Counterexample Generation for Markov Chains Using SMT-Based Bounded Model Checking*

Bettina Braitling[1], Ralf Wimmer[1], Bernd Becker[1],
Nils Jansen[2], and Erika Ábrahám[2]

[1] Albert-Ludwigs-University Freiburg, Germany
{braitlin,wimmer,becker}@informatik.uni-freiburg.de
[2] RWTH Aachen University, Germany
{abraham,nils.jansen}@informatik.rwth-aachen.de

Abstract. Generation of counterexamples is a highly important task in the model checking process. In contrast to, e. g., digital circuits where counterexamples typically consist of a single path leading to a critical state of the system, in the probabilistic setting counterexamples may consist of a large number of paths. In order to be able to handle large systems and to use the capabilities of modern SAT-solvers, bounded model checking (BMC) for discrete-time Markov chains was established.

In this paper we introduce the usage of SMT-solving over linear real arithmetic for the BMC procedure. SMT-solving, extending SAT with theories in this context on the one hand leads to a convenient way to express conditions on the probability of certain paths and on the other hand allows to handle Markov reward models. We use the former to find paths with high probability first. This leads to more compact counterexamples. We report on some experiments, which show promising results.

1 Introduction

The verification of formal systems has gained great importance both in research and industry. *Model checking* proves or refutes automatically (i. e. without user interaction) that a system exhibits a given set of properties (see, e. g., [1]). In many cases model checking also provides helpful diagnostic information; in case of a defective system a *counterexample* in form of a witnessing run is returned. The usage of symbolic representations like ordered binary decision diagrams (OBDDs) [2] assures the usability of model checking for many kinds of large systems. However, there are classes of practically relevant systems for which even these OBDD-based methods fail. To fill this gap, *bounded model checking* (BMC) was developed [3]. Thereby the existence of a path of fixed length that refutes the property under consideration is formulated as a satisfiability (SAT) problem.

* 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) and the DFG-Project "CE-Bug – CounterExample Generation for Stochastic Systems using Bounded Model Checking".

R. Bruni and J. Dingel (Eds.): FMOODS/FORTE 2011, LNCS 6722, pp. 75–89, 2011.

As the size of the corresponding formula is relatively small and as modern SAT-solvers have strongly evolved over the last 15 years, it is not surprising that this approach is very successful.

To model real-life scenarios it is often necessary to cope with specific uncertainties using probabilities. For instance, properties can be formulated in *probabilistic computation tree logic* (PCTL) [4] for models called *discrete-time Markov chains* (DTMCs). The classical model checking approach for this setting is based on the solution of a linear equation system [4]. However, it lacks the generation of counterexamples, since the solution of the equation system yields only the mere probabilities without further information.

To provide a user with such diagnostic information for probabilistic systems, there have been some recent developments during the last few years [5,6,7,8,9]. Contrary to, e. g., LTL model checking for digital systems, counterexamples for a PCTL property may consist of a large number of paths to reach certain probability bounds. Various techniques have been proposed to obtain *compact representations*: incrementally adding the paths with the highest probability [6,8], reducing the strongly connected components of the underlying digraph of a DTMC [5,9], and using regular expressions to describe whole sets of counterexamples [6]. All these techniques rely on an explicit representation of the state space.

Bounded model checking for probabilistic systems in the context of counterexample generation was introduced in [7], which represents the state space symbolically. This procedure can be used to refute probabilistic reachability problems. They can be formulated in PCTL [4] as $\mathcal{P}_{\leq p}(a \,\mathcal{U}\, b)$ with atomic propositions a and b. The meaning of this formula is that the probability to reach a b-state, passing only a-states, may be at most p. The refutation is shown by using a SAT-solver to find paths satisfying the property whose joint probability measure exceeds the bound p. The input of the solver are propositional formulae that are satisfied iff the assignment of the variables corresponds to a sequence of states of the DTMC that represents a path to a target state. This process is significantly accelerated by a loop-detection mechanism, which is used to handle sets of paths which differ only in the number of unfoldings of the same loops.

A drawback of the state-of-the-art BMC procedure for DTMCs is that paths are found in an arbitrary order while for the size of counterexamples it is often advantageous to start with paths of high probability. Moreover, it is desirable to use this procedure for *Markov reward models* (MRMs), which extend DTMCs by the possibility to model costs (or dually rewards) of operations. MRMs allow to verify properties like "The probability to finish the computation with costs larger than c is at most 10^{-3}."

In this paper we therefore extend stochastic BMC in order to handle these problems. Instead of using a SAT-solver, we use the more powerful approach of SMT-solving. SMT stands for *satisfiability modulo theories* and is a generalization of SAT [10]. It allows to express propositional formulae representing paths to target states together with conditions on the probability of such paths. Furthermore, real numbers that are allocated to the states by a cost-function can be added up and restrictions on the accumulated costs of paths can be imposed.

We will also show how this counterexample generation can be combined with minimization techniques for Markov chains in order not only to speed up the counterexample generation but also to obtain more abstract counterexamples.

Organization of the paper. At first, we give a brief introduction to the foundations of DTMCs, counterexamples, Markov reward models, and bisimulation minimization. Section 3 then explains the concept of SMT-solving for this context. In Sect. 4 we describe the results of some experiments we did on well-known test cases. In Sect. 5 we draw a short conclusion and give an outlook to future work on this approach.

2 Foundations

In this section we take a brief look at the basics of discrete-time Markov chains, Markov reward models, and bisimulation minimization.

2.1 Stochastic Models

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* $L : S \to 2^{\mathrm{AP}}$ *a labeling function that assigns each state the set of atomic propositions which hold in that state.*

P has to be a stochastic matrix that satisfies $\sum_{s' \in S} P(s, s') = 1$ for all $s \in S$. A (finite or infinite) *path* π is a (finite or infinite) sequence $\pi = s_0 s_1 \ldots$ of states such that $P(s_i, s_{i+1}) > 0$ for all $i \geq 0$. A finite path $\pi = s_0 s_1 \ldots s_n$ has length $|\pi| = n$; for infinite paths we set $|\pi| = \infty$. For $i \leq |\pi|$, π^i denotes the i^{th} state of π, i. e., $\pi^i = s_i$. The i^{th} prefix of a path π is denoted by $\pi{\uparrow}^i = s_0 s_1 \ldots s_i$. The set of finite (infinite) paths starting in state s is called $\mathrm{Path}_s^{\mathrm{fin}}$ ($\mathrm{Path}_s^{\mathrm{inf}}$).

In order to obtain a probability measure for sets of infinite paths we first need to define a probability space for DTMCs:

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

- Δ_s *is the smallest* σ*-algebra generated by* $\mathrm{Path}_s^{\mathrm{inf}}$ *and the set of basic cylinders of the paths from* $\mathrm{Path}_s^{\mathrm{fin}}$*. Thereby, for a finite path* $\pi \in \mathrm{Path}_s^{\mathrm{fin}}$*, the basic cylinder over* π *is defined as* $\Delta(\pi) = \{\lambda \in \mathrm{Path}_s^{\mathrm{inf}} \mid \lambda{\uparrow}^{|\pi|} = \pi\}$.
- $\Pr_s$ *is the uniquely defined probability measure that satisfies the equation* $\Pr_s(\Delta(ss_1s_2 \ldots s_n)) = P(s, s_1) \cdot P(s_1, s_2) \cdot \cdots \cdot P(s_{n-1}, s_n)$ *for all basic cylinders* $\Delta(ss_1s_2 \ldots s_n)$.

The properties we want to consider are formulated in PCTL [4] and are of the form $\mathcal{P}_{\leq p}(a \,\mathcal{U}\, b)$ with $a, b \in \mathrm{AP}$. This means that the probability to walk along a path from the initial state to a state in which b holds, with all intermediate states satisfying a, is less than or equal to p. More formally: A path π satisfies

$a \mathcal{U} b$, written $\pi \vDash a \mathcal{U} b$, 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 $s \in S$ satisfies the formula $\mathcal{P}_{\leq p}(a \mathcal{U} b)$ (written $s \vDash \mathcal{P}_{\leq p}(a \mathcal{U} b)$) iff $\Pr_s\big(\{\pi \in \mathrm{Path}_s^{\inf} \mid \pi \vDash a \mathcal{U} b\}\big) \leq p$.

Let us assume that such a formula $\mathcal{P}_{\leq p}(a \mathcal{U} b)$ is violated by a DTMC M. That means $\Pr_{s_I}\big(\{\pi \in \mathrm{Path}_s^{\inf} \mid \pi \vDash a \mathcal{U} b\}\big) > p$. In this case we want to compute a counterexample which certifies that the formula is indeed violated. Hence a counterexample is a set of paths that satisfy $a \mathcal{U} b$ and whose joint probability measure exceeds the bound p.

Definition 3. *Let $M = (S, s_I, P, L)$ be a discrete-time Markov chain for which the property $\varphi = \mathcal{P}_{\leq p}(a \mathcal{U} b)$ is violated in state s_I. An **evidence** for φ is a finite path $\pi \in \mathrm{Path}_{s_I}^{\mathrm{fin}}$ such that $\pi \vDash a \mathcal{U} b$, but no proper prefix of π satisfies this formula. A **counterexample** is a set $C \subseteq \mathrm{Path}_{s_I}^{\mathrm{fin}}$ of evidences such that $\Pr_{s_I}(C) > p$.*

Han and Katoen have shown that there is always a finite counterexample if $\mathcal{P}_{\leq p}(a \mathcal{U} b)$ is violated [11].

In order to be able to express properties like *"The probability to reach a target state with costs larger than c is at most p"*, Markov chains have to be extended by so-called reward functions.

Definition 4. *A **Markov reward model** (MRM) is a pair $(M, \boldsymbol{R})$ where $M = (S, s_I, P, L)$ is a DTMC and $\boldsymbol{R} : S \to \mathbb{R}$ a real-valued reward function.*

Rewards can be used to model costs of certain operations, to count steps or to measure given gratifications. The variant of rewards we use here are *state rewards*. One could also assign reward values to transitions instead of states (so-called *transition rewards*). We restrict ourselves to state rewards; all techniques that are developed in this paper can also be applied to transition rewards.

Let $\pi \in \mathrm{Path}_s^{\mathrm{fin}}$ be a finite path and $\mathbf{R}$ a reward function. The *accumulated reward* of π is given by $\mathbf{R}(\pi) = \sum_{i=0}^{|\pi|-1} \mathbf{R}(\pi^i)$. Note that the reward is granted when leaving a state.

We extend the PCTL-properties from above to Markov reward models. For a (possibly unbounded) interval $\mathcal{I} \subseteq \mathbb{R}$, a path π satisfies the property $a \mathcal{U}^{\mathcal{I}} b$, written $\pi \vDash a \mathcal{U}^{\mathcal{I}} b$, if there is $0 \leq i \leq |\pi|$ such that $b \in L(\pi^i)$, $\mathbf{R}(\pi{\uparrow^i}) \in \mathcal{I}$, and $a \in L(\pi^j)$ for all $0 \leq j < i$. A state $s \in S$ satisfies $\mathcal{P}_{\leq p}(a \mathcal{U}^{\mathcal{I}} b)$ if $\Pr_s\big(\{\pi \in \mathrm{Path}_s^{\inf} \mid \pi \vDash a \mathcal{U}^{\mathcal{I}} b\}\big) \leq p$. Our techniques can be extended in a straightforward way to $\mathcal{I}$ being the union of a finite set of intervals. Please note that the bounded until operator of PCTL is a special case of a reward condition.

Our goal is to compute counterexamples to refute such reward properties using bounded model checking (BMC).

2.2 Bisimulation Minimization

For the generation of counterexamples we work with a symbolic representation of the DTMC and the reward function. Symbolic representations have the advantage that the size of the representation is not directly correlated with the

size of the represented system. The representation can be smaller by orders of magnitude. By using algorithms whose running time only depends on the size of the representation, very large state spaces can be handled.

But even if the symbolic representation of a system is small, the number of paths that are needed for a counterexample can be very large. In order to reduce the number of paths, we first compute the smallest system that has the same behavior w. r. t. probabilities and rewards as the original one. This in general not only reduces the number of states, but also the number of paths, because all paths that stepwise consist of equivalent states are mapped onto a single path in the minimized system.

The methods for computing such minimal systems are based on bisimulation relations. A bisimulation groups states whose behavior is indistinguishable into equivalence classes [12]:

Definition 5. *Let $M = (S, s_I, P, L)$ be a DTMC and $\boldsymbol{R} : S \to \mathbb{R}$ a reward function for M. A partition $\boldsymbol{P}$ of S is a **bisimulation** if the following holds for all equivalence classes C of $\boldsymbol{P}$ and for all $s, t \in S$ such that s and t are contained in the same block of $\boldsymbol{P}$:*

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

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

The equivalence classes of bisimilarity and the coarsest bisimulation partition coincide. The equivalence class of a state $s \in S$ w. r. t. a bisimulation $\mathbf{P}$ is denoted by $[s]_P$. The classes of the bisimulation now become the states of a new DTMC.

Definition 6. *Let $M = (S, s_I, P, L)$ be a DTMC, $\boldsymbol{R}$ a reward function for M, and $\boldsymbol{P}$ be a bisimulation. The **bisimulation quotient** is the DTMC $(\boldsymbol{P}, \mathrm{C}_I, P', L')$ with reward function $\boldsymbol{R}'$ such that*

- *For all $\mathrm{C}, \mathrm{C}' \in \boldsymbol{P}$: $P'(\mathrm{C}, \mathrm{C}') = P(s, \mathrm{C}')$ for an arbitrary $s \in \mathrm{C}$,*
- *C_I is the block that contains the initial state s_I of M,*
- *$\forall\, \mathrm{C} \in \boldsymbol{P} : L'(\mathrm{C}) = L(s)$ for an arbitrary state $s \in \mathrm{C}$, and*
- *$\forall\, \mathrm{C} \in \boldsymbol{P} : \boldsymbol{R}'(\mathrm{C}) = \boldsymbol{R}(s)$ for an arbitrary state $s \in \mathrm{C}$.*

The quotient can be considerably smaller than the original system. At the same time it still satisfies the same PCTL and reward properties. All analyses can therefore be carried out on the quotient system instead of the larger original Markov model. For symbolic algorithms to compute the bisimulation quotient of a DTMC or MRM, we refer the reader to, e. g., [13,14].

3 SMT-Based Bounded Model Checking for Counterexample Generation

In this section we show how counterexamples can be computed using an SMT-based formulation of bounded model checking (BMC). BMC has already been

applied in [7] for this purpose but in a purely propositional variant. The main drawback of the existing approach is that the propositional BMC formula only contains information about the mere existence of a path, but not about its probability measure. Hence there is no direct possibility to control the SAT-solver such that paths with high probability measure are preferred.

Here we propose the usage of a more powerful formulation than purely propositional logic. The *satisfiability modulo theories* (SMT) problem is a generalization of the propositional satisfiability (SAT) problem. It combines propositional logic with arithmetic reasoning. Using SMT we can create a formula that is only satisfied by paths with a certain minimal probability. This enables us to apply binary search to find paths first whose probability measures differ from the probability of the most probable path of the current unrolling depth by at most a constant $\varepsilon > 0$. Furthermore, this formulation allows us to compute counterexamples for Markov reward models.

Since often even counterexamples of minimal size are too large to be useful, we minimize the DTMC or MRM before determining a counterexample. The counterexample obtained for the minimized system is much more compact since all equivalent evidences of the original system are merged into a single path of the minimized system. Given a counterexample of the minimized system, it can easily be translated back to the original system – either resulting in an ordinary counterexample or in one representative evidence per equivalence class.

We first describe the SMT-based BMC formula and compare it to the SAT-based approach of [7]. Then we show how to handle Markov reward models with an arbitrary number of reward functions. Finally we demonstrate how minimization can be applied to make the counterexample generation more efficient.

3.1 SMT-Based and SAT-Based SBMC

Stochastic bounded model checking (SBMC) as proposed in [7] is based on the formulation as a (propositional) satisfiability problem that a path of a certain length exists which exhibits a given property. To handle formulae of the form $\mathcal{P}_{\leq p}(a \,\mathcal{U}\, b)$, states that satisfy either $\neg a$ or b are made absorbing by removing all out-going edges. This reduces the problem to reaching a b-state.

After this preprocessing step, SBMC uses a SAT-solver to determine satisfying solutions for the BMC formula, which has the following structure:

$$\mathrm{SBMC}(k) := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T_{\mathrm{SAT}}(s_i, s_{i+1}) \wedge L_b(s_k). \tag{1}$$

$I(s_0)$ is a formula that is satisfied iff the assignment of s_0 corresponds to the initial state s_I. Accordingly, $T_{\mathrm{SAT}}(s_i, s_{i+1})$ represents the transition from a state s_i to a successor s_{i+1}, such that $T_{\mathrm{SAT}}(s_i, s_{i+1})$ is satisfied iff $P(s_i, s_{i+1}) > 0$, and $L_b(s_k)$ is the property which holds for s_k iff $b \in L(s_k)$. Each satisfying assignment of formula (1) corresponds to a path of length k that is an evidence for $a \,\mathcal{U}\, b$. The authors of [7] describe how this formula can efficiently be constructed from a BDD-based representation of the Markov chain. First they construct an

OBDD for the underlying directed graph by mapping positive probabilities in the MTBDD representation of the probability matrix to 1. Then they apply Tseitin-transformation [15] to the OBDDs to construct the different parts of the BMC formula. Fig. 1 gives a rough overview of the BMC procedure for a DTMC $M = (S, s_I, P, L)$ and a formula $\varphi = \mathcal{P}_{\leq p}(a \,\mathcal{U}\, b)$.

The probability measure of a path, however, is not considered within this formula. The probability measure has to be computed after the path has been found. Using an SMT-based formulation we can create a modified version of the BMC formula that allows us to put restrictions on the probability measure of evidences.

We define an extended transition formula $T_{\mathrm{SMT}}(s_i, s_{i+1}, \hat{p}_i)$ that is satisfied iff $P(s_i, s_{i+1}) > 0$ and the variable $\hat{p}_i$ is assigned the logarithm of this probability. Following [6], the logarithm is used in order to turn the multiplication of the probabilities along a path into a summation. This leads to SMT formulae over linear real arithmetic that can be decided efficiently. This variant of the transition predicate can also be generated from an MTBDD representation of the matrix $P(s, s')$ of transition probabilities using Tseitin-transformation. In contrast to the propositional case, where 'true' and 'false' are used as atomic formulae for the terminal nodes, we use '$\hat{p} = \log v$' for leaves with value $v > 0$ and 'false' for the leaf 0.

```
procedure Counterexample
    preprocess(M, φ);
    C ← ∅;
    k ← 0;
    Pr(C) ← 0;
    φ ← BMCformula(k, M, φ);
    while Pr(C) ≤ p do
        solution ← solve(φ)
        if solution = UNSAT then
            k ← k + 1;
            φ ← BMCformula(k, M, φ);
        else
            π ← CreatePath(solution);
            C ← C ∪̇ {π};
            Pr(C) ← Pr(C) + Pr(π);
            φ ← φ ∧ ExcludePath(π);
        end if
    end while
    return C
end procedure
```

Fig. 1. Counterexample generation

To let the solver find only evidences with a probability measure of at least $p_t \in [0, 1]$, we add the condition $\sum_{i=0}^{k-1} \hat{p}_i > \log p_t$ to the BMC formula:

$$\mathrm{SSBMC}(k) := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T_{\mathrm{SMT}}(s_i, s_{i+1}, \hat{p}_i) \wedge L_b(s_k) \wedge \left(\sum_{i=0}^{k-1} \hat{p}_i \geq \log p_t \right). \quad (2)$$

This formula is given to an SMT-solver. If the solver returns SAT, the satisfying assignment represents an evidence with a probability measure of at least p_t. If we get UNSAT, there is no such path of the current length.

This enables us to do a binary search for evidences with a high probability measure in $\mathrm{Path}_{s_I}^{\mathrm{fin}}$. In principle we could determine the most probable path of the current unrolling depth first, then the one with the second highest probability, and so on. For efficiency reasons we apply a different strategy: First, we look for paths which already exceed our probability threshold p. If this fails, we search

for paths with a probability greater or equal to $p/2$. If we have found all existing paths with such a probability and the accumulated probability mass is still less than p, we start looking for paths with a probability greater or equal $p/4$, and so on. The value p_t is decreased, until either all paths with length k have been found or the accumulated probability of the set of evidences is high enough. If the latter is not the case, we proceed to depth $k+1$.

The optimizations for SBMC – exploiting loops and improvements of the clauses to exclude paths from the search space –, which are proposed in [7], work for SSBMC as well and without further modification.

3.2 Counterexamples for MRMs

With the proposed method, we are not only able to handle DTMCs, but to handle MRMs as well. To consider the reward structure of an MRM during the search for paths, we need to integrate the rewards into the transition relation. In the preprocessing step, which is needed to turn the PCTL-Until property $\mathcal{P}_{\leq p}(a\,\mathcal{U}^{\mathcal{I}}\,b)$ into a reachability property, we must not cut the transitions from all b-states. There must be the possibility to extend a path if its accumulated reward is not within $\mathcal{I}$. Thus we cut the transitions only from states s with $a \notin L(s)$.

After that we extend the transition formula by a formula for the rewards in a similar way as we have done for the probabilities. For each time frame $0 \leq i < k$ we introduce a variable $\hat{r}_i$ such that the formula $R(s_i, \hat{r}_i)$ is satisfied iff $\hat{r}_i$ carries the value $\mathbf{R}(s_i)$. This formula can be created from an MTBDD representation of the reward function using Tseitin-transformation. The resulting SMT-formula, which takes rewards into account, has the following structure:

$$\text{R-SSBMC}(k) := \text{SSBMC}(k) \wedge \bigwedge_{i=0}^{k-1} R(s_i, \hat{r}_i) \wedge \left[\min(\mathcal{I}) \leq \left(\sum_{i=0}^{k-1} \hat{r}_i \right) \leq \max(\mathcal{I}) \right]. \tag{3}$$

Since b-states are no longer absorbing when using this formula, we have to make sure that we do not find paths of which we have already found a proper prefix. This can be guaranteed by adding clauses to the formula that exclude all paths that were found in previous iterations.

Using this technique it is possible to handle an arbitrary number of reward functions at the same time. We just add distinct variables for each reward function and build several reward sums which are checked against the corresponding intervals.

3.3 Bisimulation Minimization

We can use bisimulation minimization (cf. Sec. 2.2) as a further preprocessing step after cutting unnecessary transitions, but before constructing a counterexample. Since in many cases the quotient system is considerably smaller than the original system, fewer paths are needed for a counterexample.

Every path in the bisimulation quotient represents a set of paths in the original system. To be more precise, let $\pi, \pi' \in \mathrm{Path}^{\mathrm{fin}}(s)$ be two evidences in a DTMC $M = (S, s_I, P, L)$. They are equivalent if $|\pi| = |\pi'|$ and for all $0 \leq i \leq |\pi|$: $\pi^i \sim \pi'^i$. All equivalent paths correspond to the same path in the quotient system, namely to the path $\pi_Q = [\pi^0]_\sim [\pi^1]_\sim \dots [\pi^{|\pi|}]_\sim$. The probability of π_Q is the sum of the probabilities of the represented original paths that start in s_I. Therefore in general fewer paths are needed in the quotient system for a counterexample.

Once a counterexample has been determined for the quotient DTMC, its paths can be translated back to the original system. The result is a tree that contains all evidences that are stepwise equivalent to the given path. Let us assume that $\pi_Q = C_0 C_1 \dots C_n$ with $C_0 = [s_I]_\sim$ is such a path in the quotient system. We set $\mathrm{succ}(s) = \{s' \in S \mid P(s, s') > 0\}$. The root of the tree is the initial state s_I that corresponds to C_0 on π_Q. If s_i is a node of the tree that corresponds to C_i on π_Q, its successor nodes in the tree are $\mathrm{succ}(s) \cap C_{i+1}$. They correspond to C_{i+1}. The probability measures of the resulting tree and of π_Q coincide.

In the next section we show the effectiveness of SMT-based counterexample generation and of this optimization on a set of example benchmarks.

4 Experimental Results

We have implemented the SSBMC-tool in C++ with the refinements we presented above, including the optimizations from [7]. We used YICES [16] as the underlying SMT-solver.

Our benchmarks are instances of the following four case studies:

(1) The *contract signing protocol* [17,18] (`contract`) provides a fair exchange of signatures between two parties A and B over a network. If B has obtained the signature of A, the protocol ensures that A will receive the signature of B as well. In our experiments we examine the violation of this property.

(2) The task of the *crowds protocol* [19] (`crowds`) is to provide a method for anonymous web browsing. The communication of a single user is hidden by routing it randomly within a group of similar users. The model contains corrupt group members who try to discern the source of a message. We explore the probability that these corrupt members succeed.

(3) The *leader election protocol* [20] (`leader`) consists of N processors in a synchronous ring. Every processor chooses a random number from $\{0, \dots, M\}$. The numbers are passed around the ring, the processor with the highest number becomes the leader if this number is unique. If this fails, a new round starts. We provide a certificate that the probability to eventually elect a leader exceeds a given bound.

(4) The *self-stabilizing minimal spanning tree algorithm* [21] (`mst`) computes a minimal spanning tree in a network with N nodes. Due to, e. g., communication failures the system may leave the state where a result has been computed, and recover after some time. For our experiments we explore the probability that the algorithm does not recover within a given number of k steps. This model

is of particular interest to us, because it has a large number of paths, but only a few have a notable probability mass. Because SAT-based BMC does not find paths with high probability first, the hope is that the SMT approach finds much smaller counterexamples in less time.

We used the probabilistic model checker PRISM [22] to generate symbolic representations of these case studies. All experiments were run on a Dual Core AMD Opteron processor with 2.4 GHz per core and 4 GB of main memory. Any computation which took more than two hours ("– TO –") or needed more than 2 GB of memory ("– MO –") was aborted.

In order to compare the results we ran the same benchmarks under the same conditions also with the SBMC-tool from [7].

4.1 Counterexamples for DTMCs

In this section we present the results for the SSBMC procedure. The evaluation of bisimulation minimization and counterexamples for MRMs follows in subsequent sections.

Table 1 contains the results for counterexample generation for DTMCs using the SMT- and the SAT-approach. The first and the second column contain the names of the model and the probability threshold p. In the third column the maximal unrolling depth $k_{\max}$ is listed. The blocks titled "SSBMC" and "SBMC" present the results for SMT-based approach and the SAT-based approach, respectively. The columns "#paths", "time", and "memory" contain the number of paths in the counterexample, the needed computation time in seconds, and the memory consumption in megabytes.

Both tools are able to solve a subset of the instances within the given limits. The SSBMC-tool was aborted due to the memory limit for one instance of the `contract` benchmark, most instances of the `crowds` benchmark, and some instances of the `leader` protocol. The reason is the high memory consumption of the SMT-solver Yices compared to the SAT-solver MiniSAT, which is used in SBMC. The `contract` and `leader` benchmarks exhibit the property that all evidences of the same length have the same probability. Therefore the number of paths coincide for both tools. The running time of SSBMC for these instances is slightly higher due to the modified binary search (cf. Sec. 3.1) which increases the number of calls to the solver. The search strategy is also the reason why SSBMC may need slightly more evidences than SBMC, which is the case here for `crowds02_07`, where the probabilities of the different evidences differ only by a small amount.

Notable are the `mst` instances. They contain a large number of evidences with very different probabilities. SBMC is not able to compute a counterexample within the given time limit, while SSBMC returns less than 5000 paths in under 3 minutes. A more detailed look showed that SBMC had computed 633 729 paths for `mst-14` before the timeout occurred. This is because the SMT-approach is able to compute the more probable evidences first, only few of which are needed for a counterexample.

Table 1. Counterexample generation for DTMCs using SMT- and SAT-based BMC

			SSBMC			SBMC		
Name	p	$k_{\max}$	#paths	time	mem.	#paths	time	mem.
contract05_03	0.500	37	513	14.85	124.72	513	25.04	74.84
contract05_08	0.500	87	513	134.92	889.32	513	399.08	694.71
contract06_03	0.500	44	2049	70.21	388.81	2049	140.36	124.79
contract06_08	0.500	92		– MO –		2049	2620.12	1181.76
contract07_03	0.500	51	8193	474.59	1510.28	8193	627.56	198.27
crowds02_07	0.100	39	21306	1006.05	1394.26	21116	417.89	96.11
crowds04_06	0.050	34		– MO –		80912	1468.94	278.17
crowds04_07	0.050	34		– MO –		80926	1773.83	286.38
crowds05_05	0.050	36		– MO –			– MO –	
crowds10_05	0.025	27		– MO –		52795	1654.83	188.51
leader03_04	0.990	8	276	0.70	27.45	276	0.76	14.55
leader03_08	0.990	8	1979	28.21	101.41	1979	25.76	66.48
leader04_03	0.990	20		– MO –		347454	3959.88	720.56
leader05_02	0.900	42		– MO –			– MO –	
leader06_02	0.900	84		– MO –			– TO –	
mst14	0.990	14	426	11.64	42.99		– TO –	
mst15	0.049	15	4531	98.58	148.82		– TO –	
mst16	0.047	16	4648	107.27	158.25		– TO –	
mst18	0.036	18	4073	109.26	164.24		– TO –	
mst20	0.034	20	452	19.57	58.21		– TO –	

4.2 Bisimulation

In Table 2 we evaluate the impact of bisimulation minimization on running time and memory consumption for SSBMC and SBMC. In Table 3 we show the results of bisimulation with subsequent back-translation of the abstract evidences. In the latter case, we converted the minimized paths completely, i. e., we obtained *all* original paths which were represented by an abstract path. The running time listed in Table 2 and Table 3 include the time for bisimulation minimization, counterexample generation, and, in Table 3, path conversion. Again, the block titled "SSBMC" ("SBMC") contains the result for the SMT-based (SAT-based) approach.

As we can see, bisimulation minimization causes in most cases a significant decrease in computation time and required memory of SSBMC and SBMC. This effect is somewhat alleviated when the paths are translated back to the original system, although not dramatically. Some instances of the contract and the `crowds` protocol which could not be solved by SSBMC within the given time and memory bounds become actually solvable, even with path conversion.

However, there is also an exception to this trend: The path conversion for the minimal spanning tree benchmark cannot be done within the given memory bounds. This is due to the fact that one abstract path in these benchmarks represents a great number of original paths, too many to convert them all. While the SMT-based approach without bisimulation minimization can pick the paths

Table 2. Counterexample generation with bisimulation minimization

			SSBMC			SBMC		
Name	p	$k_{\max}$	#paths	time	mem.	#paths	time	mem.
contract05_03	0.500	37	7	23.41	61.42	7	48.50	54.55
contract05_08	0.500	87	7	467.88	179.29	7	829.43	94.72
contract06_03	0.500	44	8	57.75	84.94	8	144.06	64.56
contract06_08	0.500	97	8	1205.37	209.47	8	2213.94	120.83
contract07_03	0.500	51	9	169.37	123.93	9	407.63	79.13
crowds02_07	0.100	39	21069	629.24	633.34	21279	191.31	101.34
crowds04_06	0.050	34	3459	106.17	164.95	3624	44.19	48.76
crowds04_07	0.050	34	3459	123.32	167.61	3555	46.97	50.67
crowds05_05	0.050	36	6347	184.06	251.70	8435	55.20	50.47
crowds10_05	0.025	27	251	12.74	71.20	347	10.22	47.84
leader03_04	0.990	8	2	0.12	21.68	2	0.10	12.06
leader03_08	0.990	8	2	0.64	33.93	2	0.68	24.31
leader04_03	0.990	20	4	0.31	25.12	4	0.32	15.50
leader05_02	0.900	42	7	0.24	22.94	7	0.24	12.05
leader06_02	0.900	84	12	0.79	26.19	12	0.89	14.09
mst14	0.990	14	9	2.28	38.10	396	0.78	16.85
mst15	0.049	15	13	1.85	39.36	1648	1.40	17.92
mst16	0.047	16	13	2.19	39.73	5632	4.00	25.99
mst18	0.036	18	9	3.57	43.64	27475	30.82	69.69
mst20	0.034	20	7	5.02	44.91	20290	25.83	56.63

with the highest probability, leading to a small counterexample, this is not possible after bisimulation minimization. If we compute the most probable paths in the minimized system, they can correspond to huge numbers of paths in the original system each of which has only negligible probability.

4.3 Rewards

For the reward model checking we integrated rewards into the `leader` election protocol. A reward of 1 is granted whenever a new round starts, i. e., when each processor chooses a new ID. We want to analyze how high the probability measure is that at least three rounds are needed until a leader has been elected. For the experiments we restricted our search to a maximal path length of $\text{depth}_{\max}$. We computed all paths with the given property up to this length.

The results are shown in Table 4. The first column contains the name of the model, the second the maximal depth $\text{depth}_{\max}$. The subsequent columns contain the accumulated probability measure p, the number of found paths, the computation time (in seconds) and the required memory (in megabytes).

Compared to the results in Section 4.1 we need more and longer paths to get a noteworthy probability measure. The computation time and the amount of consumed memory are higher accordingly.

We also integrated bisimulation minimization for Markov reward models with state rewards. In this case only an appropriate initial partition has to be provided

Table 3. Counterexample generation with bisimulation and path conversion

			SSBMC			SBMC		
Name	p	$k_{\max}$	#paths	time	mem.	#paths	time	mem.
contract05_03	0.500	37	520	27.87	72.87	520	50.86	54.36
contract05_08	0.500	87	520	859.34	182.64	520	1259.94	98.06
contract06_03	0.500	44	2064	72.27	91.44	2064	168.28	75.50
contract06_08	0.500	97	2064	4181.45	224.31	2064	5927.5	131.68
contract07_03	0.500	51	8224	230.69	149.60	8224	450.20	103.13
crowds02_07	0.100	39	21069	812.51	699.80	21279	313.99	168.42
crowds04_06	0.050	34	81227	408.69	406.16	81138	218.81	289.62
crowds04_07	0.050	34	81227	426.69	409.29	80705	221.80	290.00
crowds05_05	0.050	36		– MO –			– MO –	
crowds10_05	0.025	27	54323	198.30	194.38	53507	119.83	168.93
leader03_04	0.990	8	300	0.21	21.68	300	0.16	12.06
leader03_08	0.990	8	4536	4.12	33.93	4536	3.67	24.31
leader04_03	0.990	20	583440	483.99	1123.74	583440	300.24	1108.83
leader05_02	0.900	42		– MO –			– MO –	
leader06_02	0.900	84		– MO –			– MO –	
mst14	0.990	14		– MO –			– MO –	
mst15	0.049	15		– MO –			– MO –	
mst16	0.047	16		– MO –			– MO –	
mst18	0.036	18		– MO –			– MO –	
mst20	0.034	20		– MO –			– MO –	

Table 4. Counterexample generation for MRMs using SMT-based BMC

Model	$\text{depth}_{\max}$	p	#paths	time	mem.
leader03_04	23	0.00391	20160	290.62	434.45
leader03_05	19	0.00160	18000	290.16	379.25
leader03_06	19	0.00077	52920	2134.05	1147.73
leader03_08	15	0.00024	32256	1050.96	709.98
leader04_02	25	0.21875	37376	912.11	1110.54
leader04_03	19	0.04979	26460	589.94	761.34
leader05_02	23	0.14771	4840	40.16	163.06
leader06_02	25	0.12378	32448	907.33	1360.11
leader08_02	28		– MO –		

for bisimulation computation. The results are shown in Table 5. For the `leader` benchmarks bisimulation minimization results in a enormous compression of the state space and a respective reduction of the number of evidences. Since the back-translation can be done efficiently and yields for the `leader` benchmark the same counterexample as the version without minimization, using bisimulation minimization as a preprocessing and back-translation as a postprocessing step reduces the overall computation time and memory consumption.

Table 5. Counterexample generation for MRMs with bisimulation minimization

		without conv.				with conv.			
Model	$\text{depth}_{\max}$	p	#paths	time	mem.	p	#paths	time	mem.
leader03_04	23	0.00391	3	0.15	21.70	0.00391	20160	9.37	59.36
leader03_05	19	0.00160	2	0.25	24.84	0.00160	18000	10.68	54.93
leader03_06	19	0.00077	2	0.38	26.96	0.00077	52920	34.30	119.10
leader03_08	15	0.00024	1	0.66	33.96	0.00024	32256	25.95	69.56
leader04_02	25	0.21875	3	0.11	21.29	0.21875	37376	20.43	92.57
leader04_03	19	0.04979	1	0.29	25.15	0.04979	26460	18.55	72.67
leader05_02	23	0.14771	1	0.18	22.12	0.14771	4840	3.23	31.45
leader06_02	25	0.12378	1	0.30	23.74	0.12378	32448	31.71	87.96
leader08_02	28	0.05493	1	0.98	31.33		– MO –		

5 Conclusion

In our paper we showed how SMT and BMC can be combined to efficiently generate counterexamples for DTMCs. Our SSBMC method can handle systems which could not be handled with the previously presented SAT-based approach [7]. With SSBMC it is also possible to analyze Markov reward models with an arbitrary number of reward functions. This enables us to refute reachability properties which impose restrictions on the accumulated reward of paths.

Furthermore we presented bisimulation minimization as a preprocessing step for SSBMC. It reduces the number of evidences needed for a counterexample by merging equivalent paths. This way the counterexample generation is accelerated and the memory consumption is reduced. We are able to convert these minimized paths back to the original ones.

As future work we will carry out a more detailed experimental evaluation of our methods on appropriate models. Furthermore, there are still many possible optimizations for our tool. So far, reward model checking and bisimulation minimization only work without the loop detection optimization from [7]. These combinations have to be implemented.

We plan to optimize the search for paths with higher probabilities. We want to include the BDD-based method from [23], which applies Dijkstra's shortest path algorithm to compute the most probable evidences, into our tool as another preprocessing step. The advantage of this method is that it yields counterexamples of minimal size. Preliminary experiments have shown that this method is efficient as long as the number of paths is small. Since the BDD sizes grow with each path that has been found, memory consumption and running time grow accordingly. We want to combine this approach with the SMT-approach by using the BDD-based method as long as it is efficient and switch to the SMT-approach when the BDD-approach becomes too expensive.

References

1. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press, Cambridge (2008)
2. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers 35(8), 677–691 (1986)

3. Clarke, E.M., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods in System Design 19(1), 7–34 (2001)
4. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6(5), 512–535 (1994)
5. Andrés, M.E., D'Argenio, P., van Rossum, P.: Significant diagnostic counterexamples in probabilistic model checking. In: Chockler, H., Hu, A.J. (eds.) HVC 2008. LNCS, vol. 5394, pp. 129–148. Springer, Heidelberg (2009)
6. Han, T., Katoen, J.P., Damman, B.: Counterexample generation in probabilistic model checking. IEEE Trans. on Software Engineering 35(2), 241–257 (2009)
7. Wimmer, R., Braitling, B., Becker, B.: Counterexample generation for discrete-time markov chains using bounded model checking. In: Jones, N.D., Müller-Olm, M. (eds.) VMCAI 2009. LNCS, vol. 5403, pp. 366–380. Springer, Heidelberg (2009)
8. Aljazzar, H., Leue, S.: Directed explicit state-space search in the generation of counterexamples for stochastic model checking. IEEE Trans. on Software Engineering 36(1), 37–60 (2010)
9. Ábrahám, E., Jansen, N., Wimmer, R., Katoen, J.P., Becker, B.: DTMC model checking by SCC reduction. In: Proc. of QEST, IEEE CS, pp. 37–46 (2010)
10. Nelson, G., Oppen, D.C.: Simplification by cooperating decision procedures. ACM Transactions on Programming Languages and Systems (TOPLAS) 1, 245–257 (1979)
11. Han, T., Katoen, J.-P.: Counterexamples in probabilistic model checking. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 72–86. Springer, Heidelberg (2007)
12. Katoen, J.-P., Kemna, T., Zapreev, I., Jansen, D.N.: Bisimulation Minimisation Mostly Speeds Up Probabilistic Model Checking. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 87–101. Springer, Heidelberg (2007)
13. Derisavi, S.: Signature-based symbolic algorithm for optimal Markov chain lumping. In: Proc. of QEST, IEEE CS, pp. 141–150 (2007)
14. Wimmer, R., Derisavi, S., Hermanns, H.: Symbolic partition refinement with automatic balancing of time and space. Perf. Eval. 67(9), 815–835 (2010)
15. Tseitin, G.S.: On the complexity of derivation in propositional calculus. Studies in Constructive Mathematics and Mathematical Logic Part 2, 115–125 (1970)
16. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 81–94. Springer, Heidelberg (2006)
17. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Communications of the ACM 28(6), 637–647 (1985)
18. Norman, G., Shmatikov, V.: Analysis of probabilistic contract signing. Journal of Computer Security 14(6), 561–589 (2006)
19. Reiter, M., Rubin, A.: Crowds: Anonymity for web transactions. ACM Transactions on Information and System Security (TISSEC) 1(1), 66–92 (1998)
20. Itai, A., Rodeh, M.: Symmetry breaking in distributed networks. Information and Computation 88(1), 60–87 (1990)
21. Collin, Z., Dolev, S.: Self-stabilizing depth-first search. Information Processing Letters 49(6), 297–301 (1994)
22. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Hermanns, H. (ed.) TACAS 2006. LNCS, vol. 3920, pp. 441–444. Springer, Heidelberg (2006)
23. Günther, M., Schuster, J., Siegle, M.: Symbolic calculation of k-shortest paths and related measures with the stochastic process algebra tool CASPA. In: Int'l Workshop on Dynamic Aspects in Dependability Models for Fault-Tolerant Systems (DYADEM-FTS), pp. 13–18. ACM Press, New York (2010)