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 non-deterministic choice between probability distributions and are particularly well suited to modelling distributed systems in which the outcome of some transitions is determined probabilistically, for example, 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*". Model checking of concurrent probabilistic systems against PBTL properties can be reduced to a linear programming problem and solved using e.g. the simplex algorithm with the help of a sparse matrix package. In order to represent the models in terms of MTBDDs, we adapt the Kronecker representation of (Plateau 1985) originally formulated for analysing very large Markov processes. The Kronecker representation yields an encoding of the system states which results in very compact MTBDDs. 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.

Keywords: temporal logic, symbolic model checking, probabilistic verification, Markov chains.

1 Introduction

There have been many advances in the BDD technology since BDDs [13] were first introduced and applied to symbolic model checking [14, 37]. 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 [8] and Kronos [11].

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 [50, 21, 32, 40], 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

⋆ supported in part by EPSRC grant GR/M04617
⋆⋆ supported in part by EPSRC grant GR/M13046

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 [24, 5], separate from temporal logic based 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 [19] 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 [47]. Based on [31], an MTBDD-based *symbolic* model checking procedure for purely probabilistic processes (state-labelled discrete Markov chains) for the logic PCTL of [31] (a probabilistic variant of CTL) was first presented in [4], and since extended to concurrent probabilistic systems in [2] (without implementation). The algorithm for checking PCTL "until" properties reduces to solving a system of *linear equations*.

In this paper we consider models for *concurrent probabilistic systems* based on Markov Decision Processes [9], similar to those of [50, 22, 43, 10, 6, 24]. 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 presence of nondeterminism means that no single probability space can be associated with the computation paths of our models: the probability space is determined only for a given *adversary* (or *scheduler*) which selects one of possibly several probability distributions available in each state. Consequently, the logics interpreted over concurrent probabilistic systems include quantification over adversaries, as well as probabilistic reasoning. The model checking procedure, first proposed in [10, 21] for the case without fairness and extended to incorporate fairness constraints in [6, 25], reduces to the computation of the *minimum/maximum* reachability probability. Alternatively, 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. As suggested in [10], this linear programming problem can be solved with the help of e.g. the simplex algorithm.

Multi-Terminal Binary Decision Diagrams [19] 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, for example, Apply, Abstract and Reduce, generalise to the MTBDD case. MTBDDs are known to yield a compact and efficient representation for sparse (particularly for very sparse) matrices [19]. 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 [36], are significantly less efficient than state-of-the-art sparse matrix packages due to the loss of regularity and/or fill-in. Iterative methods [29], 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 [47], 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 [38], 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 (ususally 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 Petri net tools such as [16, 39].

In this paper we adapt and extend the ideas of [4, 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 nondeterminism 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 [6] using MTBDDs; (2) we adapt the Kronecker representation of [38] and provide a translation into MTBDDs; and (3) we improve the model checking algorithm by incorporating probability-1 precomputation step of [26], both for the case of with and without fairness constraints.

We test our tool on a number of examples of varying size, and provide experimental data for the following important observations. When using the variable ordering induced by the Kronecker representation, we observe a dramatic improvement in the compactness of MTBDD representation of certain systems compared to the ordering obtained through breadth first search. In particular, we can handle purely probabilistic systems of up to 10^{25} reachable states, and concurrent systems of up to 10^{30} states, as opposed to 60,000 states with breadth-first. 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. However, we cannot as yet handle many quantitative properties efficiently, compared with sparse matrix packages, though, depending on the regularity of the vector, we are able to handle larger systems (up to 5 million states).

Related work: Much has been published concerning probabilistic logics and verification, see e.g. [50, 21, 22, 40, 41, 31, 30, 10, 6, 2], but little experimental work has been done. Probabilistic Verus [33, 3] offers an extensive C-like description language for discretely-timed systems, but does not permit nondeterminism. A canonical representation for Markov chains called Probabilistic Decision Graphs (PDGs) was introduced in [12], where early experimental results are also reported. A model checker for continuous time Markov chains is proposed in [5]. In [34] 'rules of thumb' are proposed for representing queuing systems efficiently using MTBDDs. A data structure for representing stochastic systems is Decision Node BDDs (DNBDDs) [45, 46]. Factored Edge-Valued BDDs (FEVBDDs) [49] are applied to Integer Linear Programming, resulting in space efficient but slow algorithms based on infinite precision arithmetic. It is not clear how the FEVBDD representation can be made to work with floating point arithmetic which we adhere to, as it is standard in numerical probabilistic analysis.

2 Concurrent Probabilistic Systems

In this section, we briefly summarise our underlying model for concurrent probabilistic systems; the reader is referred to [6, 2] for more details. Our model is based on "Markov decision processes", and is similar to "Concurrent Markov Chains" of [50, 22] and "simple deterministic automata" of [43]. Some familiarity with Markov chains and probability theory is assumed, see e.g. [48].

We begin by recalling the standard definition of (discrete time) Markov chains and the associated probability measure. A *Markov chain* is a pair $M = (S, \mathbf{P})$ where S is a finite set of states and $\mathbf{P} : S \times S \to [0, 1]$ a *transition matrix*, i.e. $\sum_{t \in S} \mathbf{P}(s, t) = 1$ for all $s \in S$. A *path* in M is a nonempty (finite or infinite) sequence $\pi = s_0 s_1 s_2, \ldots$ where s_i are states and $\mathbf{P}(s_{i-1}, s_i) > 0$, $i \in \mathbb{N}$. The length of a path, denoted $|\pi|$, is defined in the usual way. A path π is called a *fulpath* iff it is infinite. We model terminating behaviour by repeating the final state infinitely often. The first state of π is denoted by $first(\pi)$, and the last by $last(\pi)$ (if it exists).

$Path_{ful}$ denotes the set of fulpaths, whereas $Path_{ful}(s)$ is the set of fulpaths π with $first(\pi) = s$. For $s \in S$, let $\mathcal{F}(s)$ be the smallest σ-algebra on $Path_{ful}(s)$ which contains the basic cylinders $\{\pi \in Path_{ful} : \rho \text{ is a prefix of } \pi\}$ where ρ ranges over all finite paths starting in s. The probability measure $Prob$ on $\mathcal{F}(s)$ is the unique measure with $Prob\{\ \pi \in Path_{ful}(s)\ : \rho \text{ is a prefix of } \pi\ \} = \mathbf{P}(\rho)$ where $\mathbf{P}(s_0 s_1 \ldots s_k) = \mathbf{P}(s_0, s_1) \cdot \mathbf{P}(s_1, s_2) \cdot \ldots \cdot \mathbf{P}(s_{k-1}, s_k)$.

Concurrent probabilistic systems generalise ordinary Markov chains in that they allow possibly several probability distributions to be enabled in a given state. The selection of a probability distribution is made by an adversary (also known as a scheduler), and then the process makes a transition to a state determined by the selected distribution. We denote the set of discrete probability distributions over a set S by $Distr(S)$. Therefore, each $p \in Distr(S)$ is a function $p : S \to [0,1]$ such that $\sum_{s \in S} p(s) = 1$.

Definition 1 (Concurrent Probabilistic System). *A* concurrent probabilistic system *is a pair $\mathcal{S} = (S, Steps)$ 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 $Steps(s)$ of distributions on S. Elements of $Steps(s)$ are called* transitions. *A system $\mathcal{S} = (S, Steps)$ such that $Steps(s)$ is a singleton set for each $s \in S$ is called* purely probabilistic.

Note that purely probabilistic systems 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 $\mathcal{S} = (S, Steps)$ 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 $s_i \in S$, $p_i \in Steps(s_i)$ with $p_i(s_{i+1}) > 0$ for all $0 \leq i \leq |\pi|$.

We now introduce *adversaries* of concurrent probabilistic systems. We use *deterministic* adversaries, rather than randomized adversaries as in [10].

Definition 2 (Adversary of a Concurrent Probabilistic System). *An* adversary *(or scheduler) of a concurrent probabilistic system $\mathcal{S} = (S, Steps)$ is a function A mapping every finite path ρ of $\mathcal{S}$ to a distribution $A(\rho)$ on S such that $A(\rho) \in Steps(last(\rho))$ is a transition in $\mathcal{S}$. We use $\mathbf{A}(\mathcal{S})$ to denote the set of all adversaries of $\mathcal{S}$.*

Note that an adversary can choose a different distribution every time the system returns to the same state, which means that there are infinitely many adversaries even if the system has finitely many states. An adversary A of $\mathcal{S}$ is called *simple* iff for every state $s \in S$ there exists a transition $p_s \in Steps(s)$ with $A(\rho) = p_{last(\rho)}$ where ρ ranges over all finite paths. It turns out that for model checking it suffices to consider only the simple adversaries[1] which resolve the nondeterminism by selecting the same distribution every time a given state is reached – independent of the past history; for more detail see [6].

For an adversary A of a concurrent probabilistic system $\mathcal{S} = (S, Steps)$ we define $Path^A_{fin}$ to be the set of finite paths such that $step(\rho, i) = A(\rho^{(i)})$ for all $1 \leq i \leq |\rho|$, and $Path^A_{ful}$ to be the set of paths in $Path_{ful}$ such that $step(\rho, i) = A(\rho^{(i)})$ for all $i \in \mathbb{N}$, where $\rho(i)$ denotes the i-th state of ρ, $step(\rho, i)$ is the distribution selected by the i-th step, and $\rho^{(i)}$ is the prefix of ρ of length i.

With each adversary we associate a Markov chain, together with the induced probability measure $Prob$, which can be viewed as a set of paths in $\mathcal{S}$. Formally, if A is an adversary of the concurrent probabilistic system $\mathcal{S}$, then $MC^A = (Path^A_{fin}, \mathbf{P}^A)$ is a Markov chain where:

$$\mathbf{P}^A(\rho, \rho') = \begin{cases} p(s) & \text{if } A(\rho) = p \text{ and } \rho' = \rho \xrightarrow{p} s \\ 0 & \text{otherwise} \end{cases}$$

Since we allow nondeterministic choice between probability distributions, 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 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.

[1] These correspond to "stationary deterministic policies" in Markov decision processes.

We define a fulpath π of a concurrent probabilistic system to be *fair* iff, whenever a state s is visited infinitely often in π, each nondeterministic alternative which is enabled in s is taken infinitely often in π. More formally, we have the definition below.

Definition 3 (Fairness of Fulpaths). *Let $\mathcal{S} = (S, Steps)$ be a concurrent probabilistic system and π a fulpath in $\mathcal{S}$. π is called* fair *iff, for each $s \in inf(\pi)$ and each $p \in Steps(s)$, there are infinitely many indices i with $\pi(i) = s$ and $step(\pi, i) = p$. $Fair(\mathcal{S})$, abbreviated by $Fair$, denotes the set of fair fulpaths in $\mathcal{S}$.*

Now an adversary is *fair* if any choice of transitions that becomes enabled infinitely often along a computation path is taken infinitely often.

Definition 4 (Fair Adversaries). *Let $\mathcal{S} = (S, Steps)$ be a concurrent probabilistic system and F an adversary for $\mathcal{S}$. F is called* fair *iff $Prob(Fair^F(s)) = 1$ for all $s \in S$. We let $\mathbf{A}_{fair}(\mathcal{S})$ denote the set of fair adversaries.*

The interested reader is referred to [32, 43, 7, 6, 25, 27] for more information concerning fairness in probabilistic systems.

3 The Logic PBTL

In this section, based on [6, 10], we recall the syntax and semantics of the probabilistic branching-time temporal logic PBTL. PBTL combines features of CTL [18] and PCTL [31]. In particular, the temporal operator $\mathcal{U}$ ("until") and the path quantifiers $\forall$ and $\exists$ are taken 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 $\mathcal{S} = (S, Steps)$ 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. By abuse of notation we shall refer to the PBTL structure $(\mathcal{S}, \mathrm{AP}, L)$ over the concurrent probabilistic system $\mathcal{S} = (S, Steps)$ as $\mathcal{S}$.

The syntax of PBTL is given below. To simplify this presentation we omit the "bounded until" and "next state" operators which can be easily added.

Definition 5 (Syntax of PBTL**).** *The syntax of* PBTL *formulas is defined as follows:*

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

where a is an atomic proposition, $\lambda \in [0, 1]$, and $\sqsupseteq$ is either $\geq$ or $>$. Any formula of the form $\phi_1\ \mathcal{U}\ \phi_2$, where ϕ_1, ϕ_2 are PBTL formulas, is defined to be a path formula.

PBTL formulas are interpreted over states of concurrent probabilistic systems, whereas path formulas over fulpaths. The branching time quantifiers $\exists$ and $\forall$ involve quantification over adversaries: $\exists$ means "there exists an adversary" of a given type, and $\forall$ "for all adversaries" of a given type.

Definition 6 (Satisfaction Relation for PBTL**).** *Given a PBTL-structure $(\mathcal{S}, \mathrm{AP}, L)$, a set Adv of adversaries of $\mathcal{S}$, and* PBTL *formula ϕ, we define the satisfaction relation $s \models_{Adv} \phi$ inductively as follows:*

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

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 $\models_{Adv}$ [6], of which we only consider two, $\models$ and $\models_{fair}$.

Probabilistic extensions of temporal logics similar to PBTL have been proposed in the literature. One such is PCTL of [44]; PBTL differs from that logic in that we consider labelling of states with atomic propositions, whereas in [44] action-labelled concurrent probabilistic systems are defined.

An alternative, but equivalent formulation to ours, is that used for pCTL [10] (called PCTL in [2]) which uses a probabilistic operator $I\!P_{\bowtie\lambda}(\cdot)$ where $\bowtie \in \{<, \leq, =, >, \geq\}$. One can translate formulas of pCTL into PBTL formulas, and vice versa. For example, the PBTL formula $[\phi_1\ \forall\mathcal{U}\ \phi_2]_{\sqsupseteq\lambda}$ can be identified with the pCTL formula $I\!P_{\sqsupseteq\lambda}(\phi_1\mathcal{U}\phi_2)$, while $[\phi_1\ \exists\mathcal{U}\ \phi_2]_{>\lambda}$ corresponds to $\neg I\!P_{\leq\lambda}(\phi_1\mathcal{U}\phi_2)$.

4 Model Checking Concurrent Probabilistic Systems against PBTL

With the exception of "until" formulas and fairness, model checking for PBTL is straightforward, see [10, 6]. It proceeds by induction on the parse tree of the formula, as in the case of CTL model checking [18].

4.1 Model Checking without Fairness

We summarise the model checking algorithm for PBTL, first for the case of $\models$ without fairness.

Atomic propositions and boolean operators:

$$\begin{aligned} Sat(\texttt{true}) &= S \\ Sat(a) &= \{s \in S \mid a \in L(s)\} \\ Sat(\phi_1 \wedge \phi_2) &= Sat(\phi_1) \cap Sat(\phi_2) \\ Sat(\neg\phi) &= S \backslash Sat(\phi) \end{aligned}$$

Until formulas: We only consider existential "until". To establish whether $s \in S$ satisfies $[\phi\ \exists\mathcal{U}\ \psi]_{\sqsupseteq\lambda}$, we calculate the *maximum probability*:

$$p_s^{\max}(\phi\ \mathcal{U}\ \psi) = \sup\{p_s^A(\phi\ \mathcal{U}\ \psi) \mid A \in \mathbf{A}(\mathcal{S})\}$$

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

Definition 7 (Operator *reachE*$(\cdot,\cdot)$**).** *Let* $U_0, U_1 \subseteq S$. *Define* $reachE(U_0, U_1) = \mu Z[H]$, *the least fixed point of the map* $H : 2^S \to 2^S$, *where:*

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

Note that the above is a μ-calculus formula. The algorithm is shown below:

Algorithm **EU**

1. Compute the sets of states $Sat(\phi), Sat(\psi)$ that satisfy ϕ and ψ respectively.
2. Let
   (a) $S^{\text{yes}} := Sat(\psi)$ (the set of all states which trivially satisfy $\phi \ \mathcal{U} \ \psi$ with probability 1),
   (b) $S^{>0} := reachE(Sat(\phi), S^{\text{yes}})$ (the set of states from which, for some adversary, one can reach a state satisfying ψ via a path through the states satisfying ϕ with positive probability) where the operator $reachE(\cdot,\cdot)$ is defined in Definition 7,
   (c) $S^{\text{no}} := S \setminus S^{>0}$ (states which definitely do not satisfy $\phi \ \mathcal{U} \ \psi$),
   (d) $S^{?} := S^{>0} \setminus S^{\text{yes}}$ (states which satisfy $\phi \ \mathcal{U} \ \psi$ with some, as yet unknown, positive probability).
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 $Sat([\phi \ \exists\mathcal{U} \ \psi]_{\sqsupseteq \lambda}) := \{s \in S \mid p_s^{\max}(\phi \ \mathcal{U} \ \psi) \sqsupseteq \lambda\}$.

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 $p_s^{\max}(\phi \ \mathcal{U} \ \psi)$ can be calculated by reduction to linear optimization problems [10, 6, 2].

For purely probabilistic systems, model checking of "until" reduces to solving the following linear equation system in $|S^{?}|$ unknowns:

$$x_s = \Sigma_{t \in S^{?}} \mathbf{P}(s,t) \cdot x_t + \Sigma_{t \in S^{\text{yes}}} \mathbf{P}(s,t)$$

which can be solved either through a direct method such as Gaussian elimination, or iteratively via e.g. Jacobi or Gauss-Seidel iteration.

Universal "until" is verified through reduction to the minimum probability calculation, see [10, 6, 23, 2].

4.2 Probability-1 Precomputation Step

The model checking algorithm for "until" properties can be improved by pre-computing the set of *all* states from which the formula holds with maximal probability 1. The algorithm for this pre-computation step, based on results of [21, 22] who showed that computing the maximal probability of a temporal logic formula can be reduced to computing the maximal reachability probability of given sets (see also [10]), can be derived from that in [26] for computing the set of states that can reach a goal with probability 1. We have here adapted it to "until" formulas, and implemented it symbolically using MTBDDs.

We now describe the precomputation step. For any $Z_0, Z_1 \subseteq S$ let $Pre(Z_0, Z_1)$ be the set of states defined by:

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

Intuitively, $s \in Pre(Z_0, Z_1)$ if one can go 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.) of Algorithm **EU** to:

$$S^{\text{yes}} := prob1E(Sat(\phi), Sat(\psi)).$$

Note that BDD fixed point computation suffices to calculate S^{yes}, and that 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.

4.3 Symbolic Model Checking

A symbolic method is obtained from the above procedure by representing the system and probability vector as MTBDDs, $Sat(\phi)$ as a BDD, and expressing the probability calculations as MTBDD/BDD operations (for more details see [4, 2]). The operator $reachE(\cdot, \cdot)$ can be expressed in terms of BDD fixed point computation with respect to the transition relation extracted from the MTBDD (and likewise $reachA(\cdot, \cdot)$ needed for universal "until"). The iterative calculation of the probability vector requires matrix-by-vector multiplication and the operation ABSTRACT(max).

4.4 Model Checking with Fairness

Fairness assumptions 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". Without fairness, computations of the system which do not allow P to make progress would trivially violate this property. Unfortunately, incorporating fairness assumptions in probabilistic model checking is not as simple as for CTL, where model checking under fairness can be reduced to satisfiability checking for ordinary satisfaction by considering implications of the form $\phi_{\text{fair}} \rightarrow \phi$, where ϕ_{fair} is the fairness constraint expressed as a CTL* formula, and ϕ is the CTL formula to verify. This approach does not apply for PBTL because of a mismatch between quantification needed for $\models$ (over all adversaries) and that needed for $\models_{fair}$ (over all *fair* adversaries).

Nevertheless, model checking for $\models_{fair}$ reduces to that for ordinary satisfaction $\models$ on the basis of the following statement. Part (1) of this statement is due to [6], and part (2) is an improvement of [6] shown in [2].

Theorem 2. *If ϕ and ψ are PBTL formulas, $\mathcal{S} = (S, Steps)$ is a concurrent probabilistic system and $s \in S$, then*

1. $s \models_{fair} [\phi\ \exists\mathcal{U}\ \psi]_{\sqsupseteq\lambda}$ *iff* $p_s^{\max}(\phi\ \mathcal{U}\ \psi) \sqsupseteq \lambda$
2. $s \models_{fair} [\phi\ \forall\mathcal{U}\ \psi]_{\sqsupseteq\lambda}$ *iff* $1 - p_s^{\max}(a^?\ \mathcal{U}\ \neg a^+) \sqsupseteq \lambda$ *where $a^+, a^?$ are new atomic propositions, with*

$$Sat(a^+) := reachE(Sat_{fair}(\phi), Sat_{fair}(\psi))$$
$$Sat(a^?) := Sat(a^+) \setminus Sat_{fair}(\psi)$$

where $reachE(\cdot, \cdot)$ is as defined in Definition 7.

Observe that a^+ can be computed via BDD fixed point iteration.

5 Representing Concurrent Probabilistic Systems in MTBDDs

In this section we show how to represent concurrent probabilistic systems in terms of MTBDDs.

5.1 Multi-Terminal Binary Decision Diagrams (MTBDDs)

MTBDDs were introduced in [19] as a generalisation of BDDs [13] to allow finitely many possibly real values as the terminal nodes, instead of just 0 and 1.

Let $x_1, \ldots, x_n$ be distinct Boolean variables, which we order by $x_i < x_j$ iff $i < j$. A *multi-terminal binary decision diagram* over $(x_1, \ldots, x_n)$ is a rooted, directed graph with vertex set V containing two types of vertices, *nonterminal* and *terminal*. Each nonterminal vertex v is labelled by a variable $var(v) \in \{x_1, \ldots, x_n\}$ and two children $left(v)$, $right(v) \in V$. Each terminal vertex v is labelled by a real number $value(v)$. For each nonterminal node v, we require $var(v) < var(left(v))$ if $left(v)$ is nonterminal, and similarly, $var(v) < var(right(v))$ if $right(v)$ is nonterminal. Note that BDDs coincide with 0-1 MTBDDs. The operations on MTBDDs are derived from their BDD counter-parts, and include REDUCE, APPLY and ABSTRACT, see [1, 19].

In [19] it is shown how to represent matrices in terms of MTBDDs. Let D be a finite set of values, possibly reals. D–valued matrices can be represented by MTBDDs as follows. Consider a square $2^m \times 2^m$–matrix $\mathbf{A}$ (non-square matrices can be dealt with by padding with all-zero columns or rows). 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 value 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 D–valued 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 $y = (y_1, \ldots, y_m)$. Furthermore, we require the variables of f to alternate, thus using the MTBDD obtained from $f(x_1, y_1, x_2, y_2, \ldots, x_m, y_m)$ instead of the MTBDD for $f(x_1, \ldots, x_m, y_1, \ldots, y_m)$. This convention imposes a recursive structure on the matrix from which efficient recursive algorithms for all standard matrix operations are derived [1, 19].

Probability matrices are sparse (or even very sparse), and thus can have a compact MTBDD representation. The compactness results from *sharing* of substructures, and increases depending on the regularity of the original matrix. Though in the worst case exponential, compared to sparse matrix representation, MTBDDs provide a uniform $\log_2(N)$ access time (where N is the number of non-zero elements of the matrix) and combine efficiently with BDDs. Also, depending on the degree of regularity of the original matrix, MTBDDs can be much more space-efficient than sparse matrices.

5.2 The Kronecker Representation

Concurrent probabilistic transition systems with n states that enable at most l non-deterministic 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 non-deterministic 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. If from a state i there are less than l transitions enabled, then we repeat the last row. For a purely probabilistic system this representation specialises to a square $n \times n$ matrix.

Unfortunately, this simple MTBDD representation of concurrent probabilistic systems suffers from a disproportionately large number of internal nodes. Instead, we will use the *Kronecker representation* originally introduced for space-efficient storage of Markov processes as Stochastic Automata Networks [38], then further developed and applied to Stochastic Petri Nets in [28, 35].

Definition 8 (Kronecker product and sum). *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be matrices of size* $n \times n$ *and* $m \times m$ *respectively. The matrix* $\mathbf{C}$ *is the* Kronecker product *of* $\mathbf{A}$ *and* $\mathbf{B}$*, written* $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$*, if and only if* $\mathbf{C}$ *is a matrix of size* $nm \times nm$ *defined by*

$$\mathbf{C}(i_1 n + i_2, j_1 n + j_2) = a_{i_1, j_1} b_{i_2, j_2}$$

for $0 \leq i_1, j_1 < n$ *and* $0 \leq i_2, j_2 < m$. *The* Kronecker sum *of* $\mathbf{A}$ *and* $\mathbf{B}$ *is defined by:*

$$\mathbf{A} \oplus \mathbf{B} = \mathbf{A} \otimes \mathbf{I}_m + \mathbf{I}_n \otimes \mathbf{B}$$

where $\mathbf{I}_k$ *is the identity of size* k.

The main idea of the Kronecker representation is to decompose the system into concurrent components, where each component is represented as a set of matrices, separately for *local* events, $\mathbf{L}^i$, and for *synchronizing events*, $\mathbf{W}^i(e)$, where $e \in E$ for some finite set E of events. These are then composed into the full matrix $\hat{\mathbf{R}}$ representing the *potential* state space (for simplicity assume that there are 2 components) via the expression:

$$\hat{\mathbf{R}} = \Sigma_{e \in E} rate(e) \cdot (\mathbf{W}^1(e) \otimes \mathbf{W}^2(e)) + (\mathbf{L}^1 \oplus \mathbf{L}^2)$$

The matrix $\hat{\mathbf{R}}$ may contain unreachable states, and thus its reachable submatrix (called the *actual* state space) is identified. Instead of storing the (often very large) matrix $\hat{\mathbf{R}}$, the small component matrices are stored instead, together with the Kronecker expression, and the steady-state probability calculations appropriately reformulated. For more details see [38].

5.3 Representing Probabilistic Processes via Kronecker

We now outline our adaptation of the Kronecker representation to concurrent probabilistic systems using MTBDDs. We able to deal with *asynchronous parallel composition of processes whose local transitions may be dependent on the global state of the system.* For clarity, we will illustrate our approach by means of an example. Consider the following system consisting of two processes, P1 and P2, composed in parallel. The variables $x, y \in \{0,\ 1,\ 2\}$ range over the states of the processes P1 and P2 respectively. We use x', y' to denote successor state. The command $(x = 0) \rightarrow (\{\frac{1}{2} : x' = 0\} + \{\frac{1}{2} : x' = 1\})$ should be understood as: if P1 is in state 0, then it either returns to state 0 with probability $\frac{1}{2}$, or moves to state 1 with probability $\frac{1}{2}$. Furthermore, $(x = 1) \wedge (y \in \{0, 1\}) \rightarrow (x' = 2)$ reads as: in P1 is in state 1 and P2 is in state 0 or 1 then P1 moves to state 2.

Process P1:
$(x = 0) \rightarrow (\{\frac{1}{2} : x' = 0\} + \{\frac{1}{2} : x' = 1\})$ $(x = 1) \wedge (y \in \{0, 1\}) \rightarrow (x' = 2)$ $(x = 1) \wedge (y = 2) \rightarrow (x' = 1)$ $(x = 2) \rightarrow (\{\frac{1}{2} : x' = 0\} + \{\frac{1}{2} : x' = 2\})$
Process P2:
$(y = 0) \rightarrow (\{\frac{1}{2} : y' = 0\} + \{\frac{1}{2} : y' = 1\})$ $(y = 1) \wedge (x \in \{0, 1\}) \rightarrow (y' = 2)$ $(y = 1) \wedge (x = 2) \rightarrow (y' = 1)$ $(y = 2) \rightarrow (\{\frac{1}{2} : y' = 0\} + \{\frac{1}{2} : y' = 2\})$

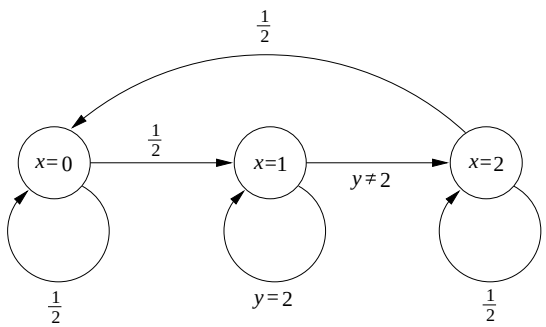

Fig. 1. (i) A system composed of two processes. (ii) Transition system of process P1

We view each process as the possible local transitions and corresponding conditions on the states of other processes, given by the commands for that process, encoded as matrices. We then construct the transition matrix for the system by means of Kronecker products of these smaller matrices. In the case of P1, there are four commands in the process description, and hence four pairs of matrices needed to build the transition matrix, namely $t_{1,i}$ (transitions) and $c_{1,i}$ (conditions) for $i = 1, \ldots, 4$, given below:

$$\mathbf{P}_{1,1} = t_{1,1} \otimes c_{1,1} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{P}_{1,3} = t_{1,3} \otimes c_{1,3} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{P}_{1,2} = t_{1,2} \otimes c_{1,2} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad \mathbf{P}_{1,4} = t_{1,4} \otimes c_{1,4} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

For example, the matrix $t_{1,1}$ represents the local probabilistic transitions of P1 in state 0 and since there is no condition on the state of P2 for this to occur, $c_{1,1}$ is the identity. By comparison, the local transition of P1 from $x = 1$ to $x = 2$ (represented by $t_{1,2}$) is conditional on P2 being in state 0 or 1. Hence, $c_{1,2}$ contains only the top two rows of the identity.

The matrices $\mathbf{P}_{2,i}$ for process P2 can be derived in a similar fashion. An important difference, due to the state ordering imposed by the Kronecker construction, is that the conditional matrices which correspond to the states of P1 must come first in the Kronecker product, i.e. $\mathbf{P}_{2,i} = c_{2,i} \otimes t_{2,i}$. Note that, more generally, in systems composed of n components, we need to apply the Kronecker product to n matrices for each command and these must adhere to a fixed ordering on the processes.

The construction of the matrices described above can be derived directly from the syntax in Figure 1(i) by means of BDD and MTBDD operations. First, we encode x, x', y, y' with pairs of Boolean variables: (x_1, x_2), (x'_1, x'_2), (y_1, y_2) and (y'_1, y'_2) respectively. We use the overall variable ordering $x_1 < x'_1 < x_2 < x'_2 < y_1 < y'_1 < y_2 < y'_2$, which is induced by the ordering of the processes in the Kronecker construction. From the syntax, the matrices can then be built with simple MTBDD operations. For example:

$$\begin{aligned} t_{1,1} &= \textsc{Apply}(\times,\ x = 0,\ \textsc{Apply}(+, \textsc{Apply}(\times,\ \tfrac{1}{2},\ x' = 0),\ \textsc{Apply}(\times,\ \tfrac{1}{2},\ x' = 1))) \\ c_{1,1} &= \textsc{Apply}(\times,\ 1,\ y' = y) \\ \mathbf{P}_{1,1} &= \textsc{Apply}(\times,\ t_{1,1},\ c_{1,1}) \end{aligned}$$

Viewed as MTBDDs, the matrices $t_{1,1}$, $c_{1,1}$ and $\mathbf{P}_{1,1}$ are shown in Figure 2.

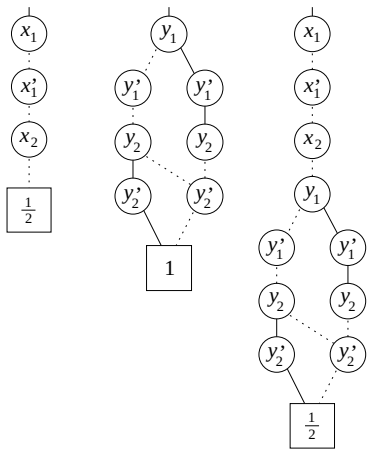

Fig. 2. MTBDD representation of matrices $t_{1,1}$, $c_{1,1}$ and $\mathbf{P}_{1,1}$.

The central observation of the paper is that if we convert each of the component matrices into an MTBDD using the boolean variables and variable 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[2]) and sharing. 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 $\textsc{Apply}(+)$.

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

Using the example above we demonstrate the types of concurrent compositions of probabilistic processes that we can model.

Probabilistic: The matrix expression:

$$\mathbf{P} = \frac{1}{2}(\mathbf{P}_{1,1} + \mathbf{P}_{1,2} + \mathbf{P}_{1,3} + \mathbf{P}_{1,4}) + \frac{1}{2}(\mathbf{P}_{2,1} + \mathbf{P}_{2,2} + \mathbf{P}_{2,3} + \mathbf{P}_{2,4})$$

where + denotes ordinary matrix sum represents the probabilistic composition of processes P1 and P2, with respect to the uniform probability distribution.

Concurrent: To define a concurrent composition of P1 and P2, we add a scheduler which selects local, independent transitions of each process for execution (these arise as *nondeterministic* choices *between* probability distributions in *global* states). This is modelled by adding MTBDD variables to encode the scheduler: one variable, s_i, for each process, where $s_1 = 1$ iff process P1 is scheduled and $s_2 = 1$ iff process P2 is scheduled.

The new variable ordering is $s_1 < x_1 < x_1' < x_2 < x_2' < s_2 < y_1 < y_1' < y_2 < y_2'$. The MTBDDs for $t_{1,1}, c_{1,1}, \ldots$ are modified accordingly, for example $t_{1,1} = \text{IfThenElse}(s_1 = 1,\ t_{1,1},\ 0)$. The matrices $\mathbf{P}_{i,j}$ are then constructed as before and the full transition matrix is given by:

$$\mathbf{P} = (\mathbf{P}_{1,1} + \mathbf{P}_{1,2} + \mathbf{P}_{1,3} + \mathbf{P}_{1,4}) + (\mathbf{P}_{2,1} + \mathbf{P}_{2,2} + \mathbf{P}_{2,3} + \mathbf{P}_{2,4})$$

Nondeterminism within processes: Our approach also allows nondeterminism *within* each component process in the concurrent case, e.g., obtained through abstracting probabilistic choice into nondeterministic. For example, in P1 and P2 we can replace the probabilistic choice in state 0 with a nondeterministic choice between remaining in state 0 and moving to state 1. This is dealt with by adding further MTBDD variables to encode the non-deterministic choices within each process. The transition matrix $\mathbf{P}$ is then computed as for concurrent composition.

6 Experimental Results

We have implemented an experimental symbolic model checking tool using the CUDD package [47]. This package provides support for BDDs and MTBDDs, together with matrix multiplication algorithms from [1, 19, 20].

Our tool performs model checking for PBTL (including "next"), with and without fairness. The MTBDD for the full system is constructed by induction on the syntax of the Kronecker expression from the matrices representing system components, then forward reachability analysis is used to filter out unreachable states. The efficient state encoding is maintained, though some of the regularity of the matrix is lost.

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 iterative process described in Section 4 (Algorithm **EU**). For purely probabilistic processes we use Jacobi iteration, which has the following advantages: it is simple and numerically stable (no need to calculate inverse), 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). Also, a large number of iterations may be required in cases of slow convergence.

We have tested our tool on several scalable examples from the literature, in particular the kanban problem [17] known from manufacturing and the randomized dining philosophers [40]. For more information see `www.cs.bham.ac.uk/~dxp/prism/`. Below is a summary of results for the dining philosophers model. The concurrent model corresponds to that presented in [40], whereas the probabilistic model corresponds to the same model with uniform probabilistic choices replacing non-determinism.

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 6	1,771,561	1.20×10^7	3,909	594,790	4,170,946	6,813
phil 7	1.95×10^7	1.53×10^8	5,927	5,454,562	441×10^7	10,149
phil 8	2.14×10^8	1.91×10^9	8,451	5.00×10^7	4.57×10^8	14,806
phil 10	2.59×10^{10}	2.86×10^{11}	14,999	4.21×10^9	4.72×10^{10}	26,269
phil 15	4.18×10^{15}	6.77×10^{16}	40,205	2.73×10^{14}	4.47×10^{15}	69,281
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.

The main observations 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.

The tables, given in Figures 3, 4 and 5, show the MTBDD statistics, construction and model checking times in seconds of the liveness property in [40] (with fairness), performed on an Ultra 10 with 380MB.

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 6	1,771,561	7.42×10^8	1,105	594,790	2.58×10^8	3,159
phil 7	1.95×10^7	1.91×10^{10}	1,323	5,454,562	5.52×10^9	3,662
phil 8	2.14×10^8	4.79×10^{11}	1,541	5.00×10^7	1.16×10^{11}	4,769
phil 10	2.59×10^{10}	2.90×10^{14}	1,977	4.21×10^9	4.87×10^{13}	6,379
phil 15	4.18×10^{15}	2.24×10^{21}	3,067	2.73×10^{14}	1.52×10^{20}	9,598
phil 20	6.73×10^{20}	1.54×10^{28}	4,157	1.77×10^{19}	4.19×10^{26}	14,429
phil 25	1.08×10^{26}	9.92×10^{34}	5,247	1.15×10^{24}	1.09×10^{33}	17,018
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.

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
phil6	0.11	2.56	36	0.09	6
phil7	0.16	5.61	42	0.11	6
phil8	0.25	9.74	48	0.17	6
phil9	0.31	16.76	54	0.20	6
phil10	0.45	28.28	60	0.23	6
phil15	1.52	139.87	90	0.43	6
phil20	4.45	404.15	120	0.65	6
phil25	10.81	766.03	150	0.99	6

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
phil6	0.08	2.55	36	0.10	6
phil7	0.11	5.20	42	0.12	6
phil8	0.16	10.77	48	0.15	6
phil9	0.20	17.77	54	0.18	6
phil10	0.24	27.03	60	0.22	6
phil15	0.65	158.61	90	0.39	6
phil20	1.45	389.56	120	0.49	6
phil25	2.42	795.84	150	0.71	6
phil30	4.10	5395.00	180	11.40	6

Fig. 5. Times for construction and model checking of both types of model.

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 model check qualitative properties of systems up to 10^{30} states in a matter of seconds. Many 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 developing a front end for our tool, comparison with the prototype tools of [5, 12, 15], and addressing the inefficiency of numerical calculations with MTBDDs.

Acknowledgements

The authors Marta Kwiatkowska, Gethin Norman and David Parker are members of the ARC project 1031 *Stochastic Modelling and Verification* funded by the British Council and DAAD.

References

1. R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. 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, submitted, 1998.
3. C. Baier, E. Clarke, and V. Hartonas-Garmhausen. On the semantic foundations of Probabilistic VERUS. In *Proceedings, Probabilistic Methods in Verification (PROBMIV'98)*, volume 21 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1999. To appear.
4. 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 *Lecture Notes in Computer Science*, pages 430–440. Springer-Verlag, 1997.
5. C. Baier, J.-P. Katoen, and H. Hermanns. Approximative symbolic model checking of continuous-time markov chains. In *CONCUR'99*, volume 1664 of *Lecture Notes in Computer Science*, pages 146–161. Springer-Verlag, 1999.
6. C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11:125–155, 1998.
7. C. Baier and M. Kwiatkowska. On the verification of qualitative properties of probabilistic processes under fairness constraints. *Information Processing Letters*, 66:71–79, 1998.
8. 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.
9. D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.
10. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proceedings, FST&TCS*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513. Springer-Verlag, 1995.
11. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: a model-checking tool for real-time systems. In *Proc. of the 10th Conference on Computer-Aided Verification*, Vancouver, Canada, 28 June - 2 July 1998. Springer Verlag.
12. M. Bozga and O. Maler. On the representation of probabilities over structured domains. In *Proc. 11th International Conference on Computer Aided Verification (CAV'99)*, Trento, Italy, 1999. Available as Volume 1633 of *LNCS*.
13. R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.
14. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, June 1990.
15. G. Ciardo and A. Miner. A data structure for the efficient Kronecker solution of GSPNs. In *Proc. 8th International Workshop on Petri Nets and Performance Models (PNPM'99)*, Zaragoza, 1999.
16. G. Ciardo and A. S. Miner. SMART: Simulation and markovian analyzer for reliability and timing. In *Tools Descriptions from the 9th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation and the 7th International Workshop on Petri Nets and Performance Models*, pages 41–43, Saint Malo, France, June 1997.
17. G. Ciardo and M. Tilgner. On the use of Kronecker operators for the solution of generalized stocastic Petri nets. Technical Report ICASE report 96-35, Institute for computer applications in science and engineering, 1996.
18. E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications: A pratical approach. In *Proceedings, 10th Annual Symp. on Principles of Programming Languages*. ACM Press, 1983.
19. E. Clarke, M. Fujita, P. McGeer, J.Yang, and X. Zhao. Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation. In *IWLS'93: International Workshop on Logic Synthesis, Tahoe City*, May 1993.
20. E. Clarke, K. McMillan, X. Zhao, M. Fujita, and J. Yang. Spectral transforms for large boolean functions with applications to technology mapping. In *Proc. ACM/IEEE Design Automation Conference (DAC'93)*, pages 54–60, Dallas, June 1993. Also available in *Formal Methods in System Design*, 10(2/3), 1997.
21. C. Courcoubetis and M. Yannakakis. Markov decision processes and regular events. In *Proc. ICALP'90*, volume 443 of *Lecture Notes in Computer Science*, pages 336–349. Springer-Verlag, 1990.
22. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.
23. L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.
24. L. de Alfaro. How to specify and verify the long-run average behavior of probabilistic systems. In *Proc. the 13th Annual IEEE Symposium on Logic in Computer Science (LICS'98)*, pages 454–465, 1998.

25. L. de Alfaro. Stochastic transition systems. In *Proc. CONCUR'98: Concurrency Theory*, Lecture Notes in Computer Science. Springer-Verlag, 1998.
26. L. de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In *Proc. 10th International Conference on Concurrency Theory (CONCUR'99)*, Lecture Notes in Computer Science. Springer-Verlag, 1999.
27. L. de Alfaro. From fairness to chance. In *Proceedings, Probabilistic Methods in Verification (PROBMIV'98)*, volume 21 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1999. To appear.
28. S. Donatelli. Superposed Stochastic Automata: a class of stochastic Petri nets amenable to parallel solution. *Performance Evaluation*, 18:21–36, 1993.
29. G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Markovian Analysis of Large Finite State Machines. *IEEE Transactions on CAD*, 15(12):1479–1493, December 1996.
30. H. Hansson. *Time and Probability in Formal Design of Distributed Systems*. Elsevier, 1994.
31. H. Hansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, 6:512–535, 1994.
32. S. Hart, M. Sharir, and A. Pnueli. Termination of probabilistic concurrent programs. *ACM Transactions on Programming Languages and Systems*, 5:356–380, 1983.
33. V. Hartonas-Garmhausen. *Probabilistic Symbolic Model Checking with Engineering Models and Applications*. PhD thesis, Carnegie Mellon University, 1998.
34. H. Hermanns, J. Meyer-Kayser, and M. Siegle. Multi Terminal Binary Decision Diagrams to represent and analyse continuous time Markov chains. In *Proc. Numerical Solutions of Markov Chains (NSMC'99)*, Zaragoza, 1999.
35. P. Kemper. Numerical analysis of superposed GSPNs. *IEEE Transactions on Software Engineering*, 22(4):615–628, September 1996.
36. 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.
37. K. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
38. 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.
39. 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*, Spain, September 1988.
40. A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1:53–72, 1986.
41. A. Pnueli and L. Zuck. Probabilistic verification. *Information and Computation*, 103:1–29, 1993.
42. A. Schrijver. *Theory of Linear and Integer Programming*. J. Wiley and Sons, 1986.
43. R. Segala. *Modelling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, 1995.
44. R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In *Proceedings, CONCUR*, volume 836 of *Lecture Notes in Computer Science*, pages 481–496. Springer-Verlag, 1994.
45. 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, Williamsburg, VA, September 1998.
46. M. Siegle. Technique and tool for symbolic representation and manipulation of stochastic transition systems. TR IMMD 7 2/98, Universität Erlangen-Nürnberg, March 1998. http://www7.informatik.uni-erlangen.de/~siegle/own.h tml.
47. F. Somenzi. CUDD: CU decision diagram package. Public software, Colorado University, Boulder, 1997.
48. W. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.
49. P. Tafertshofer and M. Pedram. Factored edge-valued binary-decision diagrams. *Formal Methods in Systems Design*, 10:137–164, 1997.
50. M. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings, FOCS'85*, pages 327–338. IEEE Press, 1987.