An MTBDD-based Implementation of Forward Reachability for Probabilistic Timed Automata

Fuzhi Wang and Marta Kwiatkowska

School of Computer Science, University of Birmingham,
Birmingham B15 2TT, United Kingdom
{F.Wang, M.Z.Kwiatkowska}@cs.bham.ac.uk

Abstract. Multi-Terminal Binary Decision Diagrams (MTBDDs) have been successfully applied in symbolic model checking of probabilistic systems. In this paper we propose an encoding method for Probabilistic Timed Automata (PTA) based on MTBDDs. The timing information is encoded via placeholders stored in the MTBDDs that are independent of how the timing information is represented. Using the Colorado University Decision Diagrams (CUDD) package, an experimental model checker is implemented, which supports probabilistic reachability model checking via the forward algorithm. We use Difference Bound Matrices (DBMs) and Difference Decision Diagrams (DDDs) for representing timing information and present experimental results on three case studies. Our key contribution is a general placeholder encoding method for Probabilistic Timed Automata and an experimental MTBDD-based model checker which has been partly integrated with PRISM. This is the first symbolic implementation of the forward probabilistic reachability algorithm.

1 Introduction

Binary Decision Diagrams (BDDs) [7] are the main data structure used in symbolic model model checking. Their success relies on the ability of BDD-like data structures to compactly represent both sets of states and the transition relation between these states. BDDs, however, cannot represent quantitative information. Recently, Multi-Terminal Decision Diagrams (MTBDDs) [8] have been successfully applied in the verification of probabilistic systems [10, 12], and various extensions, Difference Decision Diagrams (DDDs) [17], Clock Difference Diagrams (CDDs) [2] and Clock-Restriction Diagram (CRDs) [20], have been proposed for use in symbolic verification of real-time systems. However, no symbolic data structure exists for the verification of probabilistic real-time systems modelled as Probabilistic Timed Automata (PTA) [14], which contain both real-time clocks and probabilistic information. Probabilistic timed automata are a natural model for randomised distributed algorithms that use timing delays, and a number of models of such algorithms have been developed. The subclass of probabilistic timed automata with digital clocks [13] can be modelled and verified directly using the PRISM model checker [1]. For PTAs that do not comply with this

restriction, currently two, non-symbolic, methods are supported: via the forward exploration algorithm of [14] using the KRONOS to PRISM connection [9], or forward/backward using the experimental implementation of [16]. Both these methods are based on a translation into the textual modelling language of PRISM, and are therefore rather involved.

In this paper, we investigate a fully symbolic implementation of model checking for probabilistic timed automata. We propose an MTBDD-based placeholder encoding method for representing probabilistic timed automata which is independent of the data structure used to represent timing information. The advantage of our proposal is that the method is general: firstly, other data structures for representing timing information could also be integrated with this encoding method; secondly, both forward and backward analysis can be supported. The main difficulty for the fully symbolic approach is that the state space is not known in advance, since the states are extracted dynamically via forward/backward exploration, in contrast to symbolic model checking for probabilistic systems where the size of the state space can be deduced from the syntactic model description. Using the CUDD package [19] an experimental tool has been implemented and partly integrated within PRISM. We report on the performance of our implementation of the forward probabilistic reachability algorithm on three case studies. This is the first fully symbolic implementation of the forward probabilistic reachability for probabilistic timed automata originally proposed in [14].

Related Work. The most commonly used data structure for representing timing information in real-time verification tools [6, 3] is Difference Bound Matrices (DBMs) [4]. A number of BDD-like data structures, e.g. CDDs [2], DDDs [17] and CRDs [20], have been proposed for use in verifying real-time systems, but not yet extended to the probabilistic case. MTBDDs have been successfully applied in model checking of probabilistic systems, and also probabilistic timed automata and timed systems with digital clocks [12], but not in the case of the full class of PTAs. Recently, MTBDDs have also been applied to real-time systems [18]. Although the approach in [18] uses a single data structure (MTBDDs) to represent both timing and discrete information in order to leverage well-known techniques for BDDs or MTBDDs, it involves SAT-based analysis.

There are two, non-symbolic, methods for dense-time probabilistic timed automata: via the forward exploration algorithm implemented in [9], or forward/backward using the experimental implementation of [16]. The forward method is not guaranteed to produce exact reachability probabilities [14], but only requires simple operations; on the other hand, backward analysis can produce exact probabilities but at a cost of higher computational complexity. The method of [9], which combines KRONOS [6] and PRISM [1] to verify the IEEE-1394 Root Contention Protocol, requires three steps: firstly, a set of states reachable from the source state before the deadline is calculated using KRONOS [6]; secondly, the result is translated into a PRISM model description which is input into PRISM and, finally, probabilistic analysis is performed. This method is not efficient because it can generate large files that have to be parsed by PRISM. Our experimental implementation of [16] suffers from the same problem, as it

performs a translation into the PRISM modelling language. In this paper we propose a general MTBDD-based encoding scheme that can deal with both forward and backward analysis in a single step, thus avoiding the expensive translation at the level of text.

2 The Symbolic Encoding Method

2.1 Syntax of Probabilistic Timed Automata

Let $\mathcal{R}$ be the *time domain* of the non-negative reals and $\mathcal{N}$ the natural numbers. We assume a given finite set $\mathcal{X}$ of *clocks*, variables $x \in \mathcal{X}$ that take values from the time domain $\mathcal{R}$. A point $\nu \in \mathcal{R}^{\mathcal{X}}$ is referred to as a *clock valuation*, with $0 \in \mathcal{R}^{\mathcal{X}}$ the valuation that assigns 0 to all clocks in $\mathcal{X}$. We use $\nu[X := d]$ to denote the clock valuation obtained from ν by resetting all of the clocks in $X \subseteq \mathcal{X}$ to d, where d is a natural number or zero with the value of the remaining clocks unaffected.

A *zone* of $\mathcal{X}$, written ζ, is a subset of the valuation space $\mathcal{R}^{\mathcal{X}}$ described by a conjunction of constraints. Formally, for a given set of clocks $\mathcal{X}$, a *zone* ζ is defined by the following syntax:

$$\zeta \quad ::= \quad x \sim c \;\big|\; x - y \sim c \;\big|\; \neg\zeta \big|\; \zeta \wedge \zeta$$

where $x, y \in \mathcal{X}$, $c \in \mathcal{N}$ and $\sim \in \{\leq, <, >, \geq\}$.

The set of zones of $\mathcal{X}$ is denoted by $Zones(\mathcal{X})$ for a given set of clocks $\mathcal{X}$.

Below we review the definition of probabilistic timed automata [14].

Definition 1. *A* probabilistic timed automaton (PTA) *is a tuple* $= (L, \mathcal{X}, \mathcal{L}, inv, prob)$ *where:*

- L *is a finite set of* locations*;*
- *the function* $inv : L \rightarrow Zones(\mathcal{X})$ *is the* invariant condition*;*
- *the finite set* $prob \subseteq L \times Zones(\mathcal{X}) \times Dist(2^{\mathcal{X}} \times L)$ *is the* probabilistic edge relation*;*
- $\mathcal{L} : L \rightarrow 2^{AP}$ *is a labelling function assigning atomic propositions to locations.*

A *state* of a probabilistic timed automaton is a pair (l, ν) where $l \in L$ and $\nu \in \mathcal{R}^{\mathcal{X}}$ are such that $\nu \triangleleft inv(l)$. The invariant condition describes the set of admissible states. Transitions are *probabilistic edges* $(l, g, p) \in prob$ where l is the source location, g is the enabling guard and p the probability distribution on target locations, together with the set of clocks to be reset as the edge is taken. It is more convenient to treat an *edge* of a PTA as a tuple $(n, l, inv(l), l', inv(l'), g, X, p)$ where n is the non-deterministic choice between simultaneously enabled distributions, coded as a natural number, such that $(l, g, p) \in prob$ and $p(X, l') > 0$. Such a tuple contains sufficient information for making a transition, and will be directly used in our encoding method.

2.2 Multi-Terminal Binary Decision Diagrams (MTBDDs)

MTBDDs [8] are an extension of BDDs [7] which allow one to represent functions over Boolean vectors that can take any value, not just 0 or 1. In other words, BDDs can have only two terminals, while MTBDDs can have more than two terminals. An MTBDD is a Directed Acyclic Graph (DAG) whose vertices, as for BDDs, are called nodes. There are two kinds of nodes in an MTBDD, non-terminal and terminal. Like in a BDD, a non-terminal node is labelled with a single variable and each non-terminal node has exactly two children. However, unlike BDDs, the terminal node which has no children is labelled by a real number. MTBDDs can be reduced to the canonical form by imposing an ordering of the variables. However, similarly to BDDs, the size of the MTBDDs is extremely sensitive to the ordering of its variables.

Probabilistic systems, for example, Markov Decision Processes (MDPs) that are induced from PTAs, are described in terms of probability matrices, and their analysis involves numerical computation such as solving a linear equation system or (as is the case with MDPs) a linear programming problem. MTBDDs can represent both probability vectors and matrices, and can therefore serve as a symbolic representation for probabilistic systems. Given a real-valued vector of length 2^n, an MTBDD encoding can be obtained by mapping to reals from vector's indices which are encoded into n Boolean variables. As far as numerical computation is concerned, MTBDDs support methods for implementing standard matrix operations, such as scalar multiplication, matrix addition and matrix multiplication.

2.3 Representations for Zones

DBMs. Difference Bound Matrices (DBMs) are data structures that can efficiently represent sets of adjacent regions (convex union of adjacent regions which is also called a zone). Non-convex zones are represented as lists of DBMs. A DBM is a square matrix whose elements represent bounds on the difference between two clock values. For a set of n clocks $\{x_1, \ldots, x_n\}$, and by using a special clock x_0 whose value is always zero, the constraints over these clocks can be encoded as a $(n+1)*(n+1)$ square matrix D whose indices range over the interval $[0..n]$ and whose elements belong to $\mathcal{N}_\infty \times \{<, \leq\}$, where $\mathcal{N}_\infty = \mathcal{N} \vee \{0, \infty\}$.
DDDs. Difference Decision Diagrams (DDDs) are designed for representing both convex and non-convex unions of zones, which are called Difference Constraint Expressions (DCE) in DDDs. DDDs are a BDD-like data-structure. Like a BDD, a difference decision diagram is also a DAG whose vertex set contains two terminals 0 and 1, and a set of non-terminal vertices, each with two children. A non-terminal vertex v corresponds to an integer- or real-valued difference constraint between two clocks. A path in a DDD is a finite sequence of edges, and it corresponds to a conjunction of difference constraints that is called a Difference Constraint System (DCS) in DDD. In contrast to BDDs, the same pair of clocks can appear more than once along a path in a DDD. A DCS corresponds to a DBM. DDDs contain DBMs as a special case. A path that ends with true or

false is called a 1-path or 0-path respectively. A path p is feasible if and only if the corresponding DCS has a solution. If the Difference Constraint System has no solution, the path is infeasible. Unlike in BDDs, both 0- and 1-paths can be feasible or infeasible because the difference constraints can interact with each other along the path.

2.4 Encoding Probabilistic Timed Transitions

MTBDDs have been successfully applied for symbolic model checking of untimed probabilistic systems [1, 12], and specifically Markov Decision Processes (MDPs) which arise as the representation of the PTAs. The method relies on encoding sets of states as BDDs and the (probabilistic) transition relation between these states using MTBDDs, which can be done compactly if there is sufficient regularity in the model. For finite untimed probabilistic systems the potential state space is known in advance of constructing the symbolic representation of the model, as it can be deduced from the syntactic model description. This fact is exploited when formulating heuristics that determine BDD variable ordering, a good choice of which is essential to guarantee a compact model representation. The difficulty with model checking of PTAs is that the size of the state space is unknown beforehand, and states (location-zone pairs) are generated dynamically through the process of exploration of the zone graph using timed predecessor or successor operations. The resulting symbolic representation has to be amenable to such dynamic manipulation of the state space which has the potential to destroy regularity. Although, if using the region graph, the size of the state space of a PTA can be established in advance of the model construction, such an approach is impractical due to the region graph being exponential in the number of clocks and the maximal constant appearing in the model.

In this section we propose an encoding method for probabilistic timed automata based on MTBDDs. Below we describe how to encode the *states* and *probabilistic edges*. In this paper we focus on application of our method to forward probabilistic reachability analysis. However, it is also suitable for backward analysis.

Firstly, let us consider how to encode the state space. Each state in the state space has the form of (l, ζ) where $l \in L$ is the discrete part and $\zeta \in Zones(\mathcal{X})$ is the zone. Our method is to use a Boolean vector to encode the discrete part of the pair and a separate Boolean vector for the zone part. The basic idea behind the Boolean encoding is that 2^n elements of a finite set could be encoded using n bits. For the discrete part, which is finite, the Boolean vector is further divided into several groups according the structure of the system, for example, the number of subcomponents and values of the non-clock variables following well-known heuristics established in [10, 11].

The number of zones, unfortunately, could be infinite when forward analysis is used. The technique in [4] guarantees the termination of the forward reachability search, which means that a finite set of zones could be obtained. Since the set of zones is finite, informally, we use a one-to-one function to assign a unique index to each zone and similar logarithmic encoding is applied. The invariant and the

guard appearing in the probabilistic transition are also zones, so a unique index value is assigned to each of them. In this paper we use a simple method for allocating indices; later, we discuss how this can be improved.

Next, let us consider the probabilistic transition relation. Each probabilistic edge has the form of the tuple $(n, l, inv(l), l', inv(l'), g, X, p)$, where n is (the encoding of) the non-deterministic choice, l and $inv(l)$ are the current location and its invariant, l' and $inv(l')$ are the next location and its invariant, g is the guard, X is the set of clocks to be reset and p is the probability value. The probability value is natively supported by MTBDDs. The number of non-deterministic choices in each state is finite and bounded. The maximum number of non-deterministic choices for all states is determined through parallel composition, and can therefore be encoded using the logarithmic encoding. It remains to encode the set X of clocks to be reset. There are a number of issues to consider when encoding the clock reset operation in the transition relation:

- Recall that each clock $x \in \mathcal{X}$ could be set to different values and not simply zero. If we encode each reset in the transition, this means we need two sets of Boolean BDD variables for it: one for the clock and the other for the value that the clock should be set to.
- The total number of clocks to be reset appearing in the transition could vary.

Thus, we opt for a simple approach: a Boolean vector is reserved in the transition for assigning a unique index value to each distinct set X and each set X is explicitly stored as a list.

Below we summarise the main issues that have to be addressed when applying our encoding method:

- Unlike in the case of non-probabilistic timed systems, in which the on-the-fly technique [5] could be applied to make search algorithms finish as early as possible, the forward probabilistic reachability search has to construct the whole reachable zone graph in order to obtain the probability value.
- The size of the state space and transitions between these states of the generated probabilistic system is uncertain before the forward/backward algorithm, which dynamically generates these states, terminates.

As a result, we cannot fix the size of the vector of Boolean variables needed to encode the zone part in advance. Instead, we pre-allocate the vector based on an estimate.

2.5 Implementation

Our proposed forward probabilistic reachability algorithm is given in Figure 1 in terms of MTBDD-based pseudo-code. Below we described the BDD and MTBDD operations needed for the algorithm. In the following, we assume M is an MTBDD, and $\underline{\mathsf{x}}, \underline{\mathsf{y}}, \underline{\mathsf{z}}$ are the Boolean vectors which correspond to the MTBDD variables for row, column and non-deterministic choice in an MDP matrix.

- Operations $\times$ *and* $+$ are the MTBDD operations over the reals.

– Operations $\vee$, $\wedge$ *and* $\setminus$ are the BDD operations (and, or and difference) on sets.
– Function $\textsc{Threshold}(\mathsf{M}, >, 0)$ returns the BDD by replacing each terminal node with 1 if and only if its value is greater than 0.
– Function $\textsc{ThereExists}(\underline{\mathsf{x}}, \mathsf{M})$ returns the MTBDD by deleting the nodes containing the Boolean vector $\underline{\mathsf{x}}$.
– Function $\textsc{ReplaceVars}(\mathsf{M}, \underline{\mathsf{y}}, \underline{\mathsf{x}}))$ returns the MTBDD by replacing the Boolean vector $\underline{\mathsf{y}}$ with $\underline{\mathsf{x}}$.

The algorithm in Figure 1 is an MTBDD-based implementation of the algorithm of [14] with respect to our encoding. The algorithm $ModelCheckingPTA$ accepts three parameters: the probabilistic transition relation PS_{PTA}, which is an MTBDD-encoded representation of the syntax of the original probabilistic timed automaton, the initial set of states ϕ_{init} and the set of target states ϕ_{target}. Lines 1-4 deal with the initialisation: line 1 initialises the generated set of probabilistic transitions with the empty set, and lines 2-3 assign the initial set to both the front set and the reachable set. Lines 5-21 generate the finite-state graph, the edges of which are obtained in lines 8-11 by iterating timed and discrete successor operations. Each generated edge has the form of a tuple $(n, l, l', \zeta, \zeta', p)$, where n is the encoding of non-deterministic choice, (l, ζ) corresponds to current symbolic state, (l', ζ') is the next symbolic state in the generated transition, and p is the probability value. Line 6 constructs a temporary MTBDD with the information necessary for the timed and discrete successor operations by restricting to the front set: each path of the temporary MTBDD has the form of a tuple $(n, l, \zeta, inv(l), l', inv(l'), g, X, p)$, where l and l' are the current and next locations, $inv(l)$ and $inv(l')$ are the invariants associated with current and next location respectively, ζ is the current zone associated with current location, g is the guard, X is the set of clocks to be reset and p is the probability. Lines 12-19 extract the reachable states from the generated probabilistic transition set and check whether the fixed point is reached. Line 20 adds the set of newly generated edges to the old one. Finally, in line 22, model checking is performed on the resulting finite-state probabilistic system to obtain the maximum probability of reaching the set of target locations. Lines 9.1-9.3 give the MTBDD-based pseudo-code of the construction of a single probabilistic edge of the generated MDP. Line 9.1 obtains the next zone by using standard zone successor operation. Line 9.2 uses the technique in [4] to obtain the unique normal form of the next zone and adds it to the list of zones if it is a new one, and otherwise it returns the unique index to it in the list. Line 9.3 constructs and returns the probabilistic edge of the generated MDP.

Remark. For the forward reachability search, each zone obtained is convex, and can be stored as a single DBM. (A convex zone in DDDs can only have one path; this is also true for CRDs or CDDs.) The operation $\textsc{Normalise}(\zeta, k)$ (k-Normalization defined in [4]) to obtain the normal form of a zone ζ, where k is the maximal constant appearing in the model or the specification, is necessary to guarantee the termination of the forward reachability search. For DDDs, the zone can be first transformed into a DBM on which the k-Normalization can be

applied, and then transformed back into a DDD. The case of CRDs and CDDs can be handled in a similar way.

$ModelCheckingPTA(\mathsf{PS}_{PTA}, \phi_{init}, \phi_{target})$

1. $\mathsf{PS}_{MDP} := \emptyset$
2. $\phi_{frontset} := \phi_{init}$
3. $\phi_{reach} := \phi_{init}$
4. $done :=$ **false**
5. **while** ($done =$ **false**)
6. $\quad \mathsf{TmpPS} := \phi_{frontset} \times \mathsf{PS}_{PTA}$
7. $\quad \mathsf{T} := \emptyset$
8. $\quad$ **for each non-zero path** $of \mathsf{TmpPS}$
9.1. $\quad\quad \zeta' = ZoneSuccessor(l, inv(l), \zeta, l', inv(l'), g, X)$
9.2. $\quad\quad \zeta' = \textsc{AddZone}(\textsc{Normalise}(\zeta', k))$
9.3. $\quad\quad tr = n \times l \times \zeta \times l' \times \zeta' \times p$
10. $\quad\quad \mathsf{T} := \mathsf{T} + \mathsf{tr}$
11. $\quad$ **endfor**
12. $\quad \mathsf{T}_{01} := \textsc{Threshold}(\mathsf{T}, >, 0)$
13. $\quad \phi_{tmp} := \textsc{ThereExists}(\underline{\mathsf{z}}, \mathsf{T}_{01})$
14. $\quad \phi_{tmp} := \textsc{ThereExists}(\underline{\mathsf{x}}, \phi_{tmp})$
15. $\quad \phi_{tmp} := \textsc{ReplaceVars}(\phi_{tmp}, \underline{\mathsf{y}}, \underline{\mathsf{x}}))$
16. $\quad \phi_{reach'} := \phi_{reach} \vee \phi_{tmp}$
17. $\quad$ **if** $(\phi_{reach} = \phi'_{reach})$ **then** $done :=$ **true**
18. $\quad \phi_{frontset} := \phi_{reach'} \setminus \phi_{reach}$
19. $\quad \phi_{reach} := \phi_{reach'}$
20. $\quad \mathsf{PS}_{MDP} := \mathsf{PS}_{MDP} + \mathsf{T}$
21. **endwhile**
22. **return** $MaxProbReach(\phi_{init}, \phi_{reach}, PS_{MDP})$

Fig. 1. The MTBDD version of the $ModelCheckingPTA$ algorithm

2.6 Backward Adaption

In this paper, we only present the application of our encoding method to forward analysis. However, it it also suitable for backward analysis. What is needed is a replacement of lines 9.1-9.3 with the corresponding backward step, and initialisation with the target set instead of the initial set.

3 Experimental Results

3.1 Tool Overview

Using the Colorado University Decision Diagrams (CUDD) package, a symbolic model checker has been implemented which supports forward probabilistic reachability model checking via the algorithm originally presented in [14]. This is the first symbolic implementation of the forward reachability algorithm.

Our tool takes as input a description of a system written in a probabilistic variant of the guarded commands language with real-time clocks (currently, the PRISM input language does not support real-time clocks). It first parses the PTA model from this description into an MTBDD, and then computes the set of probabilistically reachable states which comprise the model, which is an MDP over location-zone pairs [14]. The tool then performs model checking over this MDP in the standard way, and calculates the probability of reaching the target set to determine whether the specification is satisfied. In this tool, the model construction and reachability analysis are implemented using MTBDDs to represent both the discrete part of the state and the placeholder, reserved in the Boolean vector of the BDD variables, that represents the zone. The tool supports two kinds of representation of timing information: DBMs and DDDs.

The transition relation of the PTA model is encoded within MTBDDs. Unlike untimed probabilistic model checking, the MDP model is dynamically constructed via forward searching which involves timing operations, for example, the timed successor. The generated MDP differs from the original PTA in that it represents the dynamic behaviour of the PTA, which is computed via forward exploration and dynamically filled using the placeholders reserved in the BDD vector. Although the tool currently supports only forward analysis and two kinds of representation for timing information, our encoding method is general: firstly, other data structures for representing timing information could be integrated; secondly, backward reachability analysis is also suitable for this encoding.

3.2 Case Studies

We present experimental results based on three case studies: the IEEE 1394 FireWire root contention protocol, the IEEE 802.3 CSMA/CD (Carrier Sense, Multiple Access with Collision Detection) protocol and Milner's scheduler. The models for FireWire and CSMA/CD are the same as those used in [16]. The model for Milner's scheduler is that used in [17] with only one clock.

In the tables, "-" denotes that the data is not available. We omit the probability values since they all agree with those calculated previously by other methods.

The results obtained from verifying the abstract and the full models of Firewire root contention protocol [15] are shown in Table 1 and Table 2. The property verified is the minimum probability that, from the initial state, a leader (root) is chosen before the deadline is reached. Table 5 and Table 6 show the memory consumption. Table 3 and Table 4 include results for the CSMA/CD protocol when computing the maximum and minimum probability of both stations correctly delivering their packets by the deadline D. Table 7 and Table 8 show the memory consumption. Table 9 shows the result of verifying Milner's scheduler when computing the maximum probability of any two cyclers being in the critical section at the same time.

To evaluate our encoding method, we also implemented an explicit version of the data structure which stores explicitly the discrete part of the symbolic states.

Table 1. Verification of the abstract model I_1^p with wire delay set to 360 ns

Deadline	States	Time(Explicit)			Time(Symbolic)			
					MTBDD/DDD		MTBDD/DBM	
		Forward	Construct.	M.C.	S.F.C.	M.C.	S.F.C.	M.C.
2000	64	0.26	0.14	0.01	0.06	0.01	0.07	0.01
2500	88	0.30	0.20	0.02	0.07	0.01	0.09	0.01
3000	88	0.28	0.20	0.02	0.08	0.01	0.09	0.01
3500	124	0.38	0.34	0.02	0.18	0.01	0.11	0.01
4000	162	0.41	0.61	0.02	0.20	0.01	0.13	0.01
4500	159	0.43	0.59	0.02	0.26	0.01	0.13	0.01
5000	208	0.51	0.96	0.026	0.29	0.01	0.16	0.01
5500	244	0.56	1.42	0.03	0.38	0.02	0.19	0.02
6000	253	0.58	1.42	0.03	0.42	0.02	0.19	0.02
7000	348	0.70	3.07	0.04	0.83	0.02	0.28	0.02
8000	438	0.80	4.67	0.05	0.98	0.03	0.35	0.03
9000	506	0.91	6.29	0.05	1.32	0.03	0.40	0.03
10000	609	1.12	10.34	0.06	1.85	0.04	0.56	0.04
20000	2124	3.20	117.37	0.13	17.77	0.19	0.56	0.04
30000	4546	10.87	615.42	0.30	76.80	0.39	2.29	0.18
40000	7851	27.20	1846.90	1.55	225.35	0.65	7.43	0.39
50000	12094	58.37	5017.27	2.77	534.15	1.00	18.54	0.65
60000	17231	103.09	-	-	1082.36	1.63	36.72	1.00
70000	23305	170.43	-	-	2492.07	2.42	66.43	1.65
80000	30251	273.99	-	-	4217.03	3.09	182.13	3.08
90000	38151	427.81	-	-	6753.72	3.95	276.52	3.97

Table 2. Verification of the full model $Impl^p$ with wire delay set to 360 ns

Deadline	States	Time(Explicit)			Time(Symbolic)			
					MTBDD/DDD		MTBDD/DBM	
		Forward	Construct.	M.C.	S.F.C.	M.C.	S.F.C.	M.C.
2000	951	6.42	9.26	0.04	25.87	0.05	7.44	0.05
2500	1415	9.89	32.47	0.09	56.66	0.10	11.71	0.10
3000	1425	9.99	32.45	0.09	57.13	0.12	11.79	0.11
3500	2092	14.79	87.48	0.14	122.52	0.19	17.59	0.20
4000	2803	19.94	186.22	0.18	219.03	0.23	24.03	0.23
4500	2799	20.53	196.01	0.21	217.41	0.24	24.84	0.24
5000	3725	27.50	375.07	0.25	385.72	0.27	34.49	0.27
5500	4432	33.02	543.03	0.33	542.73	0.39	41.79	0.39
6000	4675	35.29	697.63	0.37	609.45	0.49	44.81	0.51
7000	6545	51.25	1403.52	0.52	1180.10	0.61	66.24	0.60
8000	8437	67.95	2523.33	0.71	1966.27	0.85	90.17	0.84
9000	9879	82.23	3925.66	0.86	2694.23	0.86	110.52	0.87
10000	11988	143.79	-	-	4662.95	1.47	135.65	1.39
20000	44335	543.16	-	-	-	-	947.65	5.37
30000	96592	1693.48	-	-	-	-	3607.68	12.42
40000	168514	4135.01	-	-	-	-	10112.92	22.84
50000	261131	-	-	-	-	-	23297.78	35.03
60000	373429	-	-	-	-	-	45553.48	54.89
70000	-	-	-	-	-	-	-	-
80000	-	-	-	-	-	-	-	-
90000	-	-	-	-	-	-	-	-

Performance. In tables [1- 4], the term "Explicit" refers to the explicit version and "MTBDD/DDD" refers to the version which uses MTBDDs for encoding the discrete part and DDDs for timing information; "MTBDD/DBM" refers to the version which differs from the "MTBDD/DDD" by using DBMs instead of DDDs. However, in order to model check certain properties, the explicit version involves two steps: first, it generates the reachable states, and next it represents those as a model in the PRISM input language, which is then passed to PRISM to finish the verification process. On the other hand, in the "MTBDD/DDD" and "MTBDD/DBM" versions, as they have already been partly integrated with PRISM, the overall checking process does not go through the PRISM input language: the tool constructs the target MDP models in MTBDDs and directly calls functions provided by PRISM.

Here we only consider the algorithm performance and the number of states obtained from the experiments. The leftmost column of these tables gives the

Table 3. Verification of the full CSMA/CD model (max, backoff=1)

Deadline	States	Time(Explicit)			Time(Symbolic)			
					MTBDD/DDD		MTBDD/DBM	
		Forward	Construct.	M.C.	S.F.C.	M.C.	S.F.C.	M.C.
1000	6404	26.89	-	-	47.17	0.00	34.84	0.00
1200	9034	43.94	-	-	945.28	0.00	56.55	0.00
1400	11771	66.47	-	-	1611.72	0.01	82.79	0.01
1600	15329	100.45	-	-	2752.01	0.00	125.33	0.01
1800	19453	148.20	11021.07	0.79	4452.55	1.22	183.39	1.22
2000	23468	204.11	-	-	6517.29	2.76	255.06	2.66
2200	28516	281.92	-	-	9667.14	5.60	351.47	5.48
2400	34023	381.79	-	-	-	-	476.85	9.59
2600	39970	503.00	-	-	-	-	631.94	14.45
2800	45654	628.90	-	-	-	-	804.11	20.52
3000	52561	811.40	-	-	-	-	1028.94	27.25

Table 4. Verification of the full CSMA/CD model (min, backoff=1)

Deadline	States	Time(Explicit)			Time(Symbolic)			
					MTBDD/DDD		MTBDD/DBM	
		Forward	Construct.	M.C.	S.F.C.	M.C.	S.F.C.	M.C.
1000	6408	26.86	934.96	0.26	471.32	0.32	34.53	0.32
1200	9038	44.14	2103.15	0.33	944.00	0.42	56.82	0.41
1400	11775	66.08	3567.70	0.41	1609.68	0.51	83.00	0.49
1600	15333	100.83	6106.86	0.51	2746.67	0.60	124.82	0.59
1800	19453	147.69	-	-	-	-	182.64	1.94
2000	23468	202.60	-	-	-	-	253.95	2.55
2200	28516	280.17	-	-	-	-	349.63	3.32
2400	34023	382.47	-	-	-	-	474.25	4.08
2600	39970	503.26	-	-	-	-	631.54	5.19
2800	45654	632.84	-	-	-	-	799.57	6.42
3000	52561	814.99	-	-	-	-	1027.52	7.68

different deadlines. The second column shows the number of symbolic states generated via the forward construction. The column "S.F.C." refers to the symbolic forward and construction. The column "M.C." refers to the computation time for model checking the given properties against the MDP model encoded as an MTBDD. The unit for all columns "Time" is seconds. Notice that there are three columns under the "Explicit" version: the first column represents the time taken to generate the reachable states and translate the model to the PRISM input language, while the second refers to model construction time by PRISM via explicit encoding.

The times for model checking are nearly same for the three versions. We comment on the time spent on constructing MDP models encoded as MTBDDs via forward analysis. Compared to the explicit implementation, the symbolic encoding version based on DBMs has a significant advantage: the time spent on generating the MDP models is no longer a problem, since it took 110 seconds to perform both the forward construction and to generate the MDP in MTBDDs for the full model $Impl^p$ with deadline 9000 ns, whilst it took 3925 seconds to generate the MDP model alone with the same deadline with the explicit version.

For the abstract model, the DDD-based version performs as well as the DBM-based one. However, for the full model $Impl^p$, it is slower due to a large number of intermediate DDD nodes being generated, which forces the DDD run-time library to invoke garbage collection. The main reason is that DDDs have no canonical property.

Memory. With regard to memory usage, we need to consider two kinds of usage for each symbolic state: the memory for the discrete part and the zone

Table 5. Memory consumption of the abstract model I_1^p with wire delay set to 360 ns

Deadline	Nodes (Explicit)	Nodes (Discrete)	Mem. (Zone)		
			DDD		DBM
			Peak	Estimated	
2000	218	255	49.52	5.47	1.27
2500	289	324	77.66	6.89	1.55
3000	289	325	91.38	7.85	1.69
3500	364	364	136.94	9.49	2.07
4000	481	503	220.61	12.47	2.64
4500	474	486	253.15	13.51	2.81
5000	580	552	356.84	16.02	3.34
5500	680	625	476.98	18.76	3.90
6000	701	656	555.90	20.40	4.22
7000	905	833	942.16	26.91	5.48
8000	1087	986	1365.55	32.54	6.61
9000	1219	1075	1951.06	39.29	7.88
10000	1401	1290	2759.18	46.87	9.42
20000	3143	2893	30232.26	157.91	31.18
30000	5171	4823	136341.71	336.74	66.27
40000	6892	7003	402995.33	579.55	113.98
50000	8980	8897	225775.35	890.07	174.97
60000	10792	11085	497021.90	1268.42	249.26
70000	12938	13557	666931.81	1712.98	336.59
80000	14810	18637	229405.34	2222.45	436.61
90000	-	18679	397636.89	2799.86	550.05

Table 6. Memory consumption of the full model $Impl^p$ with wire delay set to 360 ns

Deadline	Nodes (Explicit)	Nodes (discrete)	Mem. (Zone)		
			DDD		DBM
			Peak	Estimated	
2000	1719	2489	46343.66	324.46	143.75
2500	2556	3616	93414.54	492.84	207.75
3000	2585	3718	94993.58	502.61	210.25
3500	3384	4927	184895.18	728.27	299.50
4000	4610	5834	318451.16	977.65	398.75
4500	4476	5735	322968.19	991.48	397.75
5000	5291	6695	546041.32	1311.11	525.25
5500	6003	8181	800000^+	1559.77	623.50
6000	6404	8220	800000^+	1653.37	657.75
7000	8449	10377	800000^+	2317.22	916.25
8000	9818	11865	800000^+	2983.175	1179.25
9000	11856	13735	800000^+	3519.195	1381.75
10000	-	15484	-	-	1673.50
20000	-	35157	-	-	6201.75
30000	-	56374	-	-	13534.50
40000	-	76978	-	-	23638.00
50000	-	99116	-	-	36661.75
60000	-	123220	-	-	52459.25
70000	-	-	-	-	-
80000	-	-	-	-	-
90000	-	-	-	-	-

part. In tables [5- 8], the unit for all columns under "Mem." is in kilo-Bytes and the unit for all columns under "Nodes" is the number of the nodes in the MTBDD where each node occupies 20 bytes.

Compared with the explicit version, both symbolic versions use more nodes. However, the chief contributor to the growth in the size of our symbolic data structures seems to be the fact that we do not exploit regularity in the zone representation because this first implementation allocates unique indices to zones in an arbitrary order which are then stored in the placeholders. The results of [9] show that such regularity exists and we have adapted this method and obtained preliminary results which will appear in the first author's coming thesis.

We note that the symbolic versions are performing the generation of the state space dynamically at the same time as calculating the encoding, while the explicit version does not start to encode the MTBDD until the whole state

Table 7. Memory consumption of the full model CSMA (max, backoff=1)

Deadline	Nodes (Explicit)	Nodes (discrete)	Mem. (Zone)		
			DDD		DBM
			Peak	Estimated	
1000	-	10752	577832.75	1099.57	399.71
1200	-	13434	800000^{+}	1558.76	564.45
1400	-	15968	800000^{+}	2039.87	737.11
1600	-	19426	800000^{+}	2667.49	962.01
1800	16697	22903	800000^{+}	3397.76	1223.73
2000	19276	25962	800000^{+}	4108.15	1478.71
2200	22260	30181	800000^{+}	5005.77	1800.20
2400	-	33949	-	-	2151.37
2600	-	38187	-	-	2530.37
2800	-	41611	-	-	2894.82
3000	-	45527	-	-	3337.11

Table 8. Memory consumption of the full model CSMA (min, backoff=1)

Deadline	Nodes (Explicit)	Nodes (discrete)	Mem. (Zone)		
			DDD		DBM
			Peak	Estimated	
1000	7466	10868	578877.20	1100.67	400.10
1200	9753	13492	800000^{+}	1559.85	564.84
1400	11383	16036	800000^{+}	2040.96	737.50
1600	14227	19638	800000^{+}	2668.59	962.40
1800	16714	23130	800000^{+}	3397.76	1223.73
2000	19694	26331	-	-	1478.71
2200	-	30381	-	-	1800.20
2400	-	34271	-	-	2151.37
2600	-	38092	-	-	2530.37
2800	-	41681	-	-	2894.82
3000	-	45374	-	-	3337.11

space is generated. We compare the memory usage on the zone part because those for the discrete part are the same for both symbolic versions. The memory consumption for DBMs is that actually used. For DDDs, we cannot give the actual memory consumption, and instead give both the estimated and peak time memory consumption. The column "Estimated" refers to an estimation of the memory consumption of all zones based on DDDs when reaching the fixed point. The column "Peak" refers to the highest value of memory consumption by DDDs when reaching the fixed point. The DBM-based representation uses less memory than the DDD-based representation. In practice, as shown in Table 6, the estimation of memory consumption for zones in DDDs is around 3-5 times as many as those in DBMs. However, compared with those by DBMs, the peak memory consumption of DDDs is huge, since DDDs use 1000 times more memory than DBMs for the full model $Impl^p$ with deadline 5000 ns. 800000^{+} means garbage collection occurred (DDD run-time library configuration with 800M). We note, however, that since non-convex zones arise in backward exploration it is difficult to predict how the representations will behave in the latter case.

Scalability. In Table 9, the column "Prod. const." is the time spent on parallel composition to build the product model and the column "Encoding" the time on encoding the product model into MTBDDs. In the case study of Milner's scheduler, which shows the scalability of our method and generates only five zones in total after the application of the forward algorithm, the result is not as good as those obtained with DDDs [17]. This is partly because the experimental nature of our implementation not only involves zone search but probability

Table 9. Verification of the Milner's scheduler with only one clock

N	Symbolic/Time				Prob.
	Nodes (Discrete)	Prod. const.	Encoding	S.F.C	
4	1890	0.21	0.11	0.27	0.00
5	2981	0.35	0.30	0.63	0.00
6	4345	0.93	0.79	1.76	0.00
7	5989	3.50	2.42	5.22	0.00
8	7925	19.02	7.16	15.02	0.00
9	26276	113.56	19.18	40.14	0.00
10	143762	591.91	50.41	103.50	0.00
11	-	-	-	-	-

computation as well, and partly because the parallel composition of components is constructed explicitly and does not utilise the Kronecker approach [10] implemented in PRISM which is based on good heuristics for BDD variable ordering that can yield compact MTBDDs.

4 Conclusion

We have proposed an MTBDD-based placeholder encoding method for model checking of probabilistic timed automata and implemented an experimental tool using the CUDD package. The timing information is represented as either DBMs or DDDs. Our method allows one to use other data structures, for example, CRDs [20] or CDDs [2], for representing the timing information.

Future work will address the efficiency of the symbolic implementation presented in this paper, and in particular exploiting regularity in the zone graph, implementing Kronecker-based parallel composition of probabilistic timed automata, and augmenting PRISM with real-time clocks.

Acknowledgements. We would like to thank the authors of the DDD library for letting us use their DDD implementation. We would also like to thank Gethin Norman and David Parker for helpful discussion.

References

1. PRISM WebSite. http://www.cs.bham.ac.uk/~dxp/prism/.
2. G. Behrmann, K. G. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient timed reachability analysis using clock difference diagrams. In *Proc. CAV '99*, pages 341–353, London, UK, 1999. Springer-Verlag.
3. J. Bengtsson, W. Griffioen, K. Kristoffersen, K. Larsen, F. Larsson, P. Pettersson, and W. Yi. Automated verification of an audio-control protocol using UPPAAL. *Journal of Logic and Algebric Programming*, 52(3):163–181, 2002.
4. J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In *4th Advanced Course on Petri Nets*, volume 3098 of *LNCS*, pages 87–124. Springer, 2004.
5. A. Bouajjani, S. Tripakis, and S. Yovine. On-the-fly symbolic model checking for real-time systems. In *Proc. 18TH IEEE Real-Time Systems Symposium*, pages 25–34, Los Alamitos, 1997. IEEE CS Press.

6. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In *Proc. CAV '98*, pages 546–550, London, UK, 1998. Springer-Verlag.
7. R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comput.*, C-35(8):677–691, Aug. 1986.
8. E. Clarke, M. Fujita, P. McGeer, K. McMillan, J. Yang, and X. Zhao. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. In *Proc. IWLS'93*, pages 1–15, 1993. Also available in *Formal Methods in System Design*, 10(2/3):149–169, 1997.
9. C. Daws, M. Kwiatkowska, and G. Norman. Automatic verification of the IEEE 1394 root contention protocol with KRONOS and PRISM. In *Proc. 7th FMICS'02*, volume 66.2 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002.
10. L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of concurrent probabilistic processes using MTBDDs and the Kronecker representation. In *Proc. 6th TACAS'00*, volume 1785 of *LNCS*, pages 395–410. Springer, 2000.
11. H. Hermanns, M. Kwiatkowska, G. Norman, D. Parker, and M. Siegle. On the use of MTBDDs for performability analysis and verification of stochastic systems. *Journal of Logic and Algebraic Programming*, 56(1-2):23–67, 2003.
12. M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. In *Proc. 8th TACAS'02*, volume 2280 of *LNCS*, pages 52–66. Springer, 2002.
13. M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. Performance analysis of probabilistic timed automata using digital clocks. In *Proc. FORMATS'03*, volume 2791 of *LNCS*, pages 105–120. Springer-Verlag, 2003.
14. M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282:101–150, 2002.
15. M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol. *Special Issue of Formal Aspects of Computing*, 14:295–318, 2003.
16. M. Kwiatkowska, G. Norman, J. Sproston, and F. Wang. Symbolic model checking for probabilistic timed automata. In *Joint Conference on FORMATS and FTRTFT*, volume 3253 of *LNCS*, pages 293–308. Springer, 2004.
17. J. Møller, J. Lichtenberg, H. R. Andersen, and H. Hulgaard. Fully symbolic model checking of timed systems using difference decision diagrams. In *Workshop on Symbolic Model Checking*, volume 23, The IT University of Copenhagen, Denmark, June 1999.
18. S. A. Seshia and R. E. Bryant. A boolean approach to unbounded, fully symbolic model checking of timed automata. Technical Report CMU-CS-03-117, Carnegie Mellon University, 2003.
19. F. Somenzi. CUDD: CU Decision Diagram Package Release, 1998.
20. F. Wang. Efficient verification of timed automata with BDD-like data-structures. In *Verification, model checking, and abstract interpretation*, volume 2575 of *LNCS*, pages 189–205. Springer, 2003.