Symbolic Model Checking for Probabilistic Processes

Christel Baier[1], Edmund M. Clarke[2*], Vasiliki Hartonas-Garmhausen[2], Marta Kwiatkowska[3] and Mark Ryan[3**]

[1] Fakultät für Mathematik & Informatik
Universität Mannheim
68131 Mannheim, Germany
baier@pi1.informatik.uni-mannheim.de

[2] Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{emc,hartonas}@cs.cmu.edu

[3] School of Computer Science
University of Birmingham
Birmingham B15 2TT, UK
{mzk,mdr}@cs.bham.ac.uk

Abstract. We introduce a symbolic model checking procedure for Probabilistic Computation Tree Logic PCTL over labelled Markov chains as models. Model checking for probabilistic logics typically involves solving linear equation systems in order to ascertain the probability of a given formula holding in a state. Our algorithm is based on the idea of representing the matrices used in the linear equation systems by Multi-Terminal Binary Decision Diagrams (MTBDDs) introduced in Clarke *et al* [14]. Our procedure, based on the algorithm used by Hansson and Jonsson [24], uses BDDs to represent formulas and MTBDDs to represent Markov chains, and is efficient because it avoids explicit state space construction. A PCTL model checker is being implemented in Verus [9].

1 Introduction

Probabilistic techniques, and in particular probabilistic logics, have proved successful in the specification and verification of systems that exhibit uncertainty, such as fault-tolerant systems, randomized distributed systems and communication protocols. Models for such systems are variants of probabilistic automata (such as labelled Markov chains used in e.g. [24, 34, 35, 17]), in which the usual (boolean) transition relation is replaced with its probabilistic version given in the form of a Markov probability transition matrix. The probabilistic logics are typically obtained by "lifting" a non-probabilistic logic to the probabilistic case by constructing for each formula ϕ and a real number p in the $[0, 1]$-interval the formula $[\phi]_{\geq p}$ in which p acts as a *threshold for truth* in the sense that for the formula $[\phi]_{\geq p}$ to be satisfied (in the state s) the probability that ϕ holds in s must be *at least* p (see [26, 32, 25] for a different approach). With such logics one can express *quantitative* properties such as "the probability of the message being delivered within t time steps is at least 0.75" (see e.g. the timing or average-case analysis of real-time or randomized distributed systems [24, 23, 5, 6, 2]) or (the more prevalent) *qualitative* properties, for which ϕ is required to be satisfied by almost all executions (which amounts to showing that ϕ is satisfied with probability 1, see e.g. [1, 17, 23, 24, 21, 22, 29, 30, 34]).

* This research was sponsored in part by the National Science Foundation under grant no. CCR-8722633, by the Semiconductor Research Corporation under contract 92-DJ-294, and by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, the Advanced Research Projects Agency (ARPA) under grant F33615-93-1-1330.

** This research was sponsored in part by the European Union ESPRIT projects ASPIRE and FIREworks, British Telecom, and the Nuffield Foundation.

Much has been published concerning the verification methods for probabilistic logics. Probabilistic extensions of dynamic logic [26] and temporal and modal logics, e.g. [2, 6, 17, 24, 21, 27, 30, 31, 34], and automatic procedures for checking satisfaction for such logics have been proposed. The latter are based on reducing the calculation of the probability of formulas being satisfied to a linear algebra problem: for example, in [24], the calculation of the probability of 'until' formulas is based on solving the linear equation system given by an $n \times n$ matrix where n is the size of the state space. Optimal methods are known (for sequential Markov chains, the lower bound is single exponential in the size of the formula and polynomial in the size of the Markov chain [18]), but these algorithms are not of much practical use when verifying realistic systems. As a result, efficiency of probabilistic analysis lags behind efficient model checking techniques for conventional logics, such as symbolic model checking [11, 12, 10, 8, 15, 28], for which tools capable of tackling industrial scale applications are available (cf. `smv`). This is undesirable as probabilistic approaches allow one to establish that certain properties hold (in some meaningful probabilistic sense) where conventional model checkers fail, either because the property simply is not true in the state (but holds in that state with some acceptable probability), or because exhaustive search of only a portion of the system is feasible.

The main difficulty with current probabilistic model checking is the need to integrate a linear algebra package with a conventional model checker. Despite the power of existing linear algebra packages, this can lead to inefficient and time consuming computation through the implicit requirement for the construction of the state space. This paper proposes an alternative, which is based on expressing the probability calculations in terms of Multi-Terminal Binary Decision Diagrams (MTBDDs) [16]. MTBDDs are a generalization of (ordered) BDDs in the sense that they allow arbitrary real numbers in the terminal nodes instead of just 0 and 1, and so can provide a compact representation for matrices. As a matter of fact, in [13] MTBDDs have been shown to perform *no worse* than sparse matrices. Thus, converting to MTBDDs ensures smooth integration with a symbolic model checker such as `smv` and has the potential to outperform sparse matrices due to the compactness of the representation, in the same way as BDDs have outperformed other methods. As with BDDs, the precise time complexity estimates of model checking for MTBDDs are difficult to obtain, but the success of BDDs in practice [8, 28] serves as sufficient encouragement to develop the foundations of MTBDD-based probabilistic model checkers.

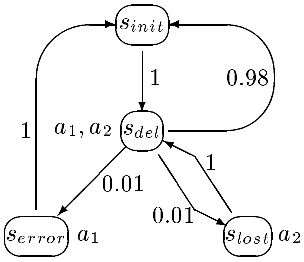
In this paper we consider a probabilistic extension of CTL called Probabilistic Computation Tree Logic (PCTL), and give a *symbolic* model checking procedure which avoids the explicit construction of the state space. We use finite-state labelled Markov chains as models. The model checking procedure is based on that of [24, 18], but we use BDDs to represent the boolean formulas, and a suitable combination of BDDs and MTBDDs for probabilistic formulas. Currently, we are implementing the PCTL symbolic model checking in Verus [9]. For reasons of space we omit much detail from this paper, which will be reported in [4]. We assume some familiarity with BDDs, automata on infinite sequences, probability and measure theory [8, 33, 20].

2 Labelled Markov chains

We use discrete time Markov chains as models (we do not consider nondeterminism). Let AP denote a finite set of atomic propositions. A *labelled Markov chain* over a set of atomic propositions AP is a tuple $M = (S, \mathbf{P}, L)$ where S is a finite set of *states*, $\mathbf{P} : S \times S \to [0, 1]$ a *transition matrix*, i.e. $\sum_{t \in S} \mathbf{P}(s, t) = 1$ for all $s \in S$, and $L : S \to 2^{AP}$ a *labelling function* which assigns to each state $s \in S$ a set of atomic propositions. We assume that there are 2^n states for some n, and that there are sufficiently many atomic propositions to distinguish them (i.e. $L(s) \neq L(s')$ for all states s, s' with $s \neq s'$). Any labelled Markov chain may be transformed into one satisfying these conditions by adding dummy states and new propositions.

Execution sequences arise by resolving the probabilistic choices. Formally, an *execution sequence* in M is a nonempty (finite or infinite) sequence $\pi = s_0 s_1 s_2, \ldots$ where s_i are states and $\mathbf{P}(s_{i-1}, s_i) > 0$, $i = 1, 2, \ldots$. The first state of π is denoted by $first(\pi)$. $\pi(k)$ denotes the $k + 1$-th state of π. An execution sequence π is also called a *path*, and a *full path* iff it is infinite. $Path_\omega(s)$ is the set of full paths π with $first(\pi) = s$. For $s \in S$, let $\Sigma(s)$ be the smallest σ-algebra on $Path_\omega(s)$ which contains the basic cylinders $\{\pi \in Path_\omega(s) : \rho \text{ is a prefix of } \pi\}$ where ρ ranges over all finite execution sequences starting in s. The probability measure $Prob$ on $\Sigma(s)$ is the unique measure with $Prob\{\, \pi \in Path_\omega(s) \ : \rho \text{ is a prefix of } \pi \,\} = \mathbf{P}(\rho)$ where $\mathbf{P}(s_0 s_1 \ldots s_k) = \mathbf{P}(s_0, s_1) \cdot \mathbf{P}(s_1, s_2) \cdot \ldots \cdot \mathbf{P}(s_{k-1}, s_k)$.

Example 1. We consider a simple communication protocol similar to that in [24]. The system consists of three entities: a sender, a medium and a receiver. The sender sends a message to the medium, which in turn tries to deliver the message to the receiver. With probability $\frac{1}{100}$, the messages get lost, in which case the medium tries again to deliver the message. With probability $\frac{1}{100}$, the message is corrupted (but delivered); with probability $\frac{98}{100}$, the correct message is delivered. When the (correct or faulty) message is delivered the receiver acknowledges the receipt of the message. For simplicity, we assume that the acknowledgement cannot be corrupted or lost. We describe the system in a simplified way where we omit all irrelevant states (e.g. the state where the receiver acknowledges the receipt of the correct message).



We use the following four states:

- s_{init} the state in which the sender passes the message to the medium
- s_{del} the state in which the medium tries to deliver the message
- s_{lost} the state reached when the message is lost
- s_{error} the state reached when the message is corrupted

The transition $s_{del} \to s_{init}$ stands for the acknowledgement of the receipt of the correct message, $s_{error} \to s_{init}$ for the acknowledgement of the receipt of the corrupted message. We use two atomic propositions a_1, a_2 and the labelling function $L(s_{init}) = \emptyset$, $L(s_{del}) = \{a_1, a_2\}$, $L(s_{lost}) = \{a_2\}$, $L(s_{error}) = \{a_1\}$. ■

3 Probabilistic branching time temporal logic

In this section we present the syntax and semantics of the logic PCTL (Probabilistic Computation Tree Logic) introduced by Hansson & Jonsson [24][4]. PCTL is a probabilistic extension of CTL which allows one to express quantitative properties of probabilistic processes such as "the system terminates with probability at least 0.75". PCTL contains atomic propositions and the operators: next-step X and until U. The operators X and U are used in connection with an interval of probabilities. The syntax of PCTL is as follows:

$$\Phi \ ::= \ tt \ | \ a \ | \ \Phi_1 \wedge \Phi_2 \ | \ \neg\Phi \ | \ [\, X\Phi \,]_{\sqsupseteq p} \ | \ [\, \Phi_1 U \Phi_2 \,]_{\sqsupseteq p}$$

where a is an atomic proposition, $p \in [0,1]$, $\sqsupseteq$ is either $\geq$ or $>$. Formulas of the form $X\Phi$ or $\Phi_1 U \Phi_2$, where Φ, Φ_1, Φ_2 are PCTL formulas, are called *path formulas*. PCTL formulas are interpreted over the states of a labelled Markov chain, whereas path formulas are interpreted over paths. The subscript $\sqsupseteq p$ denotes that the probability of paths starting in the current state fulfilling the path formula is $\sqsupseteq p$. Thus, PCTL is like CTL, except that the path operators A and E in CTL have been replaced by the operator $[\,\cdot\,]_{\sqsupseteq p}$. The usual derived constants and operators are: $ff = \neg tt$, $\Phi_1 \vee \Phi_2 = \neg(\neg\Phi_1 \wedge \neg\Phi_2)$, $\Phi_1 \rightarrow \Phi_2 = \neg\Phi_1 \vee \Phi_2$. Operators for modelling "eventually" or "always" can be derived by: $[\Diamond\Phi]_{\geq p} = [tt U \Phi]_{\geq p}$, $[\Box\Phi]_{\geq p} = \neg[\Diamond\neg\Phi]_{>1-p}$, and similarly for $[\cdot]_{>p}$.

Let $M = (S, \mathbf{P}, L)$ be a labelled Markov chain. The satisfaction relation $\models \subseteq S \times PCTL$ is given by

$s \models tt$ for all $s \in S$ $\qquad$ $s \models \Phi_1 \wedge \Phi_2$ iff $s \models \Phi_1$ and $s \models \Phi_2$

$s \models a$ iff $a \in L(s)$ $\qquad$ $s \models \neg\Phi$ iff $s \not\models \Phi$

$s \models [X\Phi]_{\sqsupseteq p}$ iff $Prob\,\{\pi \in Path_\omega(s) : \pi \models X\Phi\} \ \sqsupseteq \ p$

$s \models [\Phi_1 U \Phi_2]_{\sqsupseteq p}$ iff $Prob\,\{\pi \in Path_\omega(s) : \pi \models \Phi_1 U \Phi_2\} \sqsupseteq p$

$\pi \models X\Phi$ iff $\pi(1) \models \Phi$

$\pi \models \Phi_1 U \Phi_2$ iff there exists $k \geq 0$ with $\pi(i) \models \Phi_1, i = 0, 1, \ldots, k-1$ and $\pi(k) \models \Phi_2$.

For a path formula f the set $\{\pi \in Path_\omega(s) : \pi \models f\}$ is measurable [34, 18]. If $s \models \Phi$ then we say s satisfies Φ (or Φ holds in s). The truth value of formulas involving the linear time quantifiers $\Diamond$ and $\Box$ can be derived:

$$s \models [\Diamond\Phi]_{\sqsupseteq p} \text{ iff } Prob\{\pi \in Path_\omega(s) : \pi(k) \models \Phi \text{ for some } k \geq 0\} \ \sqsupseteq \ p$$

$$s \models [\Box\Phi\,]_{\sqsupseteq p} \text{ iff } Prob\{\pi \in Path_\omega(s) : \pi(k) \models \Phi \text{ for all } k \geq 0\} \ \sqsupseteq \ p.$$

Given a probabilistic process $\mathcal{P}$, described by a labelled Markov chain $M = (S, \mathbf{P}, L)$ with an initial state s, we say $\mathcal{P}$ satisfies a PCTL formula Φ iff $s \models \Phi$. For instance, if a is an atomic proposition which stands for termination and $\mathcal{P}$ satisfies $[\Diamond a]_{\geq p}$ then $\mathcal{P}$ terminates with probability at least p.

4 Multi-terminal binary decision diagrams

Ordered Binary Decision Diagrams (BDDs) [7, 8, 15, 28] are a compact representation of boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}$. They are based on the canonical representation of the binary tree of the function as a directed graph obtained through folding

[4] For simplicity we omit the bounded 'until' operator of [24].

internal nodes representing identical subfunctions (subject to an ordering of the variables to guarantee uniqueness of the representation) and using 0 and 1 as leaves. In [16] it is shown how one can generalize BDDs to cogently and efficiently represent matrices in terms of so-called *multi-terminal* binary decision diagrams (MTBDDs).

Formally, MTBDDs can be defined as follows. Let $x_1, \ldots, x_n$ be distinct variables, which we order by $x_i < x_j$ iff $i < j$. A *multi-terminal binary decision diagram* (MTBDD) 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. A suitable adaptation of the operator $REDUCE(\cdot)$ [7] yields an operator which accepts an MTBDD as its input and returns the corresponding reduced MTBDD.

Each MTBDD Q over $\{x_1, \ldots, x_n\}$ represents a function $F_Q : \{0,1\}^n \to I\!R$, and, vice versa, each function $F : \{0,1\}^n \to I\!R$ can be described by a unique reduced MTBDD over $(x_1, \ldots, x_n)$. In the sequel, by the MTBDD for a function $F : \{0,1\}^n \to I\!R$ we mean the unique reduced MTBDD Q with $F_Q = F$. If all terminal vertices are labelled by 0 or 1, i.e. if the associated function F_Q is a boolean function, the MTBDD specializes to a BDD over $(x_1, \ldots, x_n)$.

MTBDDs are used to represent D–valued matrices as follows. Consider a $2^m \times 2^m$–matrix A. 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 $f_A(i,j) = a_{ij}$. 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 $f : \{0,1\}^m \to D$ where $f(x,y) = f_A(c(x), c(y))$ for $x = (x_1 \ldots x_m)$ and $y = (y_1 \ldots y_m)$. This transformation now allows matrices to be represented as MTBDDs. In order to obtain an efficient MTBDD–representation, the variables of f are permuted. Instead of the MTBDD for $f(x_1 \ldots x_m, y_1 \ldots y_m)$, we use the MTBDD obtained from $f(x_1, y_1, x_2, y_2, \ldots x_m, y_m)$. This convention imposes a recursive structure on the matrix from which efficient recursive algorithms for all standard matrix operations are derived [16].

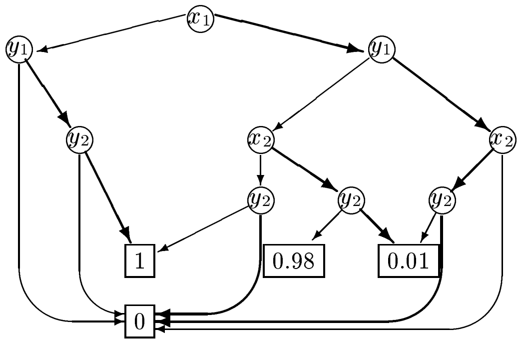
4.1 Representing labelled Markov chains by MTBDDs

To represent the transition matrix of a labelled Markov chain by a MTBDD we abstract from the names of states and instead, similarly to [8, 15], use binary tuples of atomic propositions that are true in the state. Let $M = (S, \mathbf{P}, L)$ be a labelled Markov chain. We fix an enumeration $a_1, \ldots, a_n$ of the atomic propositions and identify each state s with the boolean n-tuple $e(s) = (b_1, \ldots, b_n)$ where $b_i = 1$ iff $a_i \in L(s)$. In what follows, we identify $\mathbf{P}$ with the function $F : \{0,1\}^{2n} \to [0,1]$, $F(x_1, y_1, \ldots, x_n, y_n) = \mathbf{P}((x_1, \ldots, x_n), (y_1, \ldots, y_n))$, and represent M by the MTBDD for $\mathbf{P}$ over $(x_1, y_1, \ldots, x_n, y_n)$. The associated MTBDD is denoted by P.

Example 2. For the system in Example 1 we use the encoding $e(s_{init}) = 00$, $e(s_{del}) = 11$, $e(s_{lost}) = 01$ $e(s_{error}) = 10$. The values of the matrix $\mathbf{P}$, the function F and the MTBDD P for F are are given by:

	00	01	10	11
00	0	0	0	1
01	0	0	0	1
10	1	0	0	0
11	$\frac{98}{100}$	$\frac{1}{100}$	$\frac{1}{100}$	0

$$F(x_1, y_1, x_2, y_2) = \begin{cases} 1 & : \text{if } x_1y_1x_2y_2 \in \{0101, 0111, 1000\} \\ \frac{1}{100} & : \text{if } x_1y_1x_2y_2 \in \{1011, 1110\} \\ \frac{98}{100} & : \text{if } x_1y_1x_2y_2 = 1010 \\ 0 & : \text{otherwise.} \end{cases}$$



(The thick lines stand for the "right" edges, the thin lines for the "left" edges.) ■

4.2 Operators on MTBDDs

Our model checking algorithm makes use of several operators on MTBDDs proposed in Bryant [7] and Clarke *et al* [14]. We briefly describe them below.

Operator $BDD(\cdot)$**:** takes an MTBDD Q and an interval I, and returns the BDD representing the function $F(\overline{x}) = 1$ if $F_Q(\overline{x}) \in I$, else $F(\overline{x}) = 0$. We obtain $B = BDD(Q, I)$ from Q by changing the values of the terminal vertices (into 1 or 0 depending on whether or not $value(v) \in I$) and applying Bryant's reduction procedure $REDUCE(\cdot)$. We write $BDD(Q, > p)$ rather than $BDD(Q,]p, \infty[)$ and $BDD(Q, \geq p)$ rather than $BDD(Q, [p, \infty[)$.

Operator $APPLY(\cdot)$**:** allows elementwise application of the binary operator *op* to two MTBDDs. If *op* is a binary operator on reals (e.g. multiplication $*$ or minus $-$) and Q_1, Q_2 are MTBDDs over $\overline{x}$ then $APPLY(Q_1, Q_2, op)$ yields a MTBDD over $\overline{x}$ which represents the function $f(\overline{x}) = f_{Q_1}(\overline{x}) \; op \; f_{Q_2}(\overline{x})$.

Operator $COMPOSE^k(\cdot)$**:** This operator allows the composition of a real function $F : \{0,1\}^{n+k} \to I\!R$ and boolean functions $G_i : \{0,1\}^n \to \{0,1\}$, $i = 1, \ldots, k$ giving $H(\overline{x}) = F(\overline{x}, G_1(\overline{x}), \ldots, G_k(\overline{x}))$.

Matrix and vector operators: The standard operations on matrices and vectors have corresponding operations on the MTBDDs that represent them [13]. If MTBDDs A and Q over $2n$ and n variables represent the matrix **A** and vector **q** respectively, then $MV\_MULTI(A, Q)$ denotes the MTBDD over n variables that represents the vector $\mathbf{A} \cdot \mathbf{q}$.

Operator $SOLVE(\cdot)$: [8] presents a method to decompose a regular matrix **A** into a lower and upper triangular matrices and a permutation matrix. Using this LU-decomposition we can obtain an operator $SOLVE(A, Q)$ that takes as its input a MTBDD A over $2n$ variables where the corresponding matrix **A** is regular and a MTBDD Q over n variables which represents a vector **q**, and returns a MTBDD Q' over n variables which represents the unique solution of the linear equation system $\mathbf{A} \cdot \mathbf{x} = \mathbf{q}$. Alternatively, we can use iterative techniques to solve the equations; our experiments indicate that this performs better.

4.3 Description of (MT)BDDs by relational terms of the μ-calculus

We will use the μ-calculus as a notation for describing (MT)BDDs. In the algorithm in the next section, all our (MT)BDDs are either over $2n$ variables (in which case they represent $2^n \times 2^n$ matrices), or over n variables (in which case they represent vectors of length 2^n). For example, if B, C are BDDs over n variables and $\overline{u} = (u_1, \ldots, u_n)$, $\overline{v} = (v_1, \ldots, v_n)$, then $D = \lambda \overline{u}\,\overline{v}\ [B(\overline{u}) \wedge C(\overline{v})]$ is a BDD over $2n$ variables; if B, C represent the vectors $(b_i)_{1 \leq i \leq n}$ and $(c_i)_{1 \leq i \leq n}$ respectively, then D represents the matrix whose element in the ith row and jth column is $b_i \wedge c_j$. The BDD $E = \lambda \overline{u}\ [B(\overline{u}) \wedge C(\overline{u})]$ is a BDD over n variables, representing the vector $(b_i \wedge c_i)_{1 \leq i \leq n}$.

We write $TRUE$ for the BDD over n variables which returns 1 in all cases of its arguments. We write $\neg B$ instead of $\lambda \overline{x}[\neg B(\overline{x})]$, and $B_1 \wedge B_2$ for the BDD $\lambda \overline{x}[B_1(\overline{x}) \wedge B_2(\overline{x})]$. If $\overline{x} = (x_1, \ldots, x_n)$, $\overline{y} = (y_1, \ldots, y_n)$ then $\overline{x} = \overline{y}$ abbreviates the formula $\bigwedge_{1 \leq i \leq n}(x_i \leftrightarrow y_i)$.

We require one further operator. If the labelled Markov chain $M = (S, \mathbf{P}, L)$ is represented by a MTBDD P as described in Section 4.1, and B_1, B_2 are BDDs that represent the characteristic functions of subsets S_1, S_2 of S, then $REACH(B_1, B_2, BDD(P, > 0))$ represents the set of states $s \in S$ from which there exists an execution sequence $s = s_0, s_1, \ldots, s_k$ with $k \geq 0$ and $s_0, \ldots, s_{k-1} \in S_1$, $s_k \in S_2$, and which is used in the operator $UNTIL(\cdot)$ defined in Section 5.

Operator $REACH(\cdot)$ Let B_1, B_2 be BDDs with n variables and T a BDD with $2n$ variables. We define $REACH(B_1, B_2, T)$ to be the BDD over n variables which is given by the μ-calculus formula $\mu Z\ \lambda \overline{x}\, [B_2(\overline{x}) \vee (B_1(\overline{x}) \wedge \exists \overline{y}[Z(\overline{y}) \wedge T(\overline{x}, \overline{y})])]$. This operator uses the method of [8] to obtain the BDD for a term involving the least fixed point operator μ.

5 Model checking for PCTL

Our model checking algorithm for PCTL is based on established BDD techniques (i.e. converting boolean formulas to their BDD representation), which it combines with a new method, namely expressing the probability calculation for the probabilistic formulas in terms of MTBDDs. In the case of $[X\Phi]_{\sqsupseteq p}$ the probability is calculated by multiplying the transition matrix by the boolean vector set to 1 iff the state satisfies Φ, whereas for $[\Phi_1 U \Phi_2]_{\sqsupseteq p}$ we derive an operator called $UNTIL(\cdot)$, based on [24], which we express in terms of MTBDDs.

Let $M = (S, \mathbf{P}, L)$ be a labelled Markov chain which is represented by a MTBDD P over $2n$ variables as described in Section 4.1. For each PCTL formula Φ, we define a BDD $B[\Phi]$ over $\overline{x} = (x_1, \ldots, x_n)$ that represents $Sat(\Phi) = \{s \in S : s \models \Phi\}$. We compute the BDD representation $B[\Phi]$ of a PCTL formula Φ by structural induction:

$B[tt] = TRUE \qquad B[a_i] = \lambda \overline{x}\, [x_i]$

$B[\neg \Phi] = \neg B[\Phi] \qquad B[\Phi_1 \wedge \Phi_2] = B[\Phi_1] \wedge B[\Phi_2]$

$B[\,[X\Phi]_{\sqsupseteq p}\,] \;=\; BDD\,(\,MV\_MULTI(P, B[\Phi]),\; \sqsupseteq p\,)$
$B[\,[\Phi_1 U \Phi_2]_{\sqsupseteq p}\,] \;=\; BDD\,(\,UNTIL(B[\Phi_1], B[\Phi_2], P), \sqsupseteq p\,)\,)$
The operator $UNTIL(B[\Phi_1], B[\Phi_2], P)$ assigns to each state $s \in S$ the *probability* of the set of full paths from s satisfying $\Phi_1 U \Phi_2$; formally, it represents the function $S \to [0,1]$, $s \mapsto p_s$, where $p_s \;=\; Prob\{\pi \in Path_\omega(s) : \pi \models \Phi_1 U \Phi_2\}$. Our method for computing p_s is based on the partition of S introduced in [24, 18], but we must compute with BDDs. We first compute the set $V = \{s \in S : p_s > 0\}$ and then set $V' = V \setminus Sat(\Phi_2)$. We then have: $p_s = 1$ if $s \models \Phi_2$; $p_s = 0$ if $s \notin V$; and for the remaining cases (i.e. those such that $s \in V'$)

$$p_s = \sum_{t \in V'} \mathbf{P}(s,t) \cdot p_t + \sum_{t \in Sat(\Phi_2)} \mathbf{P}(s,t) \cdot p_t + \sum_{t \in S \setminus V} \mathbf{P}(s,t) \cdot p_t.$$

In the second term, each $p_t = 1$ and in the third term, each $p_t = 0$. Therefore p_s ($s \in V'$) satisfies a $|V'|$-dimensional equation system of the form $\mathbf{x} = \mathbf{A}\,\mathbf{x} + \mathbf{b}$, or equivalently $(\mathbf{I} - \mathbf{A})\,\mathbf{x} = \mathbf{b}$ where $\mathbf{I}$ is the $|V'| \times |V'|$ identity matrix. One can show this system has a unique solution using the method in [24, 18].

We now demonstrate how $UNTIL(\cdot)$ can be expressed in terms of MTBDDs. Let $B_i = B[\Phi_i], i = 1,2$. The set V is given by the BDD $B = REACH(B_1, B_2, BDD(P, > 0))$, V' by $B' = \lambda\overline{x}\,[B(\overline{x}) \wedge \neg B_2(\overline{x})]$. In order to avoid the BDD for the "new" transition matrix $\mathbf{A}$ with $\lceil \log_2 |V'| \rceil$ variables, we instead reformulate the equation in terms of the matrix $\mathbf{P}' = (p'_{s,t})_{s,t \in S}$ which is given by: $p'_{s,t} = \mathbf{P}(s,t)$ if $s, t \in V'$ and $p'_{s,t} = 0$ in all other cases. The MTBDD P' for $\mathbf{P}'$ can be obtained from the MTBDD P representing the Markov transition matrix. The following lemma shows that $\mathbf{I} - \mathbf{P}'$ is regular (we omit the proof).

Lemma 1. *Let V', $\mathbf{P}'$, $\mathbf{I}$ be as as above. Then, $\mathbf{I} - \mathbf{P}'$ is regular. The unique solution $\mathbf{x} = (x_s)_{s \in S}$ of the linear equation system $(\mathbf{I} - \mathbf{P}') \cdot \mathbf{x} = \mathbf{q}$ where $\mathbf{q} = (q_s)$, $q_s = \sum_{t \in Sat(\Phi_2)} \mathbf{P}(s,t)$ satisfies: $x_s = p_s$ if $s \in V'$.*

The algorithm for the operator $UNTIL(\cdot)$ is shown in Figure 1. It first calculates the MTBDDs B and B', for V and V'. B^2 is used as a mask to obtain P' from P; it sets to 0 the entries not corresponding to states in V'. We next calculate the MTBDD Q for the vector $\mathbf{q}$, and use the operator $SOLVE(\cdot)$ to obtain the MTBDD Q' satisfying $F_{Q'}(s) = p_s$ for all $s \in V'$. The result, the MTBDD Q'' for the vector $\mathbf{p} = (p_s)_{s \in S}$, is obtained from the MTBDD for the function $F(\overline{x}) \;=\; \max\{\, F_{B_2}(\overline{x}),\; F_{Q'}(\overline{x}) \cdot F_{B'}(\overline{x})\,\}$ which uses Q' for all $s \in V'$ and ensures that 1 is returned as the probability of the states already satisfying Φ_2.

Example 3. Let $\Phi \;=\; [\; try\_to\_deliver \;\; U \;\; correctly\_delivered \;]_{\geq 0.9}$ where $try\_to\_deliver = a_2$ and $correctly\_delivered = \neg a_1 \wedge \neg a_2$. We consider the system in Example 1. Our algorithm first computes the BDDs B_1 for $Sat(try\_to\_deliver) = \{s_{del}, s_{lost}\}$, B_2 for $Sat(correctly\_delivered) = \{s_{init}\}$, and then applies Algorithm $UNTIL(B_1, B_2, P)$. $V = \{s_{init}, s_{del}, s_{lost}\}$ is represented by the BDD B, $V' = \{s_{del}, s_{lost}\}$ by the BDD B'. Thus, B^2, P' and A stand for the matrices

$$\mathbf{B}^2 \;=\; \begin{pmatrix} 0\,0\,0\,0 \\ 0\,1\,0\,1 \\ 0\,0\,0\,0 \\ 0\,1\,0\,1 \end{pmatrix} \quad \mathbf{P}' \;=\; \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{1}{100} & 0 & 0 \end{pmatrix} \quad \mathbf{A} \;=\; \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & -\frac{1}{100} & 0 & 1 \end{pmatrix}$$

Algorithm: $UNTIL(B_1, B_2, P)$

Input: A labelled Markov chain represented by a MTBDD P over $2n$ variables, BDDs B_1, B_2 over n variables

Output: MTBDD X over n variables which represents the function that assigns to each state the probability of a path from the state reaching a B_2-state via an execution sequence through B_1-states

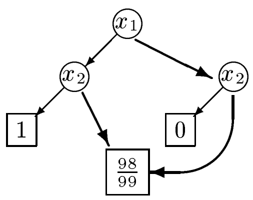
Method: $B := REACH(B_1, B_2, BDD(P, > 0))$; $B' := \lambda\overline{x}\ [\ B(\overline{x}) \wedge \neg B_2(\overline{x})\ ]$;
$B^2 := \lambda x_1 y_1 \ldots x_n y_n\ [B'(x_1, \ldots, x_n) \wedge B'(y_1, \ldots, y_n)]$;
$P' := APPLY(P, B^2, *)$; $I := \lambda x_1 y_1 \ldots x_n y_n\ [\overline{x} = \overline{y}]$;
$A := APPLY(I, P', -)$; $Q := MV\_MULTI(P, B_2)$;
$Q' := SOLVE(A, Q)$; $Q'' := APPLY(B_2, APPLY(Q', B', *), \max)$;
Return($REDUCE(Q'')$).

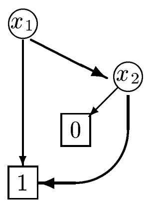
Fig. 1. Algorithm $UNTIL(B_1, B_2, P)$

B_2 (viewed as a vector) is $\mathbf{q}_2 = (1, 0, 0, 0)$. Thus, Q is the MTBDD for the vector $\mathbf{P} \cdot \mathbf{q}_2 = (0, 0, 1, 0.98)$. We solve the linear equation system

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & -\frac{1}{100} & 0 & 1 \end{pmatrix} \cdot \mathbf{x} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ \frac{98}{100} \end{pmatrix}$$

which yields the solution $\mathbf{x} = (0, \frac{98}{99}, 1, \frac{98}{99})$ (represented by the MTBDD Q'). Moreover, the MTBDD $APPLY(Q', B', *)$ can be identified with the vector $(0, \frac{98}{99}, 0, \frac{98}{99})$. $UNTIL(B_1, B_2, P)$ and the BDD $B[\Phi]$ are of the following form.





Thus, $B[\Phi]$ represents the characteristic function for $Sat(\Phi) = \{s_{init}, s_{del}, s_{lost}\}$. ■

6 Implementing PCTL model checking

We are integrating PCTL symbolic model checking within Verus [9], which is a tool specifically designed for the verification of finite-state real-time systems. Verus has been used already to verify several interesting real-time systems: an aircraft controller, a medical monitor, the PCI local bus, and a robotics controller. These examples have not been originally modeled using probabilities. However, these systems exhibit behaviors which can best be described probabilistically. The integration of PCTL model checking with Verus allows us to verify stochastic properties of these and other interesting applications.

The Verus language is an imperative language with a syntax resembling that of the C language with additional special primitives to express timing aspects such as deadlines, priorities, and delays. An important feature of Verus is the use of the `wait` statement to control the passage of time. In Verus time only passes when a wait statement is executed: non-wait statements execute in zero time. This feature allows a more accurate control of time and leads to models with less states, since consecutive statements not separated by a `wait` statement are compiled into a single state. To describe probabilistic transitions we extend the Verus language with the probabilistic `select` statement.

From the Verus description of the application, the tool generates automatically a labeled state-transition graph and the corresponding transition probability matrix using BDDs and MTBDDs respectively.

The first experimental results of our PCTL symbolic model checking implementation are promising: Parrow's Protocol (which is of a similar size to Example 1) can be verified in less than a second. We have modeled a fault tolerant system [23, p. 168–171] with three processors that has about 35000 reachable states (out of 10^8 states). A safety property of this system took only a few seconds to check. Next we plan to evaluate how well PCTL symbolic model checking performs as a formal verification tool in real applications by modeling industrial size systems.

7 Concluding remarks and further directions

We have proposed a symbolic model checking procedure for the logic PCTL which we are implementing using MTBDDs in Verus, thus forming the basis of an efficient tool for verifying probabilistic systems. Our algorithm can be extended to cater for "bounded until" of [24] which is useful in timing analysis of systems. We expect that MTBDDs can be used to derive PCTL* model checking by applying the methods of [18]. Likewise, testing of probabilistic bisimulation and simulation [3, 19] can be implemented using MTBDDs. An extension to the case of infinite state systems, perhaps by appropriate combination with induction, as well as a generalization to allow non-determinism, would be desirable.

References

1. R. Alur, C. Courcoubetis, D. Dill. Verifying Automata Specifications of Probabilistic Real-Time Systems. In *Proc. Real-Time: Theory and Practice*, LNCS 600, pp 27-44, Springer, 1991.
2. L. de Alfaro. Formal Verification of Performance and Reliability of Real-Time Systems. Techn. Report, Stanford University, 1996.
3. C. Baier. Polynomial Time Algorithms for Testing Probabilistic Bisimulation and Simulation. In *Proc. CAV'96*, LNCS 1102, pp 38-49, Springer, 1996.
4. C. Baier, S. Campos, E. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, M. Minea, and M. Ryan. Probabilistic model checking using multi terminal binary decision diagrams. In preparation.
5. C. Baier, M. Kwiatkowska. Model Checking for a Probabilistic Branching Time Logic with Fairness. Techn. Report CSR-96-12, University of Birmingham, 1996.
6. A. Bianco, L. de Alfaro. Model Checking of Probabilistic and Nondeterministic Systems. In *Proc. Foundations of Software Technology and Theoretical Computer Science*, LNCS 1026, pp 499-513, Springer, 1995.
7. R. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8), pp 677-691, 1986.

8. J. Burch, E. Clarke, K. McMillan, D. Dill, L. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computation*, 98(2), pp 142-170, 1992.
9. S. V. Campos, E. M. Clarke, W. Marrero, and M. Minea. Verus: a tool for quantitative analysis of finite-state real-time systems. In *Proc. Workshop on Languages, Compilers and Tools for Real-Time Systems*, 1995.
10. E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In D. Kozen, eds, *Proc. Logic of Programs*, LNCS 131, Springer, 1981.
11. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications: A practical approach. In *Proc. 10th Annual Symp. of Programming Languages*, 1983.
12. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Programming Lnaguages and Systems*, 1(2), 1986.
13. 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.
14. E. Clarke, M. Fujita, X. Zhao. Multi-Terminal Binary Decision Diagrams and Hybrid Decision Diagrams. In T. Sasao and M. Fujita, eds, *Representations of Discrete Functions*, pp 93-108, Kluwer Academic Publishers, 1996.
15. E. Clarke, O. Grumberg, D. Long. Verfication Tools for Finite-State Concurrent Programs. In *Proc. A Decade of Concurrency*, LNCS 803, pp 124-175, Springer, 1993.
16. E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang. Spectral transforms for large boolean functions with applications to technology mapping. In *Proc. 30th ACM/IEEE Design Automation Conference*, pp 54–60, IEEE, 1993.
17. C. Courcoubetis, M. Yannakakis. Verifying Temporal Properties of Finite-State Probabilistic Programs. In *Proc. FOCS'88*, pp 338–345, IEEE, 1988.
18. C. Courcoubetis, M. Yannakakis. The Complexity of Probabilistic Verification. *J. ACM*, 42(4), pp 857-907, 1995.
19. R. Enders, T. Filkorn, D. Taubner. Generating BDDs for Symbolic Model Checking in CCS. *Distributed Computing*, 6, 1993.
20. P. Halmos. *Measure Theory*, Springer, 1950.
21. S. Hart, M. Sharir. Probabilistic Temporal Logic for Finite and Bounded Models. In *Proc. 16th ACM Symposium on Theory of Computing*, pp 1-13, 1984.
22. S. Hart, M. Sharir, A. Pnueli. Termination of Probabilistic Concurrent Programs. *ACM Trans. Programming Languages and Systems*, 5, pp 356-380, 1983.
23. H. Hansson. *Time and Probability in Formal Design of Distributed Systems*, Elsevier, 1994.
24. H. Hansson, B. Jonsson. A Logic for Reasoning about Time and Probability. *Formal Aspects of Computing*, 6, pp 512-535, 1994.
25. M. Huth, M. Kwiatkowska. Quantitative Analysis and Model Checking, In *Proc. LICS'97*, IEEE Computer Society Press, 1997.
26. D. Kozen. A Probabilistic PDL, *JCSS*, 30(2), pp 162-178, 1985.
27. K. Larsen, A. Skou. Bisimulation through Probabilistic Testing. *Information and Computation*, 94, pp 1-28, 1991.
28. K. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*, Kluwer Academic Publishers, 1993.
29. A. Pnueli, L. Zuck. Verification of Multiprocess Probabilistic Protocols. *Distributed Computing*, 1(1), pp 53-72, 1986.
30. A. Pnueli, L. Zuck. Probabilistic Verification. *Information and Computation*, 103, pp 1-29, 1993.
31. R. Segala, N. Lynch. Probabilistic Simulations for Probabilistic Processes. In *Proc. CONCUR*, LNCS 836, pp 481-496, Springer, 1994.
32. K. Seidel C. Morgan, A. McIver and J.W. Sanders. Probabilistic Predicate Transformers. Techn. Report PRG-TR-4-95, Oxford University Computing Laboratory, 1995.
33. W. Thomas. Automata on Infinite Objects. In *Handbook of Theoretical Computer Science*, Vol. B, pp 135-191, North-Holland, 1990.
34. M. Vardi. Automatic Verification of Probabilistic Concurrent Finite-State Programs. In *Proc. FOCS'85*, pp 327-338, IEEE, 1985.
35. M. Vardi, P. Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *Proc. LICS'86*, pp 332-344, IEEE Computer Society Press, 1986.