SMT-based Counterexample Generation for Markov Chains*

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

Nils Jansen Erika Ábrahám
RWTH Aachen, Germany
{jansen,abraham}@informatik.rwth-aachen.de

Abstract

Counterexamples are a highly important feature of 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 the reals for the BMC procedure. SMT-solving, extending SAT with theories, leads in this context on the one hand to a convenient way to express conditions on the probability of certain paths and on the other hand to handle Markov Reward models. The former can be used 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. We focus on *model checking*, where systems are checked automatically against certain properties (see e. g. [1]). In many cases model checking also provides helpful diagnostic information, i. e., in case of a defective system a witnessing run is given, a *counterexample*. 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 many systems were even symbolic methods fail. In order to restrict the search for errors in a system, *bounded model checking* (BMC) was introduced [3], where a path of fixed length to a property-refuting state is formulated as a satisfiability (SAT) problem. As the size of an corresponding formula is relative small and as modern SAT solvers have strongly evolved, it seems not surprising that this approach was very successful.

To model real life scenarios it is often necessary to model specific uncertainties using probabilities. For instance, properties can be formulated in *probabilistic computation tree logic* (PCTL) [4] for a model called *discrete-time Markov chains* (DTMCs). The classical model checking approach for this setting, to which we refer as *probabilistic model checking* in what follows, is based on the solution of a linear equation system [4]. Tools like PRISM [5] and MRMC [6] use efficient implementations of this procedure but lack – for the sake of the way of computing – the generation of counterexamples.

*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 "CEBug – CounterExample Generation for Stochastic Systems using Bounded Model Checking"

To provide a user with such diagnostic information for probabilistic systems, there were recent developments during the last years [7, 8, 9, 10, 11]. 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. There are different ways to on the one hand *reduce the size* of such sets of paths or on the other hand to give *compact representations*: Incrementally adding the paths with the highest probability [8, 10] or reducing the strongly connected components of the underlying digraph of a DTMCs [7, 11], and using regular expressions to describe whole sets of counterexamples [8].
In [9], bounded model checking (BMC) for probabilistic systems was introduced. This procedure can be used to refute probabilistic reachability problems, which are intuitively of the form "The probability of reaching a state that violates the property is at most p". The refutation is shown by using a SAT-solver to find paths that altogether exceed the probability 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 built up sets of paths which differ only in the number of unfolding 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 advantageous to start with paths of high probability. Moreover, it is desirable to use this procedure for Markov models with cost functions, so-called Markov reward models.
In this paper we extend stochastic BMC in order to handle these problems. Instead of using a SAT-solver, we use the more powerful approach of SMT-solving. This allows to express propositional formulae representing paths to target states together with conditions on the probability of such a path. Furthermore, real numbers that are allocated to the states by a cost-function, can be added up, and properties for these costs can be checked.

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

2. Foundations

In this section we will 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** $\Psi_s = (\mathrm{Path}_s^{\inf}, \Delta_s, \Pr_s)$ such that*

- *Δ_s is the smallest σ-algebra generated by $\mathrm{Path}_s^{\inf}$ 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 $\Delta(\pi) = \{\lambda \in \mathrm{Path}_s^{\inf} \mid \lambda{\uparrow}^{|\pi|} = \pi\}$.*
- *$\Pr_s$ is the uniquely defined probability measure that satisfies the following constraints: $\Pr_s(\mathrm{Path}_s) = 1$ and for all basic cylinders $\Delta(ss_1s_2 \dots s_n)$:*

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

The properties we want to consider are formulated in PCTL [4] and are of the form $\mathcal{P}_{\leq p}(a \mathsf{U} b)$ 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 or equal to p. More formally: A path π satisfies $a \mathsf{U} b$, 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 $s \in S$ satisfies the formula $\mathcal{P}_{\leq p}(a \mathsf{U} b)$ (written $s \vDash \mathcal{P}_{\leq p}(a \mathsf{U} b)$) iff $\Pr_s\big(\{\pi \in \mathrm{Path}_s^{\inf} \mid \pi \vDash a \mathsf{U} b\}\big) \leq p$.
Let us assume that such a formula $\mathcal{P}_{\leq p}(a \mathsf{U} b)$ is violated by a DTMC M. That means, the probability measure of the set of paths that start in the initial state of M and that satisfy $a \mathsf{U} b$ is larger than p. In this case we want to compute a counterexample which testifies that the formula is indeed violated. Hence a counterexample is a set of paths that satisfy $a \mathsf{U} b$ 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 $\varphi = \mathcal{P}_{\leq p}(a \mathsf{U} b)$ is violated in state s_I. An **evidence** for φ is a finite path $\pi \in \mathrm{Path}_{s_I}^{\mathrm{fin}}$ such that $\pi \vDash a \mathsf{U} b$, 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 $\mathcal{P}_{\leq p}(a \mathsf{U} b)$ is violated [12]. In order to be able to express properties like *"The probability to reach a target state with costs larger than c is smaller than 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 $\boldsymbol{R} : S \to \mathbb{R}^{\geq 0}$ a real-valued reward function.*

Rewards can be used, e. g., to model costs of certain operations, to count steps or to measure given gratifications. We assume that there are no negative rewards. 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 $I \subseteq \mathbb{R}^{\geq 0}$, a path π satisfies the property $a \mathsf{U}^I b$, written $\pi \vDash a \mathsf{U}^I b$, if there is $i \geq 0$ such that $b \in L(\pi^i)$, $\mathbf{R}(\pi{\uparrow}^i) \in I$ and for all $0 \leq j < i$ the condition $a \in L(\pi^j)$ is satisfied. A state $s \in S$ satisfies $\mathcal{P}_{\leq p}(a \mathsf{U}^I b)$ if $\Pr_s\big(\{\pi \in \mathrm{Path}_s^{\inf} \mid \pi \vDash a \mathsf{U}^I b\}\big) \leq p$.
Our goal is to compute counterexamples to falsify such reward properties using bounded model checking (BMC).

2.2. Bisimulation Minimization

For the generation of counterexamples we will 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. Often the representation is smaller by

orders of magnitude. Using algorithms whose runtime 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 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:

Definition 5 *Let $M = (S, s_I, P, L)$ be a DTMC and $\boldsymbol{R}_s : S \to \mathbb{R}^{\geq 0}$ a reward function for M. A partition $\boldsymbol{P}$ of S is a **bisimulation** if, for all $s, t \in S$ such that s and t are contained in the same block of $\boldsymbol{P}$ and all blocks C of $\boldsymbol{P}$, the following holds:*

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

*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 class of a state $s \in S$ w. r. t. bisimilarity is denoted by $[s]_\sim$. These classes become now 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 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, 15].

3. SMT-based Bounded Model Checking for Counterexample Generation

In this section we will show how counterexamples can be computed using an SMT-based formulation of bounded model checking (BMC). BMC has already been applied in [9] for this purpose but with a purely propositional formulation. 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.
Instead of SAT we now want to use SMT. The satisfiability modulo theories problem (SMT) is a generalization of the propositional satisfiability problem (SAT). It combines propositional logic with arithmetic reasoning. Thus the SMT formulation that we propose in this section becomes much more powerful than the propositional formulation. We can create a formula that is only satisfied by paths with a certain minimal probability. Then we can 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 enables 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 will first describe the SMT-based BMC formula and compare it to the SAT-based approach of [9]. Then we will show how to handle Markov reward models with an arbitrary number of reward functions. Finally we will demonstrate how minimization can be applied make the counterexample generation more efficient.

3.1. SMT-based and SAT-based SBMC

Stochastic bounded model checking (SBMC) as proposed in [9] is based on the formulation that a path of a certain length exists which exhibits a given property as a (propositional) satisfiability problem. To handle formulae of the form $\mathcal{P}_{\leq p}(a \mathsf{U} b)$, 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 an evidence of length k for reaching a state that ends in a b-state. Because of the preprocessing step, all such paths are also evidences for $a \mathsf{U} b$. The authors of [9] describe how this formula can efficiently be constructed from a symbolic representation of the Markov chain by applying Tseitin-transformation [16] to the OBDD representation of the transition relation.
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. The logarithm is used in order to turn the multiplication of the probabilities along a path into a summation, leading to SMT formulae that can be decided more efficiently. This variant of the transition predicate can be generated from an MTBDD representation of the matrix $P(s, s')$ of transition probabilities using Tseitin-transformation. In contrast to the propositional case, not 'true' and 'false' are used as atomic formulae for the terminal nodes, but '$\hat{p} = \log v$' for leaves with value $v > 0$ and 'false' for the leaf 0.
To let the solver find only evidences with a probability measure of at least p_t, we add the condition $\sum_{i=0}^{i-1} \hat{p}_i > \log p_t$ to the BMC-formula:

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

This formula is given to an SMT-solver for linear arithmetic. If the solver returns SAT, the satisfying assignment represents a path with a probability measure higher than or equal to 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 $\mathrm{Path}^{\mathrm{fin}}_{s_I}$. In principle we could determine the most probable path of the current unrolling depth first, then the one with the second highest probability ... 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 $\frac{p}{2}$. 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 $\frac{p}{4}$, 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 [9], work for SSBMC as well.

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 include the rewards into the transition relation. In the preprocessing step, which is needed to turn the PCTL-Until property $\mathcal{P}_{\leq p}(a \mathsf{U}^I b)$ into a reachability property, we may not cut the transitions from all b-states. There must be the possibility to extend a path if its cumulated reward is too small. Thus we cut the transitions only from states s with $a \notin L(s)$.
After that we extend the transition formula in a similar way as we have done for the probabilities. For each timeframe $0 \leq i < k$ we introduce a variable $\hat{r}_i$ such that the formula $T_{\text{R-SMT}}(s_i, s_{i+1}, \hat{p}_i, \hat{r}_i)$ is satisfied iff $P(s_i, s_{i+1}) > 0$, $\hat{p}_i$ is assigned $\log P(s_i, s_{i+1})$, and 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:

$$\begin{aligned} \text{R-SSBMC}(k) := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T_{\text{R-SMT}}(s_i, s_{i+1}, \hat{p}_i, r_i) \wedge L_b(s_k) \\ \wedge \left[\left(\sum_{i=0}^{k-1} \hat{p}_i \right) \geq \log p_t \right] \wedge \left[\min(I) \leq \left(\sum_{i=0}^{k-1} r_i \right) \leq \max(I) \right]. \end{aligned} \tag{3}$$

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 our 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

We use bisimulation minimization (cf. Sec. 2.2) as a further preprocessing step after cutting unnecessary transitions, but before constructing a counterexample. Since in most cases the quotient system is considerable 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 can now walk through π_Q and create the original paths. We start in C_0 with the initial state s_I of the original DTMC. We collect all possible successor nodes of s_I in C_1, then we look for the successors of these states in the next block, until we reach the final block C_n: Assume $s \in S$ is a node of the tree that corresponds to C_i $(i < n)$ of the path in the quotient DTMC. The successor nodes of s are the states $C_{i+1} \cap \{t \in S \mid P(s,t) > 0\}$. They correspond to C_{i+1}.
In the next section we will 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 [9]. We used Yices [17] as the underlying SMT-solver.
Our benchmarks are instances of the following four case studies:
(1) The *contract signing protocol* [18, 19] 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* [20] 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* [21] 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 look at the probability that a leader will finally be elected and provide a certificate for it.
(4) The *self-stabilizing minimal spanning tree algorithm* [22] 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 only a few of which have a notable probability measure. 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 [5] 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 needed more than 2 GB memory ("– MO –") was aborted.

4.1. Counterexamples for DTMCs

In this section we present the results for the SSBMC procedure, including the optimizations from [9]. The evaluation of bisimulation minimization and counterexamples for MRMs will follow in subsequent sections.
Table 1 contains the results for counterexample generation for DTMCs using the SMT-approach. The first and the second column contain the names of the model and the probability threshold p. In

Table 1: Counterexample generation for DTMCs using SMT-based BMC

			without bisim.			bisim. without conv.			bisim. with conv.		
Name	p	$k_{\max}$	#paths	time	mem.	#paths	time	mem	#paths	time	mem
contract05_03	0.500	37	513	14.85	124.72	7	23.41	61.42	520	27.87	72.87
contract05_08	0.500	87	513	134.92	889.32	7	467.88	179.29	520	859.34	182.64
contract06_03	0.500	44	2049	70.21	388.81	8	57.75	84.94	2064	72.27	91.44
contract06_08	0.500	97		– MO –		8	1205.37	209.47	2064	4181.45	224.31
contract07_03	0.500	51	8193	474.59	1510.28	9	169.37	123.93	8224	230.69	149.60
crowds02_07	0.100	39	21306	1007.12	1394.26	21069	719.34	633.34	21637	865.64	699.80
crowds04_06	0.050	34		– MO –		3459	106.17	164.95	81227	408.69	406.16
crowds04_07	0.050	34		– MO –		3459	123.32	167.61	81227	426.69	409.29
crowds05_05	0.050	36		– MO –		6347	184.06	251.70		– MO –	
crowds10_05	0.025	27		– MO –		251	12.74	7.20	54323	198.30	194.38
leader03_04	0.990	8	276	0.70	27.45	2	0.12	21.68	300	0.21	21.68
leader03_08	0.990	8	1979	28.21	101.41	2	0.64	33.93	4536	4.12	33.93
leader04_03	0.990	20		– MO –		4	0.31	25.12	583440	483.99	1123.74
leader05_02	0.900	42		– MO –		7	0.24	22.94		– MO –	
leader06_02	0.900	84		– MO –		12	0.79	26.19		– MO –	
mst14	0.990	14	426	11.64	42.99	9	2.28	38.10		– MO –	
mst15	0.049	15	4531	98.58	148.82	13	1.85	39.36		– MO –	
mst16	0.047	16	4648	107.27	158.25	13	2.19	39.73		– MO –	
mst18	0.036	18	4073	109.26	164.24	9	3.57	43.64		– MO –	
mst20	0.034	20	452	19.57	58.21	7	5.02	44.91		– MO –	

the third column the maximal unrolling depth $k_{\max}$ and in the fourth column the number of paths in the counterexample are listed. We now look at the first block titled "without bisim.". 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, respectively.
Most benchmarks are solved well within the given limits, except one instance of the contract, most instances of the crowds protocol and some instances of the leader protocol. We compared the results to those of the purely propositional formulation. While the runtimes for the contract, crowds, and leader benchmarks are comparable in spite of the binary search, the SAT-based approach fails for all instances of the mst benchmark due to the time limitation. The small impact of the binary search on the runtime is due to the incremental solving technique of Yices. Information learnt during one search can be exploited in subsequent calls to the solver. We also compared the memory consumption for those instances that both approaches were able to handle. The SMT-based method used more memory. The reason is the Yices solver, which uses more memory than the SAT-solver `minisat` that is used in SBMC.

4.2. Bisimulation

In Table 1 we also evaluate the impact of bisimulation minimization (columns titled "bisim. without conv.") and of bisimulation with path conversion ("bisim. with conv.") on the runtime and memory consumption. In the latter case, we converted the minimized paths completely, i. e., we obtained *all* original paths which were represented by a minimized path. The runtimes listed in Table 1 include the time for bisimulation minimization, counterexample generation, and, in the third block, conversion. As we can see, the bisimulation minimization causes a significant decrease in computation time and required memory. 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 the original method within the given time and memory bounds become actually solvable, even with path conversion.

Table 2: Counterexample generation for MRMs using SMT-based BMC

Model	$\mathrm{depth}_{\max}$	p	#paths	mem.	time
leader03_02	25	0.06226	360	29.61	1.00
leader03_03	25	0.01234	8640	210.75	75.52
leader03_04	23	0.00391	20160	434.45	290.62
leader03_05	19	0.00160	18000	379.25	290.16
leader03_06	19	0.00077	52920	1147.73	2134.05
leader03_08	15	0.00024	32256	709.98	1050.96
leader04_02	25	0.21875	37376	1110.54	912.11
leader04_03	19	0.04979	26460	761.34	589.94
leader05_02	23	0.14771	4840	163.06	40.16
leader06_02	25	0.12378	32448	1360.11	907.33

However, there is also an exception to this trend: The path conversion for the minimal spanning tree cannot be done within the given memory bounds. This is due to the fact that one minimized path in these benchmarks represents a great number of original paths, too many to convert them all. While the approach without bisimulation minimization can pick the paths 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 know how high the probability measure is if we consider all paths with a reward of 3 or higher up to a certain maximal length $\mathrm{depth}_{\max}$. The results are shown in Table 2.
The first column contains the name of the model, the second the maximal depth $\mathrm{depth}_{\max}$. The subsequent columns contain the accumulated probability measure p, the number of found paths, the required memory (in MB) and the computation time (in seconds).
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.
In our current implementation a combination of reward model checking with bisimulation is not possible yet. State rewards can be taken into account rather easily by using an appropriate initial partition. Transition rewards are more involved because they must be considered when splitting blocks. Compared to DTMCs, we expect a similar, but weaker effect for reward models since the bisimulation quotient is larger for MRMs than for the underlying DTMC.

5. Conclusion

In our paper we showed how SMT and BMC may 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 [9]. With SSBMC it is also possible to analyze Markov reward models with an arbitrary number of reward functions. This enables us to examine Markov reward models and to build counterexamples which take conditions on the cumulated reward of paths into account.
We presented bisimulation minimization as a preprocessing step for SSBMC. It reduces the number of needed evidences for a counterexample by merging equivalent paths. Thus 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 only works without bisimulation minimization and the loop detection optimization from [9]. 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 show 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 runtime grow accordingly. The idea is to use the BDD-based method as long as it is efficient and switch to the SMT-approach when the BDD-approach becomes to expensive.

References

[1] Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)
[2] Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers **35**(8) (1986) 677–691
[3] 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
[4] Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing **6**(5) (1994) 512–535
[5] Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Proc. of TACAS. Volume 3920 of LNCS, Springer (2006) 441–444
[6] Katoen, J.P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. In: 6th Proc. of QEST, IEEE CS (2009) 167–176
[7] Andrés, M.E., D'Argenio, P., van Rossum, P.: Significant diagnostic counterexamples in probabilistic model checking. In: Haifa Verification Conference. Volume 5394 of LNCS, Springer (2008) 129–148
[8] Han, T., Katoen, J.P., Damman, B.: Counterexample generation in probabilistic model checking. IEEE Trans. on Software Engineering **35**(2) (2009) 241–257
[9] Wimmer, R., Braitling, B., Becker, B.: Counterexample generation for discrete-time Markov chains using bounded model checking. In: Proc. of VMCAI. Volume 5403 of LNCS, Springer (2009) 366–380
[10] 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) (2010) 37–60
[11] Ábrahám, E., Jansen, N., Wimmer, R., Katoen, J.P., Becker, B.: DTMC model checking by SCC reduction. In: Proc. of QEST, IEEE CS (2010) 37–46
[12] Han, T., Katoen, J.P.: Counterexamples in probabilistic model checking. In: Proc. of TACAS. Volume 4424 of LNCS, Springer (2007) 72–86
[13] Derisavi, S.: A symbolic algorithm for optimal markov chain lumping. In: Proc. of TACAS. Volume 4424 of LNCS, Springer (2007) 139–154
[14] Derisavi, S.: Signature-based symbolic algorithm for optimal Markov chain lumping. In: Proc. of QEST, IEEE CS (2007) 141–150
[15] Wimmer, R., Derisavi, S., Hermanns, H.: Symbolic partition refinement with automatic balancing of time and space. Performance Evaluation **67**(9) (2010) 815–835
[16] Tseitin, G.S.: On the complexity of derivation in propositional calculus. Studies in Constructive Mathematics and Mathematical Logic **Part 2** (1970) 115–125
[17] Dutertre, B., de Moura, L.M.: A fast linear-arithmetic solver for DPLL(T). In: Proc. of CAV. Volume 4144 of LNCS, Springer (2006) 81–94
[18] Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Communications of the ACM **28**(6) (1985) 637–647
[19] Norman, G., Shmatikov, V.: Analysis of probabilistic contract signing. Journal of Computer Security **14**(6) (2006) 561–589
[20] Reiter, M., Rubin, A.: Crowds: Anonymity for web transactions. ACM Transactions on Information and System Security (TISSEC) **1**(1) (1998) 66–92
[21] Itai, A., Rodeh, M.: Symmetry breaking in distributed networks. Information and Computation **88**(1) (1990) 60–87
[22] Collin, Z., Dolev, S.: Self-stabilizing depth-first search. Information Processing Letters **49**(6) (1994) 297–301
[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), ACM Press (2010) 13–18