On the use of MTBDDs for performability analysis and verification of stochastic systems★

Holger Hermanns[a] Marta Kwiatkowska[b] Gethin Norman[b]
David Parker[b] Markus Siegle[c]

[a]*Formal Methods and Tools Group, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands*

[b]*University of Birmingham, Birmingham B15 2TT, United Kingdom*

[c]*Lehrstuhl für Informatik 7, University of Erlangen-Nürnberg, Martensstraße 3, 91058 Erlangen, Germany*

Abstract

This paper describes how to employ Multi-Terminal Binary Decision Diagrams (MTBDDs) for the construction and analysis of a general class of models that exhibit stochastic, probabilistic and non-deterministic behaviour. It is shown how the notorious problem of state space explosion can be circumvented by compositionally constructing symbolic (i.e. MTBDD-based) representations of complex systems from small-scale components. We emphasise, however, that compactness of the representation can only be achieved if heuristics are applied with insight into the structure of the system under investigation. We report on our experiences concerning compact representation, performance analysis and verification of performability properties.

Key words: Binary decision diagram, Markov chain, Markov decision process, performability analysis, model checking.

1 Introduction and Motivation

Systems from many different areas of application, such as distributed algorithms, client-server computing or mobile computing, behave in a way that is not fully predictable and can therefore be viewed and modelled as being

★ Supported in part by DAAD/British Council ARC project StochVer, by DFG/NWO project VOSS, and by EPSRC grant GR/M04617 and MathFIT studentship for David Parker.

stochastic. The system's behaviour may either be truly stochastic, as is the case, for example, with randomised algorithms, where decisions are made on the basis of the outcome of random experiments, or it may only appear to be stochastic to an external observer, because the observable behaviour depends on many hidden parameters and conditions which cannot easily be described or understood in full detail.

Such systems can be described and analysed with the help of stochastic models, which include features such as probabilistic decisions and delays, or durations whose length is governed by a random variable. In addition to decisions with probabilistically determined outcome, it is often helpful or even mandatory to include non-deterministic decisions whose outcome is left completely unspecified, in cases where it is unrealistic or impossible to associate the possible outcomes with concrete probabilities. This leads to models such as Markov decision processes [1], also called concurrent Markov chains [2,3]. During the analysis of non-deterministic models, special techniques must be applied in order to derive maximal or minimal probabilities with which certain requirements are satisfied, conditioned on the strategy by which non-deterministic decisions are made. It is also possible to transform a non-deterministic model into a purely probabilistic model, by replacing non-deterministic decisions by probabilistic ones, and afterwards apply standard analysis techniques.

Since computer and communication systems are getting more and more complex, it is impossible for humans to describe them without the help of some high-level formalism which supports the construction of complex models from small-scale components. Process algebras [4,5] are excellent candidates for this procedure. They are equipped with operators for the parallel composition of components which can be specified in isolation, abstraction mechanisms for hiding a component's internal behaviour from the environment, and powerful notions of equivalence. Even though we do not explicitly introduce a process-algebraic specification language for stochastic models (although such languages exist, see for instance [6–8]), we do describe the compositional specification of models with the help of parallel composition and abstraction operators borrowed from the domain of process algebras. These operators are the key to the construction of complex models via composition. When constructing large and complex models, however, one is inevitably faced with the problem of state space explosion: the number of model states may easily grow to such an extent that it becomes impractical to generate, store and analyse the overall model. Among the techniques that have been developed in order to combat this notorious problem are decomposition [9], structured representation based on Kronecker expressions [10,11], and compositional state space reduction based on bisimulation equivalences and symmetry exploitation [12,13].

In this paper, we present an approach to the memory-efficient representation of very large stochastic models with the help of "symbolic" encodings in the form of Multi-Terminal Binary Decision Diagrams (MTBDDs). MTBDDs are an extension of Binary Decision Diagrams (BDD) [14], a graph-based representation of Boolean functions that is canonical and completely avoids the storing of redundant information. The sweeping success of BDDs in the fields of digital circuit verification and model checking led to the development of MTBDDs, to be applied to functions with numerical range [15,16]. We employ MTBDDs not only to represent a very general class of stochastic and probabilistic transition systems in a compact way, but also to construct large models compositionally from small components and to perform various types of analysis, including model checking and numerical analysis on them. The aim of this analysis is often to determine the performance measures of the system under investigation, which requires the computation of the state probabilities. A further aim may be to check whether the system satisfies requirements formulated with the help of a temporal logic, concerning the probability that a certain behaviour occurs within a given time. Such model checking of non-functional properties [17,18] is now considered a powerful technique that allows developers to verify complex behavioural properties which could not be tackled using traditional performance evaluation techniques.

The contribution of this paper is threefold: (i) We propose a uniform framework for representing stochastic models, and discuss analysis algorithms for both performance analysis and model checking of such models. (ii) We show how such models can be encoded effectively with the help of MTBDDs, and describe analysis and model checking algorithms based on this data structure. (iii) We report on an empirical evaluation of two software tools (which are independent, but based on the same MTBDD package), both realising the basic principles of this symbolic approach. Our experience with implementing the approach in two separate tools allows us to draw interesting conclusions that we believe are important as heuristics in the context of MTBDD-based analysis as a whole.

This paper is structured as follows: Sec. 2 introduces the general model in the form of transition systems which exhibit stochastic, probabilistic and non-deterministic behaviour. In Sec. 3, we describe mechanisms for the compositional construction of complex models from small components. We do this for the general model and for two important subclasses thereof, namely concurrent probabilistic systems (CPS) and Markov transition systems (MTS). In Sec. 3.2 and Sec. 3.3 the basic analysis algorithms for these subclasses are reviewed, and the logic PCTL is introduced as a formalism for specifying requirements of CPS models. Sec. 4 introduces the MTBDD data structure, together with the basic operations needed for the construction, manipulation and analysis of MTBDDs. We emphasise the role of MTBDDs for the representation of matrices and discuss how matrix manipulations are realised on this data structure. Sec. 5 shows how general transition systems are represented symbolically with the help of MTBDDs and explains how composition and abstraction can be realised on symbolic representations. We provide the theoretical background to show that symbolic parallel composition is the key to space-efficient representation of huge state spaces. Sec. 6 discusses MTBDD-based algorithms for model checking the logic PCTL against CPS models and for calculating stationary probabilities for MTS models. In Sec. 7 we describe our MTBDD-based software tools and present some experimental results. Our experience, especially concerning heuristics for achieving compact symbolic representations of huge transition systems, is summarised in Sec. 8, and Sec. 9 concludes the paper.

2 The general model

This section introduces *stochastic and probabilistic transition systems*, the model considered in the remainder of this paper, and isolates five specific instances thereof, namely ordinary labelled transition systems, discrete time Markov chains, continuous time Markov chains, concurrent probabilistic systems, and Markov transition systems.

Before defining the general model, we first introduce some notational conventions. We denote the set of Boolean values by $\mathbb{B} = \{0, 1\}$, the set of naturals by $\mathbb{N} = \{0, 1, 2, \ldots\}$, by $\mathbb{R}$ the set of reals, and by $\mathbb{R}^{>0}$ the set of positive reals. For a finite set S of states we use s, s', s_0, t, u, ... to range over states. We assume that states are labelled with atomic propositions, drawn from a set AP. Depending on the application domain, these atomic propositions denote for instance, an unsafe state, a failed component or an empty buffer. We also assume a (finite) set Act of actions, which contains two distinguished "internal" actions τ_i and τ_m. The set Act consists of two subsets, the set of "immediate" actions $Act_i \ni \tau_i$ and the set of "timed" or "Markovian" actions $Act_m \ni \tau_m$. These sets are disjoint, i.e. $Act_i \cap Act_m = \emptyset$. We use a, b, b', ... to refer to the elements of Act, including τ_i and τ_m.

For a given finite set S we use π, π', π_0, ... to range over probability distributions on S, i.e. functions $\pi : S \mapsto [0, 1]$ such that $\sum_{s \in S} \pi(s) = 1$. We refer to such functions as "discrete" distributions, and by $Dist(S)$ we denote the set of discrete distributions over S. A discrete distribution π is called "degenerate", if $\pi(s) = 1$ for some $s \in S$, i.e. π determines a unique state. Furthermore, λ, λ', λ_1, ... are used to range over $\mathbb{R}^{>0}$. The latter will be used to parameterise exponentially distributed random delays as follows. If X is an exponentially distributed random delay with rate parameter λ, then the probability that delay X finishes before time $t \geq 0$ is given by the continuous exponential probability distribution $P(X \leq t) = 1 - e^{-\lambda t}$. We will refer to such distributions as "continuous" distributions, in order to distinguish them from the above discrete distributions.

Definition 1 *A (finite) stochastic and probabilistic transition system (SPTS, for short) is a tuple $\mathcal{S} = (S, \bar{s}, Act, \rightarrow, \mathrm{AP}, L)$, where*

- *S is a finite set of states;*
- *$\bar{s} \in S$ is the unique initial state;*
- *$Act = Act_i \cup Act_m$ (where $Act_i \cap Act_m = \emptyset$) is a finite set of actions;*
- *$\rightarrow\ \subset\ S \times Act \times (S \mapsto \mathbb{R})$ is the transition relation, subject to the restriction that whenever $(s, a, f) \in\ \rightarrow$* either
  - (i) *$a \in Act_i$ and $f \in Dist(S)$, i.e. f defines a discrete probability distribution over S,* or
  - (m) *$a \in Act_m$ and $\exists t \in S$ such that $f(t) \in \mathbb{R}^{>0}$ and $f(t') = 0$ for all $t' \neq t$, i.e. f assigns a positive real value to precisely one state;*
- *AP is a set of atomic propositions;*
- *$L : S \mapsto 2^{\mathrm{AP}}$ is a function labelling each state with a set of atomic propositions.*

Instead of $(s, a, f) \in\ \rightarrow$ we usually write $s \xrightarrow{a} f$. If function f is of type (i) and is degenerate with $f(s') = 1$, we may write $s \xrightarrow{a} s'$. If instead f is a function of type (m), we write $s \overset{a}{\dashrightarrow}_{\lambda} s'$ if $f(s') = \lambda$.

Note that Def. 1 allows two or more transitions of type (m) with the same action label but different rates between the same ordered pair of states. We follow the convention that such "parallel" transitions will be cumulated into a single transition whose rate is the sum of the individual rates. Technically, this can be achieved with the help of a cumulation function (see Def. 5 below). The cumulation is sound, i.e. does not change the behaviour of the SPTS.

The SPTS model in its full generality is rarely used, but it is convenient to represent systems that evolve randomly, concurrently, or non-deterministically as time progresses, where time can pass either in discrete steps or flow continuously. To illustrate this, we list below five special cases that have received considerable attention in the literature – each in its own right. Other important models that can be seen as special cases of SPTS are (discrete and continuous time) Markov decision processes, and Markov reward models.

Ordinary labelled transition systems are the basic interleaving models in concurrency theory and practice. An SPTS $\mathcal{S} = (S, \bar{s}, Act, \rightarrow, \mathrm{AP}, L)$ is a labelled transition system (LTS) if
- for each $(s, a, f) \in\ \rightarrow$, f is of type (i) and degenerate.

An SPTS reduces to an LTS if continuous distributions do not occur, and the discrete probabilistic behaviour is trivial, in that each probability distribution is degenerate.

Discrete time Markov chains are widespread models for representing synchronous probabilistic systems. An SPTS $\mathcal{S} = (S, \bar{s}, Act, \rightarrow, \mathrm{AP}, L)$ is a discrete time Markov chain (DTMC) if

- Act is a singleton set, say $\{a\}$;
- for each $(s, a, f) \in\ \rightarrow$, f is of type (i);
- $(s, a, f) \in\ \rightarrow$ and $(s, a, f') \in\ \rightarrow$ imply $f = f'$.

In DTMCs there is no action labelling, and hence all transitions are labelled with the same action. In order to exclude non-determinism, there is at most one transition emanating from each state, describing a discrete probability distribution on successor states.

Continuous time Markov chains An SPTS $\mathcal{S} = (S, \bar{s}, Act, \rightarrow, \mathrm{AP}, L)$ is a continuous time Markov chain (CTMC) if

- Act is a singleton set, say $\{a\}$;
- for each $(s, a, f) \in\ \rightarrow$, f is of type (m).

All transitions are labelled with the same action, since in CTMCs the action labelling is irrelevant. In contrast to DTMCs, there can be more than one transition emanating from each state, each of them describing a continuous distribution. Such a case (as in $s'\ {}_{\lambda'}\!\overset{a}{\dashleftarrow} s \overset{a}{\dashrightarrow}_{\lambda''} s''$) does not represent a non-deterministic situation; instead, a "race" is assumed between concurrently enabled continuous distributions. This race implicitly determines probabilities for the successor states (namely $\lambda'/(\lambda' + \lambda'')$ for s' and $\lambda''/(\lambda' + \lambda'')$ for s''). As a consequence, the delay until the race finishes follows a continuous distribution with a parameter cumulated from the contributing continuous distributions ($\lambda' + \lambda''$ in this case).

Concurrent probabilistic systems are known to be conveniently captured in the "simple" probabilistic automata model of Segala [3], the sub-model of SPTS considered here. An SPTS $\mathcal{S} = (S, \bar{s}, Act, \rightarrow, \mathrm{AP}, L)$ is a concurrent probabilistic system (CPS) if

- for each $(s, a, f) \in\ \rightarrow$, f is of type (i).

In this case, only discrete, but not continuous, distributions are allowed to occur. As opposed to DTMCs, non-determinism is included in this model, i.e. it is possible to have $(s, a, f) \in\ \rightarrow$ and $(s, b, f') \in\ \rightarrow$ with $a \neq b$, or with $a = b$ and $f \neq f'$.

Markov transition systems are used as the semantics of, among others, the process algebras TIPP [19], IMC [20], and PEPA [7] (where IMC supposes Act_m to be a singleton and PEPA supposes Act_i is empty) [1]. An SPTS $\mathcal{S} = (S, \bar{s}, Act, \rightarrow, \mathrm{AP}, L)$ is a Markov transition system (MTS) if

- for each $(s, a, f) \in\ \rightarrow$, if f is of type (i) it is degenerate.

MTSs form a superset of LTSs, where continuous distributions may occur, but the discrete probabilistic behaviour is trivial.

Example 2 *Since CPSs and MTSs will play a major role in the remainder*

[1] The EMPA language family [21,22] is not directly supported by SPTSs because these languages involve the generative probabilistic model, where probabilities are assigned to actions.

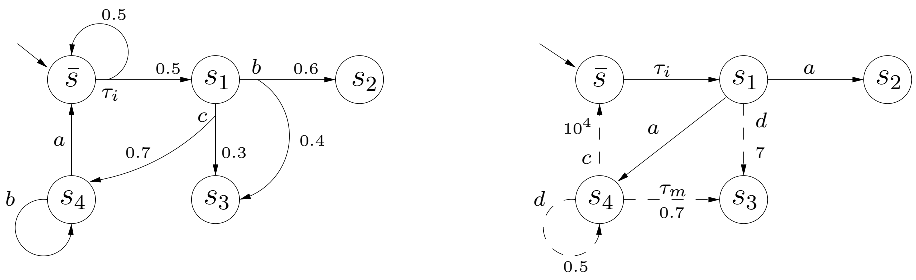


Fig. 1. Concurrent probabilistic system (left) and Markov transition system (right).

of this paper, we introduce some visualisation conventions for them. Fig.1 depicts an example of each of these models. On the left there is a CPS $(S, \bar{s}, Act, \rightarrow, \mathrm{AP}, L)$ *with*

- $S = \{\bar{s}, s_1, s_2, s_3, s_4\}$*;*
- $\bar{s}\xrightarrow{\tau_i}\pi$, $s_1\xrightarrow{b}\pi'$, $s_1\xrightarrow{b}\pi''$, $s_4\xrightarrow{b}s_4$, $s_4\xrightarrow{a}\bar{s}$ *where* $\pi(\bar{s}) = \pi(s_1) = 0.5$, $\pi'(s_3) = 0.3$, $\pi'(s_4) = 0.7$, $\pi''(s_2) = 0.6$, $\pi''(s_3) = 0.4$.

On the right, we have depicted an MTS $(S, \bar{s}, Act, \rightarrow, \mathrm{AP}, L)$ *with* S *as before and* $\rightarrow$ *is given by*

- $\bar{s}\xrightarrow{\tau_i}s_1$, $s_1\xrightarrow{a}s_2$, $s_1\xrightarrow{a}s_4$, $s_1 \overset{d}{\dashrightarrow}_7 s_3$, $s_4 \overset{\tau_m}{\dashrightarrow}_{0.7} s_3$, $s_4 \overset{d}{\dashrightarrow}_{0.5} s_4$, $s_4 \overset{c}{\dashrightarrow}_{10^4} \bar{s}$.

Note that probability 1 values are omitted, and the initial states are visualised by a dangling incoming edge. Note further that for simplicity the labelling of states with atomic propositions is not specified and not shown in the figure. Note also that Markovian loops like $s_4 \overset{d}{\dashrightarrow}_{0.5} s_4$ *are possible.*

3 Model specification and analysis

In this section we consider compositional methods for specifying and analysing SPTSs, in particular focusing on such methods for CPSs and MTSs.

3.1 Compositional model specification

Although the SPTS model suffices, at least in principle, for the description of probabilistically and stochastically evolving systems, it is too limited to be of great practical value when specifying such systems. It will often be necessary to define systems as the parallel composition of a number of subprocesses. For this purpose, we introduce a binary parallel composition operator for SPTSs which is borrowed from process algebra [23,5]. If $\mathcal{S}_1$ and $\mathcal{S}_2$ are two SPTSs, the parallel composition of $\mathcal{S}_1$ and $\mathcal{S}_2$ is denoted by the SPTS $\mathcal{S}_1 \,|[A]|\, \mathcal{S}_2$. This operator is indexed with a set A of actions on which its component SPTSs have to synchronise. All other actions, i.e. those that are not in the index set of the composition operator, can be performed independently of the other component process.

Another important feature in this context is a mechanism to abstract from internal aspects that are irrelevant at higher design levels. In process algebra, the concept of abstraction is realised by the abstraction operator. The key to this operator is the distinguished internal action τ that symbolises an internal or hidden action, e.g. a state change that cannot depend on synchronisation with the environment. If $\mathcal{S}$ is an SPTS and $B \subseteq Act$, we use **hide** B **in** $\mathcal{S}$ to denote the operation of hiding all the actions in B. We use two internal actions τ_i and τ_m, one for type (i) and one for type (m) transitions. This allows us to clearly distinguish between immediate and Markovian transitions. It is natural to assume that internal actions cannot be synchronised on, hence we require that $\tau_i, \tau_m \notin A$, where A is the above synchronisation set for parallel composition.

In the sequel, we will first define these operators for CPSs and MTSs, using the standard operational rule schemes in the style of Plotkin [24], and then generalise the definition to the SPTS setting.

3.1.1 Concurrency and Abstraction in CPS models

In this section we define the two operators introduced above for composing CPS models.

Definition 3 *Let* $\mathcal{S}_1 = (S_1, \bar{s}_1, Act_1, \rightarrow_1, \mathrm{AP}_1, L_1)$ *and* $\mathcal{S}_2 = (S_2, \bar{s}_2, Act_2, \rightarrow_2, \mathrm{AP}_2, L_2)$ *be two CPSs and* $A \subseteq (Act_1 \cup Act_2) \setminus \{\tau_i\}$. *Then* $\mathcal{S} = \mathcal{S}_1 \,|[A]|\, \mathcal{S}_2$ *denotes a CPS* $\mathcal{S} = (S_1 \times S_2, (\bar{s}_1, \bar{s}_2), Act_1 \cup Act_2, \rightarrow, \mathrm{AP}_1 \cup \mathrm{AP}_2, L)$ *with*

- $L((s_1, s_2)) = L_1(s_1) \cup L_2(s_2)$ *for all* $(s_1, s_2) \in S_1 \times S_2$;
- $\rightarrow$ *is the least relation satisfying:*

$$\frac{s_1 \xrightarrow{a}_1 \pi_1 \quad s_2 \xrightarrow{a}_2 \pi_2}{(s_1, s_2) \xrightarrow{a} (\pi_1 \otimes \pi_2)}\ (a \in A)$$

$$\frac{s_1 \xrightarrow{a}_1 \pi_1}{(s_1, s_2) \xrightarrow{a} (\pi_1 \parallel s_2)}\ (a \notin A) \qquad \frac{s_2 \xrightarrow{a}_2 \pi_2}{(s_1, s_2) \xrightarrow{a} (s_1 \parallel \pi_2)}\ (a \notin A)$$

where $(\pi_1 \otimes \pi_2)(t_1, t_2) = \pi_1(t_1) \cdot \pi_2(t_2)$ *and* $(\pi \parallel s')(t, t') = (s' \parallel \pi)(t', t) = \pi(t)$ *if* $t' = s'$ *and 0 otherwise.*

The above transition rules allow one to derive the transitions of the combined CPS from the transitions of the two participating components. The first rule, for instance is to be read as follows: if there is an a-transition $s_1 \xrightarrow{a}_1 \pi_1$ (in the first transition system) and an a-transition $s_2 \xrightarrow{a}_2 \pi_2$ (in the second transition system), then there will be an a-transition $(s_1, s_2) \xrightarrow{a} (\pi_1 \otimes \pi_2)$ in the combined system. The distribution $\pi_1 \otimes \pi_2$ is the Kronecker product of the operand distributions.

In the remaining rules (the rules for interleaving), the condition $t' = s'$ expresses idling of the non-moving partner, i.e. the fact that its source state equals its target state.

Note that in Def. 3 the set S is defined as the product space $S_1 \times S_2$. In general, it is often the case that $S \subset S_1 \times S_2$, since some states in $S_1 \times S_2$ are unreachable due to synchronisation constraints. Reachability analysis is needed in order to determine the reachable subset of $S_1 \times S_2$. Our later definition for MTBDD-based parallel composition (cf. Th. 16) also assumes this denotational view.

Definition 4 *Let $A \subseteq Act$ and $\mathcal{S}_1 = (S_1, \bar{s}_1, Act_1, \rightarrow_1, \mathrm{AP}_1, L_1)$ be a CPS. Then $\mathcal{S} =$ **hide** A **in** $\mathcal{S}_1$ denotes a CPS $\mathcal{S} = (S_1, \bar{s}_1, Act_1, \rightarrow, \mathrm{AP}_1, L_1)$ with $\rightarrow$ given as the least relation satisfying:*

$$\frac{s \xrightarrow{a}_1 \pi}{s \xrightarrow{a} \pi}\ (a \notin A) \qquad\qquad \frac{s \xrightarrow{a}_1 \pi}{s \xrightarrow{\tau_i} \pi}\ (a \in A).$$

3.1.2 Concurrency and Abstraction in MTS models

This section introduces parallel composition and abstraction on MTSs. For technical reasons, we need a function that flattens a multiset into a set, but cumulates parameters contained in this set. We need such a function for cumulating the rates of multisets of Markovian transitions which all have the same source state and the same target state. Since transitions are elements of $S \times Act \times (S \mapsto \mathbb{R})$, a multiset of transitions can be denoted as a multi-relation $(S \times Act \times (S \mapsto \mathbb{R})) \mapsto \mathbb{N}$.

Definition 5 *Let $R \subset \mathbb{R}$ be a finite set of real numbers and $M_R : R \mapsto \mathbb{N}$ denote a multiset over R. Then $C(M_R) = \sum_{\lambda \in R} \lambda \cdot M_R(\lambda)$ is called a cumulation function for M_R. We lift this cumulation function to multi-relations of the form $M_{\rightarrow} : (S \times Act \times (S \mapsto \mathbb{R})) \mapsto \mathbb{N}$ by defining the cumulated transition relation $C(M_{\rightarrow}) \subset\ S \times Act \times (S \mapsto \mathbb{R})$ as follows: $(s, a, f) \in C(M_{\rightarrow})$ if and only if one of the following conditions hold*

- *$a \in Act_i$ and $M_{\rightarrow}(s, a, f) > 0$;*
- *$a \in Act_m$ and $\exists t \in S$ such that $f(t) = \sum_{f'} f'(t) \cdot M_{\rightarrow}(s, a, f')$ and $f(t') = 0$*

for all $t' \neq t$.

Note that type (i) transitions are not cumulated (regardless of their multiplicity), and, in the case of type (m) transitions, the rates of all a-transitions leading from the same source state to the same target state are cumulated.

Definition 6 *Let $\mathcal{S}_1 = (S_1, \bar{s}_1, Act_1, \rightarrow_1, \mathrm{AP}_1, L_1)$ and $\mathcal{S}_2 = (S_2, \bar{s}_2, Act_2, \rightarrow_2, \mathrm{AP}_2, L_2)$ be two MTSs and $A \subseteq (Act_1 \cup Act_2) \setminus \{\tau_i, \tau_m\}$. Then $\mathcal{S} = \mathcal{S}_1 \mathbin{|[A]|} \mathcal{S}_2$ denotes an MTS $\mathcal{S} = (S_1 \times S_2, (\bar{s}_1, \bar{s}_2), Act_1 \cup Act_2, C(\rightarrow), \mathrm{AP}_1 \cup \mathrm{AP}_2, L)$ with*

- *$L((s_1, s_2)) = L_1(s_1) \cup L_2(s_2)$ for all $(s_1, s_2) \in S_1 \times S_2$;*
- *$\rightarrow$ is the least multi-relation satisfying:*

$$\frac{s \xrightarrow{a}_1 t \quad s' \xrightarrow{a}_2 t'}{(s,s') \xrightarrow{a} (t,t')}\ (a \in A) \qquad \frac{s \overset{a}{\dashrightarrow}_{\mu,1} t \quad s' \overset{a}{\dashrightarrow}_{\nu,2} t'}{(s,s') \overset{a}{\dashrightarrow}_{\phi(\mu,\nu)} (t,t')}\ (a \in A)$$

$$\frac{s \xrightarrow{a}_1 t}{(s,s') \xrightarrow{a} (t,s')}\ (a \notin A) \qquad \frac{s \xrightarrow{a}_2 t}{(s',s) \xrightarrow{a} (s',t)}\ (a \notin A)$$

$$\frac{s \overset{a}{\dashrightarrow}_{\lambda,1} t}{(s,s') \overset{a}{\dashrightarrow}_{\lambda} (t,s')}\ (a \notin A) \qquad \frac{s \overset{a}{\dashrightarrow}_{\lambda,2} t}{(s',s) \overset{a}{\dashrightarrow}_{\lambda} (s',t)}\ (a \notin A).$$

As in [8], the construction for synchronisation of Markovian transitions is parametric in a function ϕ determining the rate of synchronisation, in response to the fact that different synchronisation policies are possible [21,19,20,7].

Definition 7 *Let $A \subseteq Act$ and $\mathcal{S}_1 = (S_1, \bar{s}_1, Act_1, \rightarrow_1, \mathrm{AP}_1, L_1)$ be an MTS. Then $\mathcal{S} =$ **hide** A **in** $\mathcal{S}_1$ denotes an MTS $\mathcal{S} = (S_1, \bar{s}_1, Act_1, C(\rightarrow), \mathrm{AP}_1, L_1)$ with $\rightarrow$ given as the least multi-relation satisfying:*

$$\frac{s \xrightarrow{a}_1 t}{s \xrightarrow{a} t}\ (a \notin A) \qquad \frac{s \overset{a}{\dashrightarrow}_{\lambda,1} t}{s \overset{a}{\dashrightarrow}_{\lambda} t}\ (a \notin A)$$

$$\frac{s \xrightarrow{a}_1 t}{s \xrightarrow{\tau_i} t}\ (a \in A) \qquad \frac{s \overset{a}{\dashrightarrow}_{\lambda,1} t}{s \overset{\tau_m}{\dashrightarrow}_{\lambda} t}\ (a \in A).$$

3.1.3 Concurrency and Abstraction on SPTS models

We are now in a position to define what it means to compose two SPTSs in parallel, by superposing the definitions of the previous sections.

Definition 8 *Let $\mathcal{S}_1 = (S_1, \bar{s}_1, Act_1, \rightarrow_1, \mathrm{AP}_1, L_1)$ and $\mathcal{S}_2 = (S_2, \bar{s}_2, Act_2, \rightarrow_2, \mathrm{AP}_2, L_2)$ be two SPTSs, and $A \subseteq (Act_1 \cup Act_2) \setminus \{\tau_i, \tau_m\}$. Then $\mathcal{S}_1 \mathbin{|[A]|} \mathcal{S}_2$ denotes the SPTS $(S_1 \times S_2, (\bar{s}_1, \bar{s}_2), Act_1 \cup Act_2, C(\rightarrow), \mathrm{AP}_1 \cup \mathrm{AP}_2, L)$ where*

- *$L((s_1, s_2)) = L_1(s_1) \cup L_2(s_2)$ for all $(s_1, s_2) \in S_1 \times S_2$;*
- *$\rightarrow$ is the least multi-relation satisfying:*

$$\frac{s \xrightarrow{a}_1 f \quad s' \xrightarrow{a}_2 f'}{(s,s') \xrightarrow{a} (f \otimes f')} \; (a \in A \cap Act_i) \qquad \frac{s \xrightarrow{a}_1 f \quad s' \xrightarrow{a}_2 f'}{(s,s') \xrightarrow{a} \phi(f,f')} \; (a \in A \cap Act_m)$$

$$\frac{s \xrightarrow{a}_1 f}{(s,s') \xrightarrow{a} (f \parallel s')} \; (a \notin A) \qquad \frac{s \xrightarrow{a}_2 f}{(s',s) \xrightarrow{a} (s' \parallel f)} \; (a \notin A)$$

where $(f \parallel s')(t,t') = (s' \parallel f)(t',t) = f(t)$ if $t' = s'$ and 0 otherwise.

As for MTSs, the synchronisation rate of Markovian transitions is parametric in a function ϕ; on the other hand, the synchronisation probability for immediate transitions, as in CPSs, uses the Kronecker product and is the multiplication of the probabilities.

Definition 9 *Let $A_1 \subseteq Act$ and $\mathcal{S}_1 = (S_1, \bar{s}_1, Act_1, \rightarrow_1, \mathrm{AP}_1, L_1)$ be an SPTS. Then* **hide** A **in** $\mathcal{S}_1$ *denotes an SPTS $(S_1, \bar{s}_1, Act_1, C(\rightarrow), \mathrm{AP}_1, L_1)$ with $\rightarrow$ given as the least relation satisfying:*

$$\frac{s \xrightarrow{a}_1 f}{s \xrightarrow{a} f} \; (a \notin A) \qquad \frac{s \xrightarrow{a}_1 f}{s \xrightarrow{\tau_i} f} \; (a \in A \cap Act_i) \qquad \frac{s \overset{a}{\dashrightarrow}_{\lambda,1} t}{s \overset{\tau_m}{\dashrightarrow}_{\lambda} t} \; (a \in A \cap Act_m).$$

3.2 Analysing CPS models

This and the next section discuss algorithms and techniques to analyse CPS and MTS models, in order to set the ground for our MTBDD implementations.

3.2.1 The probabilistic temporal logic PCTL

Probabilistic model checking of concurrent probabilistic systems involves calculating the probability of certain temporal properties holding in a given state. In this paper we consider the probabilistic branching-time temporal logic PCTL [25,26].

Before we can introduce the syntax and semantics of the logic PCTL, we need to introduce the definitions of a *path* and an *adversary* of a CPS.

A *path* of a CPS $\mathcal{S} = (S, \bar{s}, Act, \rightarrow, \mathrm{AP}, L)$ is a non-empty finite or infinite sequence $\sigma = s_0 \xrightarrow{a_0}_{\pi_0} s_1 \xrightarrow{a_1}_{\pi_1} s_2 \xrightarrow{a_2}_{\pi_2} \cdots$ where $s_i \in S$, $s_i \xrightarrow{a_i} \pi_i$ with $\pi_i(s_{i+1}) > 0$ for all $0 \leq i < |\sigma|$ and $|\sigma|$ denotes the length of the path σ, defined in the usual way. The first state of a path σ is denoted by *first*(σ); the last state of a finite path σ is denoted by *last*(σ); $\sigma(i)$ denotes the i-th state of σ; *step*(σ, i) is the action-distribution pair selected in the i-th step; and $\sigma^{(i)}$ is the prefix of σ of length i. A path σ is called a *fulpath* if and only if it is infinite. We denote by $Path_{ful}$ the set of fulpaths of $\mathcal{S}$, whereas $Path_{ful}(s)$ is the set of fulpaths σ with $first(\sigma) = s$.

The nondeterminism present in a CPS is resolved using adversaries (or schedulers). Formally, an *adversary* A of a CPS $\mathcal{S} = (S, \bar{s}, Act, \rightarrow, \mathrm{AP}, L)$ is a function mapping every finite path σ of $\mathcal{S}$ to an action-distribution pair enabled in the last state of the path, that is $A(\sigma) = (a, \pi)$ such that $last(\sigma) \xrightarrow{a} \pi$ is a transition in $\mathcal{S}$. We use $\mathbf{A}(\mathcal{S})$ to denote the set of all adversaries of $\mathcal{S}$. For an adversary A we define $Path^A_{ful}$ as the set of paths in $Path_{ful}$ such that $step(\sigma, i) = A(\sigma^{(i)})$ for all $i \in I\!N$. Furthermore, we can associate A with a Markov chain, and hence induce the probability measure $Prob^A$ over $Path^A_{ful}$; for more detail see [26].

Since we allow non-determinism, we may have to impose *fairness constraints* in order to ensure that liveness properties can be verified. In a distributed environment fairness corresponds to a requirement for each concurrent component to progress whenever possible. Without fairness, certain liveness properties may trivially fail to hold in the presence of simultaneously enabled transitions of a concurrent component. We define a path σ of a concurrent probabilistic system to be *fair* if and only if, whenever a state s is visited infinitely often in σ, each non-deterministic alternative which is enabled in s is taken infinitely often in σ. Now an adversary is *fair* if any choice of transitions that becomes enabled infinitely often along a computation path is taken infinitely often[2]. The interested reader is referred to [27,3,26,28] for more information concerning fairness in probabilistic systems.

Based on [26,25], we now recall the syntax and semantics of the probabilistic branching-time temporal logic PCTL. Note that, to simplify this presentation, we have omitted the "bounded until" and "next state" operators which can easily be added.

Definition 10 *The syntax of* PCTL *formulas is defined as follows:*

$$\phi \ ::= \ \texttt{true} \ \mid \ \varphi \ \mid \ \phi_1 \wedge \phi_2 \ \mid \ \neg\phi \ \mid \ \mathcal{P}_{\sim q}[\phi_1 \ \mathcal{U} \ \phi_2]$$

where $\varphi \in \mathrm{AP}$*,* $q \in [0,1]$ *and* $\sim \in \{<, \leq, \geq, >\}$*. Any formula of the form* $\phi_1 \, \mathcal{U} \, \phi_2$ *where* ϕ_1*,* ϕ_2 *are PCTL formulas is defined to be a* path formula*.*

PCTL formulas are interpreted over states of a CPS, whereas path formulas over fulpaths of a CPS.

Definition 11 *Given a CPS* $(S, \bar{s}, Act, \rightarrow, \mathrm{AP}, L)$*, a set Adv of adversaries*

[2] To be precise, the measure of the fair fulpaths is 1.

of the CPS and PCTL *formula ϕ, we define the satisfaction relation $s \models_{Adv} \phi$ inductively as follows:*

$$
\begin{array}{ll}
s \models_{Adv} \texttt{true} & \text{for all } s \in S \\
s \models_{Adv} \varphi & \Leftrightarrow \varphi \in L(s) \\
s \models_{Adv} \phi_1 \wedge \phi_2 & \Leftrightarrow s \models_{Adv} \phi_1 \text{ and } s \models_{Adv} \phi_2 \\
s \models_{Adv} \neg\phi & \Leftrightarrow s \not\models_{Adv} \phi \\
s \models_{Adv} \mathcal{P}_{\sim q}[\phi_1 \,\mathcal{U}\, \phi_2] & \Leftrightarrow \mathit{Prob}^A(\{\sigma \mid \sigma \in \mathit{Path}^A_{\mathit{ful}}(s) \ \wedge\ \sigma \models_{Adv} \phi_1 \,\mathcal{U}\, \phi_2\}) \sim q \\
& \quad\ \text{for all adversaries } A \in \mathit{Adv}
\end{array}
$$

$$
\begin{array}{ll}
\sigma \models_{Adv} \phi_1 \,\mathcal{U}\, \phi_2 & \Leftrightarrow \text{there exists } k \geq 0 \text{ such that } \sigma(k) \models_{Adv} \phi_2 \\
& \quad\ \text{and } \sigma(i) \models_{Adv} \phi_1 \text{ for all } 0 \leq i \leq k.
\end{array}
$$

We denote $\models_{\mathbf{A}(\mathcal{S})}$ by $\models$ (satisfaction for all *adversaries) and $\models_{\mathbf{A}_{fair}(\mathcal{S})}$ by $\models_{fair}$ (satisfaction for* all fair *adversaries).*

The above definition in fact gives rise to an indexed family of satisfaction relations $\models_{Adv}$ [26], of which we only consider two, $\models$ and $\models_{fair}$.

3.2.2 Model checking for PCTL

With the exception of "until" formulas and fairness, model checking for PCTL is straightforward, see [25,26]. It proceeds by induction on the parse tree of the formula, as in the case of CTL model checking [29]. To establish whether $s \in S$ satisfies $\mathcal{P}_{\sqsubseteq q}[\phi \,\mathcal{U}\, \psi]$ where $\sqsubseteq \in \{<, \leq\}$, we calculate the *maximum probability*:

$$p^{\max}_s(\phi \,\mathcal{U}\, \psi) = \sup\{p^A_s(\phi \,\mathcal{U}\, \psi) \mid A \in \mathbf{A}(\mathcal{S})\}$$

where $p^A_s(\phi \,\mathcal{U}\, \psi) = \mathit{Prob}^A(\{\sigma \mid \sigma \in \mathit{Path}^A_{\mathit{ful}}(s) \ \wedge\ \sigma \ \models\ \phi \,\mathcal{U}\, \psi\})$ and compare the result to the threshold q, i.e. establish the inequality $p^{\max}_s \sqsubseteq q$.

The algorithm for finding $p^{\max}_s$ proceeds as follows. First we construct the following sets of states:

- $S^{\text{yes}} := \mathit{Sat}(\psi)$ (the set of all states which trivially satisfy $\phi \,\mathcal{U}\, \psi$ with probability 1);
- $S^{\text{no}} := S \setminus (\mathit{Sat}(\phi) \cup \mathit{Sat}(\psi))$ (states which trivially satisfy $\phi \,\mathcal{U}\, \psi$ with probability zero);
- $S^{?} := S \setminus (S^{\text{yes}} \cup S^{\text{no}})$ (states which satisfy $\phi \,\mathcal{U}\, \psi$ with some, as yet unknown, probability).

Then, we set $p_s^{\max}(\phi\, \mathcal{U}\, \psi) = 1$, if $s \in S^{\mathrm{yes}}$ and $p_s^{\max}(\phi\, \mathcal{U}\, \psi) = 0$ if $s \in S^{\mathrm{no}}$. For $s \in S^{?}$, calculate $p_s^{\max}(\phi\, \mathcal{U}\, \psi)$ iteratively as the limit, as n tends to ∞, of the approximations $\langle x_{s,n}\rangle_{n\in\mathbb{N}}$, where $x_{s,0} = 0$ and for $n = 1, 2, \ldots$

$$x_{s,n} = \max\left\{ \sum_{t\in S^{?}} \pi(t) \cdot x_{t,n-1} + \sum_{t\in S^{\mathrm{yes}}} \pi(t) \mid s \xrightarrow{a} \pi \right\}.$$

Alternatively, the values $p_s^{\max}(\phi\, \mathcal{U}\, \psi)$ can also be computed by solving linear optimization problems [25,2,30].

On the other hand, to establish whether $s \in S$ satisfies $\mathcal{P}_{\sqsupseteq q}[\phi\, \mathcal{U}\, \psi]$ where $\sqsupseteq \in \{\geq, >\}$, we calculate the *minimum probability*:

$$p_s^{\min}(\phi\, \mathcal{U}\, \psi) = \inf\{p_s^{A}(\phi\, \mathcal{U}\, \psi) \mid A \in \mathbf{A}(\mathcal{S})\}$$

and compare the result to the threshold q, i.e. establish the inequality $p_s^{\min} \sqsupseteq q$. We can calculate $p_s^{\min} \sqsupseteq q$ either using an iterative method similar to the above or by reduction to the dual linear programming problem.

The model checking algorithm for "until" properties can be improved by precomputing the sets of *all* states from which the formula holds with maximal/minimal probability 0 and maximal/minimal probability 1 by means of graph-based analysis. Further details on these precomputation algorithms can be found in [25,31,32]. Furthermore, using such precomputation steps the model checking for $\models_{fair}$ can be reduced to that for ordinary satisfaction $\models$, see [26,30,32] for further details.

For DTMCs, which are a subset of CPSs, model checking of "until" reduces to solving the following linear equation system in $|S^{?}|$ unknowns:

$$x_s = \sum_{t\in S^{?}} \pi_s(t) \cdot x_t + \sum_{t\in S^{\mathrm{yes}}} \pi_s(t)$$

where π_s is the unique distribution such that $s \xrightarrow{a} \pi_s$. This system of equations can be solved either through a direct method such as Gaussian elimination, or iteratively via e.g. Jacobi or Gauss-Seidel iteration (see Sec. 6.3).

3.3 Analysing MTS models

The analysis of MTSs proceeds via a transformation of the model into a continuous time Markov chain. Subsequently, this CTMC is analysed with standard numerical methods.

We first construct an unlabelled version of the Markov transition system, by removing all actions labelling transitions. For a given MTS $\mathcal{S} = (S, \bar{s}, Act, \rightarrow, \mathrm{AP}, L)$, this is achieved by **hide** Act **in** $\mathcal{S}$. The resulting MTS is the tuple $(S, \bar{s}, \{\tau_i, \tau_m\}, \rightarrow, \mathrm{AP}, L)$. Note that the elements of $\rightarrow$ are now either of the form $s \xrightarrow{\tau_i} s'$ or $s \overset{\tau_m}{\dashrightarrow}_\lambda s'$.

In order to associate a CTMC with the MTS under study, transitions of the form $s \xrightarrow{\tau_i} s'$ need to be eliminated. Recall from Sec. 2 that an MTS $(S, \bar{s}, \{a\}, \rightarrow, \mathrm{AP}, L)$ is a CTMC, if for each $(s, a, f) \in\ \rightarrow$, f assigns a positive real value to precisely one state. This is not the case for transitions of the form $s \xrightarrow{\tau_i} s'$. We now discuss how to eliminate these transitions, and proceed similarly to the "well-defined" and "well-specified" algorithms of [33,34], see also [35]. For $s \in S$, let $\tau_i^+(s)$ denote the set of states maximally reachable from s via $\xrightarrow{\tau_i}$, i.e., $\tau_i^+(s) = \{s' \in S \mid s \xrightarrow{\tau_i}{}^* s' \wedge s' \overset{\tau_i}{\not\rightarrow}\}$, where $\xrightarrow{\tau_i}{}^*$ denotes the transitive closure of internal immediate transitions and $s \overset{\tau_i}{\not\rightarrow}$ denotes a "stable" state s which does not possess an outgoing internal immediate transition, i.e. $s \overset{\tau_i}{\not\rightarrow}$ if and only if $\neg \exists s' \in S : s \xrightarrow{\tau_i} s'$.

- If, for each state s in S, $\tau_i^+(s)$ is a singleton, then the system is well-defined (or well-specified) according to [33,34], and the elimination proceeds as follows. Define a CTMC $(S, \bar{s}', \{\tau_m\}, \rightarrow', \mathrm{AP}, L)$ with $\rightarrow'$ the least relation satisfying that if $\tau_i^+(s') = \{s''\}$ then $s \overset{\tau_m}{\dashrightarrow}{}'_\lambda\ s''$ whenever $s \overset{\tau_m}{\dashrightarrow}_\lambda s'$. In this case, each sequence of immediate transitions (starting in s') has a unique stable end point s'' that can replace the end point s' in $s \overset{\tau_m}{\dashrightarrow}_\lambda s'$. The new initial state $\bar{s}'$ is given by $\tau_i^+(\bar{s})$ (which is equal to $\bar{s}$ if $\bar{s}$ is stable). Note that by this construction of $\rightarrow'$ Markovian transitions emanating from unstable states will no longer be reachable, but they can easily be identified by reachability analysis and then deleted.
- In the general case, there may be states s for which $\tau_i^+(s)$ is not a singleton. As a first option, the elimination can be performed by constructing an MTS which is bisimilar to the original one, with respect to weak Markovian bisimulation [20,36]. As a second option, similar to the CPS case (cf. Sec. 3.2), we may assume that for every path ending in an unstable state there is a scheduler which assigns probabilities to the successor states. In this case we can adapt the algorithm of [37], to remove immediate transitions. In the following we sketch the effect of this algorithm from an operational point of view, see e.g. [38] for a description on the level of the matrix representing the state space.

(1) Identify the unstable states. For every unstable state with more than one emanating $\xrightarrow{\tau_i}$ transition, a scheduler must assign probabilities to those transitions[3]. If probability p is assigned to transition $t \xrightarrow{\tau_i} u$, we write

[3] Lacking any further information, equiprobability can be considered as the best possible choice, because it maximises the entropy [39]. On the other hand, the use of equiprobable schedulers may yield different results for bisimilar processes.

$t \xrightarrow{\tau_i, p} u$.

(2) Delete all Markovian transitions emanating from states which have at least one outgoing $\xrightarrow{\tau_i}$ transition, since these Markovian transitions would never be taken.

(3) Step (2) may have rendered some states unreachable. Determine the unreachable states and delete them and all transitions (regardless of their type) emanating from them.

(4) While there are still unstable states, select one of them (let it be called state t) and do the following: redirect transitions leading to t (regardless of their type) to the successor states of t, thereby taking into account the probabilities. More precisely, if $s \overset{\tau_m}{\dashrightarrow}_{\lambda} t$ and $t \xrightarrow{\tau_i, p} u$, then modify the former transition as $s \overset{\tau_m}{\dashrightarrow}_{p\lambda} u$; if $s \xrightarrow{\tau_i, q} t$ and $t \xrightarrow{\tau_i, p} u$, then modify the former transition as $s \xrightarrow{\tau_i, pq} u$. Afterwards, delete t and all transitions emanating from it. This step may lead to the existence of immediate loops of the kind $s \xrightarrow{\tau_i, pq} s$. Such loops can be eliminated by deleting them and multiplying all other $\xrightarrow{\tau_i}$ transitions emanating from s with the factor $1/(1-pq)$.

In this algorithm, steps (2) and (3), which delete unreachable transitions, are optional.

Once the CTMC is constructed, i.e. its states and transition rates have been determined, it can be analysed by standard numerical techniques [40]. For instance, in order to obtain the steady-state probabilities, a linear system of equations, given by $\mathbf{x} \cdot Q = \mathbf{0}$, must be solved. Here, $\mathbf{x}$ denotes the vector of steady-state probabilities, Q denotes the infinitesimal generator matrix of the CTMC (which is equal to its rate matrix augmented by a diagonal containing the negative row sums), and $\mathbf{0}$ denotes a vector of all zeroes. The vector of transient state probabilities at a given time t, given by $\mathbf{x}(t) = \mathbf{x}_0 \cdot e^{Q \cdot t}$, can also be determined, e.g. by the method of uniformisation [41,42].

When analysing MTSs, the foremost aim of performance evaluation is to calculate the probability with which a certain property holds either at a given finite point in time or on the long run (i.e. at time infinity). For this purpose we use atomic propositions to characterise interesting state properties. So, once the state probabilities are computed, the probability that a given atomic proposition φ holds can be computed easily by summing up the matching state probabilities:

$$Prob(\varphi) = \sum_{s \in S,\, s \models \varphi} x_s$$

where we write $s \models \varphi$ if and only if $\varphi \in L(s)$ for state $s \in S$ and $\varphi \in \mathrm{AP}$ (as in the CPS context).

4 Multi-Terminal Binary Decision Diagrams

This section introduces the MTBDD data structure and operations thereon. It also explains how matrices and vectors can be represented and manipulated with the help of MTBDDs.

4.1 Basics of MTBDDs

An MTBDD [16,15] is a graph-based structure for representing functions of the type $f : \mathbb{B}^n \mapsto \mathcal{D}$, i.e. functions from a multi-dimensional Boolean domain to an arbitrary range $\mathcal{D}$ (note that for a fixed n the image of the function f is always finite). For instance, $\mathcal{D}$ can be the real numbers. In the special case $\mathcal{D} = \mathbb{B}$ the MTBDD actually reduces to a BDD [14], representing a Boolean function. Thus, MTBDDs can be seen as a generalisation of BDDs.

The main idea behind the MTBDD representation of real-valued functions is the use of a rooted directed acyclic graph as a more compact representation of the *binary decision tree* which results from the *Shannon expansion*

$$f(\mathsf{v}_1, \ldots, \mathsf{v}_n) \;=\; \mathsf{v}_1 \cdot f(1, \mathsf{v}_2, \ldots, \mathsf{v}_n) \;+\; (1 - \mathsf{v}_1) \cdot f(0, \mathsf{v}_2, \ldots, \mathsf{v}_n)$$

where $\mathsf{v}_1, \ldots, \mathsf{v}_n$ are Boolean variables and $+$ and $\cdot$ denote ordinary addition and multiplication.

Definition 12 *Let $\mathcal{D}$ be an arbitrary set, and* Var *be a finite set of Boolean variables, equipped with a total ordering $\prec \subset \mathsf{Var} \times \mathsf{Var}$. A* Multi-Terminal Binary Decision Diagram *(MTBDD) over $\langle \mathsf{Var}, \prec \rangle$ is a rooted acyclic directed graph with finite vertex set $V = V_{NT} \cup V_T$ and the following labelling:*

- *Each non-terminal vertex $v \in V_{NT}$ is labelled by a variable $\mathsf{var}(v) \in \mathsf{Var}$ and has two children $\mathsf{then}(v), \mathsf{else}(v) \in V$.*
- *Each terminal vertex $v \in V_T$ is labelled by an element of $\mathcal{D}$, denoted by $\mathsf{value}(v) \in \mathcal{D}$.*

In addition, the labelling of the non-terminal vertices by variables respects the given ordering $\prec$, i.e. if $v, \mathsf{then}(v) \in V_{NT}$ then $\mathsf{var}(v) \prec \mathsf{var}(\mathsf{then}(v))$, and if $v, \mathsf{else}(v) \in V_{NT}$ then $\mathsf{var}(v) \prec \mathsf{var}(\mathsf{else}(v))$.

The edge from v to $\mathsf{then}(v)$ represents the case where $\mathsf{var}(v)$ is true; conversely, the edge from v to $\mathsf{else}(v)$ the case where $\mathsf{var}(v)$ is false. Each MTBDD M over $(\mathsf{v}_1, \ldots, \mathsf{v}_n)$ represents a function $f_{\mathsf{M}} : \mathbb{B}^n \mapsto \mathcal{D}$ and has two *cofactors* $\mathsf{M}^{\mathsf{then}}$

v_1	v_2	v_3	v_4	value
0	1	0	0	3
0	1	0	1	3
0	1	1	1	2
1	0	0	0	4
1	0	1	0	2
1	0	1	1	1
else				0

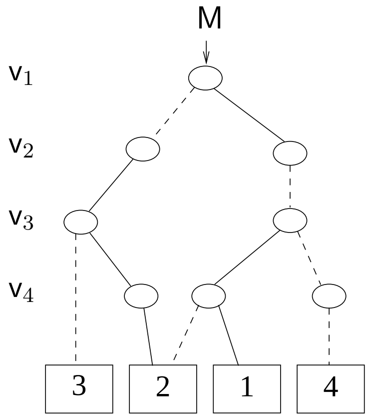


Fig. 2. MTBDD M, representing a function $f_{\mathsf{M}} : \{0,1\}^4 \mapsto \{0,\ldots,4\}$.

and $\mathsf{M}^{\mathsf{else}}$, resulting from a top-level Shannon expansion, i.e. $\mathsf{M}^{\mathsf{then}}$ and $\mathsf{M}^{\mathsf{else}}$ represent $f_{\mathsf{M}}(1, \mathsf{v}_2, \ldots, \mathsf{v}_n)$ and $f_{\mathsf{M}}(0, \mathsf{v}_2, \ldots, \mathsf{v}_n)$ respectively. [4]

Fig. 2 shows a simple MTBDD M over $(\mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3, \mathsf{v}_4)$ together with the function f_{M} it represents. In the graphical representation, the non-terminal vertices are grouped into four levels, and all vertices on the same level are assumed to be labelled with the variable denoted on the left. We adopt the convention that edges from vertex v to $\mathsf{then}(v)$ are drawn solid, while edges to $\mathsf{else}(v)$ are drawn dashed. For clarity we omit the terminal vertex 0 and all edges to it from the diagram.

Definition 13 *An MTBDD* M *is called* reduced *if and only if the following conditions hold:*

- *For each non-terminal vertex $v \in V_{NT}$ the two children are distinct, i.e.* $\mathsf{then}(v) \neq \mathsf{else}(v)$. *Each terminal vertex $v \in V_T$ has a distinct label* $\mathsf{value}(v)$.
- *For all vertices $v, v' \in V_{NT}$ with the same labelling, if the subgraphs with root v and v' respectively are identical then $v = v'$. Formally, if* $\mathsf{var}(v) = \mathsf{var}(v')$ *and* $\mathsf{else}(v) = \mathsf{else}(v')$ *and* $\mathsf{then}(v) = \mathsf{then}(v')$, *then $v = v'$.*

For a fixed ordering of Boolean variables, reduced MTBDDs form a *canonical* representation of $\mathcal{D}$-valued functions, i.e. if M, M' are two reduced MTBDDs over the same ordered set Var such that $f_{\mathsf{M}} = f_{\mathsf{M}'}$, then M and M' are isomorphic. Bryant [14] proposed a recursive procedure to reduce BDDs that can be applied to MTBDDs as well. In this paper, we assume that all MTBDDs are reduced.

[4] Note that an MTBDD over $\langle \mathsf{Var}, \prec \rangle$ is also an MTBDD over $\langle \mathsf{Var}', \prec' \rangle$ for any superset Var' of Var and total ordering $\prec'$ on Var' such that $\mathsf{v}_1 \prec \mathsf{v}_2$ if and only if $\mathsf{v}_1 \prec' \mathsf{v}_2$ for all $\mathsf{v}_1, \mathsf{v}_2 \in \mathsf{Var}$. If $\mathsf{Var} = \{\mathsf{v}_1, \ldots, \mathsf{v}_n\}$ and $\mathsf{v}_1 \prec \mathsf{v}_2 \prec \ldots \prec \mathsf{v}_n$ then we also speak about MTBDDs over $(\mathsf{v}_1, \ldots, \mathsf{v}_n)$ rather than MTDDs over $\langle \mathsf{Var}, \prec \rangle$.

MTBDD M of Fig. 2 satisfies Definition 13, i.e. it is *reduced.* Note that the valuations of some variable levels are irrelevant on certain paths through the MTBDD. For instance, for function f_{M} to return the value -3, the truth value of variable v_4 is irrelevant. Hence, the v_4-labelled vertex on this path is skipped, a consequence of the first clause of Definition 13. Variable v_4 is therefore called a *don't-care variable* for the respective path.

We remark at this point that the variable ordering chosen may have an immense effect upon the size of the MTBDD, i.e. the number of vertices. A prominent example is the identity function Id which we introduce below (in Sec. 4.3). The issue of variable ordering will be further discussed in Sec. 8.

4.2 Operations on MTBDDs

In this section, we describe how standard logical and arithmetic operations can be realised on MTBDDs. Let M, M_1, M_2 be MTBDDs over $(\mathsf{v}_1, \ldots, \mathsf{v}_n)$. In what follows, we write $v_1 \prec v_2$ if either v_2 is a terminal vertex while v_1 is non-terminal or both v_1, v_2 are non-terminal vertices and $\mathsf{var}(v_1) \prec \mathsf{var}(v_2)$. From here on, unless otherwise noted, we assume that $\mathcal{D} = I\!R$.

Variable renaming: Let $\mathsf{w} \notin \{\mathsf{v}_1, \ldots, \mathsf{v}_n\}$ and $i \in \{1, \ldots, n\}$ with $\mathsf{v}_{i-1} \prec \mathsf{w} \prec \mathsf{v}_{i+1}$. Then, $\mathsf{M}\{\mathsf{v}_i \leftarrow \mathsf{w}\}$ denotes the MTBDD over $(\mathsf{v}_1, \ldots, \mathsf{v}_{i-1}, \mathsf{w}, \mathsf{v}_{i+1}, \ldots, \mathsf{v}_n)$ that results from M where we change the variable labelling of any v_i-labelled vertex into w. For this, we set $\mathsf{var}(v) = \mathsf{w}$ for any v_i-labelled vertex v in M.[5] If $1 \leq i_1 < \ldots < i_m \leq n$ and $\mathsf{w}_1, \ldots, \mathsf{w}_m, \mathsf{v}_1, \ldots, \mathsf{v}_n$ are pairwise distinct, we write $\mathsf{M}\{\mathsf{v}_{i_1} \leftarrow \mathsf{w}_1, \ldots, \mathsf{v}_{i_m} \leftarrow \mathsf{w}_m\}$ as a shorthand for $\mathsf{M}\{\mathsf{v}_{i_1} \leftarrow \mathsf{w}_1\} \ldots \{\mathsf{v}_{i_m} \leftarrow \mathsf{w}_m\}$.

Restriction: Let $i \in \{1, \ldots, n\}$ and $b \in \{0, 1\}$. Then $\mathsf{M}|_{\mathsf{v}_i = b}$ denotes the MTBDD over $(\mathsf{v}_1, \ldots, \mathsf{v}_{i-1}, \mathsf{v}_{i+1}, \ldots, \mathsf{v}_n)$ that is obtained from M by replacing any edge from a vertex v to a v_i-labelled vertex w by an edge from v to $\mathsf{then}(w)$ if $b = 1$ ($\mathsf{else}(w)$ if $b = 0$), and by removing all v_i-labelled vertices. Thus, for instance, $\mathsf{M}|_{\mathsf{v}_1 = 0}$ represents the partial function $(\mathsf{v}_2, \ldots, \mathsf{v}_n) \mapsto f_{\mathsf{M}}(0, \mathsf{v}_2, \ldots, \mathsf{v}_n)$, which is identical to the cofactor $\mathsf{M}^{\mathsf{else}}$.

The Apply operator: If $\textsc{op}$ is a binary operator (e.g. addition $+$ or multiplication $\times$) then $\textsc{Apply}(\mathsf{M}_1, \mathsf{M}_2, \textsc{op})$ returns the MTBDD M over $(\mathsf{v}_1, \ldots, \mathsf{v}_n)$ where $f_{\mathsf{M}} = f_{\mathsf{M}_1} \;\textsc{op}\; f_{\mathsf{M}_2}$. If M_1 and M_2 are both BDDs and $\textsc{op}$ is a binary Boolean operation such as conjunction, disjunction or implication, then $\textsc{Apply}(\mathsf{M}_1, \mathsf{M}_2, \textsc{op})$ works in exactly the same manner. We often abbreviate the $\textsc{Apply}$ operator to an infix notation and simply write $\mathsf{M}_1 \;\textsc{op}\; \mathsf{M}_2$.

[5] Note that M and $\mathsf{M}\{\mathsf{v}_i \leftarrow \mathsf{w}\}$ represent the same function (when viewed as MTBDDs over $(\mathsf{v}_1, \ldots, \mathsf{v}_{i-1}, \mathsf{v}_i, \mathsf{v}_{i+1}, \ldots, \mathsf{v}_n)$ and $(\mathsf{v}_1, \ldots, \mathsf{v}_{i-1}, \mathsf{w}, \mathsf{v}_{i+1}, \ldots, \mathsf{v}_n)$, respectively).

The algorithm that realises $\textsc{Apply}(\mathsf{M}_1, \mathsf{M}_2, \textsc{op})$ calls a recursive procedure $A_{\mathrm{OP}}(v_1, v_2)$ that takes a vertex v_1 of M_1 and a vertex v_2 of M_2 as its input and returns a (reduced) MTBDD with root vertex v that represents the combination of the MTBDDs rooted at v_1 and v_2 by the operator $\textsc{op}$; for details see e.g. [14]. The worst case time complexity of $\textsc{Apply}(\mathsf{M}_1, \mathsf{M}_2, \textsc{op})$ is $O(|\mathsf{M}_1| \cdot |\mathsf{M}_2|)$, where $|\mathsf{M}_i|$ is the number of vertices of MTBDD M_i. This follows directly from the complexity analysis for $\textsc{Apply}$ for BDDs as given in [14].

Abstraction: For an associative binary operator $\textsc{op}$ we define $\textsc{Abstract}(\mathsf{M}, \mathsf{v}_i, \textsc{op}) := \mathsf{M}|_{\mathsf{v}_i=0}\ \textsc{op}\ \mathsf{M}|_{\mathsf{v}_i=1}$, which is called the abstraction of M with respect to the Boolean variable v_i and operator $\textsc{op}$. Abstracting with respect to more than one variable is defined as $\textsc{Abstract}(\mathsf{M}, (\mathsf{v}_{i_1}, \ldots, \mathsf{v}_{i_n}), \textsc{op})$ $:= \mathsf{M}|_{(\mathsf{v}_{i_1},\ldots,\mathsf{v}_{i_n})=(0,\ldots,0)}\ \textsc{op}\ \cdots\ \textsc{op}\ \mathsf{M}|_{(\mathsf{v}_{i_1},\ldots,\mathsf{v}_{i_n})=(1,\ldots,1)}$, i.e. all possible restrictions of M with respect to the Boolean variables $\mathsf{v}_{i_1}, \ldots, \mathsf{v}_{i_n}$ are combined by the operator $\textsc{op}$.

Constant: For real x, $\textsc{Constant}(x)$ returns the constant MTBDD with value x, i.e. the MTBDD consisting of a single terminal vertex v with $\mathsf{value}(v) = x$.

Absolute Maximum: Let $\{v_1 \ldots v_h\}$ be the terminal vertices of M. $\textsc{MaxAbs}(\mathsf{M})$ returns $\max_{1\le i\le h}\{\ |\mathsf{value}(v_i)|\ \}$, the maximum absolute value of the function f_{M}. This requires a simple traversal of the terminal vertices of M.

Threshold: The $\textsc{Threshold}$ operator converts an MTBDD to a BDD according to a given bound. For an MTBDD M, relational operator $\sim \in \{<, \le, \ge, >\}$ and real x, $\textsc{Threshold}(\mathsf{M}, \sim, x)$ returns the BDD representing the function $f = 1$ if $f_{\mathsf{M}} \sim x$ and 0 otherwise .

4.3 Manipulating Matrices and Vectors with MTBDDs

Vector-matrix multiplication: We first describe how (real-valued) matrices are represented by MTBDDs. For simplicity we assume square matrices whose dimension is a power of 2, i.e. 2^n. Rectangular matrices of general dimensions can be represented with the same basic scheme by padding them with an appropriate number of columns and rows of zeroes. A $2^n \times 2^n$ matrix M can be seen as a function from $\{0, \ldots, 2^n - 1\} \times \{0, \ldots, 2^n - 1\}$ to $\mathbb{R}$. Let $\mathsf{Var} = \{\mathsf{s}_1, \ldots, \mathsf{s}_n, \mathsf{t}_1, \ldots, \mathsf{t}_n\}$. If the row position s is encoded by Boolean variables s_i and the column position t by Boolean variables t_i (where in both cases $i = 1, \ldots, n$), then the MTBDD M over $\langle \mathsf{Var}, \prec \rangle$, where $f_{\mathsf{M}}(\mathsf{s}_1, \ldots, \mathsf{s}_n, \mathsf{t}_1, \ldots, \mathsf{t}_n) = M(s, t)$, is a canonical representation of matrix M.

Concerning the variable ordering, it turns out that an interleaving of the Boolean variables encoding row and column position, i.e. the ordering $\mathsf{s}_1 \prec$

$\mathsf{t}_1 \prec \ldots \prec \mathsf{s}_n \prec \mathsf{t}_n$, is usually the best choice, for the following reasons:

– The cofactors of the MTBDD correspond to block submatrices of the matrix. For instance, $\mathsf{M}^{\mathsf{else}}$ corresponds to the upper half of matrix M, $\mathsf{M}^{\mathsf{then}^{\mathsf{else}}}$ corresponds to the lower left quarter of matrix M, etc.
– The identity matrix, corresponding to the function[6] $Id = \prod_{k=1}^{n}(\mathsf{s}_k \equiv \mathsf{t}_k)$, can be represented in a number of vertices which is logarithmic in the dimension of the matrix. More precisely, the number of vertices needed to represent an identity matrix of dimension $2^n \times 2^n$ is $3n+2$. In contrast, using the straightforward ordering $\mathsf{s}_1 \prec \ldots \prec \mathsf{s}_n \prec \mathsf{t}_1 \prec \ldots \prec \mathsf{t}_n$ the size of the MTBDD would be $3 \cdot 2^n - 1$ vertices. Since identity matrices play an important role during the parallel composition of transition systems (see Sec. 5.2), their compact representation is an essential feature of MTBDDs.

Similarly as for matrices, a (row or column) vector $\mathbf{x}$ of size 2^n can be seen as a function from $\{0, \ldots, 2^n - 1\}$ to $I\!R$. Let $\mathsf{Var} = \{\mathsf{p}_1, \ldots, \mathsf{p}_n\}$. If an element's position p is encoded by Boolean variables p_i (where $i = 1, \ldots, n$), then the MTBDD X over $\langle \mathsf{Var}, \prec \rangle$, where $f_{\mathsf{X}}(\mathsf{p}_1, \ldots, \mathsf{p}_n) = \mathbf{x}(p)$, is a canonical representation of vector $\mathbf{x}$.

Based on this encoding scheme for matrices and vectors, vector-matrix multiplication can be realised on the MTBDD representation. Consider a vector $\mathbf{x}$ represented by an MTBDD X over variables $(\mathsf{s}_1, \ldots, \mathsf{s}_n)$, and a square matrix M, represented as an MTBDD M over variables $(\mathsf{s}_1, \mathsf{t}_1 \ldots, \mathsf{s}_n, \mathsf{t}_n)$. $\textsc{VMmult}(\mathsf{X}, \mathsf{M})$ produces an MTBDD Y over $(\mathsf{t}_1, \ldots, \mathsf{t}_n)$, representing the vector-matrix product $\mathbf{y} = \mathbf{x} \cdot M$. This MTBDD can be generated by *recursive descent*, according to the following idea: Compute the two halves of $\mathbf{y}$ corresponding to the cofactors $\mathsf{Y}^{\mathsf{else}}$ and $\mathsf{Y}^{\mathsf{then}}$ on the basis of the cofactors of Y and M as follows:

$$
\begin{aligned}
\mathsf{Y}^{\mathsf{else}} &= \textsc{Apply}\left(\textsc{VMmult}(\mathsf{x}^{\mathsf{else}}, \mathsf{M}^{\mathsf{else}^{\mathsf{else}}}), \textsc{VMmult}(\mathsf{x}^{\mathsf{then}}, \mathsf{M}^{\mathsf{then}^{\mathsf{else}}}), +\right) \\
\mathsf{Y}^{\mathsf{then}} &= \textsc{Apply}\left(\textsc{VMmult}(\mathsf{x}^{\mathsf{else}}, \mathsf{M}^{\mathsf{else}^{\mathsf{then}}}), \textsc{VMmult}(\mathsf{x}^{\mathsf{then}}, \mathsf{M}^{\mathsf{then}^{\mathsf{then}}}), +\right).
\end{aligned}
$$

The expression for $\mathsf{Y}^{\mathsf{else}}$ is the MTBDD reformulation of the fact that the left half of $\mathbf{x} \cdot M$ equals the sum of (1) the product of the left half of $\mathbf{x}$ and upper left quarter of M, and (2) the product of the right half of $\mathbf{x}$ and the lower left quarter of M. The four smaller size products $\textsc{VMmult}(\mathsf{x}^{\mathsf{else}}, \mathsf{M}_2^{\mathsf{else}^{\mathsf{else}}}), \ldots, \textsc{VMmult}(\mathsf{x}^{\mathsf{then}}, \mathsf{M}_2^{\mathsf{then}^{\mathsf{then}}})$ are recursively computed in the same way. The recursion terminates when the operands of $\textsc{VMmult}$ are terminal vertices v_1 and v_2, in which case a terminal vertex labelled by $\mathsf{value}(v_1) \cdot \mathsf{value}(v_2)$ is returned.

[6] Using only addition and multiplication, function Id would be written as $Id = \prod_{k=1}^{n}(\mathsf{s}_k \cdot \mathsf{t}_k + (1 - \mathsf{s}_k) \cdot (1 - \mathsf{t}_k))$.

Matrix-vector (and matrix-matrix) multiplication MVMULT (MMULT) can be performed by the same basic strategy. For two square matrices M_1 and M_2, represented as MTBDDs M_1 and M_2 over variables $(\mathsf{s}_1, \mathsf{t}_1 \ldots, \mathsf{s}_n, \mathsf{t}_n)$ and $(\mathsf{t}_1, \mathsf{t}'_1 \ldots, \mathsf{t}_n, \mathsf{t}'_n)$, MMULT$(\mathsf{M}_1, \mathsf{M}_2)$ produces an MTBDD M over $(\mathsf{s}_1, \mathsf{t}'_1 \ldots, \mathsf{s}_n, \mathsf{t}'_n)$, representing the matrix product $M = M_1 \cdot M_2$.

The naïve approach to vector-matrix multiplication which we just described is not sufficient (it does not work correctly) if X and M are reduced MTBDDs where variable levels may be skipped (as is the case, for example, when representing regularly structured vectors or matrices with repeated submatrices). The literature on MTBDDs describes algorithms for vector-matrix and matrix-matrix multiplication which overcome the shortcomings of the given simple scheme discussed above [16,43,15]. These implementations work on reduced MTBDDs and return MTBDDs that are reduced (and hence canonical) by construction. The general idea is to pass additional integer parameters to functions such as VMMULT, basically to take care of the variable levels that are skipped.

Inversion of triangular matrices: Inversion of triangular matrices, denoted INV_TRI, can be performed on their MTBDD-based representation by a straightforward algorithm, which relies on the recursive inversion of the quarters of the matrix, as described in [16].

4.4 Implementation aspects

We now briefly touch on some aspects concerning the efficient implementation of MTBDD algorithms. We focus on APPLY as the typical representative, since the other algorithms are of a similar recursive nature.

In order to make sure that the MTBDD returned by APPLY$(\cdot)$ is in reduced form, the algorithm uses a "unique table" which contains all currently existing MTBDD vertices. A unique table entry for a non-terminal vertex v consists of the vertex identifier, the vertex's variable labelling $\mathsf{var}(v)$ and the two references to the children vertices $\mathsf{then}(v)$ and $\mathsf{else}(v)$. A unique table entry for a terminal vertex consists of the vertex identifier and the vertex's value labelling $\mathsf{value}(v)$. As a result of procedure $A_{\mathrm{OP}}(v_1, v_2)$, a new vertex is inserted into the unique table if an isomorphic vertex was not yet in the table. Otherwise, a reference of the already existing vertex is returned.

Most (MT)BDD packages [44] maintain a second table called the "computed table". This contains entries of the form $(v_1, v_2, \mathrm{OP}, v)$, where v is the identifier of the vertex which had been previously obtained when computing $A_{\mathrm{OP}}(v_1, v_2)$. Whenever A_{OP} is called, the algorithm checks whether there exists a matching entry in the computed table, and if this is the case simply returns the vertex found there. This reduces the complexity of typical (MT)BDD operations from exponential to quadratic time.

For efficiency reasons, both the unique table and the computed table are usually implemented with the help of hashing functions, and the computed table is realised as a finite size cache [44].

Finally, in order to improve efficiency, the algorithm to compute $A_{\mathrm{OP}}(v_1, v_2)$ may check for the presence of special "controlling" values [14] which can receive special treatment and thereby avoid the initiation of recursive calls. For instance, if OP is multiplication and v_1 is a terminal vertex with $\mathsf{value}(v_1) = 1$, then v_2 can be immediately returned as the result.

5 MTBDD-based representation and composition of SPTS models

5.1 *Encoding of general transition systems*

In Sec. 4.2 we have already explained how (real-valued) matrices — and thus Markov chains — can be represented with the help of MTBDDs. In this section, we will discuss the general approach to the symbolic representation of stochastic and probabilistic transition systems.

In general, elements of any finite set S can be encoded by Boolean vectors of length $\lceil \log_2 |S| \rceil$. As an example, suppose we have a set of actions given by $Act = \{a, b, \tau\}$. We can encode action a as the bitstring 01, b as 10 and τ as 00 (we write $\mathcal{E}(a) = 01$, $\mathcal{E}(b) = 10$ and $\mathcal{E}(\tau) = 00$). If we use Boolean variables a_1 and a_2 to characterise the two positions of such a bitstring, then the term $\overline{\mathsf{a}_1}\mathsf{a}_2$ corresponds to 01 (i.e. action a), the term $\mathsf{a}_1\overline{\mathsf{a}_2}$ corresponds to 10 (i.e. action b) and the term $\overline{\mathsf{a}_1\mathsf{a}_2}$ corresponds to 00 (i.e. action τ). As a second example, the states from the set $S = \{0, 1, 2, 3\}$ can be encoded by bitstrings of length two in the obvious way.

To represent a transition relation, it is always necessary to encode at least the source state and the target state of a particular transition. We will use Boolean variables $\mathsf{s}_1, \ldots, \mathsf{s}_{n_s}$ to encode the source state, and $\mathsf{t}_1, \ldots, \mathsf{t}_{n_s}$ to encode the target state. Concerning the ordering of the variables in the MTBDD, unless otherwise noted, we follow the commonly accepted heuristics, already mentioned in Sec. 4.2, which interleaves the Boolean variables for source and target state. For the case where the transition relation contains a (non-trivial) action component, that component must also be encoded, say by Boolean variables $\mathsf{a}_1, \ldots, \mathsf{a}_{n_a}$, as in the example above. It is customary to place the variables encoding the actions before the state variables. Furthermore, in the general case additional Boolean variables are needed to represent non-deterministic choices between transitions. We call these the choice variables, denoted by $\mathsf{c}_1, \ldots, \mathsf{c}_{n_c}$, and place them at the beginning of the ordering. The overall variable ordering is then

$$\mathsf{c}_1 \prec \cdots \prec \mathsf{c}_{n_c} \prec \mathsf{a}_1 \prec \cdots \prec \mathsf{a}_{n_a} \prec \mathsf{s}_1 \prec \mathsf{t}_1 \prec \cdots \prec \mathsf{s}_{n_s} \prec \mathsf{t}_{n_s}.$$

We introduce the following notation: for $n > 0$ and two Boolean vectors $\underline{x} = (x_1, \ldots, x_n)$ and $\underline{y} = (y_1, \ldots, y_n)$ of length n, let $\underline{x} \bowtie \underline{y}$ abbreviate $(x_1, y_1, \ldots, x_n, y_n)$. We now give a formal definition of the correspondence between an SPTS and an MTBDD.

Definition 14 *Let $\mathcal{S}$ be an SPTS $(S, \bar{s}, Act, \rightarrow, \mathrm{AP}, L)$ and let M be an MTBDD over $(\mathsf{v}_1, \ldots, \mathsf{v}_m)$. M is said to* represent *$\mathcal{S}$ if and only if*

- *there is an injective function $\mathcal{E}_A : Act \mapsto \mathbb{B}^{n_a}$ for some $n_a \leq m$;*
- *there is an injective function $\mathcal{E}_S : S \mapsto \mathbb{B}^{n_s}$ for some $n_s \leq m$;*
- *$n_c = m - (n_a + 2n_s)$*

such that for all $a \in Act_i$ and $s \in S$ the following conditions hold:

(i1) if $s \xrightarrow{a} f$, then there exists $\underline{\mathsf{c}} \in \mathbb{B}^{n_c}$ such that for all $s' \in S$:

$$f_{\mathsf{M}}(\underline{\mathsf{c}}, \mathcal{E}_A(a), \mathcal{E}_S(s) \bowtie \mathcal{E}_S(s')) = f(s')$$

(i2) for all $\underline{\mathsf{c}} \in \mathbb{B}^{n_c}$, if there exists $s' \in S$ such that $f_{\mathsf{M}}(\underline{\mathsf{c}}, \mathcal{E}_A(a), \mathcal{E}_S(s) \bowtie \mathcal{E}_S(s')) \neq 0$, then $s \xrightarrow{a} f$ for some $f \in Dist(S)$ and for all $s' \in S$:

$$f_{\mathsf{M}}(\underline{\mathsf{c}}, \mathcal{E}_A(a), \mathcal{E}_S(s) \bowtie \mathcal{E}_S(s')) = f(s'),$$

and such that for all $a \in Act_m$ and $s \in S$ the following condition holds:

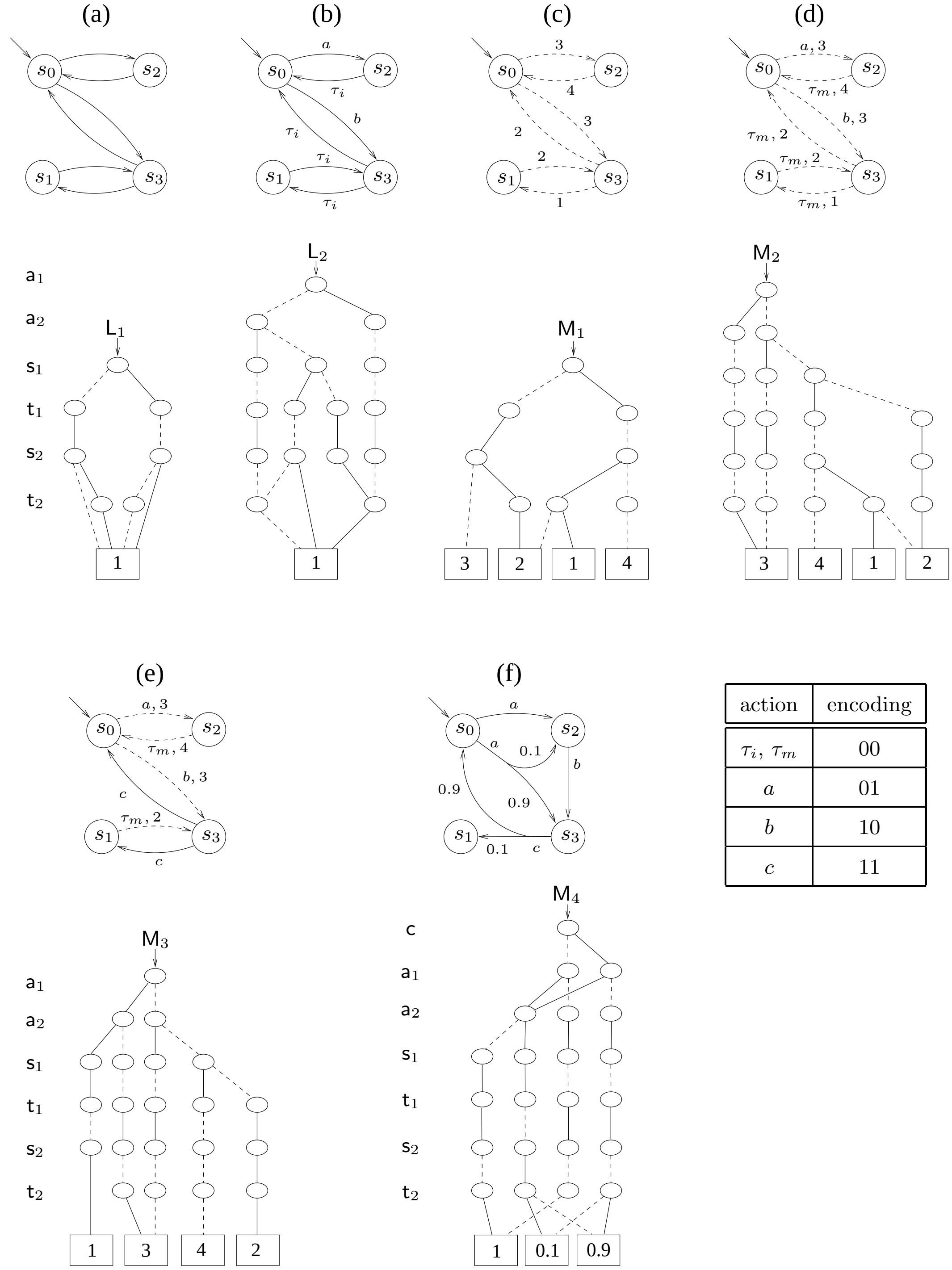
(m) $s \overset{a}{\dashrightarrow}_\lambda s'$ if and only if for all $\underline{\mathsf{c}} \in \mathbb{B}^{n_c}$:

$$f_{\mathsf{M}}\Big(\underline{\mathsf{c}}, \mathcal{E}_A(a), \mathcal{E}_S(s) \bowtie \mathcal{E}_S(s')\Big) = \lambda \text{ and } \lambda \neq 0$$

If M represents $\mathcal{S}$ we write $\mathsf{M} \triangleright \mathcal{S}$.

To illustrate this definition, Figure 3 shows six examples of how a transition system $\mathcal{S}$ can be represented by an MTBDD M.

(a) An LTS with only one single action, which does not have to be encoded explicitly. Since it is an LTS, there is no need to represent any numerical information concerning transition probabilities or rates. Between a given pair of states there either exists a transition (encoded by value 1) or there does not (encoded by value 0). Thus, the resulting structure is a BDD.



action	encoding
τ_i, τ_m	00
a	01
b	10
c	11

Fig. 3. Encoding of general transition systems. The action encodings are given in the table.

(b) An LTS with the (non-trivial) action set $Act = \{a, b, \tau_i\}$. Again, there is no numerical information and therefore we obtain a BDD.

(c) An MTS with only a single action, and where all transitions are of type (m) (this corresponds to the rate matrix of a CTMC, i.e. to a real-valued matrix). Note that this is the same function (after a renaming of Boolean variables) as the one used before in Figure 2.

(d) An MTS with a non-trivial set of actions, and where all transitions are of type (m).

(e) An MTS with a non-trivial set of actions which has both immediate and Markovian transitions. Note the non-determinism between the two c-transitions emanating from state 3.

(f) A CPS which features non-determinism. Note that apart from the Boolean variables encoding the action name, the source state and the target state, an additional "choice" variable is employed (variable c) which encodes the non-deterministic decision made by the scheduler. Note also that the value of c is immaterial for the part of the MTBDD which encodes transitions other than the two a-transitions.

Note that, for the representation of CPSs, the choice variables c_i are necessary in order to distinguish between different probability distributions labelled with the same action label, whereas in the case of MTSs, where all probability distributions are degenerate, the choice variables are not needed and can be ignored.

Note further that Def. 14 does not fix a particular initial state $\bar{s}$. Therefore, in all the examples given, the initial state is not represented by the MTBDD, only the transition relation is encoded. In general, an MTBDD M can represent several distinct SPTSs, that (apart from isomorphism in states and action labelling) differ w.r.t. to their initial state. The initial state may be stored as a scalar in a separate location. Moreover, a non-trivial initial probability distribution may be stored as a probability vector, which of course can also be represented as an MTBDD, and which may be used as a starting point for numerical analysis. Furthermore, we do not explicitly include the labelling of states with atomic propositions in Def. 14. This may, in fact, be encoded implicitly in the choice of MTBDD variables. Alternatively, the information can be stored separately in a BDD.

For convenience, we define the following functions which, for a given MTBDD $\mathsf{M}(\mathsf{v}_1, \ldots, \mathsf{v}_m)$, select different subsets of Boolean variables:

- $\mathcal{V}_c(\mathsf{M}) = \{\mathsf{v}_1, \ldots, \mathsf{v}_{n_c}\}$ which selects the choice variables of M;
- $\mathcal{V}_a(\mathsf{M}) = \{\mathsf{v}_{n_c+1}, \ldots, \mathsf{v}_{n_c+n_a}\}$ which selects the action variables of M;
- $\mathcal{V}_s(\mathsf{M}) = \{\mathsf{v}_{n_c+n_a+1}, \ldots, \mathsf{v}_m\}$ which selects the state variables of M.

In this section, we will discuss how parallel composition of components, which are given as general transition systems, can be performed at the level of their MTBDD representation.

5.2.1 MTBDD-based parallel composition

The semantic rules from Def. 8 provide the formal basis to construct the possible transitions of the combined transition system one by one from the transitions of the partner systems. If MTBDDs are used to represent the transition systems, then all transitions of the combined system can be obtained in an efficient manner by performing only a few MTBDD operations, as discussed in the sequel. In order to see how this can be done, we first define the notion of compatible representations.

Definition 15 *Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two SPTSs with the same set of actions Act. Let MTBDDs $\mathsf{M}_1(\mathsf{v}_1, \ldots, \mathsf{v}_m)$ and $\mathsf{M}_2(\mathsf{v}'_1, \ldots, \mathsf{v}'_{m'})$ be two MTBDDs such that $\mathsf{M}_1 \triangleright \mathcal{S}_1$ and $\mathsf{M}_2 \triangleright \mathcal{S}_2$. The two MTBDDs are said to be compatible representations of $\mathcal{S}_1$ and $\mathcal{S}_2$ if and only if the following conditions are satisfied:*

- $\mathcal{V}_c(\mathsf{M}_1)$ *and* $\mathcal{V}_s(\mathsf{M}_1)$ *do not appear in* M_2*;*
- $\mathcal{V}_c(\mathsf{M}_2)$ *and* $\mathcal{V}_s(\mathsf{M}_2)$ *do not appear in* M_1*;*
- $\mathcal{V}_a(\mathsf{M}_2) = \mathcal{V}_a(\mathsf{M}_1)$ *(i.e. the same set of action variables is used) and for all $a \in Act$: $\mathcal{E}_a^{(1)}(a) = \mathcal{E}_a^{(2)}(a)$, where $\mathcal{E}_a^{(1)}(a)$ and $\mathcal{E}_a^{(2)}(a)$ denote the encoding of action a in MTBDD M_1 and M_2 respectively.*

Note that one can encode any set of actions, such as the set of synchronising actions *Sync*, as a BDD Sync, by taking the disjunction of the encodings of its elements. The complementary set (of non-synchronising actions), i.e. the complement of *Sync* with respect to *Act* is encoded by simply negating BDD Sync (in MTBDD terms: by taking $1 - \mathsf{Sync}$). In addition, one needs to express idling, i.e. the fact that one of the partners does not make a move but retains its current state. This can be represented by BDDs $\mathsf{Id}_i = \bigwedge_{k=1}^{n_s^{(i)}} (\mathsf{s}_k^{(i)} \equiv \mathsf{t}_k^{(i)})$, where $i = 1, 2$ denotes the first resp. second partner.

With the help of Def. 15, we can now provide a theorem which states the relationship between the MTBDD representation of the transition system resulting from parallel composition and the MTBDD representations of the two partner systems.

Theorem 16 *Let $\mathcal{S}_1 = (S_1, \bar{s}_1, Act, \rightarrow_1, \mathrm{AP}_1, L_1)$ and $\mathcal{S}_2 = (S_2, \bar{s}_2, Act, \rightarrow_2, \mathrm{AP}_2, L_2)$ be two SPTSs. Let MTBDDs M_1 and M_2 be compatible representations of $\mathcal{S}_1$ and $\mathcal{S}_2$. Let A_i (A_m) be the encoding of the* *set Act_i (Act_m). Let* Sync *be the encoding of the set of synchronising actions* $Sync \subseteq Act \setminus \{\tau_i, \tau_m\}$*, and let* c *be a new MTBDD variable (not in* M_1 *or* M_2*). If*

$$\begin{aligned}\mathsf{M} = {} & (\mathsf{M}_1 \cdot \mathsf{Sync}) \cdot (\mathsf{M}_2 \cdot \mathsf{Sync}) \\ & + \mathsf{c} \cdot \mathsf{M}_1 \cdot \mathsf{A}_i \cdot (1 - \mathsf{Sync}) \cdot \mathsf{Id}_2 \;+\; (1 - \mathsf{c}) \cdot \mathsf{M}_2 \cdot \mathsf{A}_i \cdot (1 - \mathsf{Sync}) \cdot \mathsf{Id}_1 \\ & + \mathsf{M}_1 \cdot \mathsf{A}_m \cdot (1 - \mathsf{Sync}) \cdot \mathsf{Id}_2 \;+\; \mathsf{M}_2 \cdot \mathsf{A}_m \cdot (1 - \mathsf{Sync}) \cdot \mathsf{Id}_1,\end{aligned}$$

where

$$\mathcal{V}_c(\mathsf{M}) = \{\mathsf{c}\} \cup \mathcal{V}_c(\mathsf{M}_1) \cup \mathcal{V}_c(\mathsf{M}_2),$$
$$\mathcal{V}_a(\mathsf{M}) = \mathcal{V}_a(\mathsf{M}_1) (= \mathcal{V}_a(\mathsf{M}_2)),$$
$$\mathcal{V}_s(\mathsf{M}) = \mathcal{V}_s(\mathsf{M}_1) \cup \mathcal{V}_s(\mathsf{M}_2),$$

then $\mathsf{M} \triangleright \mathcal{S}_1 \,|[Sync]|\, \mathcal{S}_2$.

The first line of the equation in Th. 16 generates all the transitions where both partners move together [7]. The second line generates those transitions of type (i) where the first (second) partner moves while the second (first) one remains idle (note the symmetry between the two terms of the second line). Since for transitions of type (i) there is a non-deterministic choice between which of the partners moves, a new choice variable c is introduced at this point. The third line, again consisting of two symmetric terms, generates those transitions of type (m) where the first (second) partner moves while the second (first) one remains idle.

The proof of the Th. 16 involves a particular fine point related to the use of the cumulation function C in Def. 8. In short, this function cumulates parallel, duplicated Markov transitions into a single transition, i.e. it flattens a multiset to a set. The proof of the theorem now needs to ensure that such parallel, duplicated transitions are cumulated also on the MTBDD level. We can assume that the component SPTSs (and their MTBDD representations M_1 and M_2) are free of such duplicates. In the composition, a detailed case analysis is needed to identify sources of such duplicate transitions in the construction. It turns out that this situation can only arise from the interleaving of two Markovian self-loops (one in each original SPTS). In this case, the third line of the definition of M takes care of the cumulation, by adding two symmetric terms.

Note that Th. 16 says nothing about the position of the new variable c within the variable ordering. As a heuristic, one may put it at the top of MTBDD M, i.e. first in the ordering, which is consistent with the ordering mentioned in Sec 5.1.

[7] Note that in the first line both probabilities of type (i) transitions and rates of type (m) transitions are multiplied, i.e. Th. 16 assumes that the function ϕ used in Def. 6 and Def. 8 is instantiated by multiplication.

Now we consider some special cases of Th. 16:

(1) Suppose that (MT)BDDs M_1 and M_2 are compatible representations of two ordinary labelled transition systems $\mathcal{S}_1$ and $\mathcal{S}_2$. Suppose further that the set of actions *Act* does not contain the internal action τ_i, so that synchronisation is indeed allowed on all actions in the set *Act*. Then the LTS for the parallel composition $\mathcal{S} = \mathcal{S}_1|[Act]|\mathcal{S}_2$ is represented by the (MT)BDD $\mathsf{M} = \mathsf{M}_1 \cdot \mathsf{M}_2$ $(= \mathsf{M}_1 \wedge \mathsf{M}_2)$, i.e. a single MTBDD-multiplication (BDD-AND) operation suffices in order to obtain the combined transition system. Formally:

Corollary 17 *Let* $\mathcal{S}_1 = (S_1, \bar{s}_1, Act, \rightarrow_1, \mathrm{AP}_1, L_1)$ *and* $\mathcal{S}_2 = (S_2, \bar{s}_2, Act, \rightarrow_2, \mathrm{AP}_2, L_2)$ *be two LTSs whose set of actions Act does not contain an internal action* τ_i*. Let* M_1 *and* M_2 *be two compatible MTBDDs. If* $\mathsf{M}_1 \triangleright \mathcal{S}_1$ *and* $\mathsf{M}_2 \triangleright \mathcal{S}_2$*, then* $\mathsf{M}_1 \cdot \mathsf{M}_2 \triangleright \mathcal{S}_1|[Act]|\mathcal{S}_2$.

(2) In the case of only partial synchronisation, things become a bit more involved. Let us still consider ordinary labelled transition systems but assume that the set of synchronising actions *Sync* is a proper subset of $Act \setminus \{\tau_i\}$. In this case, one has to distinguish between synchronising and non-synchronising actions, i.e. both of the above semantic rules will now be needed. Formally:

Corollary 18 *Let* $\mathcal{S}_1 = (S_1, \bar{s}_1, Act, \rightarrow_1, \mathrm{AP}_1, L_1)$ *and* $\mathcal{S}_2 = (S_2, \bar{s}_2, Act, \rightarrow_2, \mathrm{AP}_2, L_2)$ *be two LTSs. Let* M_1 *and* M_2 *be two compatible MTBDDs such that* $\mathsf{M}_1 \triangleright \mathcal{S}_1$ *and* $\mathsf{M}_2 \triangleright \mathcal{S}_2$*, and* Sync *be the encoding of the set of actions* $Sync \subset Act \setminus \{\tau_i\}$*. If*

$$\begin{aligned}\mathsf{M} = & (\mathsf{M}_1 \cdot \mathsf{Sync}) \cdot (\mathsf{M}_2 \cdot \mathsf{Sync}) \\ & + \mathsf{M}_1 \cdot (1 - \mathsf{Sync}) \cdot \mathsf{Id}_2 \ \vee \ \mathsf{M}_2 \cdot (1 - \mathsf{Sync}) \cdot \mathsf{Id}_1,\end{aligned}$$

then $\mathsf{M} \triangleright \mathcal{S}_1|[Sync]|\mathcal{S}_2$.

Note that choice variables are not needed in the MTBDD representation of an LTS. Hence, they can be removed from the second line of Th. 16. This does, however, require us to use disjunction [8] in place of summation, in case identical transitions are combined.

(3) For CTMCs, the transition relation contains transitions of type (m) only, i.e. transitions of the form $s \overset{a}{\dashrightarrow}_\lambda s'$. A CTMC can be represented by an MTBDD M as shown in Figure 3 (c). Since there are no actions over which to synchronise, the parallel composition of two CTMCs is simply

[8] We define disjunction on MTBDDs in the obvious way, provided that both arguments are actually BDDs.

the interleaving of the two stochastic processes [9]. Formally:

Corollary 19 *Let* $Act = \{a\}$ *(i.e. a singleton set) and* $\mathcal{S}_1 = (S_1, \bar{s}_1, Act, \rightarrow_1, \mathrm{AP}_1, L_1)$ *and* $\mathcal{S}_2 = (S_2, \bar{s}_2, Act, \rightarrow_2, \mathrm{AP}_2, L_2)$ *be two CTMCs. Let* M_1 *and* M_2 *be two compatible MTBDDs. If* $\mathsf{M}_1 \triangleright \mathcal{S}_1$ *and* $\mathsf{M}_2 \triangleright \mathcal{S}_2$*, then* $\mathsf{M}_1 \cdot \mathsf{Id}_2 + \mathsf{Id}_1 \cdot \mathsf{M}_2 \triangleright \mathcal{S}_1|||\mathcal{S}_2$*.* [10]

We can also interpret the above equation as a matrix equation. If MTBDD M_i represents the rate matrix M_i of CTMC $\mathcal{S}_i$, then M represents the Kronecker sum $M_1 \oplus M_2$, which is the rate matrix of the combined CTMC.

(4) For DTMCs the transition relation consists of (in general non-degenerate) probability distributions. A DTMC is represented by an MTBDD in the same way as a CTMC, with the difference that the terminal vertices contain transition probabilities instead of transition rates. DTMCs are discrete-time stochastic processes which make a move at every time step (possibly back to the current state). The probability that the combined DTMC moves from state (s, s') to (t, t') is given by $p_1 \cdot p_2$, provided that the probability for the partner chain to make a move from s to t (from s' to t') is p_1 (p_2). Formally:

Corollary 20 *Let* $Act = \{a\}$ *(i.e. a singleton set) and* $\mathcal{S}_1 = (S_1, \bar{s}_1, Act, \rightarrow_1, \mathrm{AP}_1, L_1)$ *and* $\mathcal{S}_2 = (S_2, \bar{s}_2, Act, \rightarrow_2, \mathrm{AP}_2, L_2)$ *be two DTMCs. Let* M_1 *and* M_2 *be two compatible MTBDDs. If* $\mathsf{M}_1 \triangleright \mathcal{S}_1$ *and* $\mathsf{M}_2 \triangleright \mathcal{S}_2$*, then* $\mathsf{M}_1 \cdot \mathsf{M}_2 \triangleright \mathcal{S}_1|[Act]|\mathcal{S}_2$*.*

We can again interpret the above equation as a matrix equation. If MTBDD M_i represents the stochastic matrix M_i of DTMC $\mathcal{S}_i$, then M represents the Kronecker product $M_1 \otimes M_2$, which is also a stochastic matrix, namely the transition probability matrix of the combined DTMC.

(5) For CPSs, the transition relation contains no transitions of type (m), meaning that the terms on the third line of the equation in Th. 16 are not needed. Transitions of type (i), however, are allowed in their full generality, and hence the choice variables must be retained unlike in case (2) above. Formally:

Corollary 21 *Let* $\mathcal{S}_1 = (S_1, \bar{s}_1, Act, \rightarrow_1, \mathrm{AP}_1, L_1)$ *and* $\mathcal{S}_2 = (S_2, \bar{s}_2, Act, \rightarrow_2, \mathrm{AP}_2, L_2)$ *be two CPSs. Let* M_1 *and* M_2 *be two compatible MTBDDs such that* $\mathsf{M}_1 \triangleright \mathcal{S}_1$ *and* $\mathsf{M}_2 \triangleright \mathcal{S}_2$*, and* Sync *be the encoding of the set of actions* $Sync \subset Act \setminus \{\tau_i\}$*. Let* c *be a new MTBDD variable (not in* M_1

[9] The probability that both partners make a move at exactly the same time is zero.
[10] The symbol $|||$ abbreviates $|[\emptyset]|$, i.e. denotes parallel composition without synchronisation.

or M_2*). If*

$$\begin{aligned}\mathsf{M} = {} & (\mathsf{M}_1 \cdot \mathsf{Sync}) \cdot (\mathsf{M}_2 \cdot \mathsf{Sync}) \\ & + \mathsf{c} \cdot \mathsf{M}_1 \cdot (1 - \mathsf{Sync}) \cdot \mathsf{Id}_2 \ + \ (1 - \mathsf{c}) \cdot \mathsf{M}_2 \cdot (1 - \mathsf{Sync}) \cdot \mathsf{Id}_1,\end{aligned}$$

then $\mathsf{M} \triangleright \mathcal{S}_1 \,|[Sync]|\, \mathcal{S}_2$.

(6) We now consider MTSs stemming from process algebraic specifications. These models have two types of transitions: transitions of type (m), denoted $s \overset{a}{\dashrightarrow}_\lambda s'$, which are as above in CTMCs but carry an additional non-trivial action label, and transitions of type (i), restricted to degenerate probability distributions and denoted $s \xrightarrow{a} s'$. Such an MTS can be represented by a single MTBDD whose terminal vertices can have the following values: (a) value 0, (b) value 1.0 [11], or (c) a real-valued rate (which, of course, can be equal to 1.0). Formally:

Corollary 22 *Let* $\mathcal{S}_1 = (S_1, \bar{s}_1, Act, \rightarrow_1, \mathrm{AP}_1, L_1)$ *and* $\mathcal{S}_2 = (S_2, \bar{s}_2, Act, \rightarrow_2, \mathrm{AP}_2, L_2)$ *be two MTSs. Let* M_1 *and* M_2 *be two compatible MTBDDs such that* $\mathsf{M}_1 \triangleright \mathcal{S}_1$ *and* $\mathsf{M}_2 \triangleright \mathcal{S}_2$*, and* Sync *be the encoding of a set of actions* $Sync \not\ni \tau_i, \tau_m$*. If*

$$\begin{aligned}\mathsf{M} = {} & (\mathsf{M}_1 \cdot \mathsf{Sync}) \cdot (\mathsf{M}_2 \cdot \mathsf{Sync}) \\ & + \mathsf{M}_1 \cdot \mathsf{A}_i \cdot (1 - \mathsf{Sync}) \cdot \mathsf{Id}_2 \ \vee \ \mathsf{M}_2 \cdot \mathsf{A}_i \cdot (1 - \mathsf{Sync}) \cdot \mathsf{Id}_1 \\ & + \mathsf{M}_1 \cdot \mathsf{A}_m \cdot (1 - \mathsf{Sync}) \cdot \mathsf{Id}_2 \ + \ \mathsf{M}_2 \cdot \mathsf{A}_m \cdot (1 - \mathsf{Sync}) \cdot \mathsf{Id}_1,\end{aligned}$$

then $\mathsf{M} \triangleright \mathcal{S}_1 |[Sync]| \mathcal{S}_2$.

Note that, as with LTSs (case (2) above), we can remove choice variables from the equation, but must be careful to replace summation with disjunction.

For practical reasons, it may be convenient to store Markovian transitions in an MTBDD and immediate transitions in a BDD, thus making the computations disjoint. This is indeed how the tool IM-CAT proceeds (see Sec. 7.2).

5.2.2 Size of the MTBDD resulting from parallel composition

We now discuss the size of the (MT)BDD resulting from parallel composition. First note that parallel composition of two SPTSs $\mathcal{S}_1$ and $\mathcal{S}_2$ with state sets S_1 and S_2 yields an overall transition system $\mathcal{S}$ with up to $|S_1| \cdot |S_2|$ states,

[11] If non-determinism between internal immediate transitions is later resolved by probabilities (as described in Sec. 3.3), the MTBDD will have terminal vertices whose value is a probability.

i.e. in the worst case the state space grows multiplicatively. Enders et al. [45] proved for the parallel composition of BDDs generated from CCS terms that the number of BDD vertices grows only additively. This result carries over to the general SPTS case which we consider in this paper.

Theorem 23 *Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two SPTSs represented by the two compatible MTBDDs M_1 and M_2, i.e. $\mathsf{M}_i \rhd \mathcal{S}_i$ $(i = 1, 2)$, using the standard interleaved variable ordering. For $Sync \subseteq Act$ with $Sync \not\ni \tau_i, \tau_m$, let M be the MTBDD representing the parallel composition $\mathcal{S} = \mathcal{S}_1|[Sync]|\mathcal{S}_2$, written $\mathsf{M} \rhd \mathcal{S}$. Then the number of vertices of MTBDD M is bounded by $k \cdot |Act| \cdot (|\mathsf{M}_1| + |\mathsf{M}_2| + |\mathsf{Id}_1| + |\mathsf{Id}_2|)$ where k depends on the number of non-deterministic choices and on the number of distinct rate or probability values that are associated with a particular action.*

We now sketch a proof of Th. 23. Suppose that the SPTS $\mathcal{S}_i$ is represented by MTBDD M_i, i.e. $\mathsf{M}_i \rhd \mathcal{S}_i$ (for $i = 1, 2$), and the variable ordering for MTBDD M_i is

$$\mathsf{c}_1^{(i)} \prec \ldots \prec \mathsf{c}_{n_c^i}^{(i)} \prec \mathsf{a}_1 \prec \ldots \prec \mathsf{a}_{n_a} \prec \mathsf{s}_1^{(i)} \prec \mathsf{t}_1^{(i)} \prec \ldots \prec \mathsf{s}_{n_s^i}^{(i)} \prec \mathsf{t}_{n_s^i}^{(i)},$$

i.e. the Boolean variables encoding the non-deterministic choices are at the top, followed by the action names, and finally an interleaving of the variables for source and target states. For the MTBDD M resulting from parallel composition we assume variable ordering:

$$\mathsf{c} \prec \mathsf{c}_1^{(1)} \prec \ldots \prec \mathsf{c}_{n_c^1}^{(1)} \prec \mathsf{c}_1^{(2)} \prec \ldots \prec \mathsf{c}_{n_c^2}^{(2)} \prec \mathsf{a}_1 \prec \ldots \prec \mathsf{a}_{n_a} \prec$$
$$\mathsf{s}_1^{(1)} \prec \mathsf{t}_1^{(1)} \prec \ldots \prec \mathsf{s}_{n_s^1}^{(1)} \prec \mathsf{t}_{n_s^1}^{(1)} \prec \mathsf{s}_1^{(2)} \prec \mathsf{t}_1^{(2)} \prec \ldots \prec \mathsf{s}_{n_s^2}^{(2)} \prec \mathsf{t}_{n_s^2}^{(2)}$$

where c is the new MTBDD choice variable added during the construction of M. The proof considers three cases.

(1) We consider first the case of parallel composition with maximal synchronisation, i.e. synchronisation on all actions. Let $|\mathsf{M}_i|$ be the number of vertices of M_i, $\underline{c} \in I\!B^{n_c^1+n_c^2+1}$ a valuation of the non-deterministic choice variables and A_a be the BDD encoding of the action $a \in Act$. Let $\mathsf{M}_{i,\underline{c}a} = \mathsf{M}_i \cdot (\underline{c} \cdot \mathsf{A}_a)$ be the restriction of M_i to the valuation $\underline{c}$ of the non-deterministic choice variables and action a. To obtain the subgraph of the resulting MTBDD M which corresponds to the valuation $\underline{c}$ of the non-deterministic choice variables and action a one has to build $\mathsf{M}_{\underline{c}a} = \mathsf{M}_{1,\underline{c}a} \cdot \mathsf{M}_{2,\underline{c}a} = (\mathsf{M}_1 \cdot \underline{c} \cdot \mathsf{A}_a) \cdot (\mathsf{M}_2 \cdot \underline{c} \cdot \mathsf{A}_a)$ whose number of vertices

can be bounded by:

$$\begin{aligned}|\mathsf{M}_{\underline{c}a}| &\leq |\mathsf{M}_{1,\underline{c}a}| + \eta_{1,\underline{c}a} \cdot |\mathsf{M}_{2,\underline{c}a}| \\ &\leq |\mathsf{M}_1| + \eta_{1,\underline{c}a} \cdot |\mathsf{M}_2| \\ &\leq \eta_{1,\underline{c}a} \cdot (|\mathsf{M}_1| + |\mathsf{M}_2|)\end{aligned}$$

In the latter equation, $\eta_{1,\underline{c}a}$ denotes the number of terminal vertices of $\mathsf{M}_{1,\underline{c}a}$ which is usually a small value in comparison with the size of M_1. Summing up over all valuations of the non-deterministic variables and actions we obtain $|\mathsf{M}| \leq 2^{n_c} \cdot |Act| \cdot \eta_1 \cdot (|\mathsf{M}_1| + |\mathsf{M}_2|)$, where $n_c = 1 + n_c^1 + n_c^2$ and $\eta_1 = \max_{\underline{c} \in \mathbb{B}^{n_c} \wedge a \in Act}\{\eta_{1,\underline{c}a}\}$. Note that 2^{n_c} and $|Act|$ are also usually small values[12] in comparison with the size of M_1 and M_2 and that this is a rather coarse worst case bound which assumes that there is no sharing of the subgraphs which correspond to different non-deterministic choices and actions.

(2) In the case where there are no synchronising actions the picture is as follows: Let $|\mathsf{M}_i|$, $\underline{c}$, A_a and $\mathsf{M}_{i,\underline{c}a}$ be defined as in (1). If $a \in Act_i$, then $\mathsf{M}_{\underline{c}a} = \mathsf{c} \cdot \mathsf{M}_{1,\underline{c}a} \cdot \mathsf{Id}_2 + (1-\mathsf{c}) \cdot \mathsf{Id}_1 \cdot \mathsf{M}_{2,\underline{c}a} = \mathsf{c} \cdot (\mathsf{M}_1 \cdot \underline{c} \cdot \mathsf{A}_a) \cdot \mathsf{Id}_2 + (1-\mathsf{c}) \cdot \mathsf{Id}_1 \cdot (\mathsf{M}_2 \cdot \underline{c} \cdot \mathsf{A}_a)$. Therefore, if c evaluates to true in $\underline{c}$:

$$|\mathsf{M}_{\underline{c}a}| \leq |\mathsf{M}_{1,\underline{c}a}| + \eta_{1,\underline{c}a} \cdot |\mathsf{Id}_2| \leq |\mathsf{M}_1| + 1 + \eta_{1,\underline{c}a} \cdot |\mathsf{Id}_2|$$

and if c evaluates to false in $\underline{c}$:

$$|\mathsf{M}_{\underline{c}a}| \leq |\mathsf{Id}_1| + |\mathsf{M}_{2,\underline{c}a}| \leq |\mathsf{Id}_1| + |\mathsf{M}_2| + 1.$$

On the other hand, if $a \in Act_m$, then $\mathsf{M}_{\underline{c}a} = \mathsf{M}_{1,\underline{c}a} \cdot \mathsf{Id}_2 + \mathsf{Id}_1 \cdot \mathsf{M}_{2,\underline{c}a} = (\mathsf{M}_1 \cdot \underline{c} \cdot \mathsf{A}_a) \cdot \mathsf{Id}_2 + \mathsf{Id}_1 \cdot (\mathsf{M}_2 \cdot \underline{c} \cdot \mathsf{A}_a)$ whose size can be bounded by:

$$\begin{aligned}|\mathsf{M}_{\underline{c}a}| &\leq |\mathsf{M}_{1,\underline{c}a}| + \eta_{1,\underline{c}a} \cdot |\mathsf{Id}_2| + |\mathsf{M}_{2,\underline{c}a}| + |\mathsf{Id}_1| \\ &\leq |\mathsf{M}_1| + \eta_{1,\underline{c}a} \cdot |\mathsf{Id}_2| + |\mathsf{M}_2| + |\mathsf{Id}_1|.\end{aligned}$$

Therefore, in the case of pure interleaving the overall size is bounded by $|\mathsf{M}| \leq 2^{n_c} \cdot |Act| \cdot (|\mathsf{M}_1| + \eta_1 \cdot |\mathsf{Id}_2| + |\mathsf{M}_2| + |\mathsf{Id}_1|)$. Remember that Id_i is represented in a compact manner with only $3n_s^i + 2 = 3\lceil \log |S_i| \rceil + 2$ vertices, that is, $|\mathsf{Id}_i|$ is usually much smaller than $|\mathsf{M}_i|$.

(3) For the general, mixed case, where there are both synchronising and non-synchronising transitions, we combine the two extremal cases and obtain the overall bound: $|\mathsf{M}| \leq 2^{n_c} \cdot |Act| \cdot \eta_1 \cdot (|\mathsf{M}_1| + |\mathsf{M}_2| + |\mathsf{Id}_1| + |\mathsf{Id}_2|)$.

This concludes the proof.

[12] 2^{n_c} is an upper bound for the number of non-deterministic choices in any state of $\mathcal{S}$.

The relevance of the above theorem relies on the following practical considerations. First, we note that Id_1 (respectively Id_2) grows logarithmically in the size of $\mathcal{S}_1$ (respectively $\mathcal{S}_2$), and is usually dominated by the size of M_1 (M_2). The size of the constructed MTBDD M is thus in the order of $k \cdot |Act| \cdot (|\mathsf{M}_1| + |\mathsf{M}_2|)$ where k subsumes $2^{n_c} \cdot \eta_1$. All three parameters $|Act|$, n_c and η_1 are model dependent. In principle, it is possible to construct pathological cases where either n_c or η_1 become large. In practice, however, both parameters are small values, often close or equal to 1. Therefore, as is our experience indeed (and as we shall see in the examples in Sec 7.3) the size of M is virtually linear in the size of M_1 and M_2.

5.2.3 Reachability considerations

The MTBDD M resulting from the parallel composition encodes all transitions which are possible in the product space of the two partner systems $\mathcal{S}_1$ and $\mathcal{S}_2$. Given a pair of initial states for $\mathcal{S}_1$ and $\mathcal{S}_2$, only part of the product space may be reachable due to synchronisation constraints. (Symbolic) reachability analysis can be performed on the MTBDD representation, restricting M to an MTBDD M_{reach} which contains only those transitions which originate in reachable states. However, contrary to what one would expect, the MTBDD M_{reach} is typically larger than M, although it encodes fewer transitions. In general, this rather counter-intuitive increase of the MTBDD size is due to the loss of regularity, see Sec. 8.

5.3 MTBDD-based abstraction in SPTS models

This section considers symbolic abstraction on MTBDDs. More precisely, we discuss how the MTBDD representation of the transition system resulting from an abstraction operation can be constructed from the MTBDD representation of the original transition system. For notational convenience, we first give the following definition:

Definition 24 *Given n Boolean variables $\mathsf{a}_1, \ldots, \mathsf{a}_n$ and a Boolean vector $(b_1, \ldots, b_n)$ of length n, we denote by $\mathcal{M}(\mathsf{a}_1, \ldots, \mathsf{a}_n; b_1, \ldots, b_n)$ the minterm consisting of the multiplication (resp. conjunction) of n literals (a literal is either a Boolean variable or its negation), i.e. $\mathcal{M}(\mathsf{a}_1, \ldots, \mathsf{a}_n; b_1, \ldots, b_n) = \mathsf{a}_1^* \cdots \mathsf{a}_n^*$ where $\mathsf{a}_i^* = \mathsf{a}_i$ if $b_i = 1$ and $\mathsf{a}_i^* = 1 - \mathsf{a}_i$ if $b_i = 0$.*

As an example, we have $\mathcal{M}(\mathsf{a}_1, \mathsf{a}_2, \mathsf{a}_3; 0, 1, 1) = (1 - \mathsf{a}_1) \cdot \mathsf{a}_2 \cdot \mathsf{a}_3$.

Theorem 25 *Let $\mathcal{S} = (S, \bar{s}, Act, \rightarrow, \mathrm{AP}, L)$ be an SPTS. Let MTBDD M represent $\mathcal{S}$, i.e. $\mathsf{M} \triangleright \mathcal{S}$. Let A be the encoding of the set $A \subseteq Act$ of actions to be hidden, and for each $a \in A$ let c_a be a new MTBDD variables (not in M).*

If

$$\begin{aligned}\mathsf{M}' = {} & \mathsf{M} \cdot (1 - \mathsf{A}) \\ & + \textstyle\sum_{a \in A \cap Act_i} \mathsf{c}_a \cdot \mathsf{M}|_{\underline{\mathsf{a}} = \mathcal{E}_A(a)} \cdot \mathcal{M}(\underline{\mathsf{a}}, \mathcal{E}_A(\tau_i)) \\ & + \textstyle\sum_{a \in A \cap Act_m} \mathsf{M}|_{\underline{\mathsf{a}} = \mathcal{E}_A(a)} \cdot \mathcal{M}(\underline{\mathsf{a}}, \mathcal{E}_A(\tau_m)),\end{aligned}$$

where

$$\begin{aligned}\mathcal{V}_c(\mathsf{M}') &= \{\mathsf{c}_a \mid a \in Act\} \cup \mathcal{V}_c(\mathsf{M}), \\ \mathcal{V}_a(\mathsf{M}') &= \mathcal{V}_a(\mathsf{M}), \\ \mathcal{V}_s(\mathsf{M}') &= \mathcal{V}_s(\mathsf{M}),\end{aligned}$$

then $\mathsf{M}' \triangleright$ **hide** A **in** $\mathcal{S}$.

In Th. 25, we use the notation $\mathsf{M}|_{\underline{\mathsf{a}} = \mathcal{E}_A(a)}$ to denote the restriction of M to action a. Note that we add a new choice variable c_a for each synchronising action of type (m). This is because several (previously distinct) probability distributions may now be labelled with the same action (τ_i), and hence there is a non-deterministic choice between them.

6 MTBDD based analysis

The preceding sections have shown how to encode SPTSs — and in particular MTSs and CPSs — in terms of MTBDDs, and how to construct them symbolically, on the level of MTBDDs, using parallel composition and abstraction. In this section, we describe the MTBDD algorithms for model checking the logic PCTL against a concurrent probabilistic system (CPS) and calculating the steady-state distribution vector of a Markovian transition system (MTS). In the remainder of this section, for a given MTBDD M representing a stochastic and probabilistic transition system $\mathcal{S} = (S, \bar{s}, Act, \rightarrow, \mathrm{AP}, L)$, we suppose that M has choice variables $\mathsf{c}_1, \ldots, \mathsf{c}_{n_c}$, action variables $\mathsf{a}_1, \ldots, \mathsf{a}_{n_a}$ and source and target variables $\mathsf{s}_1, \mathsf{t}_1, \ldots, \mathsf{s}_{n_s}, \mathsf{t}_{n_s}$.

6.1 PCTL model checking

Given a CPS $\mathcal{S} = (S, \bar{s}, Act, \rightarrow, \mathrm{AP}, L)$, an MTBDD M representing $\mathcal{S}$ and a PCTL formula ϕ, we calculate the set of states of $\mathcal{S}$ which satisfy ϕ as a BDD as follows: we construct the parse tree for the formula ϕ and then calculate the set of states that satisfy the subformulae ψ of ϕ recursively. The cases when

```
algorithm IterativeOptimize(A, b, ε, ∼)
    res := b
    done := false
    while (done = false)
        tmp := res{s_1 ← t_1, ..., s_n ← t_n}
        tmp := MVmult(A, tmp)
        if (∼ ∈ {<, ≤})
            tmp := Abstract(tmp, {c_1, ... c_{n_c}, a_1, ... a_{n_a}}, max)
        else
            tmp := Abstract(tmp, {c_1, ... c_{n_c}, a_1, ... a_{n_a}}, min)
        endif
        tmp := Apply(b, tmp, +)
        if (MaxAbs(Apply(res, tmp, −)) < ε)
            done := true
        endif
        res := tmp
    endwhile
    return res
```

Fig. 4. MTBDD iterative algorithm for linear optimization problems.

ψ is not an until formula are the same as for CTL [46]. In the case where $\psi = \mathcal{P}_{\sim q}[\psi_1 \ \mathcal{U} \ \psi_2]$, model checking reduces to solving a linear optimization problem, as described in Sec. 3.2.2. This problem can be viewed as the matrix inequality $A\mathbf{x} \leq \mathbf{b}$ or $A\mathbf{x} \geq \mathbf{b}$ depending on whether $\sim \in \{\geq, >\}$ or $\sim \in \{<, \leq\}$. Direct methods such as Simplex have been shown to be unsuitable for MTBDD implementation [47] for the following reasons: firstly, methods which rely on access to individual rows, columns or elements of a matrix are not well suited to MTBDDs. Secondly, the change in the structure of A at every iteration of the algorithm leads to a loss in regularity. Fortunately, the optimization problems we consider can be solved iteratively.

Supposing we have already calculated the BDDs which represent the sets of states that satisfy ψ_1 and ψ_2 respectively, we now explain how to construct the MTBDDs representing A and $\mathbf{b}$. First, we compute the BDDs $\mathsf{b}_{\mathrm{yes}}$, b_{no} and $\mathsf{b}_{?}$, representing the sets of states S^{yes}, S^{no} and $S^{?}$. This is straightforward: the precomputation algorithms mentioned in Sec. 3.2.2 correspond to fixpoint algorithms based on standard BDD operations. Then, to compute the MTBDD representing A, we filter out the states which we do not need to consider (those states not in $S^{?}$):

$$\mathsf{A} = \textsc{Apply}(\mathsf{M}, \mathsf{b}_{?}, \times).$$

The vector $\mathbf{b}$ is represented by the BDD $\mathsf{b}_{\mathrm{yes}}$.

The iterative method for calculating the solution of $A\mathbf{x} \leq \mathbf{b}$ or $A\mathbf{x} \geq \mathbf{b}$ is given in Fig. 4, where ε represents the termination criterion: when the difference between the old and new iteration vectors is less than the bound ε we stop iterating. Note that we need to include $\sim$ to determine whether we are considering the case $A\mathbf{x} \leq \mathbf{b}$ or $A\mathbf{x} \geq \mathbf{b}$.

Once the MTBDD res representing the solution has been obtained, the BDD representing the set of states satisfying ψ is given by $\textsc{Threshold}(\mathsf{res}, \sim, q)$.

In the special case where $\mathcal{S}$ is a DTMC, model checking reduces to the simpler problem of solving a system of linear equations (matrix equation $A\mathbf{x} = \mathbf{b}$). We consider this problem in Sec. 6.3.

6.2 Steady-State Computation

As mentioned in Sec. 3.3, calculating the steady-state vector of an MTS reduces to transforming it to a CTMC and solving a system of linear equations, given by $\mathbf{x} \cdot Q = \mathbf{0}$, where Q denotes the infinitesimal generator matrix of the CTMC.

Given an MTS $\mathcal{S} = (S, \bar{s}, Act, \rightarrow, \mathrm{AP}, L)$ and an MTBDD M representing $\mathcal{S}$, we compute Q, the MTBDD representing the matrix Q, by performing the following steps. First we abstract the actions from M (A_i and A_m are BDDs encoding the sets Act_i and Act_m, respectively):

$$\begin{aligned}\mathsf{R} = \textsc{Apply}(&\textsc{Abstract}(\mathsf{M} \cdot \mathsf{A}_i, \{\mathsf{a}_1, \ldots, \mathsf{a}_{n_a}\}, \vee),\\ &\textsc{Abstract}(\mathsf{M} \cdot \mathsf{A}_m, \{\mathsf{a}_1, \ldots, \mathsf{a}_{n_a}\}, +), +).\end{aligned}$$

Next we remove immediate actions by an elimination procedure as described in Sec. 3.3. Location and elimination of immediate internal transitions, including the resolving of non-determinism by probabilities, can be realised by elementary MTBDD operations.

We then compute the row sums of M and use these to construct Q

$$\begin{aligned}&\mathsf{row\_sums} := \textsc{Abstract}(\mathsf{R}, \{\mathsf{t}_1, \ldots, \mathsf{t}_{n_s}\}, +)\\ &\mathsf{Q} := \textsc{Apply}(\mathsf{R}, \textsc{Apply}(\mathsf{row\_sums}, \mathsf{Id}, \times), -)\end{aligned}$$

where $\textsc{Apply}(\mathsf{row\_sums}, \mathsf{Id}, \times)$ turns the vector $\mathsf{row\_sums}$ into a diagonal matrix of the same dimension. Finally, we solve the linear system of equations $\mathbf{x} \cdot Q = \mathbf{0}$ by setting $\mathsf{Q} = \mathsf{A}$, $\mathsf{b} = \textsc{Constant}(0)$ and using the methods described in Sec. 6.3.

```
algorithm IterativeSolve(M, b', x_0, ε)
    res := x_0
    done := false
    while (done = false)
        tmp := res{s_1 ← t_1, ..., s_n ← t_n}
        tmp := MVmult(M, tmp)
        tmp := Apply(b', tmp, +)
        if (MaxAbs(Apply(res, tmp, −)) < ε)
            done := true
        endif
        res := tmp
    endwhile
    return res
```

Fig. 5. MTBDD iterative algorithm for the solution of linear equation systems.

6.3 Solving linear systems of equations

We now consider the general problem of solving a linear system of equations. We tackle the case $A \cdot \mathbf{x} = \mathbf{b}$, but the methods translate easily to the case $\mathbf{x} \cdot A = \mathbf{b}$. Suppose A is represented as an MTBDD A with variables $(\mathsf{s}_1, \mathsf{t}_1, \ldots, \mathsf{s}_{n_s}, \mathsf{t}_{n_s})$ and $\mathbf{b}$ as the MTBDD b with variables $(\mathsf{s}_1, \ldots, \mathsf{s}_{n_s})$. Direct methods such as Gaussian elimination have proved to be unsuitable for MTBDD implementation [15] due to the difficulty in accessing individual rows or columns and the loss in regularity caused by the change in structure of A at every elimination step. We therefore consider *iterative methods* where an iteration matrix M is derived from the matrix A and the matrix M remains unmodified during iteration. Each iteration takes the form $\mathbf{x}^{(k+1)} = M \cdot \mathbf{x}^{(k)} + \mathbf{b}'$ where $\mathbf{b}'$ is some modification of the vector $\mathbf{b}$. Further requirements are an initial approximation $\mathbf{x}_0$ and a termination criterion: when the difference between the old and new iteration vectors is less than some bound ε we stop iterating. This general iterative solution algorithm is given in Fig. 5.

In steady-state calculations, the vector $\mathbf{b}$ is zero and the initial approximation is usually that all states are equally probable, that is, x_0 is the MTBDD consisting of a single terminal node labelled with $1/2^m$. In the case of PCTL model checking DTMCs, both $\mathbf{b}$ and the initial vector are given by $\mathsf{b}_{\mathrm{yes}}$.

Power method: The power method can be written in the following matrix format:

$$\mathbf{x}^{(k)} = \left(I - \frac{A}{q}\right) \cdot \mathbf{x}^{(k-1)} + \frac{\mathbf{b}}{q}$$

where I is the identity matrix of the appropriate size and q is a scalar scaling factor. If A is the (negative of the) generator matrix of a CTMC, then choosing $q > \max_j |A_{jj}|$ ensures that the iteration matrix $I - A/q$ is stochastic.

The MTBDD M is therefore given by

$$\mathsf{M} := \textsc{Apply}(\mathsf{Id}(\mathsf{s}_1, \mathsf{t}_1, \ldots, \mathsf{s}_{n_s}, \mathsf{t}_{n_s}), \textsc{Apply}(\mathsf{A}, \textsc{Constant}(q), \div), -)$$

and $\mathsf{b}' := \textsc{Apply}(\mathsf{b}, \textsc{Constant}(q), \div)$.

Jacobi method: the Jacobi method has the following matrix format:

$$\mathbf{x}^{(k)} = D^{-1} \cdot (D - A) \cdot \mathbf{x}^{(k-1)} + D^{-1} \cdot \mathbf{b}.$$

where D represents the diagonal elements of A.

We can obtain the MTBDD D representing D through a pointwise multiplication with the identity matrix. We store the inverse of the diagonal in a vector, represented by MTBDD d. This is because multiplication of a diagonal matrix, such as D^{-1}, with any other matrix (or vector) can be reduced to pointwise multiplication with a vector containing the diagonal. d is obtained by abstracting the column variables from D and inverting:

$$\mathsf{D} := \textsc{Apply}(\mathsf{A}, \mathsf{Id}, \times)$$
$$\mathsf{d} := \textsc{Apply}(\textsc{Constant}(1), \textsc{Abstract}(\mathsf{D}, \{\mathsf{t}_1, \ldots, \mathsf{t}_{n_s}\}, +), \div)$$

We can then compute the matrix M and vector b' as follows:

$$\mathsf{M} := \textsc{Apply}(\mathsf{d}, \textsc{Apply}(\mathsf{D}, \mathsf{A}, -), \times)$$
$$\mathsf{b}' := \textsc{Apply}(\mathsf{d}, \mathsf{b}, \times).$$

Gauss-Seidel and Successive Overrelaxation: The Gauss Seidel scheme has the following matrix format:

$$\mathbf{x}^{(k)} = U^{-1} \cdot (U - A) \cdot \mathbf{x}^{(k-1)} + U^{-1} \cdot \mathbf{b}.$$

where U represents the upper triangular part of A (including the diagonal). A recursive MTBDD-based algorithm for the inversion of triangular matrices can easily be derived. The inversion causes a lot of fill-in and is counter–productive in the case of sparse matrices. However, in the MTBDD setting, the fill in can be tolerated as long as the regularity is not lost. In certain cases this method might prove rather effective, but in general the regularity is lost. A symbolic version of the Successive Overrelaxation method (SOR) raises essentially the same issues as Gauss–Seidel.

7 Tools and case studies

7.1 PRISM

PRISM [48] is a probabilistic symbolic model checker being developed at the University of Birmingham. It supports DTMCs, CTMCs and CPSs. Model checking algorithms are implemented in BDDs and MTBDDs. The tool is written in a combination of Java and C++. It uses CUDD [49], a publicly available BDD/MTBDD library developed at the University of Colorado at Boulder.

The main features of PRISM are:

(1) Model construction. PRISM builds DTMC, CTMC and CPS models by parsing specifications written in a custom system description language. The language is a probabilistic variant of the Reactive Modules language of [50]. We chose to use this language as input to the tool because it allows a direct (and efficient) translation into MTBDDs, and leads to structured, and hence small, MTBDDs.
(2) Reachability analysis. The tool computes the set of reachable states in the model, given an initial state. Unreachable states are then removed from the model.
(3) Model export. Once built, the transition matrix of the model can be exported for visualisation or use in other tools.
(4) PCTL model checking. Properties in the temporal logic PCTL can be parsed and then verified against constructed DTMC and CPS models (either with or without fairness). Simple PCTL operators (AND, OR, etc.) are model checked using BDDs. Model checking of the PCTL until operator reduces to solving either a linear system of equations (in the case of DTMCs), or a linear optimisation problem (in the case of CPSs). Both are solved using iterative methods. In the former case, the user can choose between the Power and Jacobi methods.
(5) Precomputation algorithms. The tool also includes BDD fixed point algorithms which can precompute partial results for the PCTL until operator. More specifically, they check qualitative properties, i.e. those where the probability is 0 or 1. These BDD methods are much more efficient than the MTBDD algorithms for quantitative reasoning. As mentioned in Sec. 3.2, using this precomputation often reduces the work to be done by the MTBDD algorithm and sometimes replaces it altogether.
(6) Steady-state computation. In the case of CTMCs, the steady-state probabilities for each state are computed. Again, this is performed by iteratively solving a linear system of equations and the user can choose between the Power and Jacobi methods.

(7) Graphical user interface. All the features described above are accessible via the PRISM Java-based graphical user interface.

More information about the tool can be found on the PRISM web page at URL www.cs.bham.ac.uk/~dxp/prism.

7.2 IM-CAT

IM-CAT is a tool for the construction, manipulation and analysis of MTSs. It has been developed at the University of Erlangen-Nürnberg and uses MTBDDs as its central underlying data structure [51,52]. IM-CAT is written in C++ and (similarly to PRISM) is built on top of the library CUDD [49].

The main features of IM-CAT are:

(1) Reading of elementary MTSs from file in a simple format as generated by the stochastic process algebra tool TIPPtool [53] and generating their MTBDD representation. Actually, the Markovian transitions of an MTS are represented by an MTBDD and immediate transitions by a separate BDD, which later is turned into an MTBDD if non-deterministic choice is resolved by probabilities as explained in Sec. 3.3.
(2) Parallel composition of two MTSs using their MTBDD representations, according to the scheme described in Sec. 5.2.1, case (6). The user needs to specify the names of the two partner MTSs, the set of synchronising actions, and the name of the resulting MTS.
(3) Reachability analysis. The tool computes the set of states reachable from the initial state. Unreachable states are then removed from the model.
(4) Hiding of actions at the MTBDD level. At any time, it is possible to hide some user-defined actions, i.e. to turn visible actions into the special invisible actions τ_i and τ_m. This feature is useful if one wishes to hide actions which are no longer needed for synchronisation. Note that the hiding of immediate actions may actually render certain states unstable, which means that they can be eliminated.
(5) Elimination of unstable states. The elimination can be triggered manually at any stage, for instance after parallel composition and subsequent hiding have been performed, and before the current MTS is composed further with other components. Elimination is invoked automatically as a mandatory first step of numerical analysis (since numerical analysis requires a CTMC). During elimination, non-determinism between several internal immediate transitions is resolved by assigning probabilities to the non-deterministic alternatives [13]. These probabilities may be as-

[13] Note that, by assigning probabilities to the internal immediate transitions, the BDD representing the immediate transitions becomes an MTBDD.

signed automatically, in which case the default option of equiprobability between all non-deterministic alternatives is assumed, or they may be assigned manually by the user in an interactive fashion.

(6) Numerical analysis. IM-CAT contains iterative algorithms for calculating the steady-state probability distribution (power method, Jacobi, different variants of Gauss-Seidel, and the projection method BiCGStab [54]). All these schemes (even Gauss-Seidel) are based on vector-matrix multiplication, which can be realised conveniently on MTBDDs.

(7) Utilities such as the output of information about the currently stored MTBDDs, the generation of graphical output for visualising the MTBDDs, the writing of transition systems to file in the original format, etc.

The parallel composition feature of IM-CAT, together with reachability analysis, hiding and elimination of unstable states, makes it possible to generate very large MTSs from small components, thereby exploiting the storage-efficiency of the MTBDD approach. Up to now, the user interacts with IM-CAT via the command line, which is somewhat tedious and requires some experience, but we are currently developing an easy-to-use Java-based graphical user interface.

7.3 Case studies

In all statistics given below for both IM-CAT and PRISM, the termination criterion (ε in Fig. 4 and Fig. 5) of 10^{-6} was used. Furthermore, all experiments using IM-CAT were carried out on a 300 MHz SUN 5/10 workstation, equipped with 1GB of main memory, while all PRISM experiments were run on a 270MHz SUN Ultra 10 workstation with 384 MB memory. All times are given in seconds.

7.3.1 A cyclic server polling system

In this section, we consider a cyclic server polling system consisting of d stations and a server, modelled as a generalized stochastic Petri net (GSPN) [14]. The example is taken from [55], where a detailed explanation can be found. For $d = 2$, i.e. a two-station polling system, the GSPN model is depicted in Fig. 6. For a d-station polling system, the Petri net is extended in the obvious way. Place $idle_i$ represents the condition that station i is idle, and place $busy_i$ represents the condition that station i has generated a job. The server visits the stations in a cyclic fashion. After polling station i (place $poll_i$), the server serves station i (place $serve_i$), and then proceeds to poll the next station. The

[14] We refer to [37,38] for details on the semantics of GSPNs, and their mapping on MTSs.

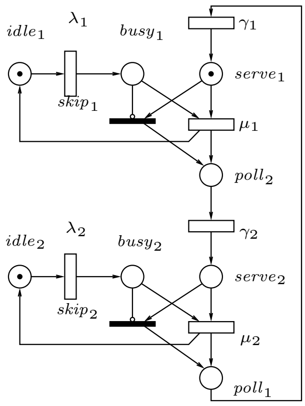


Fig. 6. The cyclic server polling system with 2 stations [55].

d	reachable states	transitions	MTBDD size			
			compositional			monolithic
			before reachability	after reachability	without actions	
3	36	84	169	203	112	351
5	240	800	387	563	271	1,888
7	1,344	5,824	624	1,087	482	9,056
10	15,360	89,600	1,163	2,459	921	69,580
15	737,280	6.144e+6	2,191	6,317	1,942	–
20	3.145e+07	3.407e+08	3,704	13,135	3,346	–

Fig. 7. Statistics for the polling system (obtained with IM-CAT).

times for generating a message, for polling a station and for serving a job are all distributed exponentially with parameters λ_i, γ_i and μ_i, respectively. If the server finds station i idle, the service time is zero. This is modelled by the immediate transition $skip_i$ and the inhibitor arc from place $busy_i$ to transition $skip_i$. In this study we consider polling systems with $d = 3$, 5, 7 and 10 stations (like in [55]). In addition, we consider the cases $d = 15$ and $d = 20$. The stations are assumed to be symmetric, i.e. λ_i, γ_i and μ_i have the same numerical value for all $i \leq d$. We set $\gamma_i = 200$, $\mu_i = 1$ and $\lambda_i = \mu_i/d$.

The MTBDD representation of the overall polling model was constructed compositionally from $d+1$ elementary transition systems (which were generated by TIPPTOOL), one for the server and one for each station, which were encoded as individual MTBDDs Server and Station$_i$ ($i = 1, \ldots, d$). The MTBDD for the overall system was computed by applying MTBDD-based parallel composition according to the following scheme:

$$(\mathsf{Station}_1 ||| \ldots ||| \mathsf{Station}_d) |[S]| \mathsf{Server}.$$

In Fig. 7, important statistics for the polling system are given for different values of d. The 2nd and 3rd columns of the table contains the number of reachable states and reachable transitions respectively. The remaining columns give the number of MTBDD vertices. Column 4 contains the number of vertices of the MTBDD that was generated compositionally from the component MTBDDs. The MTBDD generated in this way represents all transitions which are possible within the product of the $d + 1$ components' state spaces. As can be observed from the 5th column of the table, determining the set of reachable states and "deleting" the transitions which originate in unreachable states considerably increases the size of the MTBDD. The 6th column of the table contains the size of the MTBDD which one obtains by removing all action-labelling information from the MTBDD of column 5. This MTBDD, which now only depends on Boolean variables for the source and target state of transitions, but not on Boolean variables encoding action labels, represents the rate matrix of the (unlabelled) CTMC.

The last column of Fig. 7 shows the number of MTBDD vertices which one would obtain by taking the monolithic MTS of the overall model and directly encoding it as an MTBDD. Clearly, this method cannot be recommended. Apart from the fact that the MTS of the overall model may not be available due to its size, the growth of the MTBDD sizes is prohibitive [15] . As expected from Th. 23, the MTBDD sizes in column 4 (and column 6) grow linearly, whereas the ones in column 7 grow exponentially. Looking at the last row of the table (the case $d = 20$) one can observe that, even for an extremely large state space, the MTBDD representation can be very compact if it is constructed in a compositional fashion.

In Fig. 8 we give the amount of memory required to store the matrix representing the underlying CTMC of the polling system (the system with actions removed), when using MTBDDs (for both the compositional and monolithic approach) and sparse matrices. As the statistics demonstrate, using MTBDDs together with the compositional approach gives a very space-efficient encoding of the CTMC, especially for large state spaces. On the other hand, the monolithic MTBDD approach is less efficient than using sparse matrices.

In Fig. 9 we give statistics for computing the steady-state solution of the cyclic polling system using PRISM (we obtained almost identical results using IMCAT). We include results for both the power and Jacobi method. We note that when using either iteration method the time taken to construct the MTBDD representing the iteration matrix is negligible. However, as shown in Fig. 9, the size of the MTBDD representing the iteration matrix is larger than the MTBDD representing the system. In both methods this loss of structure arises

[15] The "–" entries in the last column indicate that the number could not be determined because the monolithic MTS of the overall model was never explicitly constructed due to excessive runtime and memory requirements.

d	memory (KB)		
	MTBDD compositional	MTBDD monolithic	sparse matrix
3	2.19	6.86	1.13
5	5.29	36.9	10.3
7	9.41	176	73.5
10	18.0	1,359	1,110
15	37.9	–	74,880
20	65.4	–	4.11e+06

Fig. 8. Storage requirements for the polling system.

d	power method				Jacobi method			
		MTBDD		sparse		MTBDD		sparse
	iters	iter. matrix (vertices)	time per iter. (s)	time per iter. (s)	iters	iter. matrix (vertices)	time per iter. (s)	time per iter. (s)
3	1,624	151	0.004	0.000006	75	187	0.003	0.000006
5	2,300	392	0.020	0.00007	137	628	0.014	0.00007
7	2,848	761	0.158	0.0008	196	1,453	0.101	0.0006
10	3,495	1,632	13.89	0.014	280	3,698	4.51	0.014

Fig. 9. Statistics for the steady-state analysis of the polling system (obtained with PRISM).

through the inclusion of the diagonals in the construction of the infinitesimal generator matrix Q (see Sec. 6.2). Furthermore, in the case of the Jacobi method there is a further loss of structure when we multiply by the inverse of the diagonal elements of Q (see Sec. 6.3). For comparison, we include the time per iteration for a sparse matrix implementation included in PRISM as well as when using MTBDDs. The results demonstrate that numerical analysis using MTBDDs is much slower than using a sparse implementation.

7.3.2 A tandem queueing network with blocking

As a second example we consider a tandem queueing network with blocking taken from [56] and used again in [53]. It consists of a M/Cox$_2$/1-queue sequentially composed with a –/M/1-queue, see Fig. 10. Each of the two queueing stations has a finite capacity of c jobs, $c > 0$. Jobs arrive at the first queueing station according to a Poisson stream with rate λ. The service time of the first station has a Coxian distribution with two exponential phases. The first phase has rate parameter μ_1. After completion of the first phase, with probability $b_1 = 1 - a_1$ the service is finished, and with probability a_1 the job moves to the second phase whose rate parameter is μ_2. Once served, jobs leave the first station, and are queued in the second station whose service time is exponential with rate κ. In case the second queueing station is fully occupied, i.e. its server is busy and its queue is full, the first station is said to be blocked.

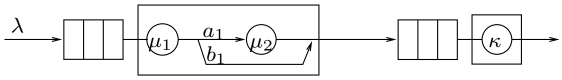


Fig. 10. A simple tandem network with blocking [56]

c	reachable states	transitions	MTBDD size	
			compositional	monolithic
7	128	378	148	723
15	512	1,650	197	1,575
127	32,768	113,538	341	11,480
1,023	2,097,152	7.33082e+06	485	–
16,383	5.36871e+08	1.8789e+09	677	–

Fig. 11. Statistics for the tandem queueing system (obtained with IM-CAT).

Note that, in this situation, the second phase of the first server is blocked and the first server can only pass a job from the first phase to the second phase (which happens with rate $\mu_1 \cdot a_1$), but the "bypass" of the second phase is also blocked.

For the experiments we use the following values for the parameters of the queue: $\lambda = 3$, $\mu_1 = \mu_2 = 2$, $\kappa = 4$, and $a_1 = 0.1$. Fig. 11 contains the number of reachable states and the number of transitions of the polling system for different values of c, and the number of vertices of the corresponding MTBDDs. The MTBDDs in column 4 were constructed as follows: the elementary MTSs for the Markovian station and for the Coxian station were encoded as two MTBDDs and subsequently composed in parallel, using MTBDD-based parallel composition. The MTBDDs in column 5 were obtained by directly encoding the MTS of the overall queueing system as an MTBDD.

From the numbers given in Fig. 11 it can be observed that the MTBDD sizes for the compositional approach are surprisingly small, whereas the MTBDD sizes for the monolithic approach become prohibitive. This result was to be expected, since it is a direct consequence of Th. 23. It supports the finding that MTBDDs are only beneficial if they are used in a structured (i.e. in our case compositional) fashion. Note that in the tandem queueing network example all states from the product state space of the two components (the Coxian station and the Markovian station) are reachable. Therefore, (MTBDD-based) reachability is not necessary in this case.

Fig. 12 contains some statistics collected when computing the steady-state solution of the tandem model for varying queue capacity c. Columns 2 to 4 contain the number of iterations until convergence, the number of MTBDD vertices of the iteration matrix, and the mean time per iteration. Columns 6 to 8 contain the corresponding figures for the Jacobi method. As a comparison, the results obtained for the same models using TIPPtool's sparse imple-

c	power method				Jacobi method			
	iters	MTBDD		sparse	iters	MTBDD		sparse (GS)
		iter. matrix (vertices)	time per iter. (s)	time per iter. (s)		iter. matrix (vertices)	time per iter. (s)	time per iter. (s)
7	180	196	0.011	0.000108	190	327	0.009	0.000151
15	380	249	0.050	0.000377	250	415	0.045	0.000481
127	1,540	399	2.068	0.04926	1,110	655	2.435	0.0500

Fig. 12. Statistics for the steady-state analysis of the tandem system (obtained with IM-CAT and TIPPTOOL).

```
while true do
    prefer_i := ⊥
    flip_i := 0.5 : heads + 0.5 : tails
    if flip_i = heads
        inc(counter)
    else
        dec(counter)
    end if
    if counter ≥ K · N
        prefer_i := 0
        return prefer_i
    else if counter ≤ −K · N
        prefer_i := 1
        return prefer_i
    end if
end while
```

Fig. 13. Shared coin protocol for process i.

mentation are given in columns 5 and 9 (note that since TIPPTOOL does not implement Jacobi, column 9 gives results for sparse Gauss-Seidel instead). In summary, as with the previous example, the statistics show that MTBDD-based numerical analysis is far slower than an efficient sparse implementation.

7.3.3 Shared coin protocol

As an example of a concurrent probabilistic system (CPS), we consider a shared coin protocol which is part of the distributed randomised consensus algorithm of [57]. The shared coin protocol implements a collective random walk, parameterised by the number of processes N and the constant $K > 1$ (independent of N). The processes access a global shared counter, initially 0. On entering the protocol, a process flips a coin, and, depending on the outcome, increments or decrements the shared counter. Having flipped the coin, the process then reads the counter. If the counter has a value greater than or equal to $K \cdot N$ it chooses 0 as its preferred value, and if the counter has a value less than or equal to $-K \cdot N$ it chooses 1. Otherwise, the process flips the coin again, and continues doing so until it observes that the counter has passed one of the barriers. The shared coin protocol for process i is given in Fig. 13.

In Fig. 14 the statistics for the shared coin protocol are given for different values of N and K, where the construction time includes the time taken to compute the reachable states. The MTBDDs were constructed as follows. The elementary CPSs for the counter and for the processes were first encoded as MTBDDs, noting that starting from the initial state (when the counter is 0) the counter will never be greater than $(K+1)\cdot N$ (less than $-(K+1)\cdot N$), and hence we only need to construct the (finite) CPSs of the counter and processes for the value of counter within these bounds. To construct the full system, these MTBDDs were composed in parallel, using MTBDD-based parallel composition.

From the results given in Fig. 14 it can be observed that the MTBDD sizes are surprisingly small and increasing K has little effect on the size of the MTBDD compared to the increase in the number of states. Furthermore, using MTBDDs the models can be constructed quickly. This is a consequence of only simple MTBDD operations being involved in constructing the parallel composition of processes.

We now consider the model checking results of this example using PRISM. The properties of the shared coin protocol required by [57] are listed below.

C1 For each fair execution of the shared coin flipping protocol that starts with a reachable state of the shared coin flipping protocol, *with probability 1* all processes that enter the shared coin flipping protocol will eventually leave.

C2 For each fair execution of the shared coin flipping protocol, and each value $v \in \{0,1\}$, the probability that all processes that enter the shared coin flipping protocol will eventually leave agreeing on the value v is at least $(K-1)/2K$.

Both properties **C1** and **C2** are expressible in PCTL. We let $\varphi_i(0)$ and $\varphi_i(1)$ denote the atomic propositions true in the states which satisfy $\mathit{prefer}_i = 0$ and $\mathit{prefer}_i = 1$ respectively. Then **C1** corresponds to the PCTL property:

$$\mathcal{P}_{\geq 1}[\,\mathtt{true}\,\mathcal{U}\,((\varphi_1(0) \vee \varphi_1(1)) \wedge \ldots \wedge (\varphi_N(0) \vee \varphi_N(1)))\,]$$

and **C2** can be represented by the PCTL properties:

$$\mathit{init} \Rightarrow \mathcal{P}_{\geq (K-1)/2K}[\,\mathtt{true}\,\mathcal{U}\,(\varphi_1(v) \wedge \ldots \wedge \varphi_N(v))\,]$$

for $v = 0, 1$ where *init* is the atomic proposition true only in the initial state. **C1** is a probability-1 property, and therefore admits efficient *qualitative* [58] probabilistic analysis, using only reachability based analysis [32], as already mentioned in Sec. 3.2.2. On the other hand, **C2** is *quantitative*, and requires calculating the minimum probability that, starting from the *initial state* of

N	K	States	Transitions	MTBDD size	Construction time (s):
4	16	166,016	692000	1,327	1.438
4	32	329,856	1,376,032	1,338	2.746
8	8	2.222e+8	1.692e+9	5,587	7.500
8	16	4.372e+8	3.332e+9	5,404	13.173
10	4	5.180e+9	4.812e+10	8,713	10.032
10	8	1.002e+10	9.319e+10	8,718	16.319

Fig. 14. Statistics for the shared coin protocol (obtained with PRISM).

N	K	**C1**		**C2**	
		iters.	time per iter. (s)	iters.	time per iter. (s)
4	16	404	0.007	59,253	1.911
4	32	788	0.008	181,791	1.170
8	8	420	0.062	59,253	8.365
8	16	804	0.074	181,791	5.969
10	4	284	0.256	26,799	8.228
10	8	524	0.266	85,641	11.51

Fig. 15. Model checking results for the coin flipping protocol.

the shared coin flipping protocol, *all* processes leave the protocol agreeing on a given value. Note that both properties involve *fairness constraints* and therefore we check these properties only against *fair adversaries* by using the satisfaction relation $\models_{fair}$.

A summary of model checking statistics obtained from the shared coin-flipping protocol using PRISM is included in Fig. 15. The results from the model checking show that **C1** and **C2** hold for all the instances of N and K given. The number of iterations for **C1** corresponds to the number of iterations of the probability-1 precomputation step, whereas for **C2** the number of iterations corresponds to those of the iterative method given in Fig. 4.

Fig. 15 clearly shows that the model checking of qualitative properties ("with probability 1" properties) when using MTBDDs is very fast. On the other hand, when one needs to calculate probabilities, as with the previous examples, MTBDDs become less efficient, but are able to handle very large state spaces.

8 Lessons learned

It is well-known that the effectiveness of BDD-based methods depends on heuristics tailored to the particular application area at hand — even though this fact is often inadequately reflected in the literature on BDDs. This section summarises some lessons learned from our experience with applying MTBDDs to SPTSs in PRISM and IM-CAT, emphasising the importance of good heuristics.

8.1 Compositional versus monolithic encoding

Our most important observation is that it is unwise to simply encode a given, monolithic transition system as an MTBDD. Instead, in order to achieve compact symbolic representation, the construction should proceed compositionally, starting from the MTBDDs for the lowest level components, and using MTBDD-based parallel composition (as described by Th. 16) in every construction step. The superiority of the compositional approach is due to the fact that it exploits structure and regularity and thus automatically yields good state encodings (see Sec. 8.2).

8.2 State encoding and state variable ordering

Most heuristics are concerned with the encoding of state identifiers as bit vectors and the ordering of the Boolean variables, two issues which are closely intertwined. To find the optimal ordering is an NP-complete problem [59], and hence one has to resort to heuristics. It is generally recommended (and we also follow this recommendation) to use an *interleaved ordering* of the Boolean vectors $(\mathsf{s}_1, \dots, \mathsf{s}_n)$ and $(\mathsf{t}_1, \dots, \mathsf{t}_n)$ encoding source and target states, i.e. to use $(\mathsf{s}_1, \mathsf{t}_1, \dots, \mathsf{s}_n, \mathsf{t}_n)$ as the variable ordering for state sets [45]. For the symbolic representation of matrices, this encoding implies that everything that "happens" on the main diagonal, or on a diagonal of some submatrix represented by a cofactor (or obtained by restriction of some arbitrary bits of the encoding) profits from this encoding. In a compositional setting, the interleaved ordering makes it possible to exploit structural information of the high-level formalism in the encoding. Any kind of structural insight that is used in the encoding of states is reflected in the encoding of both row and column positions, and therefore exploited best with the interleaved ordering [56].

Our experience has also shown that even if a composition (via $|[Sync]|$) involves heavy synchronisation among the components (implying that a large portion of the composed state space is unreachable), it is usually not recommended to shorten the bit-vector by changing the encoding, because structure would be lost. It is wise to invest in an optimal encoding of the lowest level component state spaces, but to avoid modifying the encodings after composition. That is, we propose to optimise the component encodings, either by means of the exact algorithm [16] [60] or by means of adaptions of Rudell's sifting algorithm [61] or other heuristic methods for BDDs, e.g. [62,63].

8.3 Component variable ordering

Using the compositional approach gives us a choice of where the MTBDD variables representing the state sets of sub-components appear in the ordering (while keeping the interleaved ordering). Experiments have shown that such choices can have a considerable influence on the size of the MTBDD representing the overall system. Furthermore, these experiments have led us to the following two related heuristics concerning this ordering:

- If sub-components synchronise with each other, then the Boolean variables which represent their state sets should be placed close together in the ordering.
- If a sub-component interacts (synchronises) with many of the other sub-components of the system, then the Boolean variables which represent its state set should be placed towards the root of the ordering or closer to the root than those components which it synchronises with.

To illustrate the second of these heuristics we return to the cyclic server polling system introduced in Sec 7.3.1 which is defined by the following parallel composition of sub-components:

$$(\mathsf{Station}_1 \;|||\; \ldots \;|||\; \mathsf{Station}_d) \;|[S]|\; \mathsf{Server}.$$

In this example the server synchronises with all the other components (the stations). Thus, following the heuristics given above, placing the MTBDD variables representing the state set of the server closest to the root leads to a "good" variable ordering. On the other hand, placing these variables closest to the leaf vertices leads to a "bad" ordering. To see the advantage of using this heuristic, the MTBDD sizes of the cyclic polling system under each of these orderings are given in Fig. 16. The "bad" ordering obviously yields much larger MTBDDs than the "good" one. It is interesting to observe that, while the figures in the column "before reachability" are still close to the corresponding ones (only up to 1.25 times larger), the figures in the column "after reachability" are dramatically worse for the "bad" ordering (up to 724 times larger).

[16] Since the lowest level components are not likely to have large MTBDD representations, NP-completeness of the exact algorithm is not a problem in practice.

d	MTBDD size			
	before reachability		after reachability	
	“good” ordering	“bad” ordering	“good” ordering	“bad” ordering
3	169	163	203	200
5	387	386	563	623
7	624	658	1,087	1,734
10	1,163	1,281	2,459	10,598
15	2,191	2,641	6,317	297,942
20	3,704	4,632	13,135	9,507,435

Fig. 16. Statistics for the polling system (obtained with IM-CAT).

8.4 Action and choice variable ordering

The heuristics from Sec. 8.3 can be interpreted at the MTBDD level: place variables that influence the values of each other close together in the ordering, and place variables that influence many other variables towards the root of the MTBDD.

We now consider where the MTBDD variables representing both the non-deterministic choices and actions should go in the ordering. First, the value of the variables corresponding to the target states of the system clearly depends both on the non-deterministic choice made and which action is performed. Also, what actions can be performed depends on how the non-determinism is resolved. Therefore, the above interpretation of the heuristics justifies the ordering of the variables we introduced in Sec 5.1: we place the choice variables first in the ordering (closest to the root), followed by the action variables, and finally the variables which represent the state set of the system. This issue is considered in greater depth in [64].

8.5 Reachability analysis and bisimulation

In the examples given in this paper, we observed that the restriction to the reachable part of an SPTS typically increases the size of its MTBDD representation. We have made another, related observation, namely that established techniques for state space compression, such as lumping [65] or bisimulation [66,7], are often counterproductive in the MTBDD setting, i.e. the size of the symbolic representation grows although the number of states and transitions shrinks (in the non–stochastic case a similar observation has been made in [67]). One reason for this increase is that bisimulation on MTSs, CPSs, or SPTSs cumulates transitions by adding up the respective parameters. This implies that, while the minimised model may have far fewer transitions than the original one, the former involves more distinct parameters than the latter. Since these parameters are represented as MTBDD terminal vertices, the minimised model has more terminal vertices, and hence sharing of common subgraphs is reduced. Another reason is simply that the minimised transition system is less regular and therefore not so likely to have a compact symbolic representation. Together, these factors can outweigh the gain due to the reduction of the number of states and transitions to be encoded.

8.6 Space and time efficiency

We have shown that the MTBDD encoding of a transition system can be extremely compact if it is constructed in a compositional fashion, and that this compactness carries over to the symbolic representation of the iteration matrix. Unfortunately, the MTBDDs representing the iteration vectors tend to be significantly larger than the ones for the iteration matrix. By way of example, for the polling system ($d = 10$), the MTBDD for the iteration matrix has only 1,632 vertices, but the size of the MTBDD representing the solution vector grows to 20,000 vertices in only 30 iterations.

For vectors to be represented compactly by MTBDDs, the main requirement is a limited number of distinct elements. However, in general, when performing numerical analysis, either to calculate steady-state probabilities of an MTS or during PCTL model checking of a CPS, the iteration vector quickly acquires almost as many distinct values as there are states in the system under study. We have seen that in certain cases, where there is sufficient structure in the iteration vector, MTBDDs have been able to analyse much larger models than would be feasible with a sparse implementation (for example, this holds for the case of the shared coin protocol introduced in Sec 7.3.3 and certain examples presented in [68], where systems of up to 33 million states were analysed using MTBDDs). However, it is difficult to predict when these vectors will be represented compactly, as this depends both on the structure of the model and on the property being verified.

One of the most important lessons learned, as demonstrated through the case studies, is the time inefficiency of MTBDD-based numerical analysis [15]. This has to do with the nature of the MTBDD algorithms, where many recursive function calls must be made and many pointers followed in order to access individual vector or matrix elements (stored in the terminal vertices of the MTBDDs). Contrary to typical BDD algorithms, the use of a computed table (where intermediate results are cached) is of little value in a situation where the MTBDD representing the iteration vector contains (almost) as many different numerical entries as there are states. Such a situation spoils one of the main features of the BDD approach, namely that most recursions can terminate early since the result was found in the cache.

The fact that in numerical analysis the MTBDDs representing the iteration vectors become significantly larger than the MTBDD for the transition matrix means that little or no advantage is gained from using the often smaller MTBDD representing the transition matrix of the potential (before reachability) rather than the actual state space (after reachability).

To overcome this inefficiency while maintaining the advantages of MTBDDs (compact representation of systems), PRISM has been extended to include a *hybrid* approach [69], which uses an MTBDD representation for storing matrices and a conventional representation for probability vectors. More information will be available in [64].

9 Conclusion

In this paper we presented space-efficient symbolic representations for a general class of probabilistic, stochastic and non-deterministic models. We showed that the MTBDD data structure is very well suited to compactly represent huge transition systems, provided that models are constructed compositionally with insight into the structure of the system under investigation. We showed that all steps of model construction and analysis, including PCTL model checking and numerical analysis of stationary measures, can be performed on the MTBDD data structure. While the analysis of purely functional properties is very efficient, we pointed out that existing symbolic implementations of linear algebra operations, which are needed for numerical analysis, do not match the speed of state-of-the-art sparse implementations. Improving the speed of these operations is therefore a challenging topic for future research. Furthermore, it would be interesting to compare our experience with MTBDDs with related work from the area of quantitative model checking, such as the one reported in [70] where MTBDDs are used to store the transition relation of probabilistic transition systems.

References

[1] M. Puterman, Markov Decision Processes, Wiley, 1994.

[2] C. Courcoubetis, M. Yannakakis, The complexity of probabilistic verification, Journal of the ACM 42 (4) (1995) 857–907.

[3] R. Segala, Modelling and verification of randomized distributed real time systems, Ph.D. thesis, Massachusetts Institute of Technology (1995).

[4] R. Milner, Communication and Concurrency, Prentice Hall, 1989.

[5] T. Bolognesi, E. Brinksma, Introduction to the ISO specification language LOTOS, in: P. van Eijk, C. Vissers, M. Diaz (Eds.), The Formal Description Technique LOTOS, North-Holland, Amsterdam, 1989, pp. 23–73.

[6] N. Götz, U. Herzog, M. Rettelbach, Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras, in: Proc. of the 16th International Symposium on Computer Performance Modelling, Measurement and Evaluation, PERFORMANCE 1993, Tutorial, Vol. 729 of Lecture Notes in Computer Science, Springer, 1993, pp. 121–146.

[7] J. Hillston, A Compositional Approach to Performance Modelling, Cambridge University Press, 1996.

[8] H. Hermanns, U. Herzog, J.-P. Katoen, Process algebra for performance evaluation, Th. Comp. Sci. 274 (1-2) (2002) 43–87.

[9] P. Courtois, Decomposability, queueing and computer system applications, ACM monograph series, 1977.

[10] B. Plateau, On the synchronization structure of parallelism and synchronization models for distributed algorithms, in: Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, Austin, TX, 1985, pp. 147–154.

[11] P. Buchholz, A class of hierarchical queueing networks and their analysis, Queueing Systems 15 (1994) 59–80.

[12] H. Hermanns, J. Katoen, Automated compositional Markov chain generation for a plain-old telephony system, Science of Computer Programming 36 (1) (1999) 97–127.

[13] M. Siegle, Structured Markovian performance modelling with automatic symmetry exploitation, in: G. Haring, H. Wabnig (Eds.), Short Papers and Tool Descriptions Proc. of the 7th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation, Vienna, Austria, 1994, pp. 77–81.

[14] R. Bryant, Graph-based algorithms for boolean function manipulation, IEEE ToC C-35 (8) (1986) 677–691.

[15] R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, F. Somenzi, Algebraic Decision Diagrams and their Applications, in: ICCAD-93: Int. Conference on Computer-Aided Design, ACM/IEEE, Santa Clara, CA, 1993, pp. 188–191, also available in *Formal Methods in System Design*, 10(2/3), 1997.

[16] E. Clarke, M. Fujita, P. McGeer, K. McMillan, J. Yang, X. Zhao, Multi-Terminal Binary Decision Diagrams: an effiicient data structure for matrix representation, in: Proc. International Workshop on Logic Synthesis (IWLS'93), Tahoe City, 1993, pp. 6a:1–15, also available in *Formal Methods in System Design*, 10(2/3):149–169, 1997.

[17] H. Hansson, B. Jonsson, A framework for reasoning about time and reliability, in: Proc. of 10th IEEE Real-Time Systems Symposium, IEEE Computer Society Press, Orlando, Fl., 1990, pp. 102–111.

[18] C. Baier, B. Haverkort, H. Hermanns, J.-P. Katoen, On the logical characterisation of performability properties, in: ICALP, Vol. 1853 of Lecture Notes in Computer Science, Springer, 2000, pp. 780–792.

[19] H. Hermanns, U. Herzog, V. Mertsiotakis, Stochastic process algebras - between LOTOS and Markov chains, Computer Networks and ISDN (CNIS) 30 (9-10) (1998) 901–924.

[20] H. Hermanns, Interactive Markov chains, Ph.D. thesis, Universität Erlangen-Nürnberg (1998).

[21] M. Bernardo, R. Gorrieri, A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time, Theoretical Computer Science 202 (1998) 1–54.

[22] M. Bravetti, M. Bernardo, Compositional Asymmetric Cooperations for Process Algebras with Probabilities, Priorities, and Time, in: Proc. of the 1st Int. Workshop on Models for Time Critical Systems (MTCS 2000), Electronic Notes in Theoretical Computer Science 39(3), State College (PA), 2000.

[23] E.-R. Olderog, C. Hoare, Specification-oriented semantics for communicating processes, Acta Informatica 23 (1) (1986) 9–66.

[24] G. Plotkin, A structural approach to operational semantics, technical report, Computer Science Department FN-19, DAIMI, Aarhus University (September 1981).

[25] A. Bianco, L. de Alfaro, Model checking of probabilistic and nondeterministic systems, in: P. Thiagarajan (Ed.), Proc. Foundations of Software Technology and Theoretical Computer Science, Vol. 1026 of Lecture Notes in Computer Science, Springer-Verlag, 1995, pp. 499–513.

[26] C. Baier, M. Kwiatkowska, Model checking for a probabilistic branching time logic with fairness, Distributed Computing 11 (1998) 125–155.

[27] S. Hart, M. Sharir, A. Pnueli, Termination of probabilistic concurrent programs, ACM Transactions on Programming Languages and Systems 5 (1983) 356–380.

[28] L. de Alfaro, From fairness to chance, in: C. Baier, M. Huth, M. Kwiatkowska, M. Ryan (Eds.), Proc. First International Workshop on Probabilistic Methods in Verification (PROBMIV'98), Vol. 22 of Electronic Notes in Theoretical Computer Science, 1998.

[29] E. Clarke, E. Emerson, A. Sistla, Automatic verification of finite-state concurrent systems using temporal logics, in: 10th ACM Symposium on Principles of Programming Languages, 1983, pp. 24–26, the full version is available in *ACM Transactions on Programming Languages and Systems*, 1(2), 1986.

[30] C. Baier, On algorithmic verification methods for probabilistic systems, Habilitation thesis, Fakultät für Mathematik & Informatik, Universität Mannheim (1999).

[31] L. de Alfaro, Stochastic transition systems, in: D. Sangiorgi, R. de Simone (Eds.), Proc. CONCUR'98, Lecture Notes in Computer Science, Springer-Verlag, 1998, pp. 423–438.

[32] L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, R. Segala, Symbolic Model Checking for Probabilistic Processes using MTBDDs and the Kronecker Representation, in: S. Graf, M. Schwartzbach (Eds.), TACAS'2000, LNCS 1785, Berlin, 2000, pp. 395–410.

[33] G. Ciardo, R. Zijal, Well-defined stochastic Petri nets, in: MASCOTS, 1996, pp. 278–284.

[34] M. Qureshi, W. Sanders, A. van Moorsel, R. German, Algorithms for the generation of state-level representations of stochastic activity networks with general reward structures, IEEE Transactions on Software Engineering 22 (9) (1996) 603–614.

[35] D. Deavours, W. Sanders, An efficient well-specified check, in: P. Bucholz, M. Silva (Eds.), Proc. of the 8th Int. Workshop on Petri Nets and Performance Models (PNPM '99), Zaragoza, Spain, 1999, pp. 124–133.

[36] H. Hermanns, M. Siegle, Bisimulation algorithms for stochastic process algebras and their BDD-based implementation, in: J.-P. Katoen (Ed.), ARTS'99, 5th Int. AMAST Workshop on Real-Time and Probabilistic Systems, Vol. 1601 of Lecture Notes in Computer Science, Springer, 1999, pp. 144–264.

[37] M. Ajmone Marsan, G. Balbo, G. Conte, A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems, ACM Transactions on Computer Systems 2 (2) (1984) 93–122.

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

[39] A. Shiryaev, Probability, 2nd Edition, Vol. 95 of Graduate Texts in Mathematics, Springer, 1996.

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

[41] A. Jensen, Markoff chains as an aid in the study of Markoff processes, Skand. Aktuarietiedskr. 36 (1953) 87–91.

[42] W. Grassmann, Transient solutions in Markovian queues, European Journal of Operational Research 1 (1977) 396–402.

[43] E. Clarke, K. McMillan, X. Zhao, M. Fujita, J. Yang, Spectral transforms for large boolean functions with applications to technology mapping, in: 30th Design Automation Conference, ACM/IEEE, 1993, pp. 54–60.

[44] K. Brace, R. Rudell, R. Bryant, Efficient implementation of a BDD package, in: 27th ACM/IEEE Design Automation Conference, 1990, pp. 40–45.

[45] R. Enders, T. Filkorn, D. Taubner, Generating BDDs for symbolic model checking in CCS, Distributed Computing 6 (3) (1993) 155–164.

[46] K. McMillan, Symbolic Model Checking, Kluwer Academic Publishers, 1993.

[47] M. Kwiatkowska, G. Norman, D. Parker, R. Segala, Symbolic model checking of concurrent probabilistic systems using MTBDDs and Simplex, Tech. rep., School of Computer Science, University of Birmingham (1999).

[48] M. Kwiatkowska, G. Norman, D. Parker, PRISM: Probabilistic symbolic model checker, in: T. Field, P. Harrison, J. Bradley, U. Harder (Eds.), Proc. 12th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02), Vol. 2324 of LNCS, Springer, 2002, pp. 200–204.

[49] F. Somenzi, Cudd: Colorado university decision diagram package, release 2.3.0, user's Manual and Programmer's Manual, http://vlsi.colorado.edu/~fabio (September 1998).

[50] R. Alur, T. Henzinger, Reactive modules, in: Proc. 11th Annual IEEE Symposium on Logic in Computer Science (LICS'96), IEEE Computer Society Press, 1006, pp. 207–218.

[51] E. Frank, Codierung und numerische analyse von transitionssystemen unter verwendung von MTBDDs, student's thesis, Universität Erlangen–Nürnberg, IMMD 7 (in German) (1999).

[52] E. Frank, Erweiterung eines MTBDD-basierten Werkzeugs für die Analyse stochastischer Transitionssysteme, Tech. Report Informatik 7, No. 01/00, Universität Erlangen–Nürnberg, (in German) (January 2000).

[53] H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis, M. Siegle, Compositional performance modelling with the TIPPtool, Performance Evaluation 39 (1-4) (2000) 5–35.

[54] C. Kelley, Iterative Methods for Linear and Nonlinear Systems, Series on Frontiers in Applied Mathematics, SIAM, 1995.

[55] O. Ibe, K. Trivedi, Stochastic Petri net models of polling systems, IEEE Journal on Selected Areas in Communications 8 (9) (1990) 1649–1657.

[56] H. Hermanns, J. Meyer-Kayser, M. Siegle, Multi terminal binary decision diagrams to represent and analyse continuous time Markov chains, in: B. Plateau, W. Stewart, M. Silva (Eds.), 3rd Int. Workshop on the Numerical Solution of Markov Chains, Prensas Universitarias de Zaragoza, 1999, pp. 188–207.

[57] J. Aspnes, M. Herlihy, Fast Randomized Consensus Using Shared Memory, Journal of Algorithms 11 (3) (1990) 441–460.

[58] M. Vardi, Automatic Verification of Probabilistic Concurrent Finite State Programs, in: Proc. IEEE Symposium on Foundations of Computer Science (FOCS'85), 1985, pp. 327–338.

[59] B. Bollig, I. Wegener, Improving the variable ordering of OBDDs is NP-complete, IEEE Transactions on Computers 45 (9) (1996) 993–1006.

[60] S. Tani, K. Hamaguchi, S. Yajima, The complexity of optimal variable ordering of a shared binary decision diagram, in: Proc. 45th ISAAC, Vol. 762 of Lecture Notes in Computer Science, Springer, 1993, pp. 389–398.

[61] R. Rudell, Dynamic variable ordering for ordered binary decision diagrams, in: Proc. IEEE ICCAD'93, 1993, pp. 42–47.

[62] M. Fujita, Y. Matsunaga, T. Kakuda, On variable ordering of Binary Decision Diagrams for the application of multi-level logic synthesis, in: Proc. European Design Automation Conference (EDAC 91), IEEE Computer Society Press, 1991, pp. 50–54.

[63] J. Bern, C. Meinel, A. Slobodova, Global rebuilding of OBDD's - tunneling memory requirement maxima, in: P. Wolper (Ed.), Proc. International Conference On Computer Aided Verification, Vol. 939 of Lecture Notes in Computer Science, Springer-Verlag, 1995, pp. 4–15.

[64] D. Parker, Implementation of symbolic model checking for probabilistic systems, Ph.D. thesis, School of Computer Science, University of Birmingham (2002 to appear).

[65] J. Kemeny, J. Snell, Finite Markov Chains, Springer, 1976.

[66] K. Larsen, A. Skou, Bisimulation through probabilistic testing, Information and Computation 94 (1) (1991) 1–28.

[67] E. M. Clarke, S. Jha, R. Enders, T. Filkorn, Exploiting symmetry in temporal logic model checking, Formal Methods in System Design 9 (1/2) (1996) 77–104.

[68] J. Katoen, M. Kwiatkowska, G. Norman, D. Parker, Faster and symbolic CTMC model checking, in: L. de Alfaro, S. Gilmore (Eds.), Proc. Process Algebra and Probabilistic Methods (PAPM-PROBMIV 2001), Vol. 2165 of Lecture Notes in Computer Science, Springer, 2001, pp. 23–38.

[69] M. Kwiatkowska, G. Norman, D. Parker, Probabilistic symbolic model checking with PRISM: A hybrid approach, in: J.-P. Katoen, P. Stevens (Eds.), Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02), Vol. 2280 of LNCS, Springer, 2002, pp. 52–66.

[70] P. D'Argenio, B. Jeannet, H. Jensen, K. Larsen, Reachability Analysis of Probabilistic Systems by Successive Refinements, in: L. de Alfaro, S. Gilmore (Eds.), Process Algebra and Probabilistic Methods. Joint Int. Workshop PAPM-PROBMIV 2001, Springer, LNCS 2165, 2001, pp. 39–56.