Symbolic Magnifying Lens Abstraction in Markov Decision Processes

Pritam Roy[1] David Parker[2] Gethin Norman[2] Luca de Alfaro[1]

Computer Engineering Dept, UC Santa Cruz, Santa Cruz, CA, USA [1]
Oxford University Computing Laboratory, Oxford, UK [2]

Abstract

In this paper, we combine abstraction-refinement and symbolic techniques to fight the state-space explosion problem when model checking Markov Decision Processes (MDPs). The abstract-refinement technique, called magnifying-lens abstraction *(MLA), partitions the state-space into regions and computes upper and lower bounds for reachability and safety properties on the regions, rather than the states. To compute such bounds, MLA iterates over the regions, analysing the concrete states of each region in turn - as if one was sliding a magnifying lens across the system to view the states. The algorithm adaptively refines the regions, using smaller regions where more detail is required, until the difference between the bounds is below a specified accuracy. The symbolic technique is based on Multi-Terminal Binary Decision Diagrams (MTBDDs) which have been used extensively to provide compact encodings of probabilistic models. We introduce a symbolic version of the MLA algorithm, called* symbolic MLA*, which combines the power of both practical techniques when verifying MDPs. An implementation of symbolic MLA in the probabilistic model checker PRISM and experimental results to illustrate the advantages of our approach are presented.*

1 Introduction

Markov decision processes (MDPs) provide a model for systems with both probabilistic and nondeterministic behavior, and are widely used in probabilistic verification, planning, inventory optimal control, and performance analysis [13, 3, 26, 8, 25]. At every state of an MDP, one or more *actions* are available; each action is associated with a probability distribution over the successor states. We focus on *safety* and *reachability* properties of MDPs. A safety property specifies that the MDP's behavior should not leave a *safe* subset of states; a reachability property specifies that the behavior should reach a set of *target* states. A controller can choose the actions available at each state so as to maximize, or minimize, the probability of satisfying reachability and safety properties. MDPs that model realistic systems tend to have very large state spaces, and therefore the main challenge in analysing such MDPs consists in devising algorithms that work efficiently on large state spaces.

In the non-probabilistic setting, abstraction techniques have been successful in coping with large state-spaces: by ignoring details not relevant to the property under study, abstraction makes it possible to answer questions about a system through the analysis of a smaller, more concise abstract model. This has spurred research into the use of abstraction techniques for probabilistic systems [7, 17, 22, 19]. The majority of these techniques follow a *full abstraction* approach: an abstract model is constructed and, during its analysis, all details about the concrete system are forgotten.

In [11], de Alfaro and Roy proposed an alternative approach, called *magnifying-lens abstraction* (MLA) [11]. This is based on partitioning the state space of an MDP into regions and then analysing ("magnifying") the states of each region separately. The lower and upper bounds for the magnified region are updated by computing the minimum and maximum values over the states of the region. Figuratively, MLA slides a magnifying lens across the abstraction, enabling the algorithm to see the concrete states of one region at a time when updating the region values.

Regions are refined adaptively until the difference between the lower and upper bounds for all regions is within some specified accuracy. In this way, the abstraction is refined in an adaptive fashion: smaller regions are used when finer detail is required, guaranteeing the convergence of the bounds, and larger regions are used elsewhere, saving space. When splitting regions, MLA takes care to re-use information gained in the analysis of the coarser abstraction in the evaluation of the finer one. In its ability to provide both upper and lower bounds for the quantities of interest, MLA is similar to [19].

Although experimental results have demonstrated that using MLA leads to space savings, the explicit representation of the probabilistic transition system employed in [11]

978-0-7695-3360-5/08 $25.00 © 2008 IEEE
DOI 10.1109/QEST.2008.41

placed a limit on the size of MDPs that could be analysed. A successful approach to overcome the limitations of explicit representations has been to employ symbolic data structures. In particular, BDDs (binary decision diagrams) [4] and MTBDDs (multi-terminal binary decision diagrams) [6, 1] have been shown to enable the compact representation and analysis of very large MDPs [9, 23, 15].

In this work we combine MLA with symbolic representations to improve scalability. More precisely, we adapt the MLA algorithm of [11] to the symbolic domain, yielding an approach that we call Symbolic Magnifying-Lens Abstraction (SMLA). We show that the "magnified" computation performed on the regions, and the "sliding" of the magnification from one region to the next, can be performed symbolically in a natural and efficient fashion. We have implemented SMLA in the probabilistic model checking tool PRISM [16, 24] and, through a number of case studies, demonstrate that SMLA leads to useful space savings.

MLA, and its symbolic variant SMLA, differ from other approaches to MDP abstraction [19] in that they can be profitably applied to systems where there are many states with similar value, but not necessarily similar transition structure. For instance, consider a system with an integer state variable x, with range $[0, \ldots, N]$, and assume that from every state where x has value $0 < n < N$, there are transitions to states where x has values $n-1$, n, and $n+1$. Classical abstraction schemes associate with each region (set of states) a single abstract state, whose transition relation over-approximates all the transition relations of the concrete states it represents. In such a transition-based abstraction, it is not useful to group the concrete values $[0, \ldots, N]$ for x into regions consisting of intervals $I_1, \ldots, I_k$. In fact, since the states at the endpoints of each interval can leave the interval, but the states in the interior cannot, the abstract transition relation associated with each interval would have to be a gross over-approximation of the concrete transition relations, leading to considerable loss of precision.

In MLA and SMLA, as long as the value of the property of interest is similar in states in the same interval, abstraction is possible and useful. Indeed, experimentally we noticed that SMLA performs well in many problems with integer-valued state variables, where the properties vary gradually with the value of the state variables. Problems in planning, inventory control, and similar often belong to this category. On the other hand, when it is possible to use symmetry or structural knowledge of an example, and aggregate states of similar transition relation, approaches such as [7, 19, 20] yield superior results.

The outline of the paper is the following: Section 2 presents background material, including Markov decision processes (MDPs), the Magnifying-Lens Abstraction (MLA) algorithm and symbolic (MTBDD) representations. In Section 3, we describe the symbolic version of MLA and, in Section 4, present and discuss experimental results for its implementation on a range of MDP case studies. Section 5 concludes the paper.

2 Background

For a countable set S, a *probability distribution* on S is a function $p : S \mapsto [0,1]$ such that $\sum_{s \in S} p(s) = 1$; we denote the set of probability distributions on S by $D(S)$. A *valuation* over a set S is a function $v : S \mapsto \mathbb{R}$ associating a real number $v(s)$ with every $s \in S$. For $x \in \mathbb{R}$, we denote by $\mathbf{x}$ the valuation with constant value x; for $T \subseteq S$, we indicate by $[T]$ the valuation having value 1 in T and 0 elsewhere. For two valuations v, u on S, we define $||v - u|| = \sup_{s \in S} |v(s) - u(s)|$.

A *partition* of a set S is a set $R \subseteq 2^S$, such that $\bigcup\{s | s \in R\} = S$ and $r \cap r' = \emptyset$ for all $r \neq r' \in R$. For $s \in S$ and a partition R of S, we denote by $[s]_R$ the element $r \in R$ with $s \in r$. We say that a partition R is *finer* than a partition R' if for any $r \in R$ there exists $r' \in R'$ such that $r \subseteq r'$.

2.1 Markov Decision Processes (MDPs)

Definition 1 *A* Markov decision process *(MDP)* $M = \langle S, s_{init}, A, \Gamma, p \rangle$ *consists of the following components:*

- *a finite state space S;*
- *an initial state $s_{init} \in S$;*
- *a finite set A of actions (moves);*
- *a move assignment $\Gamma : S \to 2^A \setminus \emptyset$;*
- *A probabilistic transition function $p : S \times A \to D(S)$.*

At every state $s \in S$, the controller can choose an action $a \in \Gamma(s)$; the MDP then proceeds to a successor state $t \in S$ with probability $p(s, a, t)$. A *path* of the MDP M is an infinite sequence $\overline{s} = s_0, s_1, s_2, \ldots$ of states of S; we denote by S^ω the set of all paths, and we denote by $\overline{s}_k$ the k-th state s_k of the path $\overline{s}$.

We model the choice of actions, i.e. the role of the controller, through *strategies* (these are also variously called *schedulers* [26] or *policies* [13]). A *strategy* is a mapping $\pi : S^+ \mapsto D(A)$ from finite paths to distributions over actions. Given a past history $\sigma s \in S^+$ for the MDP, the strategy π chooses the action $a \in \Gamma(s)$ with probability $\pi(\sigma s)(a)$. Since, by definition, any action $b \in A \setminus \Gamma(s)$ is not enabled in s, for any strategy π we require $\pi(\sigma s)(b) = 0$ for all $b \in A \setminus \Gamma(s)$. Under the operation of strategy π, we can define a probability measure Pr_s^π over the set of infinite paths starting from state s in the standard way [18]. We denote by Π the set of all strategies.

We consider *safety* and *reachability* goals. Given a subset $T \subseteq S$ of states, the reachability goal $\Diamond T = \{\overline{s} \in S^\omega \mid \exists k.\overline{s}_k \in T\}$ consists of the paths that reach T, and

Algorithm 1 ValIter$(T, f, g, \varepsilon_{\mathit{float}})$ Value iteration

```
1. $v := [T]$
2. **repeat**
3.   $\hat{v} := v$
4.   **for all** $s \in S$ **do**
       $v(s) := f\left([T](s),\ g\left\{\sum_{s' \in S} p(s,a,s')\cdot\hat{v}(s') \mid a \in \Gamma(s)\right\}\right)$
5. **until** $||v - \hat{v}|| \le \varepsilon_{\mathit{float}}$
6. **return** $v$
```

	Property	f	g	Converges
$V^{\max}_{\Box T}$	max. safety	min	max	from above
$V^{\min}_{\Box T}$	min. safety	min	min	from above
$V^{\max}_{\Diamond T}$	max. reachability	max	max	from below
$V^{\min}_{\Diamond T}$	min. reachability	max	min	from below

Table 1. Parameters of ValIter$(T, f, g, \varepsilon_{\mathit{float}})$ used to compute reachability and safety properties.

the safety goal $\Box T = \{\bar{s} \in S^\omega \mid \forall k.\bar{s}_k \in T\}$ consists of the paths that remain in T. More precisely, we consider computing, for all $s \in S$, the probabilities:

$$V^{\max}_{\Box T}(s) = \max_{\pi\in\Pi} \Pr^\pi_s(\Box T) \qquad V^{\max}_{\Diamond T}(s) = \max_{\pi\in\Pi} \Pr^\pi_s(\Diamond T)$$
$$V^{\min}_{\Box T}(s) = \min_{\pi\in\Pi} \Pr^\pi_s(\Box T) \qquad V^{\min}_{\Diamond T}(s) = \min_{\pi\in\Pi} \Pr^\pi_s(\Diamond T)$$

that is, over all strategies, the maximum and minimum probabilities of satisfying safety and reachability goals. The fact that we can consider maximum and minimum as opposed to supremum and infimum is a consequence of the existence of optimal strategies [13].

Such reachability and safety probabilities can be computed via the classical value-iteration scheme [13, 3, 10]. The algorithm, given in Algorithm 1, is parameterized by two operators $f, g \in \{\max, \min\}$. The operator f specifies how to merge the valuation of the current state with the expected next-state valuation; $f = \max$ is used for reachability goals, and $f = \min$ for safety ones. The operator g specifies whether to select the action that maximizes, or minimizes, the expected next-state valuation; $g = \max$ is used for computing maximal probabilities, and $g = \min$ for minimal probabilities. Table 1 summarizes the effect of different parameter combinations. The algorithm is also parameterized by $\varepsilon_{\mathit{float}}{>}0$: this is the absolute error threshold below which we consider value iteration to have converged. Some value iteration algorithms use relative error bounds to check convergence.

In the remainder of the paper, unless explicitly noted, we present algorithms and definitions for a fixed MDP $M = \langle S, s_{init}, A, \Gamma, p\rangle$ and set of states T.

2.2 Magnifying-Lens Abstraction (MLA)

Magnifying-lens abstractions (MLA) [11] is a technique for the analysis of reachability and safety properties of MDPs. MLA can cluster states based on value only, disregarding differences in their transition relation. This feature can lead to compact abstractions for systems where full abstraction provides no benefit. In our experience, MLA is particularly well-suited to problems where there is a notion of *locality* in the state space, so that it makes sense to cluster states based on variable values — even though their transition relations may not be similar. Many inventory, planning and control problems are of this type.

Let v^* be the valuation over S that is to be computed: v^* is one of $V^{\min}_{\Box T}$, $V^{\max}_{\Box T}$, $V^{\min}_{\Diamond T}$, $V^{\max}_{\Diamond T}$. Given a desired accuracy $\varepsilon_{\mathit{abs}}{>}0$, MLA computes upper and lower bounds for v^*, spaced less than $\varepsilon_{\mathit{abs}}$ apart. MLA starts from an initial partition (set of regions) R of S. The initial partition R is obtained either from the user or from the property. MLA computes the lower and upper bounds as valuations u^- and u^+ over R. The partition is refined, until the difference between u^- and u^+, for all regions, is below a specified threshold. To compute u^- and u^+, MLA iteratively considers each region $r \in R$ in turn, and performs a *magnified iteration:* it improves the bounds $u^-(r)$ and $u^+(r)$ by performing value iteration on the concrete states in r.

The MLA algorithm is shown in Algorithm 2. The algorithm has parameters T, f, g, which have the same meaning as ValIter in Algorithm 1. The algorithm also has parameters $\varepsilon_{\mathit{float}}{>}0$ and $\varepsilon_{\mathit{abs}}{>}0$. Parameter $\varepsilon_{\mathit{abs}}$ indicates the maximum difference between the lower and upper bounds returned by MLA. Parameter $\varepsilon_{\mathit{float}}$, as in ValIter, specifies the degree of precision to which the local, magnified value iteration should converge. MLA should be called with $\varepsilon_{\mathit{abs}}$ greater than $\varepsilon_{\mathit{float}}$ by at least one order of magnitude: otherwise, errors in the magnified iteration can cause errors in the estimation of the bounds. Statement 2 initializes the valuations u^- and u^+ according to the property to be computed: reachability properties are computed as least fixpoints, while safety properties are computed as greatest fixpoints [10]. A region r_2 is called successor to a region r_1 if at least one concrete state in r_1 has non-zero probability to reach concrete state(s) in r_2. A useful time optimization, not shown in Algorithm 2, consists in executing the loop at lines 6–9 only for regions r where at least one of the successor regions has changed value by more than $\varepsilon_{\mathit{float}}$.

Magnified iteration. The algorithm performing the magnified iteration is given in Algorithm 3. This is very similar

Algorithm 2 MLA$(T, f, g, \varepsilon_{float}, \varepsilon_{abs})$ Magnifying-Lens Abstraction

```
1.  $R$:= some initial partition.
2.  **if** $f = \max$ **then** $u^-$:=**0**; $u^+$:=**0** **else** $u^-$:=**1**; $u^+$:=**1**
3.  **loop**
4.  **if** $f = \max$ **then** $u^+ := u^-$ **else** $u^- := u^+$
5.      **repeat**
6.          $\hat{u}^+$:=$u^+$; $\hat{u}^-$:=$u^-$;
7.          **for** $r \in R$ **do**
8.              $u^+(r)$:= MI$(r, R, T, \hat{u}^+, \hat{u}^-, \hat{u}^+, \max, f, g, \varepsilon_{float})$
9.              $u^-(r)$:= MI$(r, R, T, \hat{u}^-, \hat{u}^-, \hat{u}^+, \min, f, g, \varepsilon_{float})$
10.             **end for**
11.     **until** $\max(||u^+ - \hat{u}^+||, ||u^- - \hat{u}^-||) \le \varepsilon_{float}$
12.     **if** $||u^+ - u^-|| \ge \varepsilon_{abs}$
13.         **then** $R, u^-, u^+$:= SplitRegions$(R, u^-, u^+, \varepsilon_{abs})$
14.         **else return** $R, u^-, u^+$
15.     **end if**
16. **end loop**
```

Algorithm 3 MI$(r, R, T, u, u^-, u^+, h, f, g, \varepsilon_{float})$

```
$v$: a valuation over $r$
1. **if** $f = \max$
2.     **then for** $s \in r$ **do** $v(s) = u^-(r)$
3.     **else for** $s \in r$ **do** $v(s) = u^+(r)$
4. **repeat**
5.     $\hat{v}$:=$v$
6.     **for all** $s \in r$ **do**
6a.    $t = g_{a \in \Gamma(s)} \left\{ \sum_{s' \in r} p(s,a,s') \cdot \hat{v}(s') + \sum_{s' \in S \setminus r} p(s,a,s') \cdot u([s]_R) \right\}$
6b.    $v(s) := f\left( [T](s), t \right)$
7. **until** $||v - \hat{v}|| \le \varepsilon_{float}$
8. **return** $h\{v(s) \mid s \in r\}$
```

to ValIter in Algorithm 1, except for three points.

First, the valuation v (which here is local to r) is initialized not to $[T]$, but rather, to $u^-(r)$ if $f = \max$, and to $u^+(r)$ if $f = \min$. Indeed, if $f = \max$, value iteration converges from below, and $u^-(r)$ is a better starting point than $[T]$, since $[T](s) \le u^-(r) \le v^*(s)$ at all $s \in r$. The case for $f = \min$ is symmetric.

Second, for $s \in S \setminus r$, the algorithm uses, in place of the value $v(s)$ which is not available, the value $u^-(r')$ or $u^+(r')$, as appropriate, where r' is such that $s \in r'$. In other words, the algorithm replaces values at concrete states outside r with the "abstract" values of the regions to which the states belong. To this end, we need to be able to efficiently find the "abstract" counterpart $[s]_R$ of a state $s \in S$. We use the following scheme, similar to schemes used in AMR (adaptive mesh refinement) [2]. The state-space S of the MDP consists of value assignments to a set of variables $X = \{x_1, x_2, \ldots, x_l\}$. In [11] we represented a partition R of S, together with the valuations u^+, u^-, via a binary decision tree. The leaves of the tree correspond to regions, and they are labeled with u^-, u^+ values. Given s, finding $[s]_R$ in such a tree requires time logarithmic in $|S|$.

Third, once the concrete valuation v is computed at all $s \in r$, Algorithm 3 returns the minimum (if $h = \min$) or the maximum (if $h = \max$) of $v(s)$ at all $s \in r$, thus providing a new estimates for $u^-(r)$, $u^+(r)$, respectively.

Adaptive abstraction refinement. We denote the *imprecision* of a region r by $\Delta(r) = u^+(r) - u^-(r)$. MLA adaptively refines a partition R by splitting all regions r having $\Delta(r) > \varepsilon_{abs}$. This is perhaps the simplest possible refinement scheme. We have experimented with alternative refinement schemes, but none were consistently better [11].

As previously explained, the partition R is represented by a decision tree, whose leaves correspond to the regions. In the refinement phase, we split a leaf according to the value of a new variable (not present in that leaf), following the variable ordering given by the user.

A call to SplitRegions$(R, u^+, u^-, \varepsilon_{abs})$ returns a triple $\tilde{R}, \tilde{u}^-, \tilde{u}^+$, consisting of the new partition with its upper and lower bounds for the valuation.

Correctness. The following theorem summarizes the correctness of the MLA algorithm.

Theorem 2 *[11] For any MDP $M = \langle S, s_{init}, A, \Gamma, p \rangle$, set of states $T \subseteq S$, and error bound $\varepsilon_{abs} > 0$, the following assertions hold.*

1. Termination. *For all $\varepsilon_{float} > 0$ and $f, g \in \{\min, \max\}$, the call MLA$(T, f, g, \varepsilon_{float}, \varepsilon_{abs})$ terminates.*
2. (Partial) correctness. *Let $g \in \{\max, \min\}$, $\varepsilon_{abs} > 0$ and $\triangle \in \{\Box, \Diamond\}$. If $f = \min$ when $\triangle = \Box$, and $f = \max$ when $\triangle = \Diamond$, then for all $\delta > 0$, there exists $\varepsilon_{float} > 0$ such that:*

$$\forall r \in R: \quad u^+(r) - u^-(r) \le \varepsilon_{abs}$$
$$\forall s \in S: \quad u^-([s]_R) - \delta \le V^g_{\triangle T}(s) \le u^+([s]_R) + \delta$$

where $(R, u^-, u^+) = $ MLA$(T, f, g, \varepsilon_{float}, \varepsilon_{abs})$.

We note that the theorem establishes the correctness of lower and upper bounds only within a constant $\delta > 0$, which depends on ε_{float}. This limitation is inherited from the value-iteration scheme used over the magnified regions. If linear programming [13, 3] were used instead, then MLA would provide true lower and upper bounds. However, in practice value iteration is preferred over linear programming, due to its simplicity and greater efficiency, and the concerns about δ are solved - in practice, albeit not in theory - by choosing a sufficiently small $\varepsilon_{\mathit{float}}$.

2.3 Symbolic model checking of MDPs

Due to the sizes of the MDPs that typically arise in probabilistic verification case studies, considerable effort has been invested into building efficient model checking implementations. In particular, *symbolic* techniques, which use extensions of Binary Decision Diagrams (BDDs), have proved successful in this area. In this paper we focus on the use of Multi-Terminal Binary Decision Diagrams (MTBDDs). This data structure lies at the heart of the probabilistic model checker PRISM and has been used to model check quantitative properties of probabilistic models with as many as 10^{10} states (see for example [21, 14]). In this section, we give a brief overview of these techniques. For more detailed coverage of the MTBDD-based implementation of MDP model checking in PRISM, see [23].

MTBDDs. Multi-terminal BDDs (MTBDDs) are rooted, directed acyclic graphs associated with a set of ordered, Boolean variables $x_1 < \ldots < x_n$. An MTBDD M represents a function $f_{\mathsf{M}}(x_1, \ldots, x_n) : \mathbb{B}^n \to \mathbb{R}$ over these variables. The graph contains two types of nodes: *non-terminal* and *terminal*. A non-terminal node m is labeled by a variable $var(m) \in \{x_1, \ldots, x_n\}$ and has two children, $then(m)$ and $else(m)$. A terminal node m is labeled by a real number $val(m)$. The Boolean variable ordering $<$ is imposed onto the graph by requiring that a child m' of a non-terminal node m is either terminal or non-terminal and satisfies $var(m) < var(m')$. The value of $f_{\mathsf{M}}(x_1, \ldots, x_n)$, the function which the MTBDD represents, is determined by traversing M from the root node, and at each subsequent node m taking the edge to $then(m)$ or $else(m)$ if $var(m)$ is 1 or 0 respectively. A BDD is simply an MTBDD with the restriction that labels on terminal nodes can only 0/1.

Representation of MDPs using MTBDDs. MTBDDs have been used, from their inception [1, 6], to encode real-valued vectors and matrices. An MTBDD v over variables $(x_1, \ldots, x_n)$ represents a function $f_{\mathsf{v}} : \mathbb{B}^n \to \mathbb{R}$. A real vector $\mathbf{v}$ of length 2^n is simply a mapping from $\{1, \ldots, 2^n\}$ to the reals $\mathbb{R}$. Using the standard binary encoding of integers, the variables $\{x_1, \ldots, x_n\}$ can represent $\{1, \ldots, 2^n\}$. Hence, an MTBDD v can represent the vector $\mathbf{v}$.

In a similar way, we can consider a square matrix M of size 2^n by 2^n to be a mapping from $\{1, \ldots, 2^n\} \times \{1, \ldots, 2^n\}$ to $\mathbb{R}$. This can be represented by an MTBDD over $2n$ variables, n for rows (current-state variables) and n for columns (next-state variables). According to the commonly-used heuristic for minimizing MTBDD size, the variables for rows and columns are ordered alternately.

MTBDDs can thus easily represent the probabilistic transition matrix of a Markov chain. Furthermore, with a simple extension of this scheme, the probabilistic transition function $p : S \times A \to D(S)$ of an MDP can also be represented. Since the set of actions A is finite, we can view p as a function $S \times A \times S \to [0, 1]$. For an MDP with 2^n states, and letting $k = \mathrm{ceil}(\log_2 |A|)$, the probabilistic transition function p is equivalently seen as a function from $\{1, \ldots, 2^n\} \times \{1, \ldots, 2^k\} \times \{1, \ldots, 2^n\}$ to $\mathbb{R}$, which can easily be represented by an MTBDD over $2n + k$ variables.

MTBDDs are efficient because they are stored in reduced form, with duplicate nodes merged and redundant ones removed. Their size (number of nodes) is heavily dependent on the ordering of their Boolean variables. Although the problem of deriving the optimal ordering for a given MTBDD is an NP-hard problem, by using heuristics [15, 23], probabilistic models with a degree of regularity can be represented extremely compactly by MTBDDs.

Model checking of MDPs using MTBDDs. Once a model's MTBDD representation has been constructed, it can be analyzed, for example using value iteration to compute minimum and maximum reachability and safety probabilities. This comprises two stages. First, a graph-based analysis is performed to find the states for which the corresponding probability is 0 or 1 [9]. This can be implemented using standard BDD techniques for calculating fixpoints. Secondly, numerical computation is applied to compute probabilities for the remaining states. For this, standard iterative methods such as value iteration, can be implemented using standard MTBDD operations, including for example algorithms from [1, 6] for matrix-vector multiplication.

3 Symbolic MLA

In this section, we present a symbolic implementation of the MLA algorithm using MTBDDs. Before doing so, we highlight some important aspects of the implementation.

We first note that a potential obstacle in the use of MLA is that, although substantial savings in terms of storage for solution vectors can be made, there is still a need to store the probabilistic transition function of the MDP in full. A symbolic approach alleviates this problem: it is often the case that a very compact MTBDD representation of the probabilistic transition function of the MDP can be constructed.

Secondly, it is also common that *qualitative* probabilistic verification (i.e. checking for which states of the MDP the probabilities for a reachability/safety property are exactly 0 or 1) can be applied to much larger models than can be analysed quantitatively. This is because qualitative properties can be model checked using only graph-based algorithms that operate on the underlying transition relation, allowing an efficient implementation with BDDs. This means that a symbolic version of MLA can also benefit from this: qualitative verification is applied to the full MDP *before* applying the MLA algorithm (this process is often referred to as *pre-computation*). Numerical computation need then only be done for states with a probability that is neither 0 or 1. Furthermore, states with probability 0 or 1 can be removed from the MDP completely, reducing computation significantly and decreasing round-off errors.

Finally, we observe that symbolic techniques are very well-suited to MLA, in terms of the representation of solution vectors. Recall that, because of the way that MLA operates, it requires separate storage of the numerical solution vector for the current region being magnified (by algorithm MI see Section 2.2) and the lower/upper bounds for each region. Furthermore when the value for a state not in the current magnified region is required, the region contains that state must be determined before the relevant value can be looked up. Because of the way that MTBDDs exploit regularity, representing real-valued vectors with many similar values is often very efficient. This allows us to store the solution vector for all states of the MDP concurrently, avoiding potentially expensive partition look-ups. Since MLA considers each region sequentially, the solution vector will contain fewer distinct values than would be required for standard value iteration. Thus, we expect a symbolic implementation of MLA to be less memory-intensive than a symbolic version of value iteration.

3.1 Symbolic Magnifying-Lens Abstraction (SMLA)

The symbolic version of MLA is shown in Algorithm 4. As for standard MLA (Algorithm 2), the symbolic version is parameterized by operators $f, g \in \{\max, \min\}$ (used to select maximum/minimum reachability/safety properties) and convergence thresholds ε_{float} and ε_{abs}. The other parameter is a BDD T representing the set of target states (T in Algorithm 2). We also assume a BDD reach representing the set of reachable states of the MDP and an MTBDD trans representing its probabilistic transition function. In the latter, the MTBDD variables representing the rows (source states), columns (target states) and nondeterminism (actions) are denoted *rvars*, *cvars* and *ndvars*, respectively.

The first part of Algorithm 4 (lines 1-5) shows the use of BDD-based pre-computation steps [9, 23] in order to obtain the BDDs yes and no, representing the sets of states for which the probability is exactly 1 or 0, respectively. If this covers all states of the MDP, no further work is required. Otherwise, rows corresponding to states in yes or no are removed from the probabilistic transition function trans (line 5). Here (and elsewhere in the algorithms) we use a simple infix notation to denote the application of binary operators (such as $\vee$ or $\times$) to BDDs or MTBDDs. This is done using the standard APPLY operator [4].

The remainder of Algorithm 4 comprises the symbolic version of MLA. We start with an initial partition R, returned by the CreateInitialPartition() routine (see Section 3.3 for details). The partition is implemented as a list of BDDs, each one representing a region in R. The main part of Algorithm 4 corresponds quite closely to the original MLA algorithm (Algorithm 2). Initialization of solution vectors (lines 8 and 9) is easily achieved using the MTBDD operation $\textsc{Const}(k)$ which returns the trivial MTBDD representing the real value k. Similarly, checking for convergence of the main loop can be done with the operation $\textsc{MaxDiff}(\mathsf{u}_1, \mathsf{u}_2)$ which computes the maximum point-wise difference between MTBDDs u_1 and u_2).

The MTBDDs representing the lower (u^-) and upper (u^+) bounds for each region are computed by the SMI function, described below. After a global iteration terminates, the algorithm calls the Split(...) method to refine the regions for which the difference between the lower and upper bounds (u^- and u^+) is greater than ε_{abs}. After each refinement, the algorithm copies u^- values to u^+ for the reachability objectives and u^+ values to u^- for safety objectives.

Algorithm 4 SMLA($\mathsf{T}, f, g, \varepsilon_{float}, \varepsilon_{abs}$) Symbolic Magnifying-Lens Abstraction

```
1. if $g = \max$
2.    then no := $\textsc{Prob0A}(\mathsf{T})$ ; yes := $\textsc{Prob1E}(\mathsf{T})$
3.    else no := $\textsc{Prob0E}(\mathsf{T})$ ; yes := $\textsc{Prob1A}(\mathsf{T})$
4. if no $\vee$ yes = reach then return yes
5. trans' := trans $\times \neg$(no $\vee$ yes)
6. $R$ := CreateInitialPartition()
7. if $f = \max$
8.    then u$^-$ := u$^+$ := $\textsc{Const}(0)$
9.    else u$^-$ := u$^+$ := $\textsc{Const}(1)$
10. loop
11.    repeat
12.       û$^+$ := u$^+$; û$^-$ := u$^-$
13.       for each r $\in R$ do
14.          û$^+$:= SMI(r, $R$, trans', yes, û$^+$, $\max, f, g, \varepsilon_{float}$)
15.          û$^-$:= SMI(r, $R$, trans', yes, û$^-$, $\min, f, g, \varepsilon_{float}$)
16.       end for
17.    until $\textsc{MaxDiff}$(u$^+$, û$^+$) $\leq \varepsilon_{float}$ &
          $\textsc{MaxDiff}$(u$^-$, û$^-$) $\leq \varepsilon_{float}$
18.    if $\textsc{MaxDiff}$(u$^+$, u$^-$) $\geq \varepsilon_{abs}$
19.       then $R$, u$^-$, u$^+$ := Split($R$, u$^-$, u$^+$, $\varepsilon_{abs}$)
20.       else return (u$^-$ + u$^+$)/2
21.    end if
22.    if $f = \max$ then u$^+$ := u$^-$ else u$^-$ := u$^+$
23. end loop
```

3.2 Symbolic Magnified Iteration (SMI)

Algorithm 5 SMI($\mathsf{r}, R, \mathsf{trans}', \mathsf{T}, \mathsf{u}, h, f, g, \varepsilon_{float}$) Symbolic Magnified Iteration

```
1. $\mathsf{v} := \mathsf{u}$
2. $\mathsf{trans}'' := \mathsf{trans}' \times \mathsf{r}$
3. $done$ := **false**
4. **while** ($done$ != **true**) **do**
5.     $\mathsf{tmp} := \textsc{Permute}(\mathsf{v},\ rvars,\ cvars)$
6.     $\mathsf{tmp} := \textsc{MVMult}(\mathsf{trans}'', \mathsf{tmp})$
7.     $\mathsf{tmp} := \textsc{Replace}(g, \mathsf{tmp}, ndvars)$
8.     $\mathsf{tmp} := \textsc{Apply}(f, \mathsf{tmp}, \mathsf{T})$
9.     $\mathsf{v}' := \mathrm{ITE}(\mathsf{r}, \mathsf{tmp}, \mathsf{u})$
10.    **if** $\textsc{MaxDiff}(\mathsf{v}', \mathsf{v}) < \varepsilon_{float}$ **then** $done$ := **true**
11.    $\mathsf{v} := \mathsf{v}'$
12. **end while**
13. **if** ($h = \max$)
14.    **then** $val := \textsc{FindMax}(\mathrm{ITE}(\mathsf{r}, \mathsf{v}, \textsc{Const}(0)))$
15.    **else** $val := \textsc{FindMin}(\mathrm{ITE}(\mathsf{r}, \mathsf{v}, \textsc{Const}(1)))$
16. **return** $\mathrm{ITE}(\mathsf{r}, \textsc{Const}(val), \mathsf{u})$
```

The core part of the MTBDD-based implementation of MLA is called *Symbolic Magnified Iteration (SMI)* and is shown in Algorithm 5. It performs a symbolic value iteration algorithm inside the region represented by BDD r from the current partition R. The algorithm is also passed the MTBDD trans' representing the (filtered) probabilistic transition function of the MDP, the BDD T representing the set of target states, and the MTBDD u, which stores the (upper or lower) bound for every state's corresponding region. The other parameters h, f, g and ε_{float}, are as for the non-symbolic version in Algorithm 3.

The algorithm initializes the solution vector v with the vector u (line 1) and then the MTBDD trans' is filtered further to include only transitions for the current region (line 2). The loop (lines 3-12) updates the solution vector v until the results of two successive iterations differ less than ε_{float}. The first two lines of the loop perform a matrix-vector multiplication of the transition probability matrix of the MDP with (a permuted copy of) the solution vector v. This corresponds to the summations in line 6a of Algorithm 3. In line 7, the operator $g \in \{\max, \min\}$ is applied over the nondeterministic variables $ndvars$ of the resulting MTBDD (the first part of line 6a from Algorithm 3). In line 8, the operator f is applied point-wise with the BDD T representing the target states (line 6b of Algorithm 3). Finally, the new solution vector v' is computed by setting values for all states not in the current region (r) to their values in u, using the MTBDD operation ITE (If-Then-Else).

Once the while loop terminates, the algorithm computes the maximum (if $h = \max$) or minimum (if $h = \min$) value val of the region by using FINDMAX (or FINDMIN). Finally the algorithm returns a solution vector with value val for the current region and the old solution value from u for all other regions.

3.3 The Splitting Order

The creation of the initial partition and the way in which it is subsequently split are governed by two user parameters: *strat* and *level*. Splitting operations are based on a priority order $X_{ord} = \langle x_1, x_2, \ldots, x_n \rangle$ of the MTBDD variables representing the state space of the MDP. In the adaptive refinement scheme of MLA, each call to the routine Split subdivides a region into two using the next MTBDD variable from the order X_{ord} (we call this the *splitting index*). Since the MLA algorithm does not refine regions with $u^+(r) - u^-(r) \le \varepsilon_{abs}$, after a refinement, different regions may have different splitting indices.

The order X_{ord} is determined by the choice of a splitting strategy *strat*: either "*consecutive*" or "*interleaved*". In the default MTBDD variable ordering (for an MDP derived from a PRISM model), MTBDD variables are grouped according to the (model-level) variable to which they correspond and ordered consecutively. For *strat*=*consecutive*, we take X_{ord} to be this default ordering. For *strat*=*interleaved*, on the other hand, the MTBDD variables corresponding to different (model-level) variables are interleaved.

The initial creation of a partition (by routine CreateInitialPartition) is determined by $X_{ord} = \langle x_1, x_2, \ldots, x_n \rangle$ and the parameter *level*. Each region in the initial partition is created by splitting on MTBDD variables $x_1, x_2, \ldots, x_{level}$ (i.e. the splitting index for each region is *level*).

4 The Case Studies and Results

We have implemented the symbolic MLA algorithm within the probabilistic model checker PRISM and, in this section, present results for the following MDP case studies.

Inventory Problem. We have modeled an inventory as an MDP. The variable "$stock$" denotes the current number of items in the inventory and "$init$" denotes the initial item count. The variable "$time$" keeps track of time elapsing. At each time step, the demand of the item is 1 with a probability p and 0 with $1 - p$. The probability p is a function of current number of items present in the inventory. The manager of the inventory visits the inventory every 7 time units and he has two actions to choose from: either place an order or do not place one. The property we are checking is the "minimum probability that the stock reach its minimum amount within MAXTIME time units". In PCTL, the reachability property can be expressed as $P_{\min=?}[\Diamond\ (stock{=}1 \wedge time{<}MAXTIME))]$.

Example	Parameters	States	Transitions	PRISM		MLA		
				Time	Nodes	Time	Nodes	Regions
Inventory	stock=512, MAXTIME=512	26,870	34,568	14	13,885	15	2,612	340
	stock=1024, MAXTIME=1,024	106,734	135,308	54	26,005	61	4,825	676
	stock=2048, MAXTIME=2,048	425,438	535,508	233	50,563	270	9,192	1,348
	stock=4096, MAXTIME=4,096	1,698,750	2,130,788	896	99,084	1,056	17,882	2,692
	stock=5120, MAXTIME=5,120	2,653,358	3,325,868	1,243	120,220	1,424	21,875	3,364
	stock=10240, MAXTIME=10,240	10,605,918	13,275,668	7,118	240,639	7,551	43,563	3,363
Minefield	n=256,m=100	65,537	299,748	75	56,971	263	8,173	2,041
	n=512,m=200	262,145	1,128,522	627	91,285	1,493	14,070	4,164
	n=1024,m=300	1,048,577	4,316,719	3,625	126,603	5,463	20,244	6,324
Hotel Booking	c=127 bk=63 MAXTIME=15	131,072	645,168	4	30,580	46	8,710	903
	c=255 bk=127 MAXTIME=31	1,048,576	5,202,016	44	118,080	1,013	37,696	6,350
	c=511,bk=255,MAXTIME=31	4,194,304	20,889,696	2,072	373,536	9,971	118,310	25,491
Secretary	c=100 MAXTIME=100	30,697	61,388	2	14,577	7	3,269	269
	c=100 MAXTIME=200	90,496	180,587	3	17,146	11	3,716	345
	c=200 MAXTIME=200	121,397	242,788	10	32,728	27	6,967	471
	c=300,MAXTIME=400	451,896	903,387	24	55,430	62	9,748	463
	c=500,MAXTIME=1000	2,252,496	4,502,987	88	105,971	199	17,425	733
	c=1000,MAXTIME=2000	9,004,996	18,005,987	392	232,946	802	32,438	768
Zeroconf	N=4,M=32,K=4	26,121	50,624	88	126,731	50	14,430	22
	N=8, M=32, K=4	552,097	1,728,272	1,307	722,224	650	49,464	64
	N=8, M=128, K=4	2,092,513	6,552,368	3,221	857,577	2,593	151,289	19

Figure 1. Experimental results: Symbolic MLA, compared to PRISM

Robot in a Minefield. We consider the problem of navigating an $n \times n$ minefield. The minefield contains m mines, each with coordinates (x_i, y_i), for $1 \leq i \leq m$, where $1 \leq x_i < n$, $1 \leq y_i < n$. We consider the problem of computing the maximal probability with which a robot can reach the target corner (n, n), from all $n \times n$ states. At interior states of the field, the robot can choose among four actions: *Up, Down, Left, Right;* at the border of the field, actions that lead outside of the field are missing. From a state $s = (x, y) \in \{1, \ldots, n\}^2$ with coordinates (x, y), each action causes the robot to move to square (x', y') with probability $q(x', y')$, and to "blow up" (move to an additional sink state) with probability $1 - q(x', y')$. For action *Right,* we have $x' = x + 1$, $y' = y$; similarly for the other actions. The probability $q(x', y')$ depends on the proximity to mines, and is given by

$$q(x', y') = \prod_i^m \exp\big(-0.7 \cdot \big((x' - x_i)^2 + (y' - y_i)^2\big)\big).$$

Optimal Stopping Game: Secretary Selection. We have modeled one application of the optimal stopping game. One boss starts interviewing c candidates for the post of secretary. After each interview, he can either select the candidate or continue the process with the remaining candidates. If the boss does not select the candidate, then the candidate is eliminated from the selection process. The variable "$time$" is used to keep track of the time that has elapsed.The boss can compare whether the current candidate is the best so far or if a better candidate was interviewed previously. If the current candidate is the best among all candidates seen, then the variable "$best$" is assigned to 1. The boss does not know the (merit) order of the candidates; hence we model assignment of the variable with a probabilistic update. The probability that the current one is the best among c candidates is set equal to $1/c$. If the boss selects a candidate, then the variable "$stop$" is assigned to 1. The property we are checking is the "maximum probability that the interviewer has selected a non-best candidate before the timeout". In PCTL, the reachability property can be expressed as $P_{\max=?}[\Diamond\ (stop{=}1 \wedge best{=}0 \wedge time{<}MAXTIME)]$.

Hotel Booking Problem. We have modeled an instance of the overbooking problem for a hotel during a multiple-day conference. The conference-chairperson books b rooms for the registered participants in a hotel with v rooms. The variable "$days$" keeps track of days that have elapsed since the start of the conference. The participants can appear at any day during the conference but some of the booked rooms remain vacant during the conference season due to "no-show" of the participants. The hotel manager takes this factor into account and overbooks the hotel during the peak seasons. When he books a hotel room and the conference participant does not appear, the manager suffers a loss. Similarly he will be in trouble whenever he allows a non-conference visitor without keeping a room booked and the conference guest appears, requiring him to find an alternative room for the guest at higher cost. The arrival of the

strat	*level*	Nodes	Time (in s)	Regions	*strat*	*level*	Nodes	Time(in s)	Regions
consecutive	1	60,563	50	191	*interleaved*	1	60,563	254	942
consecutive	4	19,732	57	191	*interleaved*	4	40,599	255	942
consecutive	7	12,129	60	214	*interleaved*	7	18,054	258	946
consecutive	11	11,491	95	752	*interleaved*	11	10,060	307	1057
consecutive	15	13,194	191	3043	*interleaved*	15	11,632	441	2705

Figure 2. Effect of splitting strategy and initial splitting index (Secretary: $c{=}300, MAXTIME{=}400$)

participants is probabilistic. The property we are checking will be the "maximum probability that a conference guest arrives within the duration of the conference and does not get a room". In PCTL, the reachability property can be expressed as $P_{\max=?}[\Diamond\ (v{=}0 \wedge b{>}0 \wedge days{<}MAXTIME)]$.

Zeroconf Protocol. The Zeroconf protocol [5] is used for the dynamic self-configuration of a host joining a network; it has been used as a testbed for the abstraction method considered in [19]. We consider a network with N existing hosts, and M total IP addresses; protocol messages have a certain probability of being lost during transmission. The variable K denotes the maximum number of probes can be sent by the new host. We consider the problem of determining the maximal probability of a host eventually acquiring an IP address.

Results. Our experiments were run on an Intel 2.16 GHz machine with 2GB RAM. We used ε_{float} = 0.01, ε_{abs}=0.1 for both PRISM and MLA and, unless otherwise stated (see next section), an initial splitting index (*level*) of $\lfloor k/2 \rfloor$, where k is the number of MTBDD variables representing the MDP's state space. For the splitting strategy (*strat*), we used "*consecutive*" for all model, except the minefield.

Figure 1 summarizes the results for all case studies. The first two columns show the name and parameters of the MDP model. The third and fourth columns gives the number of states and transitions for each model. The remaining columns show the performance of analysing the MDPs, using both PRISM and symbolic MLA. In both cases, we give the total time required (which includes model building and model checking) and the peak MTBDD node count (which includes the partial transition relation and the solution vectors). For MLA, we also show the final number of generated regions. We used the MTBDD engine of PRISM, since (a) it is generally the best performing engine for MDPs; and (b) it is the only one that can scale to the size of models we are aiming towards. More detailed experimental data is available from:
`www.soe.ucsc.edu/~pritam/qest08.html.`

Discussion. The "Nodes" columns of Figure 1 demonstrate the efficiency of the symbolic implementation of MLA: the memory requirements are significantly lower than the equivalent statistics for PRISM's MTBDD engine. As discussed earlier in Section 3, this is due to the fact that MLA analyzes each region in isolation, resulting in a smaller number of distinct values in the solution vectors. For the Zeroconf example, this phenomenon actually results in MLA also outperforming PRISM in terms of solution time.

It is also clear, from the sizes of the MDPs in the table, that the symbolic version of MLA is able to handle MDPs considerably larger than were previously feasible for the existing explicit implementation of [11]. Thanks to this, another positive conclusion which we can draw from the results is that MLA generates relatively small numbers of regions for the analysis of even large MDPs.

Finally, we also experimented with different parameter values for the splitting strategy (*strat*) and initial splitting index (*level*). Figure 2 shows results for the secretary selection case study (c = 300 and $MAXTIME$ = 400). For smaller values of the initial splitting index, there are less regions initially but these regions are relatively large, resulting in higher memory consumption. Increasing the splitting index produces smaller regions, which take less space and time to analyse, however more global iterations are required, resulting in longer total solution times. Hence, in our results (Figure 1), we opted for a trade-off by using a splitting index close to $k/2$, where k is the number of MTBDD variables representing the state space.

For the results in Figure 2 (and for most of our case studies), the "*consecutive*" strategy performs better than the "*interleaved*" strategy, both in terms of memory usage, time and number of regions. For the minefield problem, however, the reverse is true. This is due to the "grid-like" nature of the model and the fact that the state-space is described by a pair of co-ordinates, x and y. It is more effective to refine the state space into square regions of the grid.

5 Conclusion

We have presented a symbolic implementation of the magnifying-lens abstraction (MLA) technique of [11], using the multi-terminal binary decision diagram (MTBDD) data structure. This was implemented in the probabilistic model checker PRISM and applied to a range of MDP case studies. The results demonstrate that symbolic MLA yields significant gains in memory usage over standard (symbolic) implementations of MDP verification, as provided by PRISM. Furthermore, in some cases this also produce better performance in terms of time. Our results also show that symbolic MLA can be applied to much larger MDPs than its explicit counterpart.

In the future, we plan to make a comparison of our approach with other MDP abstraction techniques, including the game-based approach of [19]. We aso plan to investigate the integration of more advanced symbolic representations of state space partitions, such as [12].

Acknowledgment

Parker and Norman are supported in part by EPSRC grants GR/S11107 and GR/S46727.

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 System Design*, 10(2/3):171–206, 1997.

[2] J.B. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, 1984.

[3] D.P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995. Volumes I and II.

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

[5] S. Cheshire, B. Adoba, and E. Gutterman. Dynamic configuration of IPv4 link local addresses. Available from http://www.ietf.org/rfc/rfc3927.txt.

[6] 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. *Formal Methods in System Design*, 10((2/3):149–169, 1997.

[7] P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reachability analysis of probabilistic systems by successive refinements. In *Proc. PAPM/PROBMIV'01*, volume 2165 of *LNCS*, pages 39–56. Springer, 2001.

[8] L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997. Technical Report STAN-CS-TR-98-1601.

[9] 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. TACAS'00*, volume 1785 of *LNCS*, pages 395–410. Springer, 2000.

[10] L. de Alfaro and R. Majumdar. Quantitative solution of omega-regular games. *Journal of Computer and System Sciences*, 68:374–397, 2004.

[11] Luca de Alfaro and Pritam Roy. Magnifying-lens abstraction for Markov decision processes. In *Proc. CAV'07*, volume 4590 of *LNCS*, pages 325–338. Springer, 2007.

[12] S. Derisavi. A symbolic algorithm for optimal Markov chain lumping. In O. Grumberg and M. Huth, editors, *Proc. TACAS'07*, volume 4424 of *LNCS*, pages 139–154. Springer, 2007.

[13] C. Derman. *Finite State Markovian Decision Processes*. Academic Press, 1970.

[14] M. Duflot, M. Kwiatkowska, G. Norman, and D. Parker. A formal analysis of Bluetooth device discovery. *Int. Journal on Software Tools for Technology Transfer*, 8(6):621–632, 2006.

[15] 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.

[16] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. TACAS'06*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[17] M. Huth. On finite-state approximations for probabilistic computational-tree logic. *Theor. Comp. Sci.*, 346(1):113–134, 2005.

[18] J.G. Kemeny, J.L. Snell, and A.W. Knapp. *Denumerable Markov Chains*. D. Van Nostrand Company, 1966.

[19] M. Kwiatkowska, G. Norman, and D. Parker. Game-based abstraction for markov decision processes. In *Proc. QEST'06*, pages 157–166. IEEE CS, 2006.

[20] M. Kwiatkowska, G. Norman, and D. Parker. Symmetry reduction for probabilistic model checking. In T. Ball and R. Jones, editors, *Proc. CAV'06*, volume 4114 of *LNCS*, pages 234–248. Springer, 2006.

[21] M. Kwiatkowska, G. Norman, and R. Segala. Automated verification of a randomized distributed consensus protocol using Cadence SMV and PRISM. In G. Berry, H. Comon, and A. Finkel, editors, *Proc. CAV'01*, volume 2102 of *LNCS*, pages 194–206. Springer, 2001.

[22] D. Monniaux. Abstract interpretation of programs as Markov decision processes. *Science of Computer Programming*, 58(1–2):179–205, 2005.

[23] D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems*. PhD thesis, University of Birmingham, 2002.

[24] PRISM web site. www.prismmodelchecker.org.

[25] J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems,* P. Panangaden and F. van Breugel (eds.), volume 23 of *CRM Monograph Series*. American Mathematical Society, 2004.

[26] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, 1995. Technical Report MIT/LCS/TR-676.