Symbolic Model Checking of Concurrent Probabilistic Systems Using MTBDDs and Simplex*

Marta Kwiatkowska, Gethin Norman, David Parker
University of Birmingham, Birmingham B15 2TT, United Kingdom
{M.Z.Kwiatkowska,G.Norman,D.A.Parker}@cs.bham.ac.uk
and
Roberto Segala
Dipartimento di Scienze dell'Informazione, Università di Bologna,
Mura Anteo Zamboni 7, 40127 Bologna, Italy
segala@cs.unibo.it

January 22, 1999

Abstract

Symbolic model checking for purely probabilistic processes using MTBDDs [12] was introduced in [4] and further developed in [20, 3]. In this paper we consider models for *concurrent* probabilistic systems similar to those of [28, 7, 5] and the concurrent Markov chains of [35, 13], which extend the purely probabilistic processes through the addition of nondeterministic choice. As a specification formalism we use probabilistic branching-time temporal logic PBTL of [5, 7], which allows 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". In [5, 7] it is shown that the verification of "until" properties can be reduced to a linear programming problem and solved with the help of e.g. the simplex algorithm, but no symbolic model checking is considered. Based on the algorithms of [5, 7], we derive *symbolic* model checking procedure for PBTL over concurrent probabilistic systems using MTBDDs. We furthermore implement an experimental model checker using the Colorado University Decision Diagrams (CUDD) package [32]. Our key contribution is an implementation of the *simplex algorithm* in terms of MTBDDs.

1 Introduction

There have been many advances in the BDD technology since BDDs were first introduced [9] and applied to symbolic model checking [10, 24]. There are now several free and commercial BDD packages in existence, as well as a range of alternative techniques for improving efficiency, and 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. There have also been encouraging developments in model checking of real-time and hybrid systems [6, 8].

*supported in part by EPSRC grants GR/M04617 and GR/M13046

Technical Report CSR-99-1, School of Computer Science, University of Birmingham, January 1999.

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 [35, 19, 25], 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 performance analysis, currently separate from model checking, which allows the computation of characteristics such as long-run average, resource utilization, etc.

In order to support the 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 [12] along with matrix algorithms. Based on [18], an MTBDD-based *symbolic* model checking procedure for purely probabilistic processes (state-labelled discrete Markov chains) for the logic PCTL of [18] (a probabilistic variant of CTL which includes formulas such as $[\phi\ \mathcal{U}\ \psi]_{\geq\lambda}$ meaning the probability of satisfying $\phi\ \mathcal{U}\ \psi$ is at least λ), was first presented in [4]. The algorithm for checking $[\phi\,\mathcal{U}\,\psi]_{\geq\lambda}$ involves computing the probability measure (assuming the usual measure on the σ-algebra of sets of infinite paths induced from the cones of finite paths) of the set of all paths satisfying $\phi\,\mathcal{U}\,\psi$, and reduces to solving a system of *linear equations*. The bounded "until" variant of this algorithm was implemented within Probabilistic Verus [20, 3], a model checker for synchronous parallel randomized programs. The research concerning Probabilistic Verus focuses on semantics and case studies, featuring timing and performance analysis, but does not include support for nondeterminism.

In this paper we consider models for *concurrent probabilistic systems* similar to those of [28, 7, 5] and slightly more general than the concurrent Markov chains of [35, 13]. These 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 probabilily space is determined only for a given *adversary* (or *scheduler*) which selects one of possibly several probability distributions available in each state. Consequently, the logic we define for concurrent probabilistic systems combines features of the branching-time logic CTL with probabilistic reasoning as follows: two variants of "until" are introduced, $\exists\,\mathcal{U}$ and $\forall\,\mathcal{U}$, with quantification defined over all adversaries, and furthermore the probability measure is calculated similarly to the case for PCTL above once an adversary has been specified. The logic we use is Probabilistic Branching-Time Temporal Logic (PBTL) of [5] (it is essentially equivalent to pCTL in [7]). The model checking procedure, already proposed in [5, 7][1] albeit not in conjunction with a symbolic method, reduces to the computation of the *minimum/maximum* probability over the set of simple adversaries, as opposed to exact probability calculation possible in the case of PCTL. 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 [7], this linear programming problem can be solved with the help of e.g. the simplex algorithm, see e.g. [27].

The main contribution of this paper is a *symbolic model checking* method for PBTL over

[1]The model checking algorithms of [5, 7] coincide for the case of verification without fairness constraints. Fairness is needed for the verification of liveness properties and is considered in [5].

concurrent probabilistic systems using MTBDDs, together with an implementation of an experimental tool including an MTBDD-based implementation of simplex as its key component. Though several BDD-based packages are in existence, few support MTBDDs as well as fully fledged matrix manipulation algorithms. We use the Colorado University Decision Diagrams (CUDD) package of Fabio Somenzi [32]. Our tool inputs a sparse matrix description of the system and a PBTL formula and outputs the vector of states for which this formula holds. We defer the design of a system description language till later, concentrating instead on the model checking aspects. The minimum/maximum probability calculation for "until" formulas is implemented in two ways. The first is via the iterative refinement (known to be better for numerical stability, see e.g. [22], but may lead to slower running times if the probability function converges slowly), and the other using *simplex* adapted to MTBDDs.

The use of simplex was already suggested in [7]. Since no symbolic model checking was considered there, this implied the need for explicit state space construction or conversion to the internal representation of some linear algebra package. Coding simplex in terms of MTBDDs avoids unnecessary conversion, and yields the additional advantage that conventional, qualitative, BDD-based symbolic checks can be easily combined together with probabilistic analysis within the same tool. The reason we use simplex in preference to e.g. the elipsolid method is that it offers the best performance in practical cases [23, 27]. Though exponential in the worst case, the examples that exhibit exponential time are considered artificial and are rarely found in practical applications [27]. It has been shown that, for a certain natural probabilistic space, simplex is time polynomial on average [27]. Combined with BDDs, the simplex algorithm has the potential to benefit from the compactness of the representation, though it is difficult to predict its performance compared to sparse matrices. Indeed, our initial experiments are encouraging, though we have not as yet considered numerical stability and error analysis in our implementation. To the authors' knowledge, this is the first MTBDD-based implementation of simplex; if proved efficient in practice, this opens up the possibility of coding other linear algebra operations in terms of MTBDDs, thus making it possible for BDD-based model checkers to incorporate performability analysis.

Related work: Much has been published concerning probabilistic logics and verification, see e.g. [35, 13, 25, 26, 18, 17, 7, 5], but only a few papers deal with BDD-based approaches to probability calculation. Probabilistic Verus [20, 3] offers an extensive C-like system description language, but does not permit nondeterminism. We know of three further approaches based on BDD technology: Algebraic Decision Diagrams (ADDs) [1, 16], Factored Edge-Valued BDDs (FEVBDDs) [34], and Decision Node BDDs (DNBDDs) [30, 31]. Of these, only the latter admits action choice in addition to probabilistic choice; all three significantly differ from the discussion here in that they focus on the calculation of stationary-state probabilities instead of model checking. The ADDs of [1, 16] coincide as data structures with MTBDDs but differ as far as matrix operations are concerned. In [34] FEVBDDs are used to derive an alternative representation of matrices, together with matrix operations, and are applied to Integer Linear Programming resulting in space efficient but slow algorithms. DNDDs [30, 31] offer a data structure for representing stochastic transition diagrams efficiently, but we are not aware of experiments with probability calculations for DNBDDs.

2 Concurrent Probabilistic Systems

In this section, we briefly summarise our underlying model for concurrent probabilistic systems; the reader is referred to [5] for more details. Our model is based on "Markov decision processes"; it slightly generalises "Concurrent Markov Chains" of [35, 13] and essentially coincides with "simple deterministic automata" of [28]. Some familiarity with Markov chains and probability theory is assumed, see e.g. [33].

We begin by recalling standard definition of (discrete) 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 *full path* 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 full paths, whereas $Path_{ful}(s)$ is the set of full paths 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 a scheduler (also known as adversary), 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 $\mu(S)$. Therefore, each $p \in \mu(S)$ is a function $p : S \to [0,1]$ such that $\sum_{s \in S} p(s) = 1$.

Definition 2.1 (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.

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 $0 \le i \le |\pi|$.

We now introduce *adversaries* of concurrent probabilistic systems.

Definition 2.2 (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 $\mathcal{S}$.*

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 $\mu_s \in Steps(s)$ with $A(\rho) = \mu_{last(\rho)}$ where ρ ranges over all finite paths.

It turns out that for model checking it suffices to consider only the simple adversaries[2] 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 [5].

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

With each adversary we associate a sequential 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 $MC^A = (\mathit{Path}^A_{\mathit{fin}}, \mathbf{P}^A)$ is a Markov chain where:

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

3 The Logic PBTL and Its Model Checking Procedure

In this section, based on [5, 7], we recall the syntax and semantics of the probabilistic branching-time temporal logic PBTL and outline the corresponding model checking procedure. PBTL combines features of CTL [11] and PCTL [18]. In particular, the temporal operator $\mathcal{U}$ ("until") and the path quantifiers $\forall$ and $\exists$ are taken from CTL, and the probabilistic operator $[\cdots]_{\geq\lambda}$ 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. By abuse of notation we shall refer to the PBTL structure $(\mathcal{S}, AP, L)$ over the concurrent probabilistic system $\mathcal{S} = (S, \mathit{Steps})$ 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 3.1 (Syntax of PBTL**)** *The syntax of* PBTL *formulas is defined as follows:*

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

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

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

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

Definition 3.2 (Satisfaction Relation for PBTL) *Given a PBTL-structure $(\mathcal{S}, \mathrm{AP}, L)$ and* PBTL *formula ϕ, the satisfaction relation $s \models \phi$ is defined inductively as follows:*

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

As in [5], the above satisfaction relation can be parametrised by a set of adversaries $\mathrm{Adv} \subseteq \mathbf{A}(\mathcal{S})$ of a given type (here $\mathbf{A}(\mathcal{S})$ denotes the set of all adversaries of $\mathcal{S}$), for example the fair or strictly fair adversaries, giving rise to an indexed family of satisfaction relations $\models_{\mathrm{Adv}}$. Excluding fairness, there are obvious similarities between the logic PBTL and the probabilistic logics in the extant literature. The difference with the logic PCTL of [29] is that we label the states with atomic propositions, whereas in [29] action-labelled concurrent probabilistic systems are considered. As already mentioned, the logic pCTL of [7] essentially agrees with PBTL except that pCTL uses a probabilistic operator $\mathbb{P}_*(\cdot)$ for formulas containing the until operator. The PBTL formula $[\ \phi_1 \,\forall\mathcal{U}\, \phi_2\ ]_{\sqsupseteq\lambda}$ can be identified with the pCTL formula $\mathbb{P}_{\sqsupseteq\lambda}(\phi_1 \mathcal{U} \phi_2)$, while $[\ \phi_1 \,\exists\mathcal{U}\, \phi_2\ ]_{>\lambda}$ corresponds to $\neg\mathbb{P}_{\leq\lambda}(\phi_1 \mathcal{U} \phi_2)$.

With the exception of "until" formulas, model checking for PBTL is straightforward, see [7, 5]. To verify $[\phi \,\forall\mathcal{U}\, \psi]_{\sqsupseteq\lambda}$ we first compute the sets of states S_ϕ, S_ψ that satisfy ϕ and ψ respectively. Then we calculate the *minimum probability*:

$$p^{\min}_s(\phi \,\mathcal{U}\, \psi) = \inf\{p^A_s(\phi \,\mathcal{U}\, \psi) | A \text{ an adversary}\}$$

where $p^A_s(\phi \,\mathcal{U}\, \psi) = \mathit{Prob}(\{\pi \mid \pi \in \mathit{Path}^A_{\mathit{ful}}(s) \;\&\; \pi \models \phi \,\mathcal{U}\, \psi\})$ and establish whether the result is $\sqsupseteq \lambda$.

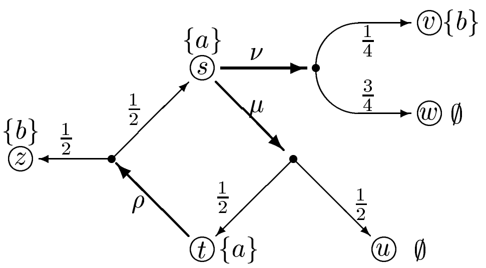
The iterative method: The minimum probability $p^{\min}_s(\phi \,\mathcal{U}\, \psi)$ can be characterized as the following limit where n tends to ∞, and hence can be approximated by iteration [5, 2]. Put $p^{\min}_n(s) = 1$ if $s \in S_\phi$, $p^{\min}_n(s) = 0$ if $s \in S \setminus (S_\phi \cup S_\psi)$, $n = 0, 1, 2, \cdots$. For $s \in S_\phi \setminus S_\psi$ define $p^{\min}_0(s) = 0$ and for $n = 0, 1, 2, \cdots$ put[3]

$$p^{\min}_{n+1}(s) = \min\{\Sigma_{t \in S}\ \mu(t) \cdot p^{\min}_n(t) \mid \mu \in \mathit{Steps}(s)\}$$

The linear optimization method: Alternatively, the values $p^{\min}(s)$ can also be computed by solving linear optimization problems [7, 13, 2]. Let $S^{>0}$ denote the set of states from which one can reach a state in S_ψ via positive probability, and put $S^{\mathrm{no}} = S \backslash S_\psi$, $S^? = S^{>0} \backslash (S^{\mathrm{no}} \cup S_\psi)$. We define $x_s = 1$ if $s \in S_\psi$ and $x_s = 0$ if $s \in S^{\mathrm{no}}$. For the remaning $s \in S^?$, the vector $p^{\min}(s)$ is the solution of the linear maximization problem $0 \leq x_s \leq 1$ and

$$x_s \leq \Sigma_{t \in S}\ \mu(t) \cdot x_t,\ \mu \in \mathit{Steps}(s)$$

[3]In the summation below it is sufficient to sum up over the set $S^? = S^{>0} \setminus (S \cup S_\psi)$ instead of S.



Maximize

$$x_s + x_t$$

subject to the constraints:

$$\begin{array}{rcl} x_s & \leq & \frac{1}{4} \\ x_s - \frac{1}{2}x_t & \leq & 0 \\ -\frac{1}{2}x_s + x_t & \leq & \frac{1}{2} \end{array}$$

Figure 1: A concurrent probabilistic system together with the linear optimization arising from the formula $\phi = [a \; \forall\mathcal{U} \; b]_{>\frac{1}{4}}$.

where $\Sigma_{s \in S^?} x_s$ is maximal.

For the case of $[\phi \; \exists\mathcal{U} \; \psi]_{\sqsupseteq\lambda}$ we calculate the maximum probability $p_s^{\max}(\phi \mathcal{U} \psi)$, which can be defined similarly [5, 2].

Example: Let $\mathcal{S}$ be the PBTL-structure of Figure 1 and $\phi = [a \; \forall\mathcal{U} \; b]_{>\frac{1}{4}}$. The linear programming problem that ϕ gives rise to is also given in Figure 1, and yields $x_s = \frac{1}{4}$ and $x_t = \frac{5}{8}$ as optimal in two iterations of simplex.

4 Representing Concurrent Probabilistic Systems with MTB-DDs

In [12] it is shown how one can generalize BDDs to represent matrices in terms of so-called *multi-terminal* binary decision diagrams (MTBDDs). One can think of MTBDDs as function graphs over boolean variables similar to BDDs, except that they allow finitely many possibly real values as the terminal nodes, instead of just 0 and 1. For the definition of MTBDDs see [12], and BDDs see e.g. [11].

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 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 $f_A : \{1, \ldots 2^m\} \times \{1, \ldots 2^m\} \to D$, where $f_A(i,j) = a_{ij}$, 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 [12].

Matrices can be compactly represented by MTBDDs if they have submatrices that are identical; since we are dealing with very sparse probability matrices this is likely to be the case here. 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, we expect MTBDDs to yield a practical advantage over sparse matrices depending on the degree of regularity of the original matrix.

We now describe the representation of concurrent probabilistic transition systems with MTBDDs, which generalise the labelled Markov chains considered in [4]. Consider a probabilistic transition system with n states that enable at most l probabilistic transitions each. Suppose also that n and l are powers of 2. We can represent the probabilistic transition system as a nl by n matrix T. Let m denote nl. Each row of T represents a single probabilistic transition, 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 have several options, including repeating the last row (adopted here) or padding with all-zero rows (in which case care must be taken so that the zeroes do not interfere with the calculations).

5 PBTL Model Checking with MTBDDs and Simplex

We now outline the symbolic model checking for PBTL formulas based on MTBDDs. We assume that the concurrent probabilistic transition system has been represented as an MTBDD as described above. Given a PBTL formula the algorithm proceeds by inductively converting the formulas to their BDD representation. With the exception of "until" formulas (and also next state not considered here) this is similar to the algorithm presented in [4].

For the PBTL formulas such as $[\psi \; \forall\mathcal{U} \; \psi]_{\sqsupseteq\lambda}$ we implement two methods: the iteration as described in Section 3 (a straightforward limiting process that computes the probability to within a specified ϵ, omitted from this discussion) and by reduction to a linear programming problem which we solve by simplex. We now describe how to generate the linear programming problem.

5.1 Generation of the Linear Programming Problem

First we need to compute the set of states from which under any scheduling there is a non-zero probability of satisfying $\phi \; \mathcal{U} \; \psi$. We start by defining two boolean column vectors $\mathbf{c\_b}_\phi$ and $\mathbf{c\_b}_\psi$ that represent the states satisfying ϕ and ψ, respectively. These correspond to the sets S_ϕ, S_ψ of Section 3. Observe that these vectors are independent of the bits representing transitions, and thus their BDD representations as n vectors and m vectors are the same. In the rest of the section we use $\mathbf{c\_b}_\phi$ and $\mathbf{c\_b}_\psi$ interchangeably as vectors and as functions depending on what is more convenient.

We define the set $REACH(\mathbf{c\_b}_\phi, \mathbf{c\_b}_\psi, T)$ as the minimum fixpoint $\mu Z[R]$ of the relation R defined by

$$R = \lambda x.(\mathbf{c\_b}_\psi[x] \vee (\mathbf{c\_b}_\phi[x] \wedge \forall_k \exists_y (Z[y] \wedge T[x.k, y]))).$$

We anticipate that iterative squaring can be incorporated, although we have not considered this question as yet.

Now we let $\mathbf{c\_b}_{>0}$ denote $REACH(\mathbf{c\_b}_\phi, \mathbf{c\_b}_\psi, T)$, the set of states that satisfy $[\phi \; \forall\mathcal{U} \; \psi]_{>0}$ ($S^{>0}$ of Section 3) and define

$$\mathbf{c\_b}_? = \lambda x.(\mathbf{c\_b}_{>0}[x] \wedge \neg \mathbf{c\_b}_\psi[x])$$

which represents the set of states where the probability of satisfying $\phi \; \mathcal{U} \; \psi$ is still unknown ($S^?$ in Section 3). We also want to identify all the transitions that leave from states of $\mathbf{c\_b}_?$.

To this purpose we define the boolean vector

$$\mathbf{c\_t}_? = \lambda xk.(\mathbf{c\_b}_?[x])$$

Observe that the rules of extending the domain of a function without changing the related BDD representation are still valid. All our arguments are up to such extensions. Indeed now we need to define a boolean matrix that identifies the relevant elements of the transition matrix T:

$$B = \lambda xky.\mathbf{c\_t}_?[x.k] \wedge \mathbf{c\_b}_?[y],$$

which is essentially $\mathbf{c\_t}_? \cdot \mathbf{c\_b}_?{}^t$, where $(\cdot)^t$ denotes transpose. Taking care to match the sizes of T and B, define

$$T' = APPLY(T, B, *)$$

to be the pointwise multiplication between T and B, which represent the set of constraints for the linear programming problem to be solved. Let

$$I = \lambda xky.(x = y)$$

denote the identity matrix on the states, irrespective of the transition enabled. Then we define

$$A = APPLY(I, T', -)$$

A is the matrix part of the tableau. In order to use the the simplex algorithm on A we need to add slack variables: increase (double) the size of A to an m by $2m$ matrix by appending the identity matrix $I_{m,m}$ to the right-hand side of A. It is assumed that the unused columns are identically 0.

Next we compute a column vector called $\mathbf{c\_q}$ (represented on the right- or left-hand side of the matrix in the usual tableau presentation of simplex, see e.g. [23]). Let

$$\mathbf{c\_q} = T\mathbf{c\_b}_\psi^t$$

express the probability that each transition reaches a state satisfying ψ. Recall that each line of T denotes one transition and that $\mathbf{c\_b}_\psi$ denotes the states that satisfy ψ.

The cost function is defined to be $\mathbf{r\_cost} = -\mathbf{c\_b}_?{}^t$, a row vector. The problem that we need to solve is to *maximize* $\mathbf{r\_cost} \cdot x$ under the constraint $Ax = \mathbf{c\_q}$. The vector x will denote the probabilities with which each state satisfies $\phi\, \mathcal{U}\, \psi$ (of course, only the x_i's that represent states).

We start by defining the base with all the slack variables, i.e., a row vector **base** with m 0's followed by m 1's. Then we implement the simplex algorithm with Bland's rule based on [23].

5.2 Implementing Simplex in MTBDDs

One iteration of the simplex algorithm is as follows.

First we find the column entering the basis – this is the column at the position of the first negative element of the cost vector. To do this we build a boolean vector (called **c_enter**) whose only true element (1 element) is the smallest i such that $\mathbf{r\_cost}[i] < 0$. To do this, first we compute the pointwise predicate that tests for negativity, and find the position of the first negative element through a BDD operation.

Next we extract the actual column that enters the basis by computing $\mathbf{c\_enter\_d} = A \cdot \mathbf{c\_enter}$. Then we compute the pointwise division operation between **c_q** and **c_enter_d** where division by 0 gives $\top$ and negative results are treated as $\top$ as well. Finally, we compute the minimum index minimum element of the result of the division, leading to a vector **r_exit** with a structure similar to **c_enter** (**c_enter** has $2m$ elements; **r_exit** has m elements).

Then we extract the vector **r_exit_d** of the row identified by **r_exit** by computing $\mathbf{r\_exit\_d} = \mathbf{r\_exit} \cdot A$. Finally, we compute the vector that identifies the column that leaves the base as $\mathbf{r\_leave\_base} = \mathbf{base} \wedge \mathbf{r\_exit\_d}$ (a value different from 0 is regarded as true; note that only one element of **base** is different from 0 in **r_exit_d**), and we compute the new base as $(\mathbf{r\_leave\_base}\ xor\ \mathbf{base}) \vee \mathbf{c\_enter}$, where all operations are taken pointwise. Note that only one element of **r_leave_base** is not 0.

It remains to change the basis: we know from **c_enter** what column will enter the base and we know from **r_exit** what row will represent the entering column. Our objective is to update the entering column so that all elements except for the element in the position identified by **r_exit** are 0 and the element identified by **r_exit** is 1. Furthermore, we need to update the cost vector so that it has a 0 in the position identified by **c_enter**. This is done through the following steps:

1. Calculate new matrix A of the tableau by subtracting from each row of A the row **r_exit_d** multiplied by an appropriate factor, then normalizing **r_exit_d** so that it has 1 in the position identified by **c_enter**.

2. Compute new cost **r_cost** by subtracting from it the row **r_exit_d** multiplied by an appropriate factor so that it has a 0 in the position identified by **c_enter**.

3. Compute new **c_q** similarly to **r_cost**.

The iteration process terminates as soon as optimal vector is found, i.e. the new **c_enter** is zero.

5.3 Computing Validity of "until" Formulas

Finally, given the solution vector x of probabilities we need to compute the validity of the initial formula $[\phi\ \forall\mathcal{U}\ \psi]_{\sqsupseteq\lambda}$. Taking appropriate care of the slack variables, this can be achieved by computing pointwise the operation $x[i] \geq \lambda$, which is an ordinary BDD operation.

The case of formulas of the form $[\phi\ \exists\mathcal{U}\ \psi]_{\sqsupseteq\lambda}$ can be dealt with by transposing the matrix A[4], the extra column and the cost function, then using the same simplex algorithm for thus reformulated problem which becomes: maximize $\mathbf{c\_q}^t \cdot y$ subject to the constraints $A^t y = \mathbf{r\_cost}^t$ (where A^t takes appropriate care of slack variables).

Example: For the PBTL-structure $\mathcal{S}$ of Figure 1 and $\phi = [a\ \exists\mathcal{U}\ b]_{>\frac{1}{4}}$. The linear programming problem that ϕ gives rise to is: minimize

$$x_s + x_t$$

[4]Technically speaking, since A already contains slacks, we transpose its left-hand side half only, excluding the slack variables, and add new slacks.

subject to the constraints:

$$\begin{array}{lcll} x_s & \geq & \frac{1}{4} & (y_0) \\ x_s - \frac{1}{2}x_t & \geq & 0 & (y_1) \\ -\frac{1}{2}x_s + x_t & \geq & \frac{1}{2} & (y_2) \end{array}$$

and yields $x_s = \frac{1}{3}$ and $x_t = \frac{2}{3}$ as optimal. When reformulated to the dual problem, this reduces to: maximize

$$\frac{1}{4}y_0 + \frac{1}{2}y_2$$

subject to the constraints:

$$\begin{array}{lcl} y_0 + y_1 - \frac{1}{2}y_2 & \leq & 1 \\ -\frac{1}{2}y_1 + y_2 & \leq & 1 \end{array}$$

which can be solved by the same method as before. The values for x_s and x_t can be read off the optimal tableau.

6 Experimental Results

We have implemented an experimental symbolic model checking tool in terms of the Colorado University Decision Diagrams (CUDD) package of Fabio Somenzi [32]. This package provides support for BDDs and MTBDDs/ADDs. ADDs and MTBDDs are the same as data structures, but differ in terms of the associated matrix multiplication; both the ADD matrix multiplication as given in [1] and the times-plus method proposed by E. Clarke are supplied with the package. The CUDD package uses its own memory management, and hence its memory requirement is relatively high (typically over 10MB). The running times, on the other hand, are fast (i.e. a fraction of a second for small examples run on Ultra 10).

Our tool accepts as input a real-valued, sparse matrix description of a concurrent probabilistic systems (we defer the implementation of the system description language till later, in the meantime using the TIPPtool [21] to construct case studies) and a PBTL formula ϕ, and outputs the states that are satisfied by ϕ. Our implementation relies on the matrix-by-matrix multiplication supplied with the CUDD package, and the derived matrix-by-vector multiplication. We work with floating point arithmetic, although we anticipate the algorithms can be modified to cater for rationals represented by integer fractions that would guarantee better precision.

The model checking of unbounded "until" properties has been implemented by two methods. The first is by iterative refinement, which has the advantages that it can start with any approximation, relies only on the matrix-by-vector multiplication, does not change the matrix representing the system, is numerically stable (has no build up of round off errors) and admits improvements to numerical precision. However, it can be inefficient in cases of slow convergence. The alternative, simplex, method is numerically not as stable as it involves subtraction, and also destroys the original matrix.

We have tested the simplex component of our tool on several small and medium-size examples (such as example in Figure 1), in each case comparing the performance of our algorithm with the Maple implementation of simplex. The largest example of a concurrent probabilistic system we have tried so far is the three randomized dining philosophers of [25] (of size 891^2). For floating-point arithmetic, this produced an answer on an Ultra 10 with 360MB virtual memory in approx. 20 secs versus 1,200 secs in Maple with similar round-off errors (though Maple also allows improved precision via fractional representation).

Shortly we intend to investigate larger examples (for example, the five dining philosophers and Rabin's mutual exclusion), and also consider the effect of variable ordering on performance. The issue of numerical stability and accuracy needs to be addressed, and likewise efficiency improvements to the simplex such as the revised simplex method [27].

7 Conclusion

We have proposed MTBDD-based symbolic model checking for the probabilistic branching-time temporal logic PBTL of [7, 5], and implemented an experimental tool using the CUDD package, with the view to investigate the feasibility of an integrated tool combining BDD-based model checking with probabilistic and performance analysis. Our initial experiments point to potential practical advantage in using MTBDDs as a representation for very sparse probability matrices. A generalisation of the model checking procedure to incorporate fairness, which is necessary when verifying liveness properties, following the method of [5] recently improved in [2] or [15, 14], is straightforward.

There are many efficiency improvements that are worth investigating, to mention heuristics concerning the variable ordering, caching policy, the use of iterative squaring, and further known improvements to the simplex method [27]. In another direction, numerical stability and accuracy of the results should be addressed following e.g. [22]. Furthermore, we would like to implement an algorithm for computing stationary state probabilities for labelled Markov chains.

Finally, we believe that the MTBDD version of the simplex algorithm that we have implemented is of independent interest. It would be interesting to establish what other linear algebra operations can be coded with MTBDDs, and for which classes of matrices they offer an efficient representation.

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 *Preliminary Proceedings, PROBMIV'98*, 1998. Technical Report CSR-98-4, University of Birmingham.

[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 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] 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.

[8] 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.

[9] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[10] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. Proceedings of the Fifth Annual Symposium on Logic in Computer Science, June 1990.

[11] 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.

[12] 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.

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

[14] L. de Alfaro. How to specify and verify the long-run average behaviour of probabilistic systems. In *Proc. 13th IEEE Symp. Logic in Computer Science*, pages 454–465, 1998.

[15] L. de Alfaro. Stochastic transition systems. In *Proc. CONCUR'98: Concurrency Theory*, Lecture Notes in Computer Science. Springer-Verlag, 1998.

[16] G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Markovian analysis of large finite state machines. *IEEE CAD*, 15(12):1479–1493, Dec. 1996.

[17] H. Hansson. *Time and Probability in Formal Design of Distributed Systems.* Elsevier, 1994.

[18] H. Hansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, 6:512–535, 1994.

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

[20] V. Hartonas-Garmhausen. *Probabilistic Symbolic Model Checking with Engineering Models and Applications.* PhD thesis, Carnegie Mellon University, 1998.

[21] H. Hermanns, V. Mertsiotakis, and M. Rettelbach. A construction and analysis tool based on the Stochastic Process Algebra TIPP. In *Proceedings, TACAS Workshop*, volume 1055 of *Lecture Notes in Computer Science.* Springer-Verlag, 1996.

[22] N. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM, 1996.

[23] H. Karloff. *Linear Programming.* Birkhauser, 1991.

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

[25] A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1:53–72, 1986.

[26] A. Pnueli and L. Zuck. Probabilistic verification. *Information and Computation*, 103:1–29, 1993.

[27] A. Schrijver. *Theory of Linear and Integer Programming.* J. Wiley and Sons, 1986.

[28] R. Segala. *Modelling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, MIT, 1995.

[29] 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.

[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, Williamsburg, VA, September 1998.

[31] 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.

[32] F. Somenzi. CUDD: CU decision diagram package. Public software, Colorado University, Boulder, 1997.

[33] W. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton University Press, 1994.

[34] P. Tafertshofer and M. Pedram. Factored edge-valued binary-decision diagrams. *Formal Methods in Systems Design*, 10:137–164, 1997.

[35] M. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings, FOCS'85*, pages 327–338. IEEE Press, 1987.