Symbolic Model Checking of Probabilistic Processes using MTBDDs and the Kronecker Representation

Luca de Alfaro[1], Marta Kwiatkowska[2⋆], Gethin Norman[2⋆], David Parker[2], and Roberto Segala[3⋆⋆]

[1] Department of Electrical Engineering and Computing Science, University of California at Berkeley. dealfaro@eecs.berkeley.edu

[2] University of Birmingham, Birmingham B15 2TT, United Kingdom {M.Z.Kwiatkowska,G.Norman,D.A.Parker}@cs.bham.ac.uk

[3] Dipartimento di Scienze dell'Informazione, Università di Bologna, Mura Anteo Zamboni 7, 40127 Bologna, Italy. segala@cs.unibo.it

Abstract. This paper reports on experimental results with symbolic model checking of probabilistic processes based on Multi-Terminal Binary Decision Diagrams (MTBDDs). We consider concurrent probabilistic systems as models; these allow nondeterministic choice between probability distributions and are particularly well suited to modelling distributed systems with probabilistic behaviour, e.g. randomized consensus algorithms and probabilistic failures. As a specification formalism we use the probabilistic branching-time temporal logic PBTL which allows one to express properties such as "under any scheduling of nondeterministic choices, the probability of ϕ holding until ψ is true is *at least 0.78/at most 0.04*". We adapt the Kronecker representation of (Plateau 1985), which yields a very compact MTBDD encoding of the system. We implement an experimental model checker using the CUDD package and demonstrate that model construction and reachability-based model checking is possible in a matter of seconds for certain classes of systems consisting of up to 10^{30} states.

1 Introduction

There have been many advances in the BDD technology since BDDs were first introduced and applied to symbolic model checking [10, 25]. There are several free and commercial BDD packages in existence, as well as a range of alternative techniques for efficient automatic verification. Model checking tools (to mention `smv`, `SPIN`, `fdr2`) are extensively used by industrial companies in the process of developing new designs for e.g. hardware circuits, network protocols, etc. More recently tremendous progress has been made with tools for the model checking of real-time systems, e.g. Uppaal [6].

⋆ Supported in part by EPSRC grant GR/M04617.

⋆⋆ Supported in part by EPSRC grant GR/M13046.

One area that is lagging behind as far as experimental work is concerned, despite the fact that the fundamental verification algorithms have been known for over a decade [32, 15, 28], is model checking of *probabilistic* systems. This is particularly unsatisfactory since many systems currently being designed would benefit from probabilistic analysis performed *in addition to* the conventional, qualitative checks involving temporal logic formulas or reachability analysis available in established model checking tools. This includes not only quality of service properties such as "with probability 0.9 or greater, the system will respond to the request within time t", but also steady-state probability (until recently, see [17, 4], separate from temporal logic model checking), which allows the computation of characteristics such as long-run average, resource utilization, etc.

In order to support efficient verification of probabilistic systems, BDD-based packages must allow a compact representation for sparse probability matrices. Such a representation, Multi-Terminal Binary Decision Diagrams (MTBDDs), was proposed in [14] along with some matrix algorithms. MTBDDs are also known as Algebraic Decision Diagrams (ADDs) [1] and are implemented in the Colorado University Decision Diagram (CUDD) package of Fabio Somenzi [31]. Based on [22], an MTBDD-based *symbolic* model checking procedure for purely probabilistic processes (state-labelled discrete Markov chains) for the logic PCTL of [22] (a probabilistic variant of CTL) was first presented in [3], and since extended to concurrent probabilistic systems in [2] (without implementation). Similarly, a symbolic model checking procedure for continuous time Markov chains is proposed in [4]. An alternative representation for Markov chains called Probabilistic Decision Graphs (PDGs) was introduced in [9], where early experimental results are also reported.

In this paper we consider *concurrent probabilistic systems* [5], based on Markov Decision Processes [7], similar to those of [32, 8]. These are state-labelled systems which admit *nondeterministic choice* between discrete probability distributions on the successor states, and are particularly appropriate for the representation of randomized distributed algorithms, fault-tolerant and self-stabilising systems. The model checking procedure, first proposed in [15, 8] for the case without fairness and extended to incorporate fairness constraints in [5, 18], reduces to the computation of the *minimum/maximum* reachability probability. We can derive a set of *linear inequalities*, and maximize/minimize the sum of the components of the solution vector subject to the constraints given by the inequalities.

Multi-Terminal Binary Decision Diagrams [14] have the same structure as BDDs, except that terminals other than 0 and 1 are allowed. The similarity between the two types of diagrams means that many BDD operations generalise to the MTBDD case. MTBDDs are known to yield a compact and efficient representation for sparse matrices [14]. They share many positive features with BDDs: because they exploit *regularity* and *sharing*, they allow the representation of much larger matrices than standard sparse matrix representations. MTBDDs also combine well with BDDs in a shared environment, thus allowing *reachability* analysis via conversion to BDDs (which coincide with 0-1 MTBDDs) and conventional BDD reachability. However, MTBDDs also inherit negative BDD features: they are exponential in the worst case, very sensitive to variable ordering heuristics, and may be subject to a sudden, unpredictable, increase in size as the regularity of the structure is lost through performing operations on it. As a consequence, algorithms that *change* the structure of the matrix, such as Gaussian elimination for solving linear equations [1] or simplex for solving systems of linear inequalities [24], are significantly less efficient than state-of-the-art sparse matrix packages due to the loss of regularity. Iterative methods [21, 23], on the other hand, which rely on matrix-by-vector multiplication without changing the matrix structure, perform better.

There has been very little work concerning MTBDD-based numerical linear algebra; a notable exception is the CUDD package [31], a free library of C routines which supports matrix multiplication in a shared BDD and MTBDD environment. In contrast, numerical analysis of Markov chains based on sparse matrices is much more advanced, particularly in the context of Stochastic Petri Nets. There, with the help of a Kronecker representation originally introduced by Brigitte Plateau [26], systems with millions of states can be analysed. The Kronecker representation applies to systems composed of parallel components; each component is represented as a set of (comparatively small) matrices, with the matrix of the full system defined as the reachable subspace of a Kronecker-algebraic expression (usually referred to as the *actual*, versus the *potential*, state space). Then one can avoid having to store the full size matrix by storing the component matrices instead and reformulating steady-state probability calculation in terms of the component matrices. Existing implementation work in this area includes tools such as SMART [11] and PEPS [27].

In this paper we adapt and extend the ideas of [3, 2] in order to represent *concurrent probabilistic systems* in terms of MTBDDs. The differences with the corresponding work in numerical analysis of Markov chains are: we allow non-determinism as well as probability; we work with probability matrices, not generator matrices of continuous time Markov chains; we generate the matrix in full, then perform BDD reachability analysis to obtain the actual state space; and we perform model checking against PBTL through a combination of reachability analysis and numerical approximation instead of steady-state probability calculation. The main contribution of the paper is threefold: (1) we implement an experimental symbolic model checker for PBTL [5] using MTBDDs; (2) we adapt the Kronecker representation of [26] and provide a translation into MTBDDs; and (3) we improve the model checking algorithm by incorporating the probability-1 precomputation step of [19].

2 Concurrent Probabilistic Systems

In this section, we briefly summarise our underlying model for concurrent probabilistic systems; the reader is referred to [5, 2] for more details. Our model is based on "Markov decision processes", and is similar to "Concurrent Markov Chains" of [32, 16] and "simple deterministic automata" of [29]. Some familiarity with Markov chains and probability theory is assumed.

Concurrent probabilistic systems generalise ordinary Markov chains in that they allow a nondeterministic choice between possibly several probability distributions in a given state. Formally, a *concurrent probabilistic system* is a pair $\mathcal{S} = (S, \mathit{Steps})$ where S is a finite set of states and *Steps* a function which assigns to each state $s \in S$ a finite, non-empty set $\mathit{Steps}(s)$ of distributions on S. Elements of $\mathit{Steps}(s)$ are called *transitions*. Systems $\mathcal{S} = (S, \mathit{Steps})$ such that $\mathit{Steps}(s)$ is a singleton set for each $s \in S$ are called *purely probabilistic* and coincide with discrete time Markov chains.

Paths in a concurrent probabilistic system arise by resolving both the nondeterministic and probabilistic choices. A *path* of the system $\mathcal{S} = (S, \mathit{Steps})$ is a non-empty finite or infinite sequence $\pi = s_0 \xrightarrow{p_0} s_1 \xrightarrow{p_1} s_2 \xrightarrow{p_2} \cdots$ where $s_i \in S$, $p_i \in \mathit{Steps}(s_i)$ with $p_i(s_{i+1}) > 0$. We let $\pi(i)$ denote the ith state of the path π.

The selection of a probability distribution is made by an adversary (also known as a scheduler), a function mapping every finite path of the system onto one of the distributions in $\mathit{Steps}(s)$ where s is the last state of the path. Note we use *deterministic* adversaries, rather than randomized adversaries as in [8]. For an adversary A of a concurrent probabilistic system $\mathcal{S} = (S, \mathit{Steps})$ we define $\mathit{Path}^A_{\mathit{ful}}$ to be the set of infinite paths corresponding to the choices of the adversary. In the standard way, we define the measure *Prob* over infinite paths.

Since we allow nondeterministic choice between probability distributions, we may have to impose *fairness constraints* to ensure that liveness properties can be verified. In a distributed environment fairness corresponds to a requirement for each 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. An adversary is called *fair* if any choice of transitions that becomes enabled infinitely often along a computation path is taken infinitely often. The interested reader is referred to [5, 20] for more information on the subject.

3 The Logic PBTL

In this section, based on [5, 8], we recall the syntax and semantics of the probabilistic branching-time temporal logic PBTL. PBTL derives from CTL [13] and PCTL [22], borrowing the temporal operator $\mathcal{U}$ ("until") and the path quantifier $\exists$ from CTL, and the probabilistic operator $[\,\cdot\,]_{\sqsupseteq\lambda}$ from PCTL.

Let AP denote a finite set of atomic propositions. A *PBTL structure* is a tuple $(\mathcal{S}, \mathrm{AP}, L)$ where $\mathcal{S} = (S, \mathit{Steps})$ is a concurrent probabilistic system and $L : S \to 2^{\mathrm{AP}}$ is a labelling function which assigns to each state $s \in S$ a set of atomic propositions. The syntax of PBTL is:

$$\phi ::= \mathtt{true} \;\mid\; a \;\mid\; \phi_1 \wedge \phi_2 \;\mid\; \neg\phi \;\mid\; [\phi_1 \;\exists\mathcal{U}\; \phi_2]_{\sqsupseteq\lambda}$$

where a is an atomic proposition, $\lambda \in [0, 1]$, and $\sqsupseteq$ is either $\geq$ or $>$.

The branching time quantifier $\exists$ involves quantification over adversaries, meaning "there exists an adversary" of a given type. Note that to simplify this presentation, we have omitted the "bounded until", "next state" and "universal until" operators which can easily be added. The latter is defined similarly to the "existential until" operator included above. For a PBTL formula ϕ and set Adv of adversaries we define the satisfaction relation $s \models_{Adv} \phi$ inductively as follows:

$$
\begin{array}{lll}
s \models_{Adv} \texttt{true} & & \text{for all } s \in S \\
s \models_{Adv} a & \Leftrightarrow & a \in L(s) \\
s \models_{Adv} \phi_1 \wedge \phi_2 & \Leftrightarrow & s \models_{Adv} \phi_1 \text{ and } s \models_{Adv} \phi_2 \\
s \models_{Adv} \neg\phi & \Leftrightarrow & s \not\models_{Adv} \phi \\
s \models_{Adv} [\phi_1 \,\exists\mathcal{U}\, \phi_2]_{\sqsupseteq\lambda} & \Leftrightarrow & Prob(\{\pi \mid \pi \in Path^A_{ful}(s) \;\&\; \pi \models_{Adv} \phi_1 \,\mathcal{U}\, \phi_2\}) \sqsupseteq \lambda \\
& & \text{for some adversary } A \in Adv \\
& & \\
\pi \models_{Adv} \phi_1 \,\mathcal{U}\, \phi_2 & \Leftrightarrow & \text{there exists } k \geq 0 \text{ such that } \pi(k) \models_{Adv} \phi_2 \\
& & \text{and for all } j = 0, 1, \ldots, k-1,\ \pi(j) \models_{Adv} \phi_1
\end{array}
$$

We denote satisfaction for *all* adversaries by $\models$ and satisfaction for *all fair* adversaries by $\models_{fair}$.

4 PBTL Model Checking

With the exception of "until" formulas and fairness, model checking for PBTL is straightforward, see [8, 5]. It proceeds by induction on the parse tree of the formula, as in the case of CTL model checking [13].

We only consider existential "until" for reasons of space. To establish whether $s \models_{Adv} [\phi \,\exists\mathcal{U}\, \psi]_{\sqsupseteq\lambda}$, we calculate the *maximum probability*:

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

where $p^A_s(\phi \,\mathcal{U}\, \psi) = Prob(\{\pi \mid \pi \in Path^A_{ful}(s) \;\&\; \pi \models \phi \,\mathcal{U}\, \psi\})$ and compare the result to the threshold λ, i.e. establish the inequality $p^{\max}_s \sqsupseteq \lambda$. First we introduce an operator on sets of states which will be used in the algorithm.

For $U_0, U_1 \subseteq S$, define $reachE(U_0, U_1) = \mu Z[H]$ as the least fixed point of the map $H : 2^S \to 2^S$, where:

$$H = \lambda x.(((x \in U_0) \wedge \exists p \in Steps(x)\, \exists y(p(y) > 0 \wedge y \in Z)) \vee (x \in U_1)).$$

The algorithm is shown in Figure 1. We use $\varepsilon = 10^{-6}$ as the termination criterion for the iteration in step 3. Observe that we compute *approximations* to the actual (minimum/maximum) probabilities from below to within ε. Alternatively, the values $p^{\max}_s(\phi \,\mathcal{U}\, \psi)$ can be calculated by reduction to linear optimization problems [8, 5, 2].

Fairness assumptions, which are necessary in order to verify liveness properties of concurrent probabilistic processes, for example "under any scheduling, process P will eventually enter a successful state with probability at least 0.7", can also be handled. This is possible via reduction of the model checking for $\models_{fair}$ to that for ordinary satisfaction $\models$ using results from [5, 2].

1. Compute the sets of states $Sat(\phi), Sat(\psi)$ that satisfy ϕ, ψ.
2. Let (a) $S^{\text{yes}} := Sat(\psi)$
   (b) $S^{>0} := reachE(Sat(\phi), S^{\text{yes}})$
   (c) $S^{\text{no}} := S \setminus S^{>0}$
   (d) $S^{?} := S \setminus (S^{\text{yes}} \cup S^{\text{no}})$
3. Set $p_s^{\max}(\phi\ \mathcal{U}\ \psi) = 1$ if $s \in S^{\text{yes}}$ and $p_s^{\max}(\phi\ \mathcal{U}\ \psi) = 0$ if $s \in S^{\text{no}}$. 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 $n = 1, 2, \ldots$

$$x_{s,n} = \max\left\{ \sum_{t\in S^{?}} r(t)\cdot x_{t,n-1} + \sum_{t\in S^{\text{yes}}} r(t) \mid r \in Steps(s)\right\}.$$

4. Finally, let $Sat([\phi\ \exists\mathcal{U}\ \psi]_{\sqsupseteq\lambda}) := \{s \in S \mid p_s^{\max}(\phi\ \mathcal{U}\ \psi) \sqsupseteq \lambda\}$.

Fig. 1. The Algorithm **EU**

For purely probabilistic systems, model checking of "until" reduces to a linear equation system in $|S^{?}|$ unknowns which can be solved either through a direct method such as Gaussian elimination, or iteratively via e.g. Jacobi or Gauss-Seidel iteration.

4.1 Probability-1 Precomputation Step

The model checking algorithm for "until" properties given below can be improved by pre-computing the set of *all* states from which the formula holds with maximal probability 1. The algorithm for this precomputation step is based on results of [15, 16] and can be derived from that in [19] for computing the set of states that can reach a goal with probability 1. We have here adapted it to "until" formulas.

For any $Z_0, Z_1 \subseteq S$ let $Pre(Z_0, Z_1)$ be the set of states defined by:

$$Pre(Z_0,Z_1) = \\ \{x \mid \exists p \in Steps(x)(\forall y(p(y) > 0 \to y \in Z_0) \wedge \exists y(p(y) > 0 \wedge y \in Z_1))\}.$$

Intuitively, $s \in Pre(Z_0, Z_1)$ if one can go from s to Z_1 with positive probability without leaving Z_0.

Theorem 1. *Let $\mathcal{S} = (S, Steps)$ be a concurrent probabilistic transition system, $U_0, U_1 \subseteq S$ subsets of states and $prob1E(U_0, U_1)$ be the set of states given by the solution to $\nu Z_0 \mu Z_1[G]$ where*

$$G = \lambda x.((x \in U_1) \vee ((x \in U_0) \wedge x \in Pre(Z_0, Z_1))).$$

Then $s \in prob1E(U_0, U_1)$ if and only if from s, for some adversary, one reaches a state in U_1 via a path through states in U_0 with probability 1.

It follows from this theorem that we can strengthen the assignment to S^{yes} at step 2(a) of Algorithm **EU** to: $S^{\text{yes}} := \mathit{prob1E}(\mathit{Sat}(\phi), \mathit{Sat}(\psi))$. Hence, in cases of *qualitative* properties, i.e. properties which are required to hold with probability 1, no further computation of the probability vector will be required. In particular, this avoids potential difficulties with approximations.

4.2 Symbolic Model Checking

A symbolic method is obtained from the above procedure by representing the system and probability vector as MTBDDs, $\mathit{Sat}(\phi)$ as a BDD, and expressing the probability calculations as MTBDD/BDD operations (for more details see [3, 2]). The operators $\mathit{reachE}(\cdot, \cdot)$ and $\mathit{prob1E}(\cdot, \cdot)$ can be expressed in terms of BDD fixed point computation with respect to the transition relation extracted from the MTBDD. The iterative calculation of the probability vector requires matrix-by-vector multiplication and the operation ABSTRACT(max).

5 Representing Probabilistic Processes with MTBDDs

MTBDDs were introduced in [14] as a generalisation of BDDs. Like BDDs, they take the form of a rooted directed acyclic graph, the nonterminal nodes of which are labelled with Boolean variables from an ordered set. Unlike BDDs however, the terminal nodes are labelled with values taken from a finite set D (usually a subset of the reals), not just 0 and 1. The operations on MTBDDs are derived from their BDD counter-parts, and include REDUCE, APPLY and ABSTRACT, see [1, 14]. An MTBDD with n Boolean variables and terminals taken from the finite set D, can be considered as a map $f : \{0, 1\}^n \to D$.

In [14] it is shown how to represent matrices in terms of MTBDDs. Consider a square $2^m \times 2^m$–matrix $\mathbf{A}$ with entries taken from D. Its elements a_{ij} can be viewed as the values of a function $f_{\mathbf{A}} : \{1, \ldots, 2^m\} \times \{1, \ldots, 2^m\} \to D$, where $f_{\mathbf{A}}(i, j) = a_{ij}$, mapping the position indices i, j to the matrix element a_{ij}. Using the standard encoding $c : \{0, 1\}^m \to \{1, \ldots, 2^m\}$ of Boolean sequences of length m into the integers, this function may be interpreted as a Boolean function $f : \{0, 1\}^{2m} \to D$ where $f(x, y) = f_A(c(x), c(y))$ for $x = (x_1, \ldots, x_m)$ and $y = (y_1, \ldots, y_m)$. We require the variables for the rows and columns to alternate, that is, use the MTBDD obtained from $f(x_1, y_1, x_2, y_2, \ldots, x_m, y_m)$. This convention imposes a recursive structure on the matrix from which efficient recursive algorithms for all standard matrix operations are derived [1, 14].

Probability matrices are sparse, and thus can have a compact MTBDD representation. This compactness results from *sharing* of substructures, and increases with the regularity of the original matrix. Though in the worst case exponential, compared to sparse matrix representation and depending on the degree of regularity of the original matrix, MTBDDs can be much more space-efficient than sparse matrices. They also combine efficiently with BDDs.

Concurrent probabilistic transition systems with n states that enable at most l nondeterministic transitions each can be represented as a $nl \times n$ matrix, which can then be stored as an MTBDD. (For simplicity assume that n and l are powers of 2). Each row of the matrix represents a single nondeterministic choice, where the element in position $(i.k, j)$ represents the probability of reaching state j from state i in the k^{th} transition that leaves from i.

Unfortunately, experimental evidence has shown that this simple MTBDD representation of concurrent probabilistic systems suffers from a disproportionately large number of internal nodes, due to the lack of regularity. Instead, we will adapt the *Kronecker representation* originally introduced for space-efficient storage of Markov processes as Stochastic Automata Networks [26].

5.1 A Modular Description Language for Probabilistic Processes

We propose a modular description language for concurrent probabilistic systems in an attempt to derive a more efficient MTBDD encoding. The system is considered as a composition of *modules*, acting concurrently, more specifically via the asynchronous parallel composition of probabilistic processes whose local transitions may be dependent on the global state of the system.

This model bears similarities to the Stochastic Automata Networks (SANs) of [26]. One difference between the two approaches is that we consider probabilistic, as opposed to stochastic processes. Secondly, SANs permit two types of process interaction: *synchronization* between components, and *functional transitions*, where the rate or probability with which one component makes a transition may depend on the state of another component. For simplicity, we discuss here only the latter type of interaction. Most importantly, the motivation is different: the fundamental idea with SANs is that since the transition matrix for the composed system is formulated as a Kronecker expression, only the small matrices which make up this expression need to be stored and explicit construction of the whole (often huge) transition matrix can be avoided. Although our aim is also to obtain a space efficient method of storage, we construct the whole matrix and use a Kronecker expression to derive an efficient MTBDD variable ordering.

We consider the system as a composition of n modules $M_1, \ldots, M_n$, each with a set of local variables Var_i. Each variable $x \in Var_i$ has a finite range of values, $range(x)$. The local state space S_i of module M_i is $\prod_{x \in Var_i} range(x)$. The global state space of the combined system is then $S = \prod_{i=1}^{n} S_i$.

Each module defines the transitions that it can make, depending on its current state and the state of the other modules in the system. The behaviour of a module M_i is given by a finite non-empty set L_i of tuples of the form (c, p), where $c = \wedge_{j=1}^{n} c_j$ is a conjunction of n variable constraints, c_j is a formula over Var_j and p is a probability distribution over S_i. Intuitively, c represents the condition under which transitions corresponding to the probability distribution p can be made. We can associate with a tuple $l = (c, p)$ the set of global states $S_l = \{s \in S \mid s \models c\}$ which satisfy the variable contraints. We require, for all modules M_i, that the sets S_l where $l \in L_i$ form a disjoint union of S.

We interpret the formal description of the behaviour of the modules as follows. If the global state of the system is $s = (s_1, \ldots, s_n)$ and $s \in S_l$ for a tuple $l = (c, p) \in L_i$ then the probability of module M_i moving from its current local state s_i to the local state t_i is $p(t_i)$. Hence, in each global state of the system, any of the n modules can make a move. The behaviour of each individual module is essentially that of a Markov chain. It is necessary to decide on some form of scheduling between the modules to define the behaviour of the composed system. We consider two possibilities: probabilistic and nondeterministic scheduling. In the former, each module has an equal probability of being scheduled, giving a Markov chain. In the latter, we allow a nondeterministic choice between modules, which gives a concurrent probabilistic system as described in Section 2.

Module M_1:
$(x = 0) \rightarrow \frac{1}{2} : (x' = 0) + \frac{1}{2} : (x' = 1)$
$(x = 1) \wedge (y \leq 1) \rightarrow (x' = 2)$
$(x = 1) \wedge (y = 2) \rightarrow (x' = 1)$
$(x = 2) \rightarrow \frac{1}{2} : (x' = 0) + \frac{1}{2} : (x' = 2)$
Module M_2:
$(y = 0) \rightarrow \frac{1}{2} : (y' = 0) + \frac{1}{2} : (y' = 1)$
$(y = 1) \wedge (x \leq 1) \rightarrow (y' = 2)$
$(y = 1) \wedge (x = 2) \rightarrow (y' = 1)$
$(y = 2) \rightarrow \frac{1}{2} : (y' = 0) + \frac{1}{2} : (y' = 2)$

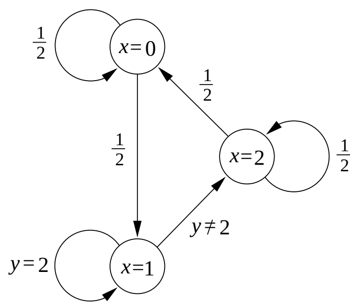

Fig. 2. (i) A system composed of two modules. (ii) Transition system of module M_1

Consider the example shown in Figure 2 of the modules M_1, M_2, with corresponding variable sets $Var_1 = \{x\}$ and $Var_2 = \{y\}$, where x and y have range $\{0, 1, 2\}$. Figure 2(i) shows a textual description of the modules. Each line corresponds to a tuple (c, p) where the condition c and the probability distribution p are separated by a $\rightarrow$ symbol. For example, in line 1 of module M_1, the condition is $(x = 0)$ and the distribution is $\frac{1}{2} : (x' = 0) + \frac{1}{2} : (x' = 1)$, where x' denotes the value of x in the state after the transition. Note $c = (x = 0)$ is in fact $c = c_1 \wedge c_2$ where $c_1 = (x = 0)$ and $c_2 = \texttt{true}$, i.e. there is no constraint on y.

5.2 A Kronecker Expression for the Modular Description

We now derive a Kronecker expression for the transition matrix of the composed system. As pointed out in [26], the transition matrix for the composition of n non-interacting stochastic automata can be written as:

$$\mathbf{Q} = \oplus_{i=1}^{n} \mathbf{L}_i = \sum_{i=1}^{n} \mathbf{Q}_i \text{ where } \mathbf{Q}_i = (\otimes_{j=1}^{i-1} \mathbf{I}_{n_j}) \otimes \mathbf{L}_i \otimes (\otimes_{j=i+1}^{n} \mathbf{I}_{n_j}).$$

In the above, $\mathbf{L}_i$ is the local transition matrix for component i, $\mathbf{I}_n$ the identity matrix of size n, and n_j the dimension of local matrix $\mathbf{L}_j$. We use the same basic construction here, but with *probability* matrices $\mathbf{P}_i$ for each module, rather than $\mathbf{Q}_i$. These local matrices are then combined through scheduling rather than simply summation. We also account for the fact that the behaviour of one module can depend on the state of another. As with functional transitions in SANs, this does not affect the overall structure of the Kronecker expression:

some transitions are simply removed by zeroing out the relevant entry in the component matrix.

For a given module M_i, we want an expression for the corresponding transition matrix $\mathbf{P}_i$. Since $+$ distributes over $\otimes$, we can break the expression up into separate matrices $\mathbf{P}_{i,l}$, such that $\mathbf{P}_i = \sum_{l \in L_i} \mathbf{P}_{i,l}$. The restrictions on transitions are handled by zeroing out some entries of the identity matrices. We first convert the information from the module descriptions to vectors. For module M_i and line $l \in L_i$ where $l = (c, p)$ and $c = \wedge_{j=1}^{n} c_j$, we associate with each c_j the column vector $\mathbf{c}_j$, indexed over local states S_j, with $\mathbf{c}_j(s) = 1$ if $s \models c_j$ and $\mathbf{c}_j(s) = 0$ otherwise. Similarly, the probability distribution p is converted to a row vector $\mathbf{p}$, indexed over local states from S_i, with $\mathbf{p}(s) = p(s)$. The unwanted elements of the jth identity matrix are removed by a pointwise multiplication with the vector $\mathbf{c}_j$. Then:

$$\mathbf{P}_{i,l} = (\otimes_{j=1}^{i-1} \mathbf{c}_j \cdot \mathbf{I}_{n_j}) \otimes (\mathbf{c}_i \otimes \mathbf{p}) \otimes (\otimes_{j=i+1}^{n} \mathbf{c}_j \cdot \mathbf{I}_{n_j})$$

Consider the example given previously. We can write the matrices $\mathbf{P}_{1,1}$ and $\mathbf{P}_{2,2}$ for line 1 of module M_1 and line 2 of module M_2 respectively, as:

$$\mathbf{P}_{1,1} = \left(\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix} \right) \otimes \left(\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \right) = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{P}_{2,2} = \left(\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \right) \otimes \left(\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \right) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

5.3 Module to MTBDD Translation

The construction of the transition matrix, as described above, can be derived directly from the syntax in Figure 2(i) by means of BDD and MTBDD operations. First, we encode all the module variables with Boolean variables. For convenience, we assume that the range of each variable, x, is a power of 2, i.e. $range(x) = \{0, \ldots, 2^k - 1\}$ for some k. Hence, x can be encoded with k Boolean variables $x_1, \ldots, x_k$, and x' with $x_1{}', \ldots, x_k{}'$. This gives a set of MTBDD row (unprimed) and column (primed) variables for each module variable in the system. The ordering of the modules in the textual description and of the module variables within them gives us an overall ordering for the Boolean variables in our MTBDDs. In our small example, we get $x_1 < x_1' < x_2 < x_2' < y_1 < y_1' < y_2 < y_2'$.

The column vectors $\mathbf{c}_j$, row vector $\mathbf{p}$ and identity matrices $\mathbf{I}_{n_j}$ can then be represented as MTBDDs, using the appropriate Boolean variables. The Kronecker product operation on matrices and vectors represented as MTBDDs can be performed using the APPLY($\times$) operator. The only precaution which must be taken is to ensure that the relative order of the MTBDD variables is correct. If the MTBDDs f and g represent the matrices F and G respectively and all the variables in f precede all those of g in the overall variable ordering then $\textsc{Apply}(\times,\ f,\ g)$ gives the MTBDD for the matrix $F \otimes G$ which depends on the variables of both. Because we have ensured that our Boolean variables are grouped and ordered by module, the Kronecker expression can be computed easily with $\textsc{Apply}(\times)$. Since pointwise multiplication is also carried using $\textsc{Apply}(\times)$, the MTBDD expression for $\mathbf{P}_{i,l}$ is as shown below.

$$\mathbf{P}_{i,l} = \textsc{Apply}(\times, \mathbf{c}_1, \mathbf{I}_{n_1}, \ldots, \mathbf{c}_{i-1}, \mathbf{I}_{n_{i-1}}, \mathbf{c}_i, \mathbf{p}, \mathbf{c}_{i+1}, \mathbf{I}_{n_{i+1}}, \ldots, \mathbf{c}_n, \mathbf{I}_{n_n}).$$

Since $\times$ is commutative and $\textsc{Apply}(\times, \mathbf{c}_1, \ldots, \mathbf{c}_n) = \mathbf{c}$, rearranging:

$$\mathbf{P}_{i,l} = \textsc{Apply}(\times, \mathbf{c}, \mathbf{p}, \mathbf{I}_{n_1}, \ldots, \mathbf{I}_{n_{i-1}}, \mathbf{I}_{n_{i+1}}, \ldots, \mathbf{I}_{n_n}).$$

We then obtain $\mathbf{P}_i$ by summing the $\mathbf{P}_{i,l}$ for $l \in L_i$ using $\textsc{Apply}(+)$. Finally, we compute the MTBDD $\mathbf{P}$ for the whole system. For probabilistic scheduling:

$$\mathbf{P} = \textsc{Apply}(\times, \frac{1}{n}, \textsc{Apply}(+, \mathbf{P}_1, \ldots, \mathbf{P}_n)).$$

For nondeterministic scheduling, we add MTBDD variables to encode the scheduler's choice: one variable, s_i, for each process, where $s_i = 1$ iff module M_i is scheduled. This variable can be inserted in the variable ordering next to the variables for M_i, to preserve regularity. Returning to our simple example, we would have the variable ordering $s_1 < x_1 < x_1' < x_2 < x_2' < s_2 < y_1 < y_1' < y_2 < y_2'$. The computation of $\mathbf{P}$ becomes:

$$\mathbf{P} = \textsc{Apply}(+, \mathrm{ITE}(s_1 = 1, \mathbf{P}_1, 0), \ldots, \mathrm{ITE}(s_n = 1, \mathbf{P}_n, 0)).$$

where $\mathrm{ITE}(\cdot,\cdot,\cdot)$ refers to the MTBDD operation $\textsc{IfThenElse}(\cdot,\cdot,\cdot)$.

The central observation of the paper is that if we convert each of the component matrices into an MTBDD using the Boolean variables and ordering given above, then this yields a very efficient state-space encoding through increase in regularity (for example, over the matrix obtained through breadth-first search[1]) and sharing. This complies with similar results in [23, 30]. Moreover, Kronecker product and ordinary sum of two matrices are also very efficient as they respectively correspond to the MTBDD operations $\textsc{Apply}(\times)$ and $\textsc{Apply}(+)$.

6 Experimental Results

We have implemented an experimental symbolic model checking tool using the CUDD package [31]. This package provides support for BDDs and MTBDDs, together with matrix multiplication algorithms from [1, 14].

Our tool performs model checking for PBTL, with and without fairness. The MTBDD for a system of modules is automatically generated from its textual description using the translation described above. Forward reachability analysis is then performed to filter out the unreachable states.

[1] See `www.cs.bham.ac.uk/~dxp/prism/` for MATLAB spy plots of matrices obtained via breadth-first search and Kronecker.

To model check qualitative properties (typically 'with probability 1'), reachability analysis through fixed point computation suffices. Quantitative properties, however, require numerical computation. We work with double-precision floating point arithmetic, which is standard for numerical calculations. Our implementation relies on the matrix-by-vector multiplication obtained from the matrix-by-matrix multiplication algorithms supplied with the CUDD package. The model checking of unbounded "until" properties is through the Algorithm **EU**. For purely probabilistic processes we use Jacobi iteration, which has the following advantages: it is simple and numerically stable, relies only on the matrix-by-vector multiplication, and can deal with very large matrices (since it does not change the matrix representing the system). The limiting factor is thus the size of the probability vector (which can only be represented efficiently in MTBDDs if it has regularity).

We have tested our tool on several scalable examples from the literature, in particular the kanban problem [12] known from manufacturing and the randomized dining philosophers [28]. For more information see `www.cs.bham.ac.uk/~dxp/prism/`. Figures 3–6 show a summary of results for the dining philosophers model. The concurrent model corresponds to that presented in [28], whereas the probabilistic model corresponds to the same model with probabilistic scheduling. The tables give the MTBDD statistics, construction and model checking times in seconds of the liveness property in [28] (with fairness), performed on an Ultra 10 with 380MB.

The main observations we have made so far are as follows: (1) the variable ordering induced from the Kronecker representation results in very compact MTBDDs (see comparison of the number of internal nodes in breadth-first and Kronecker); (2) because of sharing, MTBDDs allow space-efficiency gains over conventional sparse matrices for certain systems; (3) the model construction is very fast, including the computation of the reachable subspace; (4) through use of probability-1 precomputation step, model checking of qualitative properties (with and without fairness) is very fast, and results in orders of magnitude speed-up; (5) performance of numerical calculation with MTBDDs is considerably worse than with sparse matrices, though MTBDDs can potentially handle larger matrices and vectors (e.g. up to 5 million) depending on their regularity.

7 Conclusion

We have demonstrated the feasibility of symbolic model checking for probabilistic processes using MTBDDs. In particular, the state encoding induced from the Kronecker representation allows us to verify qualitative properties of systems containing up to 10^{30} states in a matter of seconds. Moreover, model creation is very fast (typically seconds) due to the close correspondence between Kronecker product and Apply($\times$), and efficiency of reachability analysis implemented as the usual BDD fixed point computation. Likewise, model checking of qualitative properties (expressed as 'with probability 1' PBTL formulas) is very fast. Many

Model:	Breadth-first:		
	States:	NNZ:	Nodes:
phil 3	770	2,845	3,636
phil 4	7,070	34,125	30,358
phil 5	64,858	384,621	229,925

Model:	Kronecker:			After reachability:		
	States:	NNZ:	Nodes:	States	NNZ:	Nodes:
phil 3	1,331	4,654	647	770	2,845	873
phil 4	14,641	67,531	1,329	7,070	34,125	2,159
phil 5	161,051	919,656	2,388	64,858	384,621	3,977
phil 10	2.59×10^{10}	2.86×10^{11}	14,999	4.21×10^{9}	4.72×10^{10}	26,269
phil 20	6.73×10^{20}	1.43×10^{22}	174,077	1.77×10^{19}	3.81×10^{20}	291,760
phil 25	1.08×10^{26}	2.86×10^{26}	479,128	1.14×10^{24}	3.06×10^{25}	798,145

Fig. 3. Statistics for probabilistic models and their MTBDD representation.

Model:	Breadth-first:		
	States:	NNZ:	Nodes:
phil 3	770	2,910	4,401
phil 4	7,070	35,620	41,670
phil 5	64,858	408,470	354,902

Model:	Kronecker:			After reachability		
	States:	NNZ:	Nodes:	States	NNZ:	Nodes:
phil 3	1,331	34,848	451	770	20,880	779
phil 4	14,641	1,022,208	669	7,070	511,232	1556
phil 5	161,051	2.81×10^{7}	887	64,858	1.17×10^{7}	2,178
phil 10	2.59×10^{10}	2.90×10^{14}	1,977	4.21×10^{9}	4.87×10^{13}	6,379
phil 20	6.73×10^{20}	1.54×10^{28}	4,157	1.77×10^{19}	4.19×10^{26}	14,429
phil 30	1.75×10^{31}	6.13×10^{41}	6,337	7.44×10^{28}	2.71×10^{39}	22,479

Fig. 4. Statistics for concurrent models and their MTBDD representation.

quantitative properties, however, are not handled efficiently by our present tool. This is due to poor efficiency of numerical computation, such as Jacobi iteration or simplex, compared to the corresponding sparse matrix implementation. The causes of this are a sudden loss of regularity of the probability vector due to explosion in the number of distinct values computed in the process of approximation (in the case of Jacobi) or fill-in of the tableau (in the case of simplex).

Future work will involve further development of the front end for our tool, comparison with the prototype tools of [4, 9, 12], and addressing the inefficiency of numerical calculations with MTBDDs.

Acknowledgements

The authors Kwiatkowska, Norman and Parker are members of the ARC project 1031 *Stochastic Modelling and Verification* funded by the British Council and DAAD. We also thank the anonymous referees for their helpful comments.

Model:	Construction:	Reachability:		Model checking:	
	Time (s):	Time (s):	Iterations:	Time (s):	Iterations:
phil3	0.02	0.06	18	0.02	6
phil4	0.04	0.33	24	0.04	6
phil5	0.07	1.08	30	0.06	6
phil10	0.45	28.28	60	0.23	6
phil20	4.45	404.15	120	0.65	6
phil25	10.81	766.03	150	0.99	6

Fig. 5. Times for construction and model checking of probabilistic models

Model:	Construction:	Reachability:		Model checking:	
	Time (s):	Time (s):	Iterations:	Time (s):	Iterations:
phil3	0.02	0.07	18	0.02	6
phil4	0.03	0.35	24	0.04	6
phil5	0.05	1.00	30	0.07	6
phil10	0.24	27.03	60	0.22	6
phil20	1.45	389.56	120	0.49	6
phil30	4.10	5395.00	180	11.40	6

Fig. 6. Times for construction and model checking of concurrent models

References

1. I. Bahar, E. Frohm, C. Gaona, G. Hachtel, E.Macii, A. Pardo, and F. Somenzi. Algebraic Decision Diagrams and their Applications. *Journal of Formal Methods in Systems Design*, 10(2/3):171–206, 1997.
2. C. Baier. On algorithmic verification methods for probabilistic systems. Habilitation thesis, 1998.
3. C. Baier, E. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In *Proceedings, 24th ICALP*, volume 1256 of *LNCS*, pages 430–440. Springer-Verlag, 1997.
4. C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In *CONCUR'99*, volume 1664 of *LNCS*, pages 146–161. Springer-Verlag, 1999.
5. C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11:125–155, 1998.
6. J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, W. Yi, and C. Weise. New generation of uppaal. In *Proceedings of the International Workshop on Software Tools for Technology Transfer*, Aalborg, Denmark, July 1998.
7. D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.
8. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proceedings, FST&TCS*, volume 1026 of *LNCS*, pages 499–513. Springer-Verlag, 1995.
9. M. Bozga and O. Maler. On the representation of probabilities over structured domains. In *Proc. CAV'99*, 1999. Available as Volume 1633 of *LNCS*.
10. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *LICS'90*, June 1990.

11. G. Ciardo and A. Miner. SMART: Simulation and markovian analyzer for reliability and timing. In *Tools Descriptions from PNPM'97*, pages 41–43, 1997.
12. G. Ciardo and A. Miner. A data structure for the efficient Kronecker solution of GSPNs. In *Proc. PNPM'99*, 1999.
13. E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach. In *Proceedings, 10th Annual Symp. on Principles of Programming Languages*, 1983.
14. E. Clarke, M. Fujita, P. McGeer, J.Yang, and X. Zhao. Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation. In *International Workshop on Logic Synthesis*, 1993.
15. C. Courcoubetis and M. Yannakakis. Markov decision processes and regular events. In *Proc. ICALP'90*, volume 443 of *LNCS*, pages 336–349. Springer-Verlag, 1990.
16. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.
17. L. de Alfaro. How to specify and verify the long-run average behavior of probabilistic systems. In *Proc. LICS'98*, pages 454–465, 1998.
18. L. de Alfaro. Stochastic transition systems. In *Proc. CONCUR'98*, volume 1466 of *LNCS*. Springer-Verlag, 1998.
19. L. de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In *Proc. CONCUR'99*, volume 1664 of *LNCS*, 1999.
20. L. de Alfaro. From fairness to chance. In *Proc. PROBMIV'98*, volume 21 of *ENTCS*. Elsevier, 1999.
21. G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Markovian Analysis of Large Finite State Machines. *IEEE Transactions on CAD*, 15(12):1479–1493, 1996.
22. H. Hansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, 6:512–535, 1994.
23. H. Hermanns, J. Meyer-Kayser, and M. Siegle. Multi Terminal Binary Decision Diagrams to represent and analyse continuous time Markov chains. In *Proc. NSMC'99*, 1999.
24. M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of concurrent probabilistic systems using MTBDDs and simplex. Technical Report CSR-99-1, University of Birmingham, 1999.
25. K. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
26. B. Plateau. On the Stochastic Structure of Parallelism and Synchronisation Models for Distributed Algorithms. In *Proc. 1985 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 147–153, May 1985.
27. B. Plateau, J. M. Fourneau, and K. H. Lee. PEPS: a package for solving complex Markov models of parallel systems. In R. Puigjaner and D. Potier, editors, *Modelling techniques and tools for computer performance evaluation*, 1988.
28. A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1:53–72, 1986.
29. R. Segala. *Modelling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, 1995.
30. M. Siegle. Compact representation of large performability models based on extended BDDs. In *Fourth International Workshop on Performability Modeling of Computer and Communication Systems (PMCCS4)*, pages 77–80, 1998.
31. F. Somenzi. CUDD: CU decision diagram package. Public software, Colorado University, Boulder, 1997.
32. M. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings, FOCS'85*, pages 327–338. IEEE Press, 1987.