Symblicit Calculation of Long-Run Averages for Concurrent Probabilistic Systems*

Ralf Wimmer Bettina Braitling Bernd Becker
Albert-Ludwigs-University Freiburg, Germany
{wimmer|braitlin|becker}@informatik.uni-freiburg.de

Ernst Moritz Hahn Pepijn Crouzen Holger Hermanns
Saarland University, Saarbrücken, Germany
{emh|crouzen|hermanns}@cs.uni-sb.de

Abhishek Dhama Oliver Theel
Carl von Ossietzky University Oldenburg, Germany
{abhishek.dhama|theel}@informatik.uni-oldenburg.de

***Abstract*—Model checkers for concurrent probabilistic systems have become very popular within the last decade. The study of long-run average behavior has however received only scant attention in this area, at least from the implementation perspective. This paper studies the problem of how to efficiently realize an algorithm for computing optimal long-run average reward values for concurrent probabilistic systems. At its core is a variation of Howard and Veinott's algorithm for Markov decision processes, where symbolic and non-symbolic representations are intertwined in an effective manner: the state space is represented using binary decision diagrams, while the linear equation systems which have to be solved for the induced Markov chains to improve the current scheduler are solved using an explicit state representation. In order to keep the latter small, we apply a symbolic bisimulation minimization algorithm to the induced Markov chain. The scheduler improvement step itself is again performed on symbolic data structures. Practical evidence shows that the implementation is effective, and sometimes uses considerably less memory than a fully explicit implementation.**

I. Introduction

Model checking of concurrent probabilistic systems with tools like PRISM [1] is very popular these days. One of the core concurrency models in this context is the model of probabilistic automata (PA), which are akin to Markov decision processes (MDPs) [2]. The study of long-run behavior for such models has however received only scant attention, especially from the implementation perspective. On the other hand, the study of long-run behavior is well understood for Markov chain models, and efficient implementations exist in many tools.

This paper aims at filling this gap, and aims at calculating minimum or maximum long-run average reward properties for MDPs. The computation of such properties is important not only in the concurrent system context. Applications involve models from such different areas as highway pavement maintenance, inventory management, biological models [2], and self-stabilizing systems [3]. Depending on the application, long-run average rewards may correspond to the maximum expected fraction of time a system is working if it is scheduled in an optimal way, the minimum cost per time unit on the long run if an optimal policy is used, the extremal expected fractions of time a self-stabilizing system is up and working, etc.

De Alfaro and Bianco [4], [5] have put forward techniques to specify general properties based on average long-run values within a probabilistic logic. Their algorithm is based on solving an explicit-state problem via linear programming, after a number of transformation steps. While this approach is conceptually very elegant, we are not aware of attempts to arrive at an efficient and scalable implementation that solves the problems specified. When considering complex systems comprising multiple components, the number of states to consider grows rapidly, often exponentially with the number of processes. Because of this, the applicability of an explicit-state implementation of such an algorithm is severely constrained by the available memory. To alleviate such problems, symbolic, i. e., BDD-based techniques are known to have potential. To the best of our knowledge however, no symbolic approach to compute long-run average reward properties for MDPs has been devised yet.

This paper develops an efficient algorithm for computing long-run average reward properties for MDPs. It harvests experience gained in the combination of symbolic and explicit algorithms [6], pioneered by the hybrid approach [7] used within PRISM. Our *symblicit* approach combines symbolic and explicit analysis steps in a sophisticated manner. As an algorithmic basis, we employ a variant of the algorithm of Howard and Veinott [8], [9], [2] to compute minimum or maximum long-run average reward properties of MDPs. The basic idea of our approach is as follows. We maintain

*This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS) (see www.avacs.org for more information) and the Graduiertenkolleg "Leistungsgarantien für Rechnersysteme"

the MDP using a symbolic representation. However, certain parts of our calculations are done for very small explicit models. Doing so, we swap back and forth between the symbolic and explicit world, exploiting the benefits of both. This way we are able to handle very large models, which are impossible to store in an explicit-state manner. Using a fully symbolic approach instead is feasible, but is unfavorable owed to time requirement for computing necessary solutions of linear equation systems.

To illustrate our approach, we report on applications of the symblicit algorithm to a number of case studies. Some of these cases are well known from other publications [10], [11], and one of them [12] is a protocol that arose in the study of self-stabilizing algorithms [3]. For each case, we compare runtime and memory usage with an explicit-state implementation. The results are encouraging. With a slight modification of the symblicit algorithm, we can also compute unbounded reachability probabilities for MDPs [4] in an effective way. However for unbounded reachability the results turn out to be inconclusive, as we will discuss.

II. Preliminaries

In this section we briefly restate the general theory of Markov decision processes, the policy iteration algorithm for optimal long-run average rewards by Howard and Veinott [8], [9] as described by Puterman [2] and finally the symbolic representation of our models.

A. Stochastic Models and Bisimulation

We denote the collection of all probability distributions on (Borel) subsets of a set S as $\mathit{Distr}(S)$.

Definition 1: A *discrete-time Markov chain (MC)* is a tuple $(S, \mathbf{P})$ where S is a finite set of states and $\mathbf{P} : S \times S \to [0,1]$ with $\sum_{s' \in S} \mathbf{P}(s,s') = 1$ (or equivalently $\mathbf{P} : S \to \mathit{Distr}(S)$ is a stochastic matrix).
An MC $\mathbf{D}$ together with an initial state $s_0 \in S$ is a discrete-time stochastic process $(X_{\mathbf{D},s_0}(n))_{n \in \mathbb{N}}$ such that $P(X_{\mathbf{D},s_0}(0) = s_0) = 1$ and $P(X_{\mathbf{D},s_0}(n+1) = s' \mid X_{\mathbf{D},s_0}(n) = s) = \mathbf{P}(s,s')$.

Definition 2: A *Markov decision process* (MDP) is a tuple $(S, A, \mathbf{P})$ where S is the finite state space, A is a finite set of actions and $\mathbf{P} \subseteq S \times A \times \mathit{Distr}(S)$ is the transition relation such that for every pair $(s,a) \in S \times A$ we have at most one transition $(s,a,\pi) \in \mathbf{P}$.
We say that an action $a \in A$ is *enabled* in a state $s \in S$ if there exists a triple $(s,a,\pi) \in \mathbf{P}$. We denote the set of enabled actions in a state $s \in S$ as A_s and require $A_s \neq \emptyset$. In each state $s \in S$, we have a nondeterministic choice between the enabled actions. To obtain a stochastic process, this nondeterminism must be resolved by a *scheduler*.

Definition 3: A *stationary deterministic scheduler* is a function $\sigma : S \to A$ mapping each state s to an enabled action $a \in A_s$.
A scheduler σ selects a probability distribution for each state. This induces an MC with transition matrix $\mathbf{P}_\sigma : S \to \mathit{Distr}(S)$ such that $\mathbf{P}_\sigma(s) = \pi$ with $(s, \sigma(s), \pi) \in \mathbf{P}$.

To the choice of an action, we can assign a *reward* which we can interpret as a cost or bonus.

Definition 4: Given an MDP $(S, A, \mathbf{P})$, a *reward function* $\mathbf{R} : S \times A \rightharpoonup \mathbb{R}$ maps a combination of a state and an enabled action to a real value.
A scheduler for an MDP also assigns a reward structure $\mathbf{R}_\sigma : S \to \mathbb{R}$ to the states of the induced MC, such that $\mathbf{R}_\sigma(s) = \mathbf{R}(s, \sigma(s))$.

For an MDP $\mathbf{M}$, given a scheduler σ and a reward function $\mathbf{R}$, we can define the long-run average reward when beginning in state s as the Cesàro limit

$$g_\sigma(s) = \left(\lim_{N \to \infty} \frac{1}{N} \sum_{i=0}^{N-1} \mathbf{P}_\sigma^i \mathbf{R}_\sigma \right)(s).$$

We are interested in the minimum and maximum average rewards we can obtain in the long run. Other scheduler classes than the ones from Definition 3 exist, which may have more information, like the history of the stochastic process, or which may take randomized decisions. However, within the given conditions, there always exists a stationary deterministic scheduler which minimizes or maximizes the long-run average reward probability for all possible initial states at the same time [2].

B. Computing long-run average rewards

The policy iteration algorithm given in Algorithm 1 works by starting with an arbitrary scheduler and improving it until a scheduler which yields the extremal average reward is obtained. We have specified the algorithm for obtaining maximal values. If minimal values are to be computed, $\arg\max$ in lines 4 and 7 has to be changed to $\arg\min$.

In line 1, an arbitrary initial scheduler is chosen. Then, in lines 2–10, this policy is improved, until it is maximal or minimal, respectively.

In line 3 we solve an equation system from which we obtain the *gain* value g_n under the current scheduler, as well as a helper value b_n, the *bias*. The gain corresponds to the long-run average reward. To guarantee termination, the equation system has to be equipped with an additional constraint. We require that $\mathbf{P}^*_{\sigma_n} b_n = 0$ with $\mathbf{P}^*_{\sigma_n} = \lim_{N \to \infty} \frac{1}{N} \sum_{i=0}^{N-1} \mathbf{P}^i_{\sigma_n}$ [2]. This restriction also allows us to state the bias as $b_n(s) = E_s \{ \sum_{t=0}^{\infty} \mathbf{R}(X_t) - g_n(X_t) \}$. Here, $(X_t)_{t \in \mathbb{N}}$ is the stochastic process of the induced MC and E_s is the expectation under the condition that we start $(X_t)_{t \in \mathbb{N}}$ in s. Thus, we can interpret the bias as the expected total difference between the reward and the long-run average reward. As an illustrating example [2], consider the MC $\mathbf{D} = (\{s_0, s_1\}, \mathbf{P})$ with $\mathbf{P}(s_0, s_1) = \mathbf{P}(s_1, s_0) = 1$ and reward structure $\mathbf{R}(s_0) = 1$ and $\mathbf{R}(s_1) = -1$. Starting in state s_0, the sequence of rewards is $(1, -1, 1, \ldots)$, leading to a long-run average reward of 0, in the limit. The same value is obtained for s_1. When considering the bias, we see that $m \mapsto \sum_{t=0}^{m} \mathbf{R}(X_t) - g(X_t)$ alternates between 1 and 0, for the only path in the model when starting in s_0. Thus, the limit as well as the expectation in the bias interpretation for

s_0 is $1/2$ in this example. For s_1 we obtain the value $-1/2$ since the sequence above alternates between -1 and 0.

In line 5 we try to locally improve the gain value, thus obtaining an improved scheduler. If this is not possible for any state, we try to locally improve the bias value in line 7, also to obtain an improved scheduler. When doing so, we must take care only to choose from actions which also optimize the gain. If with neither of the above an improvement is possible, we have found an optimal scheduler and terminate the algorithm in line 10. Otherwise, we restart by solving the equation system of the new induced MC, try to improve the scheduler again, and so on.

Algorithm 1 Computing long-run average rewards [8], [9], [2]

LongRunAverageReward(MDP $\mathbf{M}$, Rewards $\mathbf{R}$)
begin

(1) $n := 0$, $\sigma_0 :=$ arbitrary scheduler

(2) **repeat**

(3) Obtain g_n and b_n which satisfy

$$(\mathbf{P}_{\sigma_n} - \mathbf{I})g_n = 0$$
$$\mathbf{R}_{\sigma_n} - g_n + (\mathbf{P}_{\sigma_n} - \mathbf{I})b_n = 0$$
$$\mathbf{P}^*_{\sigma_n} b_n = 0.$$

(4) $\bar{A}_s := \arg\max_{a \in A_s} \sum_{s' \in S} \mathbf{P}(s \xrightarrow{a} s') \cdot g_n(s')$,

(5) Choose σ_{n+1} such that $\sigma_{n+1}(s) \in \bar{A}_s$, setting $\sigma_{n+1}(s) := \sigma_n(s)$ if possible.

(6) **if** $\sigma_{n+1} = \sigma_n$ **then**

(7) Choose σ_{n+1} such that

$$\sigma_{n+1}(s) \in \arg\max_{a \in \bar{A}_s} \Big(\mathbf{R}(s,a) + \sum_{s' \in S} \mathbf{P}(s \xrightarrow{a} s') \cdot b_n(s') \Big),$$

setting $\sigma_{n+1}(s) = \sigma_n(s)$ if possible.

(8) **end if**

(9) $n := n + 1$

(10) **until** $\sigma_{n-1} = \sigma_n$

(11) **return** (σ_{n-1}, g_{n-1})

end

C. Lumping

The idea of bisimulation lumping [13], [14], [15] is to subsume certain states which behave equivalently for the class of properties under consideration. For our purposes, we can use the following notion of equivalence of two states.

Definition 5: Let $(S, \mathbf{P})$ be an MC and $\mathbf{R} : S \to \mathbb{R}$ a reward structure on its states. A partition P of S is a *bisimulation* if, for all $s, t \in S$ such that s and t are contained in the same block of P and all blocks C of P, the following holds:

$$\mathbf{R}(s) = \mathbf{R}(t) \quad \text{and} \quad \mathbf{P}(s, C) = \mathbf{P}(t, C),$$

where $\mathbf{P}(s, C) = \sum_{s' \in C} \mathbf{P}(s, s')$. Two states $s, t \in S$ are *bisimilar* ($s \sim t$) if there exists a bisimulation P such that s and t are contained in the same block of P.

Given a bisimulation, we can define the *quotient* of the model, which is a minimized model equivalent to the original one. Equivalence means that the quotient satisfies the same transient and steady-state properties (including long-run average rewards) as the original model [16].

Definition 6: Let $(S, \mathbf{P})$ be an MC and P be a bisimulation. The *bisimulation quotient* is the MC $(P, \mathbf{P}')$ such that $\mathbf{P}'(C, C') = \mathbf{P}(s, C')$ where $s \in C$ and $C, C' \in P$.

We transfer the reward function to the quotient of an MC by assigning each of its states the value of one of its constituent states. This notion can be lifted to MDPs. Usually, we are interested in the unique largest bisimulation, which leads to the smallest bisimulation quotient.

D. Symbolic Model Representation

Algorithms that rely on explicit state space representations (e. g., sparse matrix representations) are severely limited by the size of systems that they can handle. To circumvent this problem, we employ symbolic data structures, in particular (Multi-terminal) Binary Decision Diagrams ((MT)BDDs) [17], [18], in our algorithm to represent data.

BDDs are a data structure for the compact representation of binary functions of n binary variables, i. e., $\{0,1\}^n \to \{0,1\}$. A BDD is a directed acyclic graph, consisting of decision nodes, each labeled with one of the variables, and two constant nodes, labeled with 0 and 1, respectively. One node is marked as the root node. Decision nodes have two edges, *high* referring to setting the associated variable to 1 and *low* for 0. The function value is given by the label of the leaf reached by following the path that is induced by the variable assignment.

MTBDDs [18] are a generalization of BDDs used to represent functions of n binary variables, i. e., $\{0,1\}^n \to V$ where V is a finite set. We use calligraphic letters to denote (MT)BDDs.

By putting restrictions on the order in which the variables occur along paths and by removing redundant nodes, resulting in so-called reduced ordered (MT)BDDs, we obtain a canonical data structure for Boolean functions [17]. We assume that all (MT)BDDs are ordered and reduced. We denote the number of nodes (or the size) of an (MT)BDD $\mathcal{G}$ by $|\mathcal{G}|$.

A Markov decision process can be represented symbolically using BDDs and MTBDDs as the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T})$: the state space S is represented by a BDD $\mathcal{S}(s)$ such that $\mathcal{S}(s) = 1$ iff $s \in S$. For the set of actions which are enabled in state s, we use a BDD $\mathcal{A}(s, a)$ such that $\mathcal{A}(s, a) = 1$ iff $a \in A_s$. For the transitions we use an MTBDD $\mathcal{T}(s, a, t)$ such that $\mathcal{T}(s, a, t) = \mathbf{P}(s \xrightarrow{a} t)$ if s has an out-going a-transition, and 0 otherwise. The reward function can be represented similarly by an MTBDD $\mathcal{R}(s, a)$ such that $\mathcal{R}(s, a) = \mathbf{R}(s, a)$.

A scheduler σ can be represented by a BDD $\Sigma(s,a)$ such that $\Sigma(s,a) = 1$ iff $\sigma(s) = a$. Given a Markov decision process and a scheduler, we can compute the induced MC and reward by

$$\mathcal{P}(s,t) = \mathcal{Q}_a^+\big(\mathcal{T}(s,a,t) \cdot \Sigma(s,a)\big) \text{ and}$$
$$\mathcal{U}(s) = \mathcal{Q}_a^+\big(\mathcal{R}(s,a) \cdot \Sigma(s,a)\big),$$

where $\mathcal{Q}_x^{\odot}$ is the quantification operator w. r. t. an associative and commutative operator $\odot$:

$$\mathcal{Q}_x^{\odot}(\mathcal{A}(x,y)) = \bigodot_{a \in \{0,1\}^n} \mathcal{A}(x := a, y).$$

In the formulae above, $\mathcal{R}(s,a)\cdot\Sigma(s,a)$ yields an MTBDD in which there is for each state s exactly one action a leading to a non-zero value. Hence, summing over all possible actions a, we obtain an MTBDD which assigns each state this unique value. The same is done in the other formula.

Symbolic representation and computation of quotients is more involved. To represent a partition of n blocks $C_0, \ldots, C_{n-1}$, we introduce a vector v of (at least) $m = \lceil \log_2 n \rceil$ new variables $v = (v_0, \ldots, v_{m-1})$. The partition is represented using the BDD

$$\mathcal{P}(s,v) = \bigvee_{i=0}^{n-1} \mathrm{encBin}(i,v) \wedge \mathcal{C}_i(s).$$

Here, $\mathcal{C}_i(s)$ is the BDD representing C_i and $\mathrm{encBin}(i,v)$ returns the BDD for the binary encoding of the natural number i using the variables v.

We start the refinement with a partition of the state space where we relate all states to which the same reward is assigned, i. e.,

$$P_{\text{init}} = \begin{cases} \big\{\{s \in S \,|\, \mathbf{R}(s) = \mathbf{R}(t)\} \,\big|\, t \in S\big\} & \text{for MCs,} \\ \big\{\{s \in S \,|\, A_s = A_t \wedge \\ \qquad \forall a \in A_s : \mathbf{R}(s,a) = \mathbf{R}(t,a)\} \,\big|\, t \in S\big\} & \text{for MDPs.} \end{cases}$$

We refine our partition until a fixed point is reached. Afterward, we can easily transform the quotient (which is often much smaller than the original model in particular in the case of models that are constructed through parallel composition) to an explicit form, e. g., using a sparse matrix representation. A detailed description of our partition refinement algorithm can be found in [19], [20].

III. Symblicit Computation of Long-run Average Rewards

A. Basic Algorithm

The underlying principle of our symblicit algorithm is the same as in Algorithm 1. The most important change is the combination of symbolic and explicit data structures in order to yield efficiency w. r. t. memory consumption and runtime. We use a BDD-based representation of the MDP, the schedulers, and the induced MCs. Solving linear equation systems is much more efficient using an explicit representation of the matrix, provided that they are small enough to fit into memory [7], [21]. Therefore we apply a symbolic lumping algorithm to the induced MC first and convert the resulting – often much smaller – quotient system to an explicit sparse matrix representation. Using this representation we compute the gain and bias values for the quotient system. These values are converted to a symbolic BDD-based representation and transferred back to the original (non-minimized) system. The update of the scheduler is finally performed in a symbolic manner.

Algorithm 2 depicts the pseudo-code of this procedure. In line 1 we choose an initial scheduler. This can easily be implemented by setting $\sigma = \textsc{MakeUnique}(\mathcal{A}, \mathrm{cube}_a, \emptyset)$ (see Algorithm 4). We improve the scheduler along lines 2–10. In line 3, we compute the MC and reward structure induced by the current scheduler, as described in Section II-D.

In lines 4 and 5 we generate the smallest bisimulation quotient of the symbolic model and convert it to an explicit-state form. Using the explicit state quotient, in line 6 we solve the equation system of Algorithm 1 line 3 to obtain gain and bias values. Then, in line 7, we map these values back to the symbolic model such that each state gets assigned the values of the partition block it is contained in. If $\mathcal{P}$ is the BDD of the bisimulation and g the vector of gain values, this operation corresponds to the following formula:

$$\mathfrak{g}(s) := \mathcal{Q}_v^+\left(\mathcal{P}(s,v) \cdot \bigvee_{i=0}^{n-1}\Big(\mathrm{encBin}(i,v) \cdot \mathrm{leaf}\big(g(C_i)\big)\Big)\right).$$

The function $\mathrm{leaf}(v)$ creates a new MTBDD leaf with value $v \in \mathbb{R}$.

The bias values $\mathfrak{b}(s)$ for the original system can be computed in the same way, replacing g by b in this formula.

We use the values $\mathfrak{g}$ and $\mathfrak{b}$ to improve the scheduler in line 8. In Section III-B we give a detailed description of the symbolic implementation of this update.

The validity of this method is justified by the following lemma. The proof can be found in the extended version [22] of this paper.

Lemma 1: Let $\mathbf{D} = (S, \mathbf{P})$ be an MC and $\mathbf{D}'$ its quotient induced by a bisimulation P. Let $g' : P \to \mathbb{R}$ and $b' : P \to \mathbb{R}$ be such that the equations of Algorithm 1 line 3 are fulfilled in $\mathbf{D}'$. Then, the gain and reward functions $g : S \to \mathbb{R}$ and $b : S \to \mathbb{R}$ with $g(s) = g'(C)$ and $b(s) = b'(C)$ where $s \in C$ and $C \in P$ also fulfill the given equations.

B. Improving Schedulers

For the update step (line 8 of Algorithm 2), we have to solve the following problem: given a BDD $\mathcal{A}(s,a)$ specifying which actions a are selectable in state s and an MTBDD $\mathcal{V}(s,a)$ which assigns each action (of a superset of $\mathcal{A}(s,a)$) a value, compute a BDD $\mathcal{B}(s,a)$ which contains for each state s exactly one action that is selectable and that maximizes (or minimizes, resp.) the value of $\mathcal{V}(s,a)$ among the selectable actions of state s. Furthermore, the algorithm for long-run

Algorithm 2

LRAR_SemiSymb(MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T})$, Reward $\mathcal{R}$)
begin
$n := 0, \Sigma_n :=$ ChooseInitialScheduler (1)
repeat (2)
$(\mathcal{P}, \mathcal{U}) := \text{InducedMCAndRew}(\mathcal{M}, \mathcal{R}, \Sigma_n)$ (3)
$(\mathcal{P}', \mathcal{U}') := \text{Lump}(\mathcal{P}, \mathcal{U})$ (4)
$(\mathbf{D}', \mathbf{R}') := \text{Explicit}(\mathcal{P}', \mathcal{U}')$ (5)
$(g, b) := \text{GainAndBias}(\mathbf{D}', \mathbf{R}')$ (6)
$(\mathfrak{g}, \mathfrak{b}) := \text{MapValuesToSym}(g, b)$ (7)
$\Sigma_{n+1} := \text{ImproveScheduler}(\Sigma_n, \mathfrak{g}, \mathfrak{b})$ (8)
$n := n + 1$ (9)
until $\Sigma_{n-1} = \Sigma_n$ (10)
return $(\Sigma_n, \mathfrak{g})$ (11)
end

average rewards requires us to make the same choice as in the previous iteration whenever possible.

A solution to this problem is given in Algorithm 3. First, we determine the values of the selectable actions in line 1. All non-selectable actions are thereby mapped to $-\infty$ (when maximizing) or $+\infty$ (when minimizing). Then we traverse $\mathcal{V}'(s, a)$ and determine the optimal value v_{opt}. In another pass through this MTBDD we replace the leaf with value v_{opt} by 1 and all other leaves by 0. This yields a BDD $\mathcal{O}(s, a)$ which assigns each state all selectable actions that lead to the optimal value.

Algorithm 3

Select(BDD $\mathcal{A}$, MTBDD $\mathcal{V}$, BDD Σ_n)
begin
$\mathcal{V}' := \big(\mathcal{V} \cdot \mathcal{A}\big) + \big(\neg\mathcal{A} \cdot \text{leaf}(\pm\infty)\big)$ (1)
$v_{\text{opt}} := \text{Maximum}(\mathcal{V}')$ (or $\text{Minimum}(\mathcal{V}')$, resp.) (2)
$\mathcal{O} := \text{EqualValue}(\mathcal{V}', v_{\text{opt}})$ (3)
$\mathcal{B} := \mathcal{O} \wedge \Sigma_n$ (4)
$\mathcal{U} := \mathcal{O} \wedge \neg\mathcal{Q}_a^{\vee}\big(\mathcal{B}(s, a)\big)$ (5)
$\mathcal{N} := \text{MakeUnique}(\mathcal{U}, \text{cube}_a, \emptyset)$ (6)
$\mathcal{B} := \mathcal{B} \vee \mathcal{N}$ (7)
return $(\mathcal{O}, \mathcal{B})$ (8)
end

If $\mathcal{O}$ permits the same choice as in Σ_n, we have to make this choice again. Therefore we first intersect $\mathcal{O}(s, a)$ with $\Sigma_n(s, a)$. For the other states, stored in $\mathcal{U}$, for which the old choice is no longer optimal, we have to select a new action.

If we presuppose a variable order of the BDDs such that the a_i-variables precede the s_j-variables, we can use the function MAKEUNIQUE (see Algorithm 4) to make this choice.[1]

[1]For the reverse variable order, in which the state variable precede the action variable, a similar algorithm can be used.

Algorithm 4

MakeUnique(MTBDD $\mathcal{F}$, BDD $\mathcal{C}^a$, BDD& seen)
begin
if $\mathcal{F} = 0$ **then** (1)
return 0 (2)
else if $a \neq \text{const}$ **then** (3)
$a_i := \text{topVar}(\mathcal{C}^a)$ (4)
$\text{high} := \text{MakeUnique}(\mathcal{F}_{|a_i=1}, \mathcal{C}^a{}_{|a_i=1}, \text{seen})$ (5)
$\text{low} := \text{MakeUnique}(\mathcal{F}_{|a_i=0}, \mathcal{C}^a{}_{|a_i=1}, \text{seen})$ (6)
$\text{result} := \text{FindOrCreate}(a_i, \text{high}, \text{low})$ (7)
return result (8)
else (9)
$\text{result} := \mathcal{F} \wedge \neg\text{seen}$ (10)
$\text{seen} := \text{seen} \vee \text{result}$ (11)
return result (12)
end if (13)
end

MAKEUNIQUE takes three parameters: $\mathcal{F}$ is the BDD of state/action pairs we have to make the selection from, $\mathcal{C}^a$ is the cube (i. e., conjunction) of all action variables, and seen contains all states for which we have already determined an action. Please note that seen may be changed by recursive function calls. Initially, seen is the empty set.

We traverse the top part of the BDD, which is labeled with action variables, until we reach a node labeled with a state variable (in this case $\mathcal{C}^a$ becomes constant). This node represents all states for which the action that is encoded by the path from the root node to the current node is optimal. We remove from this sub-BDD all states for which we have already determined an action and add the remaining states to the seen states.

The function SELECT (Algorithm 3) can now be used to perform the scheduler update operation. The details are given in Algorithm 5. Lines 1 and 2 directly correspond to lines 4 and 5 of Algorithm 1. The same holds for lines 3–5 and lines 6–7 of Algorithm 1.

Algorithm 5

ImproveScheduler(MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T})$, BDD Σ_n)
begin
$\mathcal{V}(s, a) := \mathcal{Q}_t^{+}\big(\mathcal{T}(s, a, t) \cdot \mathfrak{g}(t)\big)$ (1)
$(\bar{\mathcal{A}}, \Sigma_{n+1}) := \text{Select}(\mathcal{A}, \mathcal{V}, \Sigma_n)$ (2)
if $\Sigma_{n+1} = \Sigma_n$ **then** (3)
$\mathcal{V}(s, a) := \mathcal{R} + \mathcal{Q}_t^{+}\big(\mathcal{T}(s, a, t) \cdot \mathfrak{b}(t)\big)$ (4)
$(\cdot, \Sigma_{n+1}) := \text{Select}(\bar{\mathcal{A}}, \mathcal{V}, \Sigma_n)$ (5)
end if (6)
return Σ_{n+1} (7)
end

C. Bisimulation Reuse

In order to speed up the algorithm, we try to improve the repeated lumping of induced MCs by reusing already computed partitions.

With the exception of the first few iterations, we expect that the scheduler update does not change the induced Markov chain substantially. Since the bisimulation minimization is the most expensive step (see Section IV), we want to reduce the minimization time by reusing bisimulation partitions from previous iterations. This does not affect the correctness, since the result is still a bisimulation relation, although in general not the coarsest possible one.

The first possibility is to compute the bisimulations during the first n iterations as before, but in the subsequent iterations, to use the partition of the n-th iteration as initial partition. Alternatively, we can use, from the $(n+1)$-th iteration on, the bisimulation from the previous partition as starting point for refinement. Since the reward structure of the induced MC also depends on the scheduler, we need an extra refinement step to take the reward structure into account if it has changed.

The hope is that only few iterations are then necessary to reach the new fixed point, thereby saving computation time. The drawback is that the bisimulation gets finer than necessary. This results in time and space overhead and larger equation systems for computing gain and bias values.

D. Unbounded Reachability

The algorithm for long-run average rewards can also be used to compute probabilities for another important class of properties, namely unbounded reachability properties of the form $\varphi_1 \,\mathrm{U}\, \varphi_2$. A path $\pi = s_0 s_1 s_2 \ldots$ in an MDP satisfies such a property if there is $i \geq 0$ such that s_i satisfies φ_2 and all states s_k with $k < i$ satisfy φ_1.

Computing the probability to walk along such a path can be reduced to computing the long-run average reward of a system as follows: states which satisfy $\neg\varphi_1 \vee \varphi_2$ are made absorbing by removing all out-going edges and adding a self-loop with probability 1 [23]. We define the following reward function

$$\mathbf{R}(s,a) = \begin{cases} 1, & \text{if } s \text{ satisfies } \varphi_2 \text{ and } a \in A_s, \\ 0, & \text{otherwise.} \end{cases}$$

We refer the reader to the extended version [22] of this paper for a proof that the long-run average reward of a state s and the probability to walk along a path that satisfies $\varphi_1 \,\mathrm{U}\, \varphi_2$ when starting in s agree.

IV. Case Studies

We have implemented three different versions of the algorithm for long-run average rewards in a prototypical implementation: the first version follows directly Algorithm 1 and uses explicit sparse matrix representations. The second version computes the bisimulation quotient of the MDP using the symbolic signature-based algorithm [24], then converts the quotient to the explicit sparse matrix representation, and finally applies Algorithm 1. The third version is the symblicit algorithm described in Section III (Algorithm 2).

For the (MT)BDDs we use the Colorado University decision diagram package (Cudd) [25]. We solve all linear equation systems using the SOR-method [26], with the relaxation parameter ω set to 0.9 (except for a few cases where this value prevented termination; then we used $\omega = 0.4$).

All experiments (with one exception) were run on a 3.0 GHz Intel Core2 Duo processor with 3 GB of main memory running Linux (Kubuntu 9.10). We stopped any experiment which took more than 10 hours. For each case study, we give a short description as well as a table stating the resulting values and performance statistics for several variants of the model. We give the number of states ("States") and transitions ("Trans.") of the original MDP, as well as time ("Time") (in seconds) and memory ("Mem.") (in megabytes) for each of the three engines. For the symblicit algorithm, we consider the basic version ("S-Basic") as well as the heuristic where we reuse existing partitioning information, either using the bisimulation computed in the first iteration as initial partition for the subsequent ones ("S-First") or always the bisimulation from the previous iteration ("S-Last"). Information is given for both minimal and maximal values ("Mode").

As case studies we used a number of models most of which are publicly available on the web page of the PRISM model checker http://www.prismmodelchecker.org. For a more exhaustive description of the case studies, we refer to this website and the literature cited. We give results and performance figures in Table I. At first, we consider the case studies for which we compute long-run average rewards.

Dining Philosophers (Lehmann, Rabin) [10]. We consider a probabilistic extension of the classical dining philosophers model by Lehmann and Rabin. The randomness is introduced by having the philosophers pick up their left or right spoon with a probability of one half each. A scheduler nondeterministically allows one of the N different dining philosophers to make his next move. We can prevent participants from starving by restricting to fair schedulers. However, we consider all schedulers and thus the minimum average long-run number of philosophers eating is always zero. This means that a nasty scheduler may decide to let every philosopher die of hunger. The maximal number of philosophers eating is about half the number of philosophers. This is the case because a scheduler can decide to place as many philosophers as possible in their eating phase. These stay there, while the other philosophers starve.

The model size grows with increasing number of participants. For those instances where the explicit state implementation did not run out of memory, it is much faster than the symbolic variants. The relatively large memory usage of the symbolic techniques for the smallest model variant is due to the overhead of BDDs when applying them on small state

Table I: Case Studies Performance Results (Long-run Average Rewards)

Dining Philosophers (Lehman)

N	States	Trans.	Mode	Explicit		MDP Min		S-Basic		S-First		S-Last		Value
				Time	Mem.	Time	Mem.	Time	Mem.	Time	Mem.	Time	Mem.	
5	93068	599600	max	3.80	20.67	101.83	201.39	3.33	40.79	6.62	51.33	3.36	42.84	2
			min	1.29	20.43	100.65	201.17	0.03	8.24	0.03	8.24	0.02	8.24	0
6	917424	7092696	max	37.76	229.17	1617.24	2127.62	82.68	57.33	186.78	72.64	175.24	104.89	3
			min	16.70	227.11	1606.10	2125.44	0.06	9.88	0.04	9.88	0.05	9.88	0
7	9043420	81568144	max	540.07	2513.61	– *Memory out* –		154.01	83.02	361.37	117.69	253.29	170.05	3
			min	199.52	2493.07	– *Memory out* –		0.09	11.17	0.07	11.17	0.07	11.17	0
8	89144512	918913056	max	– *Memory out* –		– *Memory out* –		5330.52	614.64	10998.45	1210.20	– *Memory out* –		4
			min	– *Memory out* –		– *Memory out* –		0.12	13.52	0.09	13.52	0.08	13.52	0
9	878732012	10190342448	max	– *Memory out* –		– *Memory out* –		5131.26	1099.93	10879.45	2297.83	– *Memory out* –		4
			min	– *Memory out* –		– *Memory out* –		0.16	17.98	0.12	15.15	0.11	15.15	0
10	8662001936	111611282280	max	– *Memory out* –		– *Memory out* –		– *Memory out* –		– *Memory out* –		– *Memory out* –		5
			min	– *Memory out* –		– *Memory out* –		0.22	19.27	0.16	18.36	0.16	18.36	0
15	806171451829916	15581472413070480	max	– *Memory out* –		– *Memory out* –		– *Memory out* –		– *Memory out* –		– *Memory out* –		7
			min	– *Memory out* –		– *Memory out* –		0.73	34.80	0.48	34.80	0.49	34.80	0

Dining Philosophers (Duflot)

N	States	Trans.	Mode	Explicit		MDP Min		S-Basic		S-First		S-Last		Value
				Time	Mem.	Time	Mem.	Time	Mem.	Time	Mem.	Time	Mem.	
3	956	3048	max	0.11	0.15	0.19	9.44	0.54	9.25	1.10	10.06	0.53	9.41	0.842105
			min	0.37	0.15	0.55	9.44	3.51	18.88	2.90	10.36	1.44	10.57	0.0169492
4	9440	40120	max	1.56	1.64	4.53	53.80	16.45	38.73	31.84	53.77	18.69	49.61	1.21429
			min	2.25	1.65	6.07	53.80	77.70	59.08	68.46	60.19	32.81	56.97	0.010989
5	93068	494420	max	16.95	18.21	82.25	180.83	283.11	76.54	421.71	76.41	283.49	73.89	1.84211
			min	32.23	18.30	89.45	180.92	2073.27	92.18	1895.02	103.11	821.04	107.06	0.00763359
6	917424	5848524	max	361.38	200.66	1262.60	1929.03	15393.95	236.33	9098.57	373.38	6267.19	590.17	2.19512
			min	549.61	201.62	1388.16	1930.11	– *Time out* –		– *Time out* –		– *Time out* –		0.00507482

Minimal Spanning Tree

N	L	R	States	Trans.	Mode	Explicit		MDP Min		S-Basic		S-First		S-Last		Value
						Time	Mem.	Time	Mem.	Time	Mem.	Time	Mem.	Time	Mem.	
4	3	5	6087	31238	max	0.87	1.23	0.90	17.54	3.60	22.08	2.52	23.96	2.17	22.32	0.82022
					min	2.84	1.40	1.69	17.54	3.76	21.50	3.12	20.68	2.34	20.70	0.666325
5	3	6	389163	3067562	max	181.17	95.23	62.77	66.52	128.15	65.23	109.95	64.67	90.10	64.36	0.807324
					min	403.69	106.00	92.09	66.52	153.63	63.02	146.15	67.82	123.92	63.48	0.569486
5	4	6	283263	1879262	max	79.27	67.02	18.59	52.43	62.99	66.51	59.19	66.49	47.01	66.49	0.746579
					min	343.81	69.17	32.26	52.43	85.58	66.48	76.65	66.49	62.11	66.49	0.595175
5	4	7	535263	3967262	max	201.03	121.28	22.14	66.89	123.99	66.31	127.08	65.53	89.40	65.55	0.759498
					min	328.02	132.49	25.76	66.89	120.70	66.97	112.47	65.53	88.21	68.02	0.537119
6	5	7	20440893	167251772	max	– *Memory out* –		692.84	177.77	1633.75	135.33	1499.91	175.40	1252.96	171.25	0.733402
					min	– *Memory out* –		686.29	178.34	1665.59	142.39	1427.16	163.54	1275.14	170.37	0.589117
7	6	8	2110978929	20601368048	max	– *Memory out* –		– *Memory out* –		1256380.25	10127.43	–		–		0.59571

spaces, as they were not developed for such models. The implementation minimizing the MDP is rather ineffective in this case study, in both memory usage and speed, since the application of bisimulation minimization to the MDP state space does not reduce its size. In contrast, the symblicit algorithm, which minimizes the induced MCs, is able to handle much larger state spaces than the other versions. Among the symblicit variants, the basic one performs best since when reusing an already computed bisimulation, the quotient system becomes much finer than necessary, causing large memory and runtime overhead.

We were able to complete model variants with several hundred millions of states. The analysis of minimal values always succeeded and never took many resources, whereas the analysis of maximal values was more involved. The maximal value for $N = 15$ was obtained by manual analysis.

Dining Philosophers (Duflot et al.) [11]. This case study by Duflot, Fribourg and Picaronny is a variant of the dining philosophers model by Lehman and Rabin. This protocol works correctly even under unfair schedulers. The difference is that philosophers cannot be scheduled if their next transition is a loop. Because of this restriction, the extremal values are less diverse than for the previous model.

Performance results are worse in that more time and memory was needed for models of the same size. We believe that the reason for this is, that the restriction of the schedulers to schedule only some philosophers breaks symmetries in the model, leading to larger BDDs and quotient MCs.

In this model, the explicit-state engine performed better than the symbolic and symblicit ones. For the symblicit implementations, partition reuse resulted in a speed up of the analysis, but lead to a significantly increased memory usage. For the next larger instance (i. e., for $n = 7$), the explicit tool, the variant which minimizes the MDP, and S-Last run out of memory, while the S-Basic and S-First fail due to the time limit.

Minimal Spanning Tree [12]. We model a self-stabilizing distributed algorithm which computes a minimal spanning tree in a network with N nodes. We compute the maximum and minimum availability, i. e., the fraction of time in which

the system is in a safe state. Assuming absence of failures, this number would be one [12]. However, due to, e. g., communication failures the system may leave the state where a result has been computed, and recover after some time. We have to take into account that nodes in principal work in parallel, at equal speed, but do not work in a lock-step manner. To obtain realistic results, we added restrictions on the number of steps the scheduler can wait to schedule a certain node. These scheduler classes refer to restrictions on the relative clock drifts of the network nodes. We use schedulers for which we have a lower bound L on the number of steps a node may be delayed on scheduling, as well as an upper bound R, in contrast to the dining philosophers model of Lynch et al. [27], where only an upper bound on the step number to wait is possible.

With decreasing L and increasing R, because of the increased scheduling freedom, the minimum average reward decreases while the maximum probability increases. An increasing number of nodes leads to an increase in the state spaces. The same also holds for decreasing L and increasing R, because of the growing amount of information the scheduler has to maintain. The explicit state implementation quickly runs out of memory for larger models. The implementation which minimizes the MDP is faster than the symblicit implementation, but needs more memory and thus cannot be applied on the model variant with $N = 7$. Concerning different heuristics of the symblicit algorithm we observe a trade-off between time and space.

Notably, the analysis of the variant with $N = 7$ had to be performed on a different architecture, namely a computation server with four AMD Quad-Core Opteron processors with 2.3 GHz CPU frequency and 64 GB of main memory, running in 64 bit mode. Because of the very large number of states, the explicit state implementation as well as the MDP minimizing implementation ran out of memory. The symblicit implementation took considerable time, but was finally able to compute maximum long-run average rewards.

To get a better feel of the various steps of the symblicit approach, we studied the accumulated times for key operations over all benchmarks for which the symblicit algorithm (without bisimulation reuse, i. e., "S-Basic") was able to compute the maximum or minimum long-run average rewards. The following table contains the fractions of the total runtime spent for the different operations. The bisimulation computation clearly dominates the runtime.

Operation	**Fraction of time**
Bisimulation computation	90.81 %
Quotient extraction	3.08 %
Solution of linear equation systems	0.67 %
Scheduler updates	4.01 %

As described in Section III-D, we can also use the symblicit approach to compute time-unbounded reachability properties. To study the effectiveness of this approach, we considered a number of case studies, comparing with a standard implementation of unbounded reachability, implemented in PRISM. The results are given in Table II.

Asynchronous Leader Election Protocol [28]. In this case study, we consider an asynchronous ring of N processors which use a protocol such that they will be able to elect a leader (a uniquely designated processor) by sending messages around the ring. The exact content of messages is chosen probabilistically. With a certain probability, processes become inactive, until finally the leader is fixed. The protocol is asynchronous in the sense that processes do not work in lock step but for each time step the scheduler schedules one of them. We consider the minimal and maximal probability that one specific process finally becomes the leader. We see that the probability is $1/n$ in both cases. Thus, it is guaranteed that under all possible schedulers, each participant has the same chance of becoming the leader finally.

In this setting, PRISM is considerably more effective in solving the unbounded reachability problem, due to the large memory usage of our implementation.

Mutual Exclusion Protocol (Rabin) [29]. We consider a case study by Rabin et al. in which a number of N processes compete with each other to enter a critical section. We consider the minimal and maximal probability that a fixed process, for instance the first one, enters the critical section next. As on the PRISM homepage, we restrict to those states where the values drawn by the other processes are below K.

Notably, our symblicit implementation is more or less comparable to PRISM in this case, though not as efficient in time and memory usage. The explicit and symbolic implementations run out of memory rather early.

Dining cryptographers [30]. This case study considers a group of N cryptographers which have dinner at a restaurant. An agreement has been made that the dinner will be paid anonymously. The cryptographers want to find out whether one of them has to pay for the dinner, or whether their master will pay it. In case one of the cryptographers has to pay for the dinner, the group wants him to stay anonymous. The problem is solved as follows: each member of the group flips an unbiased coin and only tells the outcome to his right neighbor. In case a cryptographer is not paying the bill, it will tell "agree" if his coin and the one of his left neighbor agree, and "disagree" else. However, if a cryptographer is paying the bill, he will inverse the answers. Then, if N is odd and an odd number of "agree" occur, the master pays the bill. The same is the case if N is even as well as the number of "agree". In the other cases, one of the cryptographers pays the bill. The nondeterminism results because there are $N+1$ possible persons to pay the bill, and from the unspecified order in which the cryptographers toss their coins. The property we consider here is the anonymity of the group members. Anonymity is guaranteed if in the case that one of the cryptographers pays, every possible combination of "agree" and "disagree" along the group members has the same probability, independent of the member who paid. To check this, we compute the minimal and maximal probability of each of the 2^{N-1} possible outcome under the condition

that a cryptographer pays. As we see from the result table, anonymity is guaranteed.

Concerning performance, the general picture is like the one of the mutual exclusion protocol case study.

General Observations. For the prime concern of this paper, long-run average properties, our implementation performed well for a number of case studies. Especially for some large cases, our symblicit technique outperforms both the explicit state implementation as well as an implementation which is based on minimizing the MDPs directly. For the ones where it failed, we observed in many cases that the first policy improvement steps where effective, but later steps lead to excessive memory usage, prematurely terminating the run. To avoid such problems, we are planning to tune the policy selection algorithm such that from the legal policies the policy which leads to the smallest memory usage will be selected. In addition, the approach could profit very much from heuristics leading to an improved choice of the initial scheduler.

Unbounded reachability analysis generally performed worse when using the symblicit implementation compared to when using PRISM. One reason is that our prototypical implementation uses more memory in some places than would be necessary. Furthermore, we are currently using strong bisimulation when calculating unbounded reachability, while weak bisimulation would also be sufficent to preserve the unbounded reachability probabilities, but allows to subsume more states. This would lead to smaller quotients, and thus a decreased memory usage.

V. CONCLUSION

This paper has introduced a new algorithmic approach to compute long-run average properties for Markov decision processes. The symblicit algorithm combines symbolic and explicit computation steps in a way that exploits the respective benefits. We have shown the correctness of the approach and have reported on empirical evidence that the symblicit technique outperforms for large models both the explicit-state implementation as well as an implementation which is based on minimizing the MDPs directly. The algorithm has applications in many diverse areas, among them long-run stability assessment of self-stabilizing systems. We are focusing especially on this application context. Further algorithmic work will especially focus on improved memory consumption in the directions identified in the experimental studies.

REFERENCES

[1] A. Hinton, M. Z. Kwiatkowska, G. Norman, and D. Parker, "PRISM: A tool for automatic verification of probabilistic systems," in *12th Int'l Conf. on Tool and Algorithm for the Construction and Analysis of Systems (TACAS)*, ser. LNCS, vol. 3920. Springer, 2006, pp. 441–444.

[2] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 1994.

[3] A. Dhama, O. Theel, P. Crouzen, H. Hermanns, R. Wimmer, and B. Becker, "Dependability engineering of silent self-stabilizing systems," in *11th Int'l Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS)*, ser. LNCS, vol. 5873. Springer, 2009, pp. 238–253.

[4] A. Bianco and L. de Alfaro, "Model checking of probabilistic and nondeterministic systems," in *15th Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, ser. LNCS, vol. 1026. Springer, 1995, pp. 499–513.

[5] L. de Alfaro, "How to specify and verify the long-run average behavior of probabilistic systems," in *13th Annual IEEE Symp. on Logic in Computer Science (LICS)*. Indianapolis, Indiana, USA: IEEE CS, 1998, pp. 454–465.

[6] E. Böde, M. Herbstritt, H. Hermanns, S. Johr, T. Peikenkamp, R. Pulungan, J. Rakow, R. Wimmer, and B. Becker, "Compositional dependability evaluation for statemate," *IEEE Transactions on Software Engineering*, vol. 35, no. 2, pp. 274–292, 2009.

[7] M. Z. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic symbolic model checking with PRISM: A hybrid approach," *Software Tools for Technology Transfer*, vol. 6, no. 2, pp. 128–142, 2004.

[8] R. A. Howard, *Dynamic Programming and Markov Processes*. John Wiley and Sons, Inc., 1960.

[9] A. F. Veinott, "On finding optimal policies in discrete dynamic programming with no discounting," *Annals of Mathematical Statistics*, vol. 37, no. 5, pp. 1284–1294, 1966.

[10] D. Lehmann and M. O. Rabin, "On the advantage of free choice: A symmetric and fully distributed solution to the dining philosophers problem (extended abstract)," in *8th Annual ACM Symp. on Principles of Programming Languages (POPL)*. ACM Press, 1981, pp. 133–138.

[11] M. Duflot, L. Fribourg, and C. Picaronny, "Randomized dining philosophers without fairness assumption," *Distributed Computing*, vol. 17, no. 1, pp. 65–76, 2004.

[12] Z. Collin and S. Dolev, "Self-stabilizing depth-first search," *Information Processing Letters*, vol. 49, no. 6, pp. 297–301, 1994.

[13] J. G. Kemeney and J. L. Snell, *Finite Markov Chains*. D. Van Nostrand Company, Inc., 1960.

[14] K. G. Larsen and A. Skou, "Bisimulation through probabilistic testing," *Information and Computation*, vol. 1, no. 94, pp. 1–28, 1991.

[15] P. Buchholz, "Exact and ordinary lumpability in finite Markov chains," *Journal of Applied Probability*, vol. 31, pp. 59–74, 1994.

[16] C. Baier, J.-P. Katoen, H. Hermanns, and V. Wolf, "Comparative branching-time semantics for Markov chains," *Information and Computation*, vol. 200, no. 2, pp. 149–214, 2005.

[17] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, 1986.

Table II: Case Studies Performance Results (Unbounded Reachability)

Asynchronous Leader Election Protocol

N	States	Trans.	Mode	MDP Min		S-Basic		PRISM (Sparse)		PRISM (Hybrid)		PRISM (MTBDD)		Value
				Time	Mem.	Time	Mem.	Time	Mem.	Time	Mem.	Time	Mem.	
3	364	654	min	0.09	9.47	0.10	9.47	0.04	14.86	0.04	14.95	0.54	16.28	0.333333
4	3172	7144	min	1.12	21.76	1.03	17.12	0.20	18.57	0.26	18.62	5.49	23.41	0.25
5	27299	74365	min	16.52	68.34	10.85	52.69	1.23	26.03	1.90	27.45	48.55	53.60	0.2
6	237656	760878	min	218.45	337.78	107.39	115.99	7.70	64.18	15.91	66.96	373.65	148.82	0.166667
7	2095783	7714385	min	2969.44	2446.65	913.47	620.69	48.40	268.93	156.85	228.67	4557.53	221.67	0.142857
8	18674484	77708080	min	– *Memory out* –		– *Memory out* –		404.02	1278.23	1616.20	645.28	– *Memory out* –		0.125

Mutual Exclusion Protocol (Rabin)

N	States	Trans.	Mode	MDP Min		S-Basic		PRISM (Sparse)		PRISM (Hybrid)		PRISM (MTBDD)		Value
				Time	Mem.	Time	Mem.	Time	Mem.	Time	Mem.	Time	Mem.	
4	668836	3637488	min	13.06	69.25	2.69	53.79	1.18	56.57	0.95	51.59	0.45	29.99	0.180015
5	27381358	177834300	min	406.92	646.57	9.47	103.13	36.41	1112.28	24.89	734.45	1.80	54.34	0.144385
6	624265622	4207964676	min	– *Memory out* –		22.51	158.85	– *Memory out* –		– *Memory out* –		3.53	54.39	0.119937
7	13500805210	93439432632	min	– *Memory out* –		52.21	204.81	– *Memory out* –		– *Memory out* –		6.50	91.33	0.102551
8	280776218242	1982001054416	min	– *Memory out* –		112.24	289.06	– *Memory out* –		– *Memory out* –		14.90	164.92	0.089578
9	16136875996952	129440687593176	min	– *Memory out* –		452.26	568.16	– *Memory out* –		– *Memory out* –		56.48	177.52	0.079959

Dining Cryptographers

N	States	Trans.	Mode	MDP Min		S-Basic		PRISM (Sparse)		PRISM (Hybrid)		PRISM (MTBDD)		Value
				Time	Mem.	Time	Mem.	Time	Mem.	Time	Mem.	Time	Mem.	
3	380	776	min	0.01	6.33	0.01	6.33	0.01	14.05	0.01	13.69	0.01	13.58	0.25
5	11850	38772	min	0.18	10.40	0.05	9.44	0.07	16.20	0.06	16.31	0.06	15.39	0.0625
7	328760	1499472	min	3.60	51.96	0.28	15.27	0.39	27.21	0.48	29.86	0.28	18.44	0.015625
9	8550950	50114780	min	291.08	85.70	1.56	25.70	4.47	241.65	8.52	248.99	1.02	26.61	0.00390625
10	42906171	279384204	min	1397.82	215.70	3.63	56.64	18.99	1086.35	43.58	1107.37	1.87	34.06	0.001953125
15	123256716400	1203834043808	min	– *Memory out* –		280.40	320.01	– *Memory out* –		– *Memory out* –		21.21	155.72	0.0000610352

[18] M. Fujita, P. C. McGeer, and J. C.-Y. Yang, "Multiterminal binary decision diagrams: An efficient data structure for matrix representation," *Formal Methods in System Design*, vol. 10, no. 2/3, pp. 149–169, 1997.

[19] S. Derisavi, "Signature-based symbolic algorithm for optimal Markov chain lumping," in *4th Int'l Conf. on Quantitative Evaluation of Systems (QEST)*. IEEE CS, 2007, pp. 141–150.

[20] R. Wimmer and B. Becker, "Correctness issues of symbolic bisimulation computation for Markov chains," in *15th Int'l Conf. on Measurement, Modelling and Evaluation of Computing Systems (MMB)*, ser. LNCS, vol. 5987. Springer, 2010, pp. 287–301.

[21] H. Hermanns, M. Z. 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*, vol. 56, no. 1–2, pp. 23–67, 2003.

[22] R. Wimmer, B. Braitling, B. Becker, E. M. Hahn, P. Crouzen, H. Hermanns, A. Dhama, and O. Theel, "Symblicit calculation of long-run averages for concurrent probabilistic systems (extended version)," http://ira.informatik.uni-freiburg.de/research/qest2010-extended.pdf, 2010.

[23] C. Baier and J.-P. Katoen, *Principles of Model Checking*. The MIT Press, 2008.

[24] R. Wimmer, M. Herbstritt, H. Hermanns, K. Strampp, and B. Becker, "Sigref – A symbolic bisimulation tool box," in *4th Int'l Symp. on Automated Technology for Verification and Analysis (ATVA)*, ser. LNCS, vol. 4218. Springer, 2006, pp. 477–492.

[25] F. Somenzi, "CUDD: Colorado university decision diagram package, release 2.4.2," 2009.

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

[27] N. Lynch, I. Saias, and R. Segala, "Proving time bounds for randomized distributed algorithms," in *13th Annual ACM Symp. on Principles of Distributed Computing (PODC)*. ACM Press, 1994, pp. 314–323.

[28] A. Itai and M. Rodeh, "Symmetry breaking in distributed networks," *Information and Computation*, vol. 88, no. 1, 1990.

[29] M. O. Rabin, "N-process mutual exclusion with bounded waiting by $4 \log_2 N$-valued shared variable," *Journal of Computer and System Sciences*, vol. 25, no. 1, pp. 66–75, 1982.

[30] D. Chaum, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *Journal of Cryptology*, vol. 1, pp. 65–75, 1988.