MTBDD-based Activity-Local State Graph Generation

Kai Lampka, Markus Siegle
Institute of Computer Science 7, Friedrich-Alexander-University Erlangen-Nuremberg
Email:{kilampka,siegle}@informatik.uni-erlangen.de

***Abstract*— We describe a method for constructing compact representations of labelled continuous-time Markov chains which are derived from high-level model descriptions, such as stochastic Petri nets, stochastic process algebras, etc.. Our approach extends existing techniques in that symbolic (i.e. MTBDD-based) representations are constructed for non-modular, flat model descriptions. The symbolic representation of the overall model is obtained by merging symbolic activity-local representations of transitions. Such a scheme will not only yield a compact symbolic representation, we also show that the state space of the overall model may only need to be explored explicitly in parts.**

I. Introduction

High-level performability models are usually specified by means of stochastic Petri nets, stochastic process algebra, etc., where the low-level model or state graph consists of all possible states and the set of all possible transitions between these states. Concurrency, compactly described in the high-level model, must be made explicit when deriving such a state graph. I.e. conventional state graph exploration algorithms expand every permutation of independent activities, where the number of visited states will grow exponentially. The goal is to explore huge state spaces in a time-efficient manner, and store the associated labelled continuous-time Markov chain in a compact way. We use binary decision diagrams (BDD) and extensions thereof for representing the state graph symbolically. It is known that extremely compact symbolic representations can be constructed if the compositional structure of the high-level model at hand is taken into consideration [6]. Our approach extends existing techniques in that symbolic representations are constructed for non-modular, flat model descriptions. In the approach described here, the symbolic representation of the overall model is obtained by merging symbolic activity-local representations of transitions. Such a scheme of activity-local state graph generation does not require the high-level description to be hierarchically structured in order to obtain a compact symbolic representation of the underlying state graph. We also show that the state space of the overall model may only need to be explored partially. Therefore the here illustrated approach is highly suited for performance evaluation tools, such as Möbius [2], where the high-level description is mapped onto the corresponding state graph in a monolithic manner. By applying this scheme, one can expect both runtime and memory savings.

The paper is organised as follows: In Sec. II, we briefly introduce BDDs and MTBDDs and the general idea of symbolic state graph representation. In Sec. III, the new scheme of activity-local state graph generation is described. Sec. IV discusses the issue of partial state space exploration, and Sec. V concludes the paper.

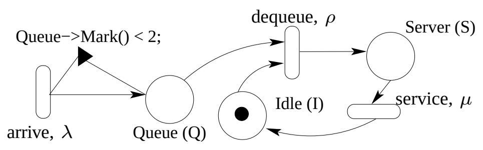


Fig. 1. A simple producer-consumer system described as a monolithic SAN

II. Symbolic State Graph Representation

A. State spaces and their encoding

During state space exploration, a state of the overall system is represented by a state descriptor which is a vector $\vec{s}$ consisting of n elements, i.e. $\vec{s} = (s_1, \ldots, s_n)$ where $s_i \in \{0, \ldots, K_i\}$ for all $1 \leq i \leq n$. The vector elements are called state variables (SVs). The state space can be explored in a monolithic fashion, by executing each enabled activity, one at a time, and determining the value of each SV s_i. As running example, we consider the SAN [5] model given in Fig. 1. Here, each state is described by a 3-dimensional state descriptor, where each element s_i indicates the number of tokens contained in the corresponding place p_i. The initial state is given by $\vec{s} = (0, 1, 0)$. The whole state graph is given in Fig. 2 (A), where the capacity of place Queue is bounded by 2. One may encode each SV s_i by applying an injective encoding function $\mathcal{E} : \{0, \ldots, K_i\} \mapsto \mathbb{B}^{n_{s_i}}$, where we may choose $n_{s_i} \geq \lceil \log_2(K_i + 1) \rceil$. In our running example the encoding of the initial state is given by the Boolean vector $\vec{b} = (00, 1, 0)$. One may also encode the activity labels in a similar way: If there are K_{Act} different activities, we can define a function $\mathcal{I} : \mathcal{A}ct \mapsto \{0, \ldots, K_{Act} - 1\}$ which returns an index for each activity. We can then encode the index of each activity by applying an encoding function $\mathcal{E}$ on the set of activity indices.

For encoding the state graph, Boolean vector $\vec{b}$ encodes the values of the SVs before (source state) and $\vec{b}\,'$ (target state) after an activity's execution, and the activity label is encoded by the Boolean vector $\vec{a}$. The execution of an activity l is thus encoded by the following scheme:

$$\Big((s_1,\ldots,s_n) \xrightarrow{l} (s'_1,\ldots,s'_n)\Big) \equiv (\vec{a},\vec{b},\vec{b}\,')$$

Note that, for simplicity, we concentrate on tangible states only, i.e. we assume that vanishing states are eliminated "on-the-fly".

B. MTBDD-based state graph representation

An ordered MTBDD M is a canonical representation of a function of type $f_{\mathsf{M}} : \mathbb{B}^n \mapsto \mathbb{D}$, where $\mathbb{D}$ is a finite set. We assume that the MTBDD variables have the following ordering: At the first n_A levels from the root are the variables a_i, encoding the activity labels. On the remaining levels we have $2n_s := 2\sum_{i=1}^{n} n_{s_i}$ variables, encoding the source- and target values of the n SVs. In order to obtain small MTBDD sizes, the variables b_i and b'_i are ordered in an interleaved fashion, yielding the following overall variable ordering [6]:

$$\mathsf{a}_1 \prec \ldots \mathsf{a}_{n_A} \prec \mathsf{b}_1 \prec \mathsf{b}'_1 \prec \ldots \prec \mathsf{b}_{n_s} \prec \mathsf{b}'_{n_s}$$

Standard arithmetic (and Boolean) operators can be implemented efficiently on the MTBDD data structure with the help of the so-called APPLY algorithm. The table in Fig. 2 (B) shows the binary encoding of the state graph of the running example, and part (C) shows the corresponding symbolic representation by means of an MTBDD. In the MTBDD, a dashed (solid) line indicates the value 0 (1) of the corresponding Boolean variable.

III. Symbolic activity-local state graph generation

A. Partitioning of the state descriptor

The execution of an activity l depends on a set of SVs, denoted as *pre*-set ($\bullet S_l$), where the execution itself will change a set of SVs, denoted as *post*-set ($S_l\bullet$). The union of these sets yields the set of dependent SVs, $S_{d_l} := \bullet S_l \cup S_l\bullet$, and its complement is denoted as S_{c_l}. The elements of S_{c_l} are the SVs, which are neither affected by l's execution, nor do they influence its enabling condition.

This concept of dependent and independent SVs yields the following encoding scheme for a transition induced by an activity l:

$$\Big((S_{d_l}, S_{c_l}) \xrightarrow{l} (S'_{d_l}, S_{c_l})\Big) \equiv (l, S_{d_l}, S'_{d_l}),$$

where S_{d_l} refers to the SV before and S'_{d_l} after l's execution. S_{c_l} can be omitted, since its elements are immaterial for l's execution. The possible values of S_{c_l} and S'_{c_l} will be inserted later during the stage of symbolic completion and composition.

We can apply the concept of pre- and post-set directly to the Boolean vectors which encode the state variables. For our running example, the activity-dependent binary encodings as well as the activity-dependent sets of Boolean vectors S_{d_l} and S_{c_l} are given in Fig. 2 (D) For example, the execution of activity *arrive* yields the following encoding scheme:

$$\Big((Q,I,S) \xrightarrow{arrive} (Q',I,S)\Big) \equiv (00, \mathsf{b}_1\mathsf{b}_2, \mathsf{b}'_1\mathsf{b}'_2)$$

B. Generation of the symbolic state graph representation

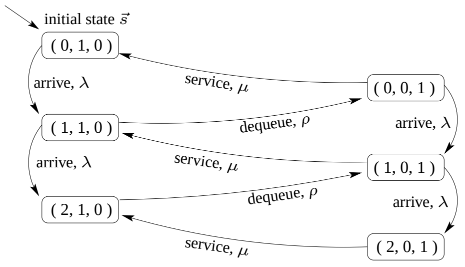
The idea behind activity-local state graph generation is as follows: At the first stage, K_{Act} MTBDDs M_l are constructed. These encode the set of dependent SVs (S_{d_l}) before and after a specific execution of activity l. Once all transitions are generated, each of these activity-local MTBDDs M_l needs to be supplemented by its individual set of symbolically encoded not-dependent SVs S_{c_l}, yielding the symbolic representation of the set of potential transitions induced by activity l. Finally the K_{Act} supplemented MTBDDs, denoted $\widetilde{\mathsf{M}_l}$, must be merged in order to encode the transition relation of the overall model. A symbolic reachability analysis needs to be carried out then, in order to restrict the potential transition relation to the actually reachable states.

1) Layout of the state graph generation procedure: We propose to break the major task of conventional state space exploration and symbolic encoding into two parts:
(a) A conventional state space exploration algorithm finds all transitions between reachable states, by successively firing all enabled transitions, one at a time, for each detected state descriptor. As a consequence the algorithm needs to operate on two data structures: (i) A state buffer, containing the already detected but not yet explored states. (ii) A transition buffer, holding detected transitions of the form $(\vec{s}, l, \lambda, \vec{s}\,')$ which are to be entered into the activity-local MTBDD M_l, where $\vec{s}$ and $\vec{s}\,'$ are the state descriptors before and after the execution of activity l, and where λ is the rate of the activity.
(b) The exploration part is complemented by an administration part, which collects and encodes the detected transitions from the transition buffer and inserts them into the activity-local MTBDDs. Furthermore, it must decide whether a state needs to be entered into the state buffer or not. Since the state space should be only visited partially, in order to save time, the conditions of inserting a state into the state buffer need to be considered carefully, see Sec. IV.

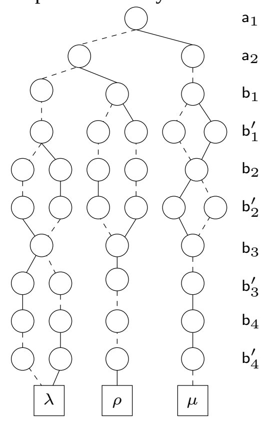
2) Generating the symbolic activity-local state graphs: Each transition $(\vec{s}, l, \lambda, \vec{s}\,')$ is taken from the transition buffer, the dependent SVs S_{d_l} are encoded in binary form and inserted into the respective MTBDD M_l. We propose to employ temporarily unreduced MTBDDs at this stage, in order to minimize the runtime of the insertion procedure [1]. The algorithm for inserting the MTBDD-based representation of an activity-local transition into M_l can be sketched as follows:

```
(0) SymbolicEnc($\mathsf{M}_l, \vec{s}_{d_l}, l, \lambda, \vec{s}_{d_l}\,'$)
(1)     $NewTrans := \mathcal{M}(\vec{b}; \mathcal{E}(\vec{s}_{d_l})) \wedge \mathcal{M}(\vec{b}\,'; \mathcal{E}(\vec{s}_{d_l}\,'))$
(2)     $NewTrans := NewTrans \cdot \lambda$
(3)     $\mathsf{M}_l := \mathsf{M}_l + NewTrans$
(4) return
```

(A) State graph of the running example



(C) Symbolic representation by MTBDD M



(B) Binary encoding of the state graph

$\vec{a}$	$\mathsf{b}_1\mathsf{b}_2$	b_3	b_4	$\mathsf{b}'_1\mathsf{b}'_2$	b'_3	b'_4	f_M
00	00	1	0	01	1	0	λ
00	01	1	0	10	1	0	λ
00	00	0	1	01	0	1	λ
00	01	0	1	10	0	1	λ
01	01	1	0	00	0	1	ρ
01	10	1	0	01	0	1	ρ
10	00	0	1	00	1	0	μ
10	01	0	1	01	1	0	μ
10	10	0	1	10	1	0	μ

(D) Activities, their encoding and their sets S_{d_l} and S_{c_l}

activity	j	$\vec{\mathsf{a}}$	$S_{d_l} := \bullet S_l \cup S_l \bullet$	$S_{c_l} := S \setminus S_{d_l}$
arrive	0	00	$\mathsf{b}_1, \mathsf{b}_2$	$\mathsf{b}_3, \mathsf{b}_4$
dequeue	1	01	$\mathsf{b}_1, \mathsf{b}_2, \mathsf{b}_3, \mathsf{b}_4$	$\emptyset$
service	2	10	$\mathsf{b}_3, \mathsf{b}_4$	$\mathsf{b}_1, \mathsf{b}_2$

Fig. 2. State graph, binary encodings, corresponding MTBDD and activity-dependent sets of SVs

Hereby $\mathsf{M}_l := 0$ before the first insertion. The vectors $\vec{s}_{d_l}$ and $\vec{s}_{d_l}{}'$ encode the values of the dependent SVs before and after the execution of l. Parameter λ is the rate between these two states, which will be stored in a terminal vertex of M_l. $\mathcal{M}$ is the minterm function which constructs the conjunction of n literals given as first argument (e.g. $\vec{b}$) according to the value given as second argument (e.g. $\mathcal{E}(\vec{s}_{d_l})$).

3) Merging of activity-local MTBDDs: After the activity-local MTBDDs are generated, the overall state graph can be generated by merging the activity-local MTBDDs. Before the actual merging, the symbolic encoding of the set S_{c_l} must be inserted into each MTBDD M_l. For the variables from S_{c_l} the condition $\vec{b} = \vec{b}'$ must hold, because under activity l the SVs $s_i \in S_{c_l}$ do not change their value, they stay stable. This condition is achieved by the BDD $Stab_l$ defined as follows:

$$f_{Stab_l}(\vec{b}, \vec{b}') := \bigwedge_{\forall \mathsf{b}_i \in S_{c_l}} (\mathsf{b}_i = \mathsf{b}'_i)$$

It is interesting to note that the interleaved ordering of the variables minimises the number of vertices of $Stab_l$. In order to enable the calculation of impulse rewards, we need to insert the encoding of the activity labels (represented by BDD $\mathcal{M}(\vec{\mathsf{a}}, \mathcal{E}(\mathcal{I}(l)))$ into M_l as well. Now one can sum the K_{Act} activity-local MTBDDs, where the whole process of completion and merging is given by:

$$\mathsf{M} := \sum_{l \in \mathcal{A}ct} \mathcal{M}(\vec{\mathsf{a}}, \mathcal{E}(\mathcal{I}(l))) \cdot \widetilde{\mathsf{M}}_l = \sum_{l \in \mathcal{A}ct} \mathcal{M}(\vec{\mathsf{a}}, \mathcal{E}(\mathcal{I}(l)) \cdot \mathsf{M}_l \cdot Stab_l,$$

The MTBDD M thus constructed encodes the potential set of transitions of the overall model.

IV. Partial state space exploration

In the following, we consider the condition for entering a state into the state buffer, in order to do as few exploration steps as necessary. Following a standard breadth-first search strategy, one may enter a target state $\vec{s}\,'$ (resulting from the execution of activity l) into the state buffer, if for all activities k the activity-dependent marking $\vec{s}_{d_k}{}' \subseteq \vec{s}\,'$ is not contained (as source state) in M_k. If one only checked M_l for the containment of $\vec{s}_{d_l}{}'$, the algorithm might stop too soon and states might be omitted. On the other hand, in many cases M_k will not contain $\vec{s}_{d_k}{}'$, since k may not be enabled in this marking. As a consequence, a state would be always entered into the state buffer, thus the algorithm would never stop. Therefore we need to supplement each state by a list of activities which could possibly be executed by the exploration part. The criterion for an activity k to become a member of such a list is given by $\vec{s_{d_k}}{}' \notin \mathsf{M}_k$. However, since we follow a breadth-first search strategy, one needs to check only those activities which might be newly enabled or re-enabled after the execution of activity l. Therefore we define two activities l and k as *dependent* if they share at least one SV, i.e. $S_{d_l} \cap S_{d_k} \neq \emptyset$, otherwise the activities are considered as *independent*. The set of dependent activities for activity l is thus given by:

$$T_{d_l} := \{k \in \mathcal{A}ct \mid S_{d_l} \cap S_{d_k} \neq \emptyset\}$$

Note that according to this definition we have $l \in T_{d_l}$, since it is possible that l is enabled again in $\vec{s}\,'$. Thus T_{d_l} contains at least activity l. For each activity $k \in T_{d_l}$ one needs to check whether or not MTBDD M_k contains the encoding of the activity-dependent marking $\vec{s}_{d_k}{}'$ given by $\vec{s}\,'$. In case it does not, activity k is considered as being potentially enabled,

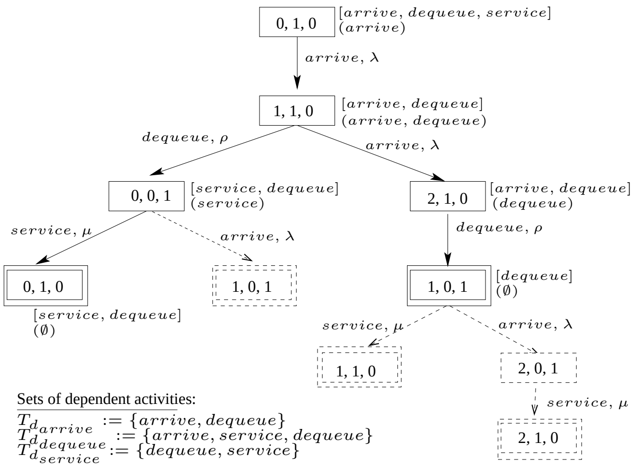


Fig. 3. State Tree under a partial exploration scheme

which gives us the set of potentially enabled activities for each state $\vec{s}\,'$ reached through activity l:

$$E_{\vec{s}\,',l} := \{k \in T_{d_l} \mid \vec{s}_{d_k}\,' \notin \mathsf{M}_k\}$$

In the initial state all activities are candidates for being enabled, thus $E_{\vec{s}_1,\epsilon} = \mathcal{A}ct$, where $\vec{s}_1$ is the initial state. Fig. 3 shows the state tree, where each activity-dependent set T_{d_l} is given in $\{\ldots\}$ and each set of potentially enabled activities $E_{\vec{s}\,',l}$ is given in $[\ldots]$. The sets of activities given in $(\ldots)$ contain those activities from $E_{\vec{s}\,',l}$ whose enabling condition is evaluated to true by the exploration part. As illustrated in the figure, the scheme introduced above reduces the number of transitions explicitly established, e.g. activity $service$ is only executed once. It is interesting to note that some reachable states (here $(2, 0, 1)$) will not be visited at all during this phase, since the states in the dashed boxes would be only (re-) visited in case a conventional state space layout and exploration routine were employed. In contrast, the algorithm described here will stop at the states framed by double boxes. Our algorithm can be sketched as follows:

```
(0) EvaluateTransition($TransBuffer$)
(1)     read($TransBuffer, \vec{s}, l, \lambda, \vec{s}\,'$)
(2)     $E_{\vec{s}\,',l} := \emptyset$
(3)     **for each** $k \in T_{d_l}$ **do**
(4)         **if** $\vec{s}_{d_k}\,' \notin \mathsf{M}_k$ **then** $E_{\vec{s}\,',l} := E_{\vec{s}\,',l} \cup \{k\}$ **fi**
(5)     **od**
(6)     **if** $E_{\vec{s}\,',l} \neq \emptyset$ **then** insert($StateBuffer, \vec{s}\,', E_{\vec{s}\,',l}$) **fi**
(7)     SymbolicEnc($\mathsf{M}_l, \vec{s}_{d_l}, l, \lambda, \vec{s}_{d_l}\,'$)
(8) return
```

One may note that the insertion of a transition into the respective M_l (line (7)) is performed after the generation of $E_{\vec{s}\,',l}$, in order to make l a member of the latter. However doing so induces an overhead in case of loops, since they are detected at their second generation, where $E_{\vec{s}\,',l} := \emptyset$. On the other hand, one is enabled now, to check target and source states encoded in M_k for the existence of $\vec{s_{d_l}}\,'$. Under such a procedure, the complementatory exploration part works then as follows:

```
(0) ExploreState($StateBuffer$)
(1)     read($StateBuffer, \vec{s}, E_{\vec{s},l}$)
(2)     **for each** $k \in E_{\vec{s},l}$ **do**
(3)         **if** $k$ enabled in $\vec{s}$ **then**
(4)             $(\vec{s}\,', \lambda)$ := succ($\vec{s}, k$)
(5)             insert($TransBuffer, \vec{s}, k, \lambda, \vec{s}\,'$) **fi**
(6)     **od**
(7) return
```

where succ$(\vec{s}, k)$ returns the successor state $\vec{s}\,'$ in case activity k is executed in $\vec{s}$, as well as the respective rate λ (line(4)).

V. Summary and Future work

In this short paper we have shown how symbolic state graph representations can be constructed in the context of monolithic models. We saw that only parts of the state graph need to be generated, yielding advantages concerning run-time behaviour. The complete state graph of the overall model is constructed by merging the activity-local state graphs and subsequent symbolic state space exploration. So far, we have not considered the problem that the bounds K_i for the state variables may not be known a priori. However, we plan to employ Z-BDDs [4] and their extension to the multi-terminal case, yielding Z-MTBDDs. The reduction rules for this type of decision diagram enable an efficient handling of this problem. Furthermore the use of Z-MTBDDs reduces the memory requirements for representing f_{Stab} [3].

The performance evaluation tool Möbius [2] is capable of exploiting symmetries specified within a model, generating a reduced state space by applying the lumpability theorem on-the-fly [5]. We plan to support this feature during the symbolic state space generation as well. Furthermore, we also plan to develop an efficient scheme for handling reward variables in the symbolic context, especially in combination with reduced overall models.

References

[1] I. Davies, W.J. Knottenbelt, and P.S. Kritzinger. Symbolic Methods for the State Space Exploration of GSPN Models. In *Proc. of the 12th Int. Conf. on Modelling Techniques and Tools (TOOLS 2002)*, LNCS 2324, pages 188 – 199. Springer, April 2002.

[2] Daniel D. Deavours, Graham Clark, Tod Courtney, David Daly, Salem Derisavi, Jay M. Doyle, William H. Sanders, and Patrick G. Webster. The Moebius Framework and Its Implementation. *IEEE Transactions on Software Engineering (TSE)*, 28(10):956–969, October 2002.

[3] K. Lampka. Z-BDD-based State Graph Representation for Monolithic Model Descriptions. Technical Report 02/03, Universität Erlangen-Nürnberg, Institut für Informatik 7, 2003.

[4] S. Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In *Proc. of the 30th Design Automation Conference*, pages 272–277, Dallas (Texas), USA, June 1993. ACM Press.

[5] W.H. Sanders and J.F. Meyer. Reduced Base Model Construction Methods for Stochastic Activity Networks. *IEEE Journal on Selected Areas in Communications*, 9(1):25–36, January 1991.

[6] M. Siegle. *Behaviour analysis of communication systems: Compositional modelling, compact representation and analysis of performability properties.* Berichte aus der Informatik. Shaker Verlag, Aachen, Germany, 2002. Habilitation Thesis Friederich-Alexander-University Erlangen-Nürnberg.