Counterexample Generation for Discrete-time Markov Chains using Bounded Model Checking*

Ralf Wimmer, Bettina Braitling, and Bernd Becker

Chair of Computer Architecture
Albert-Ludwigs-University Freiburg im Breisgau, Germany
{wimmer,braitlin,becker}@informatik.uni-freiburg.de

Abstract. Since its introduction in 1999, bounded model checking has gained industrial relevance for detecting errors in digital and hybrid systems. One of the main reasons for this is that it always provides a counterexample when an erroneous execution trace is found. Such a counterexample can guide the designer while debugging the system.
In this paper we are investigating how bounded model checking can be applied to generate counterexamples for a different kind of model—namely discrete-time Markov chains. Since in this case counterexamples in general do not consist of a single path to a safety-critical state, but of a potentially large set of paths, novel optimization techniques like loop-detection are applied not only to speed-up the counterexample computation, but also to reduce the size of the counterexamples significantly. We report on some experiments which demonstrate the practical applicability of our method.

1 Introduction

Nowadays, formal verification plays a crucial role in the design process of digital circuits. In particular model checking, i. e., the proof that a system exhibits a set of properties, which are part of the specification, has gained great importance in industry. The reasons for its success are that it can be performed automatically and—in case a property is violated—that it is often able to generate a counterexample, which guides the designer when debugging the system. By using symbolic methods (e. g. ordered binary decision diagrams, OBDDs [1]) model checking can be applied to fairly large systems. Nevertheless, there are many practically important systems which cannot be verified using OBDDs (e. g., if they contain multipliers), because the size of the OBDD representations may explode.

To overcome this problem, Clarke et al. [2] suggested a method called Bounded Model Checking (BMC). It aims at the *refutation* of invariant properties. The

* 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.

reachability of a state within a fixed number of steps which violates an invariant is thereby formulated as a satisfiability problem in conjunctive normal form (CNF). The advantage is that the size of this CNF is linear in the number of gates of the digital circuit. Due to the enormous improvements in solving such satisfiability problems during the last two decades, BMC has successfully been applied to industrial systems.

On the other hand, on a higher modelling level, the designer often has to cope with uncertainties. They can result from unpredictable behavior of the environment, component failures, user interaction, ... One of the most common models for this scenario are discrete-time Markov chains (DTMCs). Model checking, which has been extended to DTMCs, is not as successful as model checking for digital circuits for three reasons: (1) The size of the systems which can be handled by state-of-the-art tools is still quite limited compared to the size of verifiable circuits, (2) all state-of-the-art model checkers use inexact arithmetic for efficiency reasons (i. e., using IEEE 754 floating point arithmetic and iterative solution methods for linear equation systems). This can cause numerical instabilities which themselves may lead to wrong results [3]. (3) The model checker is not able to provide a counterexample for violated properties, since the check is done by solving a linear equation system and not by traversing the state space.

In this paper we tackle the latter two problems. We propose an extension to bounded model checking such that we can use it to certify that invariant properties of the form "The probability to reach a state which violates the invariant is at most p" do not hold. For this purpose we provide counterexamples which can be checked easily for correctness by using exact arithmetic.

Related work. The generation of counterexamples for various kinds of properties has been studied extensively for non-stochastic models (see e. g. [4–7]). However, little research has been done on counterexamples for stochastic models. Aljazzar et al. [8, 9] apply informed search methods like Best First Search or A* to generate counterexamples for continuous-time Markov chains, whereas Han and Katoen [10] propose a different method to obtain counterexamples: They apply a k-shortest-paths algorithm to a Markov chain. Therewith they compute a counterexample consisting of a minimal number of paths. Although Han and Katoen do not provide any experimental results in their paper, we suppose that their method—as well as the method by Aljazzar et al.—does not scale well to large systems due to the explicit state representation it relies on. Further on both approaches do not apply any method to reduce the size of the counterexamples. As our experiments will show, it is crucial to reduce the size of the counterexamples in order to get useful counterexamples in reasonable time. We will overcome both problems by using a symbolic representation and effective techniques to reduce the size of counterexamples via loop detection.

The necessity of reducing the size of counterexamples has also been recognized by Damman et al. [11]. They compute regular expressions describing a sufficiently large set of paths from the initial state to a target state. This can be considered as a generalization of the loop-detection technique we will present later, but is restricted to explicitly represented state spaces.

In the paper [12] by Andrés et al. abstract away the details of strongly connected components of Markov chains by replacing the SCCs by edges that are labeled with the probability to walk through the SCC. On the resulting acyclic Markov chain, counterexamples are computed which are more compact by collapsing the SCCs, but lack the details how the SCCs can be traversed.

Recently, Hermanns et al. extended counterexample-guided abstraction refinement (CEGAR) to Markov decision processes (MDPs) [13]. In this scenario a counterexample is an adversary which resolves the nondeterminism in each state, such that the probability of reaching a set of states exceeds some bound. For the computation of counterexamples in the abstract system, they apply the shortest-paths algorithm of [10], but they could also use our method.

Organization of this paper. In the next section, we will briefly review the foundations of discrete-time Markov chains and bounded model checking. In Section 3 we will show how to bring together DTMCs and BMC. Its practical applicability will be demonstrated experimentally in Section 4, before we conclude in Section 5.

2 Foundations

In this section we will briefly review the basics of discrete-time Markov chains and bounded model checking for digital systems.

2.1 Discrete-time Markov Chains and Reachability Properties

Definition 1. *Let AP be a set of atomic propositions. A* discrete-time Markov chain *(DTMC) is a tuple $M = (S, s_I, P, L)$ such that S is a finite, non-empty set of states; $s_I \in S$, the initial state; $P : S \times S \to [0, 1]$, the matrix of the one-step transition probabilities; and $L : S \to 2^{AP}$, a labeling function that assigns each state the set of atomic propositions which hold in that state.*

We require the matrix P to be a stochastic matrix, i. e., $\sum_{s' \in S} P(s, s') = 1$ for all $s \in S$. A *path* π is a (finite or infinite) sequence $\pi = s_0 s_1 \dots$ of states such that $P(s_i, s_{i+1}) > 0$ for all $i \geq 0$. For a finite path $\pi = s_0 s_1 \dots s_n$, $|\pi| = n$ denotes its length. The i^{th} state of π is denoted by π^i, i. e., $\pi^i = s_i$. Furthermore let $\pi\uparrow^i$ be the i^{th} prefix of π ($\pi\uparrow^i = s_0 s_1 \dots s_i$). The set of infinite paths starting in state s is denoted by Path_s.

Definition 2. *Let $M = (S, s_I, P, L)$ be a DTMC and $s \in S$. We define a* probability space $\Psi_s = (\text{Path}_s, \Delta_s, \Pr_s)$ *such that*

- *Δ_s is the smallest σ-algebra generated by* Path_s *and the basic cylinders that are subsets of* Path_s*. Thereby, for a finite path π, the basic cylinder over π is defined as $\Delta(\pi) = \{\pi' \in \text{Path}_s \mid \pi'\uparrow^{|\pi|} = \pi\}$.*
- *$\Pr_s$ is the uniquely defined probability measure that satisfies the following constraints:* $\Pr_s(\text{Path}_s) = 1$ *and for all basic cylinders $\Delta(ss_1s_2 \dots s_n)$ over S:*

$$\Pr_s(\Delta(ss_1s_2 \dots s_n)) = P(s, s_1) \cdot P(s_1, s_2) \cdot \dots \cdot P(s_{n-1}, s_n).$$

Many investigations can be carried out on even simpler systems, namely Kripke structures, which are obtained by omitting the transition probabilities of the Markov chain:

Definition 3. *Let $M = (S, s_I, P, L)$ be a discrete-time Markov chain. The underlying* Kripke structure *is a labeled graph $G = (S, s_I, T, L)$ such that $(s, s') \in T$ iff $P(s, s') > 0$.*

The properties we are considering are PCTL formulae [14] of the form $\mathcal{P}_{\leq p}(a \mathsf{U} b)$ with $a, b \in AP$, meaning that the probability to walk along a path from the initial state to a state which satisfies b, passing only states in which a holds, is less or equal p. More formally:

Definition 4. *Let π be a path in a Markov chain $M = (S, s_I, P, L)$. It satisfies the formula $a \mathsf{U} b$ (written $\pi \vDash a \mathsf{U} b$) iff*

$$\exists i \geq 0 : \big(b \in L(\pi^i) \wedge \forall\, 0 \leq j < i : a \in L(\pi^j)\big).$$

A state s satisfies the formula $\mathcal{P}_{\leq p}(a \mathsf{U} b)$ (written $s \vDash \mathcal{P}_{\leq p}(a \mathsf{U} b)$) iff

$$\Pr_s\big(\{\pi \in \mathrm{Path}_s \mid \pi \vDash a \mathsf{U} b\}\big) \leq p.$$

If such a property is violated in the initial state s_I, i. e., if the actual probability is greater than p, there is a finite set of finite paths, starting in s_I and satisfying $a \mathsf{U} b$, whose probability measure is greater than p [10].

Definition 5. *Let $M = (S, s_I, P, L)$ be a discrete-time Markov chain for which the property $P_{\leq p}(a \mathsf{U} b)$ is violated in state s_I. A* counterexample *is a set C of paths which start in s_I and satisfy $a \mathsf{U} b$, such that $\Pr_{s_I}(C) > p$.*

Our goal is to provide the user with such a set of paths which testifies that the probability indeed exceeds the bound p.

2.2 Bounded Model Checking for Digital Systems

Bounded model checking [2] is an automatic technique to prove that some invariant is violated in a transition system, i. e., that a state in which the invariant does not hold is reachable from the initial state of the system. The reachability of such a state in a fixed number k of steps is thereby formulated as a satisfiability problem.

Let $I(s)$ be a predicate which is true if s is the initial state and false otherwise. Accordingly, let $P(s)$ be a predicate for the states violating the invariant. The transition relation is encoded by a predicate $T(s, t)$ which is true iff there is a transition from s to t. The existence of a run of length k starting in an initial state and ending in a state in which the invariant does not hold can then be described by the following formula:

$$BMC(k) = I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge P(s_k). \tag{1}$$

If this formula is unsatisfiable for the current unrolling depth k, there is no path of length k from an initial state to a safety-critical state. Otherwise we get a satisfying assignment which directly corresponds to a run of the system. It can be considered a counterexample for the invariant property.

The predicates are constructed in conjunctive normal form (CNF) from a digital circuit by applying Tseitin transformation [15]. It works by introducing auxiliary variables for internal signals. Then only local operations are necessary to get a satisfiability equivalent CNF: If G is an AND gate with inputs a, b and output c, then we construct the formula $c \leftrightarrow (a \wedge b)$, which is equivalent to $(\neg c \vee a) \wedge (\neg c \vee b) \wedge (c \vee \neg a \vee \neg b)$. A CNF for the whole circuit is given by the conjunction over the equivalences for each gate. Since this CNF is satisfied for an assignment iff it assigns consistent values to the signals in the circuit, we have to add a unit clause which sets the value of the output signal to true.

The advantage of applying Tseitin transformation is that the size of the predicates is linear in the size of the circuit. In contrast to that, the size, e. g., of OBDD representations, which are often used for symbolic model checking, might explode during the model checking process.

3 Bounded Model Checking for DTMCs

In this section we will turn our attention to how bounded model checking can be applied to discrete-time Markov chains in order to compute counterexamples for violated safety properties.

The differences to traditional bounded model checking are the following: (1) In general, it is not sufficient to compute a single path to a state which violates the invariant. Even the probability of the most probable path may be too small to exceed the bound p. Instead, we need a potentially large set of paths whose probability measure exceeds the given bound. (2) We normally cannot start from a circuit description of the system. Stochastic systems are usually modelled using a process algebraic description language from which the state space and the transition probabilities can be computed using techniques like parallel composition. An example for this formalism is the input language of the stochastic model checker PRISM [16], which we will later use for our experiments. Among other tools, PRISM is able to generate symbolic representations of the Markov chains. In particular, the transition probabilities are represented using Multi-terminal binary decision diagrams (MTBDDs) [17].

Since the matrix is normally sparse and well-structured, the symbolic MTBDD representation is in many cases much more compact than explicit representations. For this reason, we start with an MTBDD $\mathcal{P}(s,t)$ for the transition probability matrix, an OBDD $\mathcal{I}(s)$ for the initial state and an OBDD $\mathcal{L}_a(s)$ for each atomic proposition $a \in AP$ such that $\mathcal{L}_a(s) = 1$ iff $a \in L(s)$.

In the following, we will first describe some preprocessing steps, before we continue with the formula generation from OBDDs and the bounded model checking itself.

3.1 Preprocessing

Before we start the bounded model checking process, we reduce the computation of paths satisfying $a\mathsf{U}b$ to computing arbitrary paths ending in a b-state. Since we do not need the probabilities for the path computation, we can use the associated Kripke structure for this operation. The OBDD-representation[1] $\mathcal{T}(s,t)$ of its transition relation can be obtained from the transition probability matrix $\mathcal{P}(s,t)$ by a threshold operation, i. e., $\mathcal{T}(s,t) = 1$ iff $\mathcal{P}(s,t) > 0$.

Removing edges. We have to ensure that we will never find a path of which a proper prefix satisfies the path formula $a\mathsf{U}b$. This can be guaranteed if the b-states do not have any out-going edges. Furthermore we remove all out-going edges from states in which neither a nor b hold. This makes sure that all paths ending in a b state satisfy $a\mathsf{U}b$. Therefore we will call the b-states the *target states* of the system.

Often we can cut even more edges of the resulting graph: Only those a- (and target) states are relevant which can be reached from the initial state and from which a target state can be reached. These states can be determined using well-known symbolic graph traversal algorithms.

While the first edge cutting is mandatory for the correctness of the stochastic BMC algorithm, the latter is only an optimization. It may be skipped if the time or memory consumption of the reachability analysis is too high or if the resulting OBDD of the transition relation is considerably larger than the original one.

3.2 CNF generation

From the OBDD representations $\mathcal{I}(s)$, $\mathcal{T}(s,t)$, and $\mathcal{L}_b(s)$, we have to generate the predicates in conjunctive normal form (CNF). They will be denoted by $I(s)$, $T(s,t)$, and $L_b(s)$. Since OBDDs can be considered as a special form of circuits such that each node corresponds to a multiplexer, we can apply Tseitin transformation to obtain a CNF as already described in Section 2.2. The transformation results in (at most) four clauses per OBDD-node. Our experiments have shown that it is beneficial to reduce the size of the CNFs as much as efficiently possible. We apply the following ideas:

- To avoid unnecessary auxiliary variables when both successor nodes are leaves. This situation results in equivalences of the form $(n_1 \leftrightarrow x)$, where n_1 is the auxiliary variable associated with an internal signal and x the label of an OBDD node. Then every occurrence of n_1 can be replaced by x.
- To use reordering (e. g., sifting [18]) to reduce the size of the OBDD for the transition relation.

[1] In the following, we use calligraphic letters $\mathcal{L}, \mathcal{P}, \mathcal{T}, \ldots$ for OBDDs and MTBDDs.

3.3 Bounded Model Checking

Recall that for a given unrolling depth k the formula describing paths starting in the initial state s_I and ending in a target state is given by (cf. Eq. (1)):

$$BMC(k) = I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge L_b(s_k).$$

We feed it into a SAT-solver and try to find a satisfying solution. If the formula is unsatisfiable, there is no path of length k from the initial state to a target state. Otherwise, the satisfying assignment we get from the SAT-solver corresponds to a path in the original Markov chain which satisfies $a \mathsf{U} b$.

In contrast to model checking for digital or hybrid systems, we cannot stop once we have found a satisfying solution. Instead we have to continue until we have found enough paths such that their probability measure exceeds the given bound p.

For a boolean variable a, let a^1 denote the positive literal a and a^0 the negative literal $\neg a$. Let $s_{i,0}, \ldots, s_{i,n}$ be the variables of the state visited at time step i, and $v_{i,0}, \ldots, v_{i,n}$ their values assigned by the SAT-solver. To prevent the solver from finding the same solution again, we add the following clause to its clause database:

$$\left(\bigvee_{i=1}^{k} \bigvee_{j=0}^{n} s_{i,j}^{1-v_{i,j}} \right) \tag{2}$$

Thereby, we can ignore the auxiliary variables introduced by the Tseitin transformation, because their values are implied by the values of the original state variables. Furthermore we may start with step $i = 1$ instead of 0. Since the CNF contains unit literals which enforce that all solutions start in the initial state, the literals in the exclusion clause (2) which correspond to step 0 would be false and can therefore be left out completely. The same applies to step k if there is a unique target state.

We iterate the solution process until we have either found enough paths or the formula becomes unsatisfiable. If the latter is the case before we have found enough paths, we increase the unrolling depth k by one and continue.

Theorem 1. *If $s_I \not\models \mathcal{P}_{\leq p}(a \mathsf{U} b)$, the algorithm described above terminates and returns a set of paths whose probability measure is greater than p.*

Proof. (1) All paths which are found by the SAT-solver satisfy $a \mathsf{U} b$: Since all states which satisfy neither a nor b do not have any out-going edges anymore, the target state cannot be reached from such a state.
(2) The algorithm terminates after a finite number of steps: If $\mathcal{P}_{\leq p}(a \mathsf{U} b)$ is violated, there is a finite counterexample C (see [10]). This is found after at most $k = \max\{|\pi| \mid \pi \in C\}$ iterations. Since there is only a finite number of paths with length $\leq k$, the SAT-solver is only called a finite number of times.
(3) The probabilities are computed correctly: In our counterexample, no path is a prefix of another path. Hence the total probability is the sum of the probabilities of the single paths. □

3.4 Optimizing the BMC process

Having all the paths computed by a SAT-solver is a time-consuming task, in particular when the counterexample consists of a large number of paths. Therefore we need to reduce the number of calls to the SAT-solver.

Minimal distance initial state → target state. We would like to avoid unnecessary calls to the SAT-solver of which we can know in advance that they return UNSAT. This is the case if the current unrolling depth k is smaller than the length of the shortest path from the initial state to a target state. We therefore compute the minimal distance from s_I to a target state using the OBDD representation.

Algorithm 3.1: MinDist$\big($Initial state $\mathcal{I}(s)$, Target states $\mathcal{L}_b(s)\big)$
Begin
$\mathcal{R}(s) \leftarrow \mathcal{I}(s)$, $old(s) \leftarrow \emptyset$, $dist \leftarrow 0$ (1)
While $\mathcal{R}(s) \neq old(s)$ **And** $\mathcal{L}_b(s) \wedge \mathcal{R}(s) = \emptyset$ **Do** (2)
$old(s) \leftarrow \mathcal{R}(s)$ (3)
$\mathcal{R}(s) \leftarrow \mathcal{R}(s) \vee$ renameVariables$\big(t \rightarrow s$ in $\exists s : \mathcal{R}(s) \wedge \mathcal{T}(s,t)\big)$ (4)
$dist \leftarrow dist + 1$ (5)
End While (6)
If $\mathcal{L}_b(s) \wedge \mathcal{R}(s) \neq \emptyset$ **Then Return** $dist$ (7)
Else Return ∞ (8)
End

Later we will start with the minimal distance as the initial unrolling depth of the bounded model checking process.

Incomplete Assignments. Modern SAT-solvers detect satisfiability when all variables have been assigned a boolean value without resulting in a conflict. But often all clauses are satisfied before the solver has assigned each variable a boolean value.

If we are able to obtain a partial satisfying assignment in which n state variables are unassigned, this assignment corresponds to 2^n different paths. Furthermore, to exclude all these paths from the solution space only a single clause is necessary, which is shorter than a clause only excluding a single path.

Loop Detection. Let us assume we have found a path on which a state occurs twice, e. g., $\pi = s_I \xrightarrow{p_1} s_1 \xrightarrow{p_2} s_2 \xrightarrow{p_3} s_3 \xrightarrow{p_4} s_2 \xrightarrow{p_5} s_4$; s_I is the initial state, s_4 a target state. We can easily detect that there is a loop $s_2 \xrightarrow{p_3} s_3 \xrightarrow{p_4} s_2$. This loop can be taken arbitrarily often. Instead of returning a set of "flat" paths—one path for each unrolling of the loop—we construct counterexamples which consist of acyclic paths whose states may be annotated with loops (see Fig. 1).

Definition 6. *Let L be a set of loops, i. e., of paths σ such that $\sigma(0) = \sigma(|\sigma|)$ and for all $0 < i < |\sigma|$: $\sigma(i) \neq \sigma(0)$. An* annotated path $\hat{\pi}$ *is a pair $\hat{\pi} = (\pi, l)$*

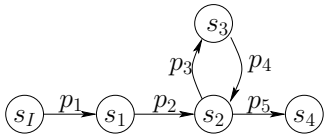


Fig. 1. A Path that is annotated with a loop.

such that π is an acyclic path and $l : \{0, \ldots, |\pi|\} \to 2^L$ assigns each position on the path a set of loops with $\forall i \in \{0, \ldots, |\pi|\}\ \forall \sigma \in l(i) : \pi(i) = \sigma(0) = \sigma(|\sigma|)$.

Such an annotated path $\hat{\pi}$ represents an infinite set of "flat" paths. For instance, the probability measure of the annotated path in Fig. 1 is

$$\Pr_{s_0}(\hat{\pi}) = \sum_{i=0}^{\infty} p_1 \cdot p_2 \cdot (p_3 \cdot p_4)^i \cdot p_5 = p_1 \cdot p_2 \cdot \frac{1}{1 - (p_3 \cdot p_4)} \cdot p_5.$$

We have to take into account that this probability includes the probability of the path with 0 unrollings of the loop. This acyclic path has been found by the SAT-solver in one of the previous iterations.

If we have attached m different loops with probabilities $p_1, p_2, \ldots, p_m$ to the same state, we can increase the probability of the underlying acyclic path by the following factor:

$$\begin{aligned} p &= 1 + (p_1 + p_2 + \cdots p_m) + (p_1^2 + p_1 p_2 + p_2 p_1 + p_2^2 + p_1 p_3 + \cdots + p_m^2) + \cdots \\ &= \sum_{n=0}^{\infty} \sum_{\substack{i_1, i_2, \ldots, i_m \in \mathbb{N} \\ i_1 + i_2 + \cdots + i_m = n}} \binom{n}{i_1\ i_2\ \cdots\ i_m} p_1^{i_1} p_2^{i_2} \cdots p_m^{i_m} \\ &= \sum_{n=0}^{\infty} (p_1 + p_2 + \cdots + p_m)^n = \frac{1}{1 - (p_1 + p_2 + \cdots + p_m)}. \end{aligned} \tag{3}$$

The advantages of generating sets of annotated paths are: (1) The user learns more about the structure of the system. This makes the diagnosis of faults easier. (2) The size of the counterexample can become considerably smaller since an infinite number of paths is represented by one annotated path. (3) The number of calls to the SAT-solver is reduced and thereby also the runtime of the bounded model checking process.

Path exclusion. Before calling the SAT-solver, we have to exclude all those paths from the solution space of the SAT-problem which originate from unrolling annotated paths we have already found in previous iterations and whose length is identical to the current unrolling depth k.

Assume, we have an annotated path $\hat{\pi} = (\pi, l)$, which is annotated with loops $\pi_1, \ldots, \pi_p$. We have to decide how often we have to unroll which loop in order

to obtain a path of total length k. These unrollings correspond to the solutions of the following constraint with $r_1, \ldots, r_p \in \mathbb{N}$:

$$|\pi| + |\pi_1| \cdot r_1 + |\pi_2| \cdot r_2 + \cdots + |\pi_p| \cdot r_p = k. \tag{4}$$

We solve this constraint system using a simple enumeration algorithm. More sophisticated methods are certainly available.

If there are several loops attached to the same state, we need to exclude them in all possible orders. This can result in a large number of long clauses which need to be added to the SAT-solver's clause database. All paths generated by a single solution have the literals corresponding to the acyclic base path in common. The remaining literals can be grouped according to the state of the base path their loop corresponds to. Let Π be the set of paths corresponding to a single solution of Eq. 4. They all have the acyclic base path $s_0 s_1 \ldots s_{n-1}$ in common, and the states s_i occur at the same positions for all $\pi \in \Pi$. Each clause which excludes a path $\pi \in \Pi$ can therefore be split into C^{base}—the literals to exclude the base path—and $C_\pi^{s_i}$ for $i = 0, \ldots, n-2$—the sets of literals to exclude the unrolling of the loops attached to s_i. Using the naive approach, the clauses to exclude the paths in Π are given by $\{C^{\text{base}}\} \times \{C_\pi^{s_0} \mid \pi \in \Pi\} \times \cdots \times \{C_\pi^{s_{n-2}} \mid \pi \in \Pi\}$.

By introducing auxiliary variables, this set can be reduced using the following observation: Assume we have two clauses $(C \vee S_1) \wedge (C \vee S_2)$, whereby C, S_1, and S_2 are disjunctions of literals and C are the literals which both clauses have in common. We introduce a new auxiliary variable a and replace the two clauses by $(C \vee a) \wedge (\neg a \vee S_1) \wedge (\neg a \vee S_2)$. Applied to our path exclusion problem this leads to the following set of clauses:

$$\left\{C^{\text{base}} \dot\cup \{a_{s_0}, \ldots, a_{s_{n-2}}\}\right\} \dot\cup \bigcup_{i=0}^{n-2} \left\{C_\pi^{s_i} \dot\cup \{\neg a_{s_i}\} \mid \pi \in \Pi\right\}. \tag{5}$$

The variables a_{s_i} are the auxiliary variables used to split off the loop unrolling at state s_i of the base path. The following lemma counts the number of clauses and literals for both approaches:

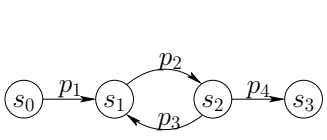
Lemma 1. *Let L be a set of loops; $\hat\pi = (\pi, l)$, an annotated path over L; $\pi = s_0 s_1 \ldots, s_{n-1}$; and k, the current unrolling depth. Furthermore let $u : L \to \mathbb{N}$ be a solution of Eq. (4). Let $u(L') = \sum_{l' \in L'} u(l')$ for $L' \subseteq L$ and $l(s_i) = \{l_1^i, \ldots, l_{n_i}^i\}$ the loops attached to state s_i. The number of clauses of the naive path exclusion vs. the optimized path exclusion is given by*

$$\#_c^{naive} = \prod_{i=0}^{n-2} \binom{u(l(s_i))}{u(l_1^i) \cdots u(l_{n_i}^i)} \qquad \textit{vs.} \qquad \#_c^{opt} \leq 1 + \sum_{i=0}^{n-2} \binom{u(l(s_i))}{u(l_1^i) \cdots u(l_{n_i}^i)}.$$

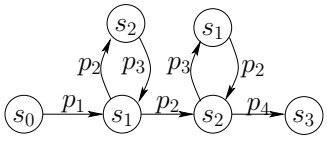
If bps denotes the number of bits per state, the total number of literals for both approaches is

$$\#_l^{naive} = \#_c^{naive} \cdot (k+1) \cdot bps$$

$$\#_l^{opt} \leq (|\pi| + 1) \cdot bps + |\pi| + \sum_{i=0}^{n-2} \left(\binom{u(l(s_i))}{u(l_1^i) \cdots u(l_{n_i}^i)} \cdot \left(1 + bps \cdot \sum_{l' \in l(s_i)} |l'| \cdot u(l')\right)\right).$$



(a) A path with only one loop ...



(b) ... but we might detect two loops if we are not careful enough.

Fig. 2. A Path with an embedded loop.

Correctness issues. We have to guarantee that we do not count the same path twice when we compute the probability measure of the set of annotated paths. This happens if the same path can be generated by unrolling loops in different ways.

Example 1. Let us assume the SAT-solver returns the path $s_0 \xrightarrow{p_1} s_1 \xrightarrow{p_2} s_2 \xrightarrow{p_3} s_1 \xrightarrow{p_2} s_2 \xrightarrow{p_4} s_3$. In principle, we could detect two loops: $s_1 \to s_2 \to s_1$ and $s_2 \to s_1 \to s_2$, which would result in the annotated path depicted in Fig. 2(b). One might think the probability of this annotated path was $p_1 \cdot \frac{1}{1-(p_2 \cdot p_3)} \cdot p_2 \cdot \frac{1}{1-(p_3 \cdot p_2)} \cdot p_4$, but this is not correct. The reason is that the same paths are generated by unrolling the first loop as by the second loop. The correct probability is therefore only $p_1 \cdot \frac{1}{1-(p_2 \cdot p_3)} \cdot p_2 \cdot p_4$. □

We can avoid to over-estimate the probability measure with the following two observations:

(1) On each path returned by the SAT-solver, there can be at most one loop which we may attach to a state of the underlying acyclic path. If it was a path with two loops, the solver would have returned two shorter paths—each with only one of the loops—in previous iterations. But then, we would have excluded the path with both loops in advance. If the path contains more than one state twice, this means that either the loop itself consists of sub-loops or that we have the situation depicted in Fig. 2(a). Both problems can be solved if we attach the loop to the first state which occurs twice on the path.

(2) We only attach loops to that position on the current path where the loop has been found by the SAT-solver. If the SAT-solver returns a path with a loop we can conclude that this path cannot be generated by unrolling any existing annotated path in our collection. So we may attach the loop to the corresponding basic path without over-estimating the correct probability measure.

The pseudocode of the optimized algorithm with loop detection is sketched in Algorithm 3.2. Starting with the minimal distance from the initial state to a target state, we iterate lines 1–14 until either the counterexample exceeds the probability bound p or the maximal unrolling depth has been finished. First, we generate the CNF for the current unrolling depth k (line 2), then we exclude all paths of length k, which originate from unrolling loops (line 3).

Algorithm 3.2: StochBMC(CNF I, CNF T, CNF L_b, Probability p)
Begin

```
(1)  For k ← minDist To maxDist Do
(2)      φ ← CreateCNF(I, T, L_b, k)
(3)      φ ← φ ∧ ExcludePrecomputedPaths(C, k)
(4)      While φ is satisfiable Do
(5)          π ← Solution2Path(φ)
(6)          φ ← φ ∧ ExcludePath(π)
(7)          If π contains a cycle Then
(8)              (π_base, π_loop) ← split(π)
(9)              b ← findBasePath(C, π_base)
(10)             C ← (C \ {b}) ∪̇ {attachLoop(b, π_loop)}
(11)         Else C ← C ∪ {π}
(12)         If Pr_{s_I}(C) > p Then Return C
(13)     End While
(14) End For
(15) Return "I could not generate a counterexample!"
```

End

As long as the resulting formula is satisfiable, we exclude the newly found path from the solution space of the CNF. Furthermore we test if the path that corresponds to the satisfying assignment contains a cycle. If this is the case, split the path into the acyclic base path and the loop which we attach to the base path that is already contained in the counterexample (lines 8–10). Otherwise we insert the (acyclic) path into the counterexample (line 11). The algorithm terminates as soon as the probability measure of C is larger than p.

4 Experiments

We have implemented the BMC algorithm in C++ with the optimizations that were presented above. We use Cudd [19] for the OBDDs and Minisat [20] as state-of-the-art SAT-solver. We ran all our experiments on a Dual Core AMD Opteron™ processor with 2.4 GHz and 4 GB of main memory. For all experiments we set a time limit of 2 hours and a memory limit of 1 GB. The results of the experiments are listed in Table 1. Experiments which were aborted due to the time limit are marked with "— TL —". We used the probabilistic model checker PRISM [16] to generate symbolic representations for benchmarks, all of which are available from the PRISM website `http://prismmodelchecker.org`.

The first benchmark we used is the synchronous leader election protocol by Itai and Rodeh [21]. It models a ring of N processors with a common clock. The goal is to elect a leader, i. e. a uniquely designated processor, by sending messages around the ring. The protocol proceeds in rounds. During each round the processors draw a random number from the range $\{1, \ldots, K\}$ as an id. If there is a unique maximal id, the processor with this id becomes the leader. Otherwise a new round is initiated. The Markov chains consist of a few hundred (for $N = 3$) up to 12 500 states (for $N = 5$, $K = 5$). We checked the property that finally a leader is elected. Since this happens with probability 1.0 when

Table 1. Experimental results for the leader election protocol

			without loop detection				*with loop detection*				
N	K	bound	paths	depth	SAT-calls	time [s]	paths	loops	depth	SAT-calls	time [s]
3	2	0.99	66	16	79	0.15	6	12	8	23	0.05
3	3	0.99	143	12	152	0.39	24	66	8	95	0.20
3	4	0.99	276	8	281	0.72	60	202	8	267	0.51
3	5	0.99	589	8	594	3.08	120	451	8	576	2.41
3	6	0.99	1040	8	1045	3.97	210	808	8	1023	3.76
3	8	0.99	1979	8	1984	8.31	504	1455	8	1964	5.42
3	10	0.99	990	4	991	5.63	990	0	4	991	5.16
3	12	0.99	1711	4	1712	8.82	1711	0	4	1712	8.20
4	2	0.99	— TL —				8	64	10	78	0.23
4	3	0.99	— TL —				60	1215	10	1281	9.22
4	4	0.90	3903	12	3909	79.52	216	3223	10	3445	61.84
4	5	0.90	2123	10	2129	99.01	560	1483	10	2049	85.45
4	6	0.90	1167	5	1168	30.80	1167	0	5	1168	28.13
4	8	0.90	3687	5	3688	113.95	3687	0	5	3688	102.58
5	2	0.90	— TL —				10	204	12	221	1.06
5	3	0.90	9585	12	9592	382.00	180	7508	12	7695	272.06
5	4	0.90	23019	12	23026	2948.00	900	20270	12	21177	2438.50
5	5	0.90	2813	6	2814	1102.67	2813	0	6	2814	1044.63
6	2	0.90	— TL —				12	606	14	626	6.56
7	2	0.90	— TL —				14	1571	16	1594	31.46
8	2	0.90	— TL —				16	3796	18	3822	137.46

starting in the initial state, we generated paths to provide a certificate that the given bounds (0.99 and 0.90, respectively) are indeed exceeded.

One can see that the version with loop detection always performs better than the version without. In those cases where the counterexample does not contain any loops, both versions of the algorithm perform similarly. For other instances loop detection turns out to be essential for counterexample generation. E. g., for the `leader`-benchmark with $N = 4$ and $K = 2$, the algorithm without loop detection found 239 138 paths (maximal length: 29) with a total probability measure of 0.980 077 2 before the time limit was exceeded. In contrast to this, the optimized algorithm only needed 8 paths with 64 loops and an unrolling depth of 10 to reach the probability bound of 0.99.

In addition to the leader election protocol, we applied BMC to a contract signing protocol [22, 23]. Its purpose is to exchange pieces of information (e. g. digital signatures) between two parties A and B over a network. One important property that such a protocol should exhibit is fairness. This means, whenever B has obtained A's commitment, B cannot prevent A from getting B's commitment. For the variant of the protocol we used, the actual probability to reach an unfair state is slightly larger than 0.5. The violated property for which we generated counterexamples is $\mathcal{P}_{<\frac{1}{2}}(\mathrm{true}\,\mathsf{U}\,\mathrm{unfair})$. Since the optimized algorithm did not find any loops before reaching the probability bound and since the results for both variants do not differ substantially, we only show the results of the algorithm without loop detection in Table 2. N denotes the number of data pairs to exchange; and L, the size of each piece of data.

Table 2. Experimental results for the contract signing protocol

N	L	states	prob.	bound	SAT-calls	paths	depth	time [s]
5	3	53041	0.515625	0.50	513	512	31	49.67
5	4	73211	”	”	513	512	41	158.53
5	5	93381	”	”	513	512	51	278.90
5	6	115710	”	”	513	512	61	441.64
6	2	159742	0.5078125	0.50	2049	2048	25	127.12
6	3	258046	”	”	2049	2048	37	293.14
6	4	356350	”	”	2049	2048	49	758.91
6	5	454654	”	”	2049	2048	61	1778.75
7	2	737278	0.50390625	0.50	8193	8192	29	668.51
7	3	1196030	”	”	8193	8192	43	2231.64

The results for the contract signing protocol show that we are able to handle quite large Markov chains with more than 10^6 states. More states are possible if we decrease the probability bound for the counterexample.

5 Conclusion

In the previous sections we have shown how bounded model checking can successfully be applied to generate counterexamples when invariant properties of Markov chains are violated. By returning not simply “flat” paths, but paths which are annotated with loops, we make not only the algorithm more efficient, but also the counterexamples more useful for the designer: (1) In many cases fewer paths are needed such that the given probability bound is exceeded and (2) the structured counterexamples provide more information about the system: Each annotated path represents an acyclic execution trace together with possible deviations from this direct path in the form of loops. Additionally, bounded model checking returns a counterexample which consist of shortest possible paths (although they might be not the most probable ones).

Points for further research are to make the detection of loops more powerful such that loops themselves may consist of loops. This will reduce the size of the counterexample further. Other optimizations aim at reducing the number of SAT-calls, e. g. by re-using loops on different annotated paths. Furthermore, optimization techniques known from conventional bounded model checking should be transferred to the stochastic world to speed-up the solution of the SAT-instances. The extension of bounded model checking to Markov decision processes (MDPs) will be another topic of our future research.

References

1. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers **35**(8) (1986) 677–691
2. Clarke, E.M., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods in System Design **19**(1) (2001) 7–34

3. Wimmer, R., Kortus, A., Herbstritt, M., Becker, B.: Probabilistic model checking and reliability of results. In: 11th IEEE Int'l Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS), IEEE CS (2008) 207–212
4. Hojati, R., Brayton, R.K., Kurshan, R.P.: BDD-based debugging of design using language containment and fair CTL. In: 5th Int'l Conf. on Computer Aided Verification (CAV). Vol. 697 of LNCS., Springer (1993) 41–58
5. Clarke, E.M., Grumberg, O., McMillan, K.L., Zhao, X.: Efficient generation of counterexamples and witnesses in symbolic model checking. In: 32nd Design Automation Conference. (1995) 427–432
6. Clarke, E.M., Jha, S., Lu, Y., Veith, H.: Tree-like counterexamples in model checking. In: Symposium on Logic in Computer Science (LICS), IEEE CS (2002) 19–29
7. Gurfinkel, A., Chechik, M.: Proof-like counter-examples. In: 9th Int'l Conf. on TACAS. Vol. 2619 of LNCS., Springer (2003) 160–175
8. Aljazzar, H., Hermanns, H., Leue, S.: Counterexamples for timed probabilistic reachability. In: 3rd Int'l Conf. on Formal Modeling and Analysis of Timed Systems (FORMATS). Vol. 3829 of LNCS., Springer (2005) 177–195
9. Aljazzar, H., Leue, S.: Extended directed search for probabilistic timed reachability. In: 4th Int'l Conf. on Formal Modeling and Analysis of Timed Systems (FORMATS). Vol. 4202 of LNCS., Springer (2006) 33–51
10. Han, T., Katoen, J.P.: Counterexamples in probabilistic model checking. In: 13th Int'l Conf. on TACAS. Vol. 4424 of LNCS., Springer (2007) 72–86
11. Damman, B., Han, T., Katoen, J.P.: Regular expressions for PCTL counterexamples. In Rubino, G., ed.: 5th QEST, Saint-Malo, France, IEEE CS (2008) 179–188
12. Andrés, M.E., D'Argenio, P., van Rossum, P.: Significant diagnostic counterexamples in probabilistic model checking. In: Haifa Verification Conference. LNCS, Springer (2008)
13. Hermanns, H., Wachter, B., Zhang, L.: Probabilistic CEGAR. In: Int'l Conf. on Computer Aided Verification (CAV). Vol. 5123 of LNCS., Springer (2008) 162–175
14. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing **6**(5) (1994) 512–535
15. Tseitin, G.S.: On the complexity of derivation in propositional calculus. Studies in Constructive Mathematics and Mathematical Logic, Part 2 (1970) 115–125
16. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: 12th Int'l Conf. on TACAS. Vol. 3920 of LNCS., Springer (2006) 441–444
17. Fujita, M., McGeer, P.C., Yang, J.C.Y.: Multiterminal binary decision diagrams: An efficient data structure for matrix representation. Formal Methods in System Design **10**(2/3) (1997) 149–169
18. Rudell, R.: Dynamic variable ordering for ordered binary decision diagrams. In: IEEE/ACM Int'l Conf. on Computer Aided Design (ICCAD). (1993) 42–47
19. Somenzi, F.: CUDD: CU Decision Diagram Package Release 2.4.1. University of Colorado at Boulder (2005)
20. Eén, N., Sörensson, N.: An extensible SAT-solver. In: 6th Int'l Conf. on Theory and Applications of Satisfiability Testing. Vol. 2919 of LNCS., Springer (2003) 502–518
21. Itai, A., Rodeh, M.: Symmetry breaking in distributed networks. Information and Computation **88**(1) (1990) 60–87
22. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Communications of the ACM **28**(6) (1985) 637–647
23. Norman, G., Shmatikov, V.: Analysis of probabilistic contract signing. Journal of Computer Security **14**(6) (2006) 561–589