldsymbol{m}_0$ *and matrix of minimal support S-invariants* $\boldsymbol{S}$*.*

Then the structure of the MDD is as follows:

1. *The top level* n *contains a single root node, denoted here with* $\langle n.\boldsymbol{V}\rangle$*, which encodes the renamed reachability set* $\mathcal{S}_\phi(\mathcal{P}_n,\boldsymbol{V})$*.*
2. *Levels* $l \in \{n-1,\ldots,1\}$ *each contain exactly the nodes which encode sets from the family* $\{\mathcal{S}_\phi(\mathcal{P}_l,\boldsymbol{U}-i\boldsymbol{S}_{l+1}) : \exists$ *node on level* $l+1$ *that encodes* $\mathcal{S}_\phi(\mathcal{P}_{l+1},\boldsymbol{U})$ *and* $i \in M_{l+1}(\mathcal{P}_{l+1},\boldsymbol{U})\}$*. A node that encodes the set* $\mathcal{S}_\phi(\mathcal{P}_l,\boldsymbol{W})$ *from this family is denoted with* $\langle l.\boldsymbol{W}\rangle$*.*
3. *Each non-terminal node* $\langle l.\boldsymbol{W}\rangle$ *has exactly* $|M_l(\mathcal{P},\boldsymbol{V})|$ *outgoing arcs with destination nodes as follows. For all* $i \in M_l(\mathcal{P},\boldsymbol{V})$*,*

$$\langle l.\boldsymbol{W}\rangle[\phi_l(i)] = \begin{cases} \langle(l-1).(\boldsymbol{W}-i\boldsymbol{S}_l)\rangle & \text{if } l > 1 \text{ and } i \in M_l(\mathcal{P}_l,\boldsymbol{W}), \\ \langle 0.1\rangle & \text{if } l = 1 \text{ and } i \in M_l(\mathcal{P}_l,\boldsymbol{W}), \\ \langle 0.0\rangle & \text{otherwise.} \end{cases}$$

Proof. Since, by definition, the root node $\langle n.\boldsymbol{V}\rangle$ is the only node at level n and it encodes set $\mathcal{S}_\phi(\mathcal{P}_n, \boldsymbol{V})$ by assumption, statement 1. holds.

We now prove statements 2. and 3. for the lower levels by finite induction, starting from the level $n-1$. For $n=1$, statement 2. is trivial and statement 3. can be easily checked by inspection of the decision diagram. In the following we assume that $n > 1$.

Induction base. By applying Lemma 5.2 on the node $\langle n.\boldsymbol{V}\rangle$, it follows that statement 2. holds for level $n-1$ and statement 3. holds for the single node in level n.

Induction step. Let $l \in \{n-1, \ldots, 2\}$ and assume that statement 2. holds for level l and that statement 3 holds for all nodes in level $l+1$. Applying the Lemma 5.2 to each of the nodes $\langle l.\boldsymbol{W}\rangle$ from statement 2. in turn, and noting that, due to canonicity, same sets cannot be encoded by different nodes, we obtain that statement 2. holds for level $l-1$, and that statement 3. holds for level l.

By induction, statement 2. holds for all levels $1, \ldots, n-1$ and statement 3. holds for all nodes in all levels $2, \ldots, n$.

We still need to show that statement 3. holds for all nodes in level $l = 1$. In this case, since $\mathcal{B}(\langle 1.\boldsymbol{W}\rangle) = \mathcal{S}_\phi(\mathcal{P}_1, \boldsymbol{W}) = \{\phi_1(m_1) : m_1 \in M_1(\mathcal{P}_1, \boldsymbol{W})\}$, it is easy to establish:

$$\langle l.\boldsymbol{W}\rangle[\phi_l(i)] = \begin{cases} \langle 0.1\rangle & \text{if } i \in M_l(\mathcal{P}_l, \boldsymbol{W}), \\ \langle 0.0\rangle & \text{otherwise} \end{cases}$$

which concludes the proof. □

Proposition 5.2. Computing marking sets. *Let $(\mathcal{P}, \mathcal{T}, I, O, W, \boldsymbol{m}_0)$ be an S-invariant reachable SPN with n places and load vector $\boldsymbol{V} = \boldsymbol{S}\boldsymbol{m}_0$. Consider a nonempty subset $\mathcal{P}'$ of places, $\varnothing \neq \mathcal{P}' \subseteq \mathcal{P}$, place $p \in \mathcal{P}'$ and vector $\boldsymbol{W} \in \mathbb{N}^n$ such that $\boldsymbol{0} \leqslant \boldsymbol{W} \leqslant \boldsymbol{V}$.*

Then for every $i \in \mathbb{N}$, $i \in M_p(\mathcal{P}', \boldsymbol{W})$ if and only if the ILP

$$\begin{array}{ll} \textit{maximize} & 0 \\ \textit{subject to} & \boldsymbol{S}\boldsymbol{x} = \boldsymbol{W} - i\boldsymbol{S}_p \\ & x_q = 0, \forall q \in \mathcal{P}\backslash(\mathcal{P}'\backslash\{p\}) \\ & x_q \geqslant 0, \forall q \in \mathcal{P}'\backslash\{p\} \\ & \boldsymbol{x} \in \mathbb{Z}^n \end{array} \tag{10}$$

is feasible.

Proof. Let $i \in \mathbb{N}$ be arbitrary. From the definition (7), we obtain the equivalence

$$i \in M_p(\mathcal{P}', \boldsymbol{W}) \iff \exists \boldsymbol{m} \in \mathcal{E}(\mathcal{P}', \boldsymbol{W}) \text{ such that } m_p = i\,.$$

By applying Lemma 5.1 to $\mathcal{E}(\mathcal{P}', \boldsymbol{W})$, we obtain that the right-hand side of the above is equivalent to

$$\exists \boldsymbol{m} \in \bigcup_{j \in M_p(\mathcal{P}', \boldsymbol{W})} \left[j\boldsymbol{e}_p + \mathcal{E}(\mathcal{P}'\backslash\{p\}, \boldsymbol{W} - j\boldsymbol{S}_p) \right] \text{ such that } m_p = i\,.$$

From (6), it is clear that all markings in the sets $\mathcal{E}(\mathcal{P}'\backslash\{p\}, \boldsymbol{W} - j\boldsymbol{S}_p)$ in the above union have the p-th component equal to 0. From this, we conclude that the set $i\boldsymbol{e}_p + \mathcal{E}(\mathcal{P}'\backslash\{p\}, \boldsymbol{W} - i\boldsymbol{S}_p)$, obtained for $j = i$, is equal to the set of *all* markings from the above union whose p-th component is equal to i. Therefore, the above is equivalent to

$$i\boldsymbol{e}_p + \mathcal{E}(\mathcal{P}'\backslash\{p\}, \boldsymbol{W} - i\boldsymbol{S}_p) \neq \varnothing$$

which is equivalent to

$$\mathcal{E}(\mathcal{P}'\backslash\{p\}, \boldsymbol{W} - i\boldsymbol{S}_p) \neq \varnothing\,.$$

Using the definition (6) again, we finally obtain that this is equivalent to

$$\exists \boldsymbol{x} \in \mathbb{N}^n \text{ such that } \boldsymbol{S}\boldsymbol{x} = \boldsymbol{W} - i\boldsymbol{S}_p \text{ and } \forall q \in \mathcal{P}\backslash(\mathcal{P}'\backslash\{p\}), x_q = 0\,.$$

It is clear that the last statement is true if and only if the ILP (10) is feasible. □

Appendix B. Comparison of MDD-rec algorithm with convolution on queueing networks

In this section, a class of queueing networks considered in [18] is taken into consideration and the algorithm MDD-rec proposed in Section 4 is compared with the convolution algorithm for computation of the normalising constant.

QNs considered here consist of a number of service stations in which a number of customers is served. State of a QN is a tuple of the numbers of customers in the stations. After a customer is served at a station, it is probabilistically routed to some (possibly the same) station. The networks are assumed to be closed (there are no arrivals of customers into the network and no departures from the network), single-class and single-chain (there is only a single type of customers; thus the service times and the routing probabilities are the same for all customers and do not change), with state-independent customer routing and with exponential service times.

The (constant) number of customers in the network is denoted with n and the number of service stations with m. The service time for a customer at the i-th service station, $i \in \{1, \ldots, m\}$, is exponentially distributed with mean $(a_i(n_i)\mu_i)^{-1}$, where $a_i : \{1, 2, \ldots, n\} \to \mathbb{R}_{>0}$ is a positive function, $\mu_i \in \mathbb{R}_{>0}$ is a positive real number, and n_i is the number of customers at the i-th station. After completing the service at the i-th station, the probability that the customer will require service at the j-th station is given by p_{ij}, for $i, j \in \{1, 2, \ldots, m\}$.

In this case, submodels correspond to the service stations of the queueing network, thus $K = m$, local state spaces $\mathcal{S}_m, \ldots, \mathcal{S}_1$ of submodels are all equal to the set $\{0, \ldots, n\}$ and the potential state space $\hat{\mathcal{S}}$ is equal to the set $\{0, \ldots, n\}^m$. In [18], authors define sets

$$\mathcal{S}(l, p) = \left\{ (s_l, \ldots, s_1) \in \{0, \ldots, n\}^l : \sum_{k=1}^{l} s_k = p \right\}, l \in \{1, \ldots, m\}, p \in \{1, \ldots, n\}.$$

The state space $\mathcal{S}$ of the model is then

$$\mathcal{S} = \mathcal{S}(m,n) = \left\{ (s_m, \ldots, s_1) \in \hat{\mathcal{S}} : \sum_{k=1}^{m} s_k = n \right\}.$$

Let $(x_1, \ldots, x_m) \in \mathbb{R}^m_{>0}$ be a real positive solution to the equations

$$\mu_j x_j = \sum_{i=1}^{m} \mu_i x_i p_{ij}, i, j \in \{1, \ldots, m\}$$

and let functions $A_i : \mathcal{S}_i \to \mathbb{R}_{>0}, i \in \{1, \ldots, m\}$ be defined with

$$A_i(s) = \begin{cases} 1 & \text{if } s = 0, \\ \prod_{j=1}^{s} a_i(j) & \text{if } s > 0. \end{cases}$$

The state probabilities can then be written as a product over the submodels:

$$P(\boldsymbol{s}) = \frac{1}{G} \prod_{k=1}^{m} \frac{x_k^{s_k}}{A_k(s_k)},$$

which is in product form (1) with functions g_k set to

$$g_k(s_k) = \frac{x_k^{s_k}}{A_k(s_k)}, s_k \in \mathcal{S}_k, k \in \{1, \ldots, m\},$$

and with the normalising constant G defined by (2).

In the following subsections, the standard convolution algorithm [18] for computation of the normalising constant for the considered class of QNs is first recalled and then compared to MDD-rec. By examining the structure of the MDD encoding the state space of the QN it is shown that the two algorithms are equivalent in terms of computational complexity and data flow.

B.1. Convolution algorithm for computation of normalising constant for queueing networks

In [18] the authors define

$$G(l,p) = \sum_{\boldsymbol{s} \in \mathcal{S}(l,p)} \prod_{k=1}^{l} g_k(s_k), l \in \{1, \ldots, m\}, p \in \{0, \ldots, n\}.$$

Clearly, $G = G(m,n)$. The following recurrence relation is then derived:

$$G(l,p) = \sum_{i=0}^{p} g_l(i) G(l-1, p-i), l \in \{2, \ldots, m\}, p \in \{1, \ldots, n\}. \tag{B.1}$$

with boundary conditions

$$G(1,p) = g_1(p), p \in \{1,\ldots,n\};$$
$$G(l,0) = 1, l \in \{1,\ldots,m\}.$$

To implement the computation of $G(m,n)$ that uses the above reccurence relation, one would use a cache table with m rows indexed with $1,\ldots,m$ and $n+1$ columns indexed with $0,\ldots,n$, and store $G(l,p)$—once computed—in the table at position (l,p). At the start of the algorithm, the table positions $(1,1),\ldots,(1,n)$ and $(1,0),\ldots,(m,0)$ would be populated with initial values derived from the corresponding boundary conditions, and the algorithm would proceed by recursively computing $G(m,n)$, caching in the table and reusing any previously computed values $G(l,p)$. It is easy to see that such a recursive computation would require $\mathcal{O}(mn^2)$ applications of the functions g_i and the same number of additions and multiplications and that the data flow of the computation is defined by (B.1). In the next section, it is shown that MDD-rec is equivalent to this scheme.

The above computation can be reordered in order to minimize the recursion depth and/or space complexity of the algorithm. This is what the authors in [18] do by exploiting the fact that in the reccurence relation (B.1) term $G(l,p)$ is expressed using terms $G(l-1,0),\ldots,G(l-1,p)$ (where the first parameter is always lower by 1) to reorder the computation so that only two neighbouring rows of the cache table need to be stored at any one time: first $G(2,1),\ldots,G(2,n)$ are computed from the initial values (stored in the first row) and stored in the second row of the cache table, then $G(3,1),\ldots,G(3,n)$ are computed using the values in the second row and so on. While this lowers the space complexity of the algorithm, the data flow and the needed number of operations are the same as in the computation described above. In the next section, it is shown that the proposed recursion on the MDD encoding $\mathcal{S}$ can be equivalently reordered so that labels $L(\cdot)$ from the algorithm MDD-rec 1 need to be stored for only two neighbouring levels of the MDD at any one time.

B.2. Comparison of MDD-rec with the convolution algorithm for queueing networks

In this subsection it is assumed that the MDD encoding the state space $\mathcal{S}$ of the QN model from the considered class is available, and that the nodes encoding the empty sets have been removed from this MDD, as remarked at the end of Section 4.1.

Lemma B.2.1. *Let an ordered quasi-reduced MDD over $\hat{\mathcal{S}} = \{0,1,\ldots,n\}^m$ with m levels corresponding to the QN service stations encode the state space $\mathcal{S} = \{(s_m,\ldots,s_1) \in \hat{\mathcal{S}} : \sum_{k=1}^{m} s_k = n\}$ of a closed queueing network with m service stations and n customers. Assume that a non-terminal node $\langle l.p\rangle$ on a level greater than 1, $l > 1$ encodes the set $\mathcal{S}(l,p)$ for some $p \in \{0,\ldots,n\}$, that is $\mathcal{B}(\langle l.p\rangle) = \mathcal{S}(l,p)$.*

Then there exist nodes on level $l-1$ that encode the sets $\mathcal{S}(l-1,0),\ldots,\mathcal{S}(l-1,p)$; denote these nodes with $\langle(l-1).0\rangle,\ldots,\langle(l-1).p\rangle$, respectively. The node $\langle l.p\rangle$ has $n+1$ outgoing arcs with destination nodes as follows:

$$\langle l.p\rangle[i] = \begin{cases} \langle(l-1).(p-i)\rangle & \text{if } p \geqslant i, \\ \langle 0.0\rangle & \text{otherwise.} \end{cases}$$

Proof. From the following equalities

$$\begin{array}{c} \mathcal{S}(l,p) = \bigcup_{i=0}^{p}(i)\mathcal{S}(l-1,p-i) \\ \| \\ \mathcal{B}(\langle l.p\rangle) = \bigcup_{i=0}^{n}(i)\mathcal{B}(\langle l.p\rangle[i]) \end{array}$$

it easily follows that

$$\mathcal{B}(\langle l.p\rangle[i]) = \begin{cases} \mathcal{S}(l-1,p-i) & \text{if } i \in \{0,\ldots,p\}, \\ \varnothing & \text{if } i \in \{p+1,\ldots,n\}. \end{cases}$$

Therefore, there exist nodes on level $l-1$ that encode the sets $\mathcal{S}(l-1,0),\ldots,\mathcal{S}(l-1,p)$; we denote these nodes as in the statement of the Lemma and the configuration of the arcs follows. Note that these $p+1$ sets are all different (due to canonicity, there can be no duplicate nodes among these $p+1$ nodes). □

Proposition B.2.1. Structure of the MDD encoding the state space of a QN. *Let an ordered quasi-reduced MDD over $\hat{\mathcal{S}} = \{0,1,\ldots,n\}^m$ with m levels corresponding to the QN service stations encode the state space $\mathcal{S} = \{(s_m,\ldots,s_1) \in \hat{\mathcal{S}} : \sum_{k=1}^{m} s_k = n\}$ of a closed queueing network with m service stations and n customers.*

Then the structure of the MDD is as follows:

1. *The top level m contains a single root node, denoted here with $\langle m.n\rangle$, which encodes the set $\mathcal{S}(m,n)$.*
2. *Levels $l \in \{m-1,\ldots,1\}$ each contain exactly $n+1$ nodes, denoted here with $\langle l.0\rangle,\ldots,\langle l.n\rangle$, which encode the sets $\mathcal{S}(l,0),\ldots,\mathcal{S}(l,n)$, respectively.*
3. *Each non-terminal node $\langle l.p\rangle$, $l>0$ has $n+1$ outgoing arcs with destination nodes as follows:*

$$\forall i \in \{0,\ldots,n\},\ \langle l.p\rangle[i] = \begin{cases} \langle(l-1).(p-i)\rangle & \text{if } l>1 \text{ and } p \geqslant i, \\ \langle 0.1\rangle & \text{if } l=1 \text{ and } p=i, \\ \langle 0.0\rangle & \text{otherwise.} \end{cases}$$

Proof. Since, by definition, root node $\langle m.n\rangle$ is the only node at level m and it encodes the set $\mathcal{S} = \mathcal{S}(m,n)$ by assumption, statement 1. holds.

We now prove statements 2. and 3. for the lower levels by finite induction, starting from the level $m-1$. For $m=1$, statement 2. is trivial and statement 3. can be easily checked by inspection of the decision diagram. In the following we assume that $m>1$.

Induction base. By applying Lemma B.2.1 to node $\langle m.n \rangle$, it follows that statement 2. holds for level $m-1$ and statement 3. holds for the single node in level m.

Induction step. Let $l \in \{m-1, \ldots, 2\}$ and assume that statement 2. holds for level l and that statement 3 holds for all nodes in level $l+1$. Applying the Lemma B.2.1 to each of the nodes $\langle l.0 \rangle, \ldots, \langle l.n \rangle$ from statement 2. in turn, and noting that, due to canonicity, same sets cannot be encoded by different nodes, we obtain that statement 2. holds for level $l-1$, and that statement 3. holds for level l.

By induction, statement 2. holds for all levels $1, \ldots, m-1$ and statement 3. holds for all nodes in all levels $2, \ldots, m$.

We still need to show that statement 3. holds for all nodes in level $l = 1$. In this case, since $\mathcal{B}(\langle 1.p \rangle) = \mathcal{S}(1, p) = \{p\}$, it is easy to establish:

$$\langle l.p \rangle[i] = \begin{cases} \langle 0.1 \rangle & \text{if } p = i, \\ \langle 0.0 \rangle & \text{otherwise} \end{cases}$$

which concludes the proof. □

From the just established structure of the MDD encoding the state space of a QN model and from the definition of the mass of an MDD node it follows that

$$M(\langle 1.p \rangle) = g_1(p) = G(1, p), p \in \{1, \ldots, n\},$$
$$M(\langle l.0 \rangle) = 1 = G(l, 0), l \in \{1, \ldots, m\},$$

and from the recurrence relation for the mass of the MDD node and the structure of the MDD it is easy to obtain:

$$M(\langle l.p \rangle) = \sum_{i=0}^{p} g_l(i) M(\langle (l-1).(p-i) \rangle), l \in \{2, \ldots, m\}, p \in \{1, \ldots, n\}.$$

which, after identifying $M(\langle l.p \rangle) = G(l, p)$, is equivalent to the recurrence relation (B.1). Thus, the data flow of MDD-rec is equivalent to the data flow of the convolution algorithm for the considered class of QNs. Furthermore, the computational complexity of these two algorithms is the same: $\mathcal{O}(mn^2)$ applications of the functions g_i and the same number of multiplications and additions. As with the convolution for QNs, if the computations of the masses of nodes are reordered so that the masses are computed from the bottom level upwards, the labels $L(\cdot)$ need to be stored only for two neighbouring levels at any one time. We stress again that the above considerations of the computational complexity do not take into account the generation and storage of the MDD itself.

Because the state spaces of the considered class of QNs are simply the sets $\mathcal{S}(m, n)$, it makes no sense in practice to generate the MDD encoding of the state space and run the proposed algorithm MDD-rec—the convolution algorithm is perfectly adequate and more efficient, both in time and space.