Symbolic Counterexample Generation for Discrete-Time Markov Chains*

Nils Jansen[1], Erika Ábrahám[1], Barna Zajzon[1], Ralf Wimmer[2],
Johann Schuster[3], Joost-Pieter Katoen[1], and Bernd Becker[2]

[1] RWTH Aachen University, Germany
[2] Albert-Ludwigs-University Freiburg, Germany
[3] University of the Federal Armed Forces Munich, Germany

Abstract. In this paper we investigate the generation of *counterexamples* for discrete-time Markov chains (DTMCs) and PCTL properties. Whereas most available methods use explicit representations for at least some intermediate results, our aim is to develop *fully symbolic* algorithms. As in most related work, our counterexample computations are based on *path search.* We first adapt *bounded model checking* as a path search algorithm and extend it with a novel *SAT-solving heuristics* to prefer paths with higher probabilities. As a second approach, we use *symbolic graph algorithms* to find counterexamples. Experiments show that our approaches, in contrast to other existing techniques, are applicable to very large systems with millions of states.

1 Introduction

Model checking is a very successful technique to automatically analyze the correctness of a system. During the last two decades, a lot of work has been done to develop model checking techniques for different kinds of systems like digital circuits, hybrid and probabilistic systems.

One feature which made model checking for digital circuits a standard technology in industry is the ability to deliver a *counterexample* if a desired property is violated. Counterexamples, which provide an explanation for the violation, are indispensable for reproducing and fixing errors in the design. They are also crucial for so-called CEGAR frameworks [1,2], in which the system is abstracted for verification. In case the abstraction is too coarse, verification might yield a spurious counterexample, which is used to refine the abstraction accordingly.

This paper addresses counterexample generation for probabilistic systems modeled as *discrete-time Markov chains (DTMCs)* and properties formalized in the logic *PCTL* [3]. Standard model checking algorithms for PCTL properties of DTMCs are based on probabilistic reachability analysis: they compute

* This work was partly supported by the German Research Council (DFG) as part of the research project CEBug (AB 461/1-1), the Transregional Collaborative Research Center AVACS (SFB/TR 14) and by the Netherlands Organisation for Scientific Research (NWO) as part of the DFG/NWO Bilateral Research Programme ROCKS.

C.S. Păsăreanu and G. Salaün (Eds.): FACS 2012, LNCS 7684, pp. 134–151, 2013.

the probability of reaching a given set of states by solving a linear equation system [4]. However, if a PCTL property is violated, e. g., if the probability to reach a set of unsafe states is larger than a certain value, these model checking algorithms are not able to return any information about the reason of the violation.

Therefore, in the last few years intensive research was carried out to develop methods which allow to generate *counterexamples* for PCTL properties of DTMCs. For digital circuits a single execution that leads from an initial state to a safety-critical state suffices as a counterexample, for DTMCs a *set* of such executions is required whose cumulated probability mass exceeds the maximally tolerated value. While some of the available counterexample generation methods [5,6,7,8] represent counterexamples as such sets of paths, other methods use alternative representations: Counterexamples are represented as regular expressions in [8] and as winning strategies for probabilistic games in [9,10]. In [11], abstractions of strongly connected components of DTMCs are used. Most relevant for our work is the representation of counterexamples as paths of a *subsystem* of the given DTMC [12,13,14]. In [13] we proposed two methods to build such subsystems. The *global* search starts with an empty subsystem, searches incrementally for paths to be included and extends the subsystem with the states along the paths and all induced transitions until the subsystem is large enough to violate the given property. The *local* search not only finds further violating paths but also path fragments which connect parts of already included paths.

Practically relevant systems are often too large to be represented explicitly, i. e., by enumerating all the states and transitions. To overcome this problem, large DTMCs can be represented *symbolically* by *binary decision diagrams (BDDs)* [15,16]. Sets of states and transitions are encoded by acyclic graphs, with the elements in the set being represented by paths in the graph. Symbolic representations are often smaller by orders of magnitude than explicit ones.

Symbolic model checking has been successfully established for DTMCs [17,18]. However, there is still a lack of symbolic algorithms for counterexample generation. In order to take full advantage of efficient representations of DTMCs and path sets, the applied path search methods should work on symbolic representations without using any explicit representations for intermediate results. In [5,6] approaches for symbolic counterexamples are presented, but all paths forming a counterexample are enumerated explicitly. For very large systems, this approach is not scalable, as (1) a counterexample may consist of a very large or even infinite number of paths. Their explicit representation has to be computed which may consist of a very large number of states. An alternative symbolic path search algorithm was introduced in [7]. This algorithm calculates the k most probable paths of a symbolically represented DTMC. Although this algorithm is well-suited for fully symbolic counterexample generation, due to some auxiliary data structures, the memory requirements increase strongly with increasing k.

As mentioned above, most of the available counterexample generation approaches for DTMCs apply *path search algorithms* (e. g., k shortest paths search [8] or heuristic search [12]). A suitable path search method which works on symbolic system representations is *bounded model checking*, encoding paths of a given length from the initial state of a DTMC to a target state by a formula such that each satisfying solution corresponds to such a path. In [5,6], counterexamples are generated by searching for solutions until enough paths have been found to form a counterexample. The method in [5] encodes paths without their probabilities in propositional logic and uses SAT-solving to find satisfying solutions. A disadvantage of this method is that it finds paths with fewer steps first, in contrast to more probable ones. The approach [6] uses SMT-solving to search for paths having at least a given minimal probability, which leads to longer running times while more probable paths are found earlier.

In this paper we first adapt SAT-based bounded model checking to support the ideas of local and global search from [13] and suggest a heuristic for SAT-solving that allows to influence the SAT search to find more probable paths first, without the need to invoke SMT-solving. Furthermore, we do not restrict the search to paths of a fixed length as suggested by standard bounded model checking, but search for paths whose length is between a given lower and upper bound.

As a second approach, we propose in this paper novel fully symbolic methods based on BDDs for the generation of counterexamples for DTMCs and PCTL properties. Our methods take as input a DTMC which is symbolically represented by BDDs. The counterexample computation uses the algorithm from [7] to find most probable paths of a DTMC. In our first BDD-based method, we combine the symbolic k-shortest path search with the idea of global search from [13] to compute a symbolically represented subsystem of the original DTMC, whose paths form a counterexample. However, this suffers from very high memory consumption, while by not enumerating the paths some computation time can be saved. As our best approach, we adapt the idea for local search, also presented in [13] which is applicable to systems with up to $1.2 \cdot 10^8$ states.

The contribution of this paper is the development of *fully symbolic* algorithms, which overcome the main disadvantages of previous approaches:

- No explicit representation of states is needed during the counterexample generation. This is crucial for handling large systems.
- In comparison to other approaches we are now able to generate counterexamples for systems with millions of states.
- As in [12,13] the counterexample is not represented by an enumeration of paths which yields a counterexample that is smaller by orders of magnitude.

In the next section we briefly introduce some theoretical foundations. Section 3 describes the general framework of our symbolic methods for counterexample generation. The usage of SAT-based path search is described in Section 4 and the application of BDD-based graph search algorithms in Section 5. These methods are evaluated experimentally on some case studies in Section 6. We conclude our work and discuss future work in Section 7.

2 Preliminaries

We introduce the basic definitions and concepts used in this paper. For more details we refer to [4].

2.1 Discrete-Time Markov Chains and Critical Subsystems

Definition 1. *A* discrete-time Markov chain (DTMC) *is a tuple $M = (S, I, P, L)$ with S being a finite set of states, $I : S \to [0,1] \subseteq \mathbb{R}$ with $\sum_{s \in S} I(s) \leq 1$ an initial distribution, $P : S \times S \to [0,1] \subseteq \mathbb{R}$ a matrix of transition probabilities such that $\sum_{s' \in S} P(s,s') \leq 1$ for all $s \in S$, and L a labeling function with $L : S \to 2^{AP}$ with AP a denumerable set of atomic propositions.*

Please note that we allow *sub-stochastic* distributions $\sum_{s \in S} I(s) \leq 1$ and $\sum_{s' \in S} P(s,s') \leq 1$ for all $s \in S$. Usually, these sums of probabilities are required to be exactly 1. This can be obtained by defining $M' = (S \cup \{s_\perp\}, I', P', L')$ with $s_\perp$ a fresh sink state such that for all $s, s' \in S$ we have $I'(s) = I(s)$ and $I'(s_\perp) = 1 - \sum_{s \in S} I(s)$, $P'(s,s') = P(s,s')$, $P'(s, s_\perp) = 1 - \sum_{s' \in S} P(s,s')$, $P'(s_\perp, s_\perp) = 1$ and $P'(s_\perp, s) = 0$, and finally $L'(s) = L(s)$ and $L'(s_\perp) = \emptyset$.

For simplicity, in the following we restrict ourselves to DTMCs (S, I, P, L) having a *single initial state* $s_I \in S$ with $I(s_I) = 1$ and use the notation (S, s_I, P, L). Note that every DTMC having an arbitrary initial distribution can be transformed to this form by adding a fresh unique initial state.

Assume in the following a DTMC $M = (S, s_I, P, L)$. We say that there is a *transition* (s, s') from a state $s \in S$ to a state $s' \in S$ iff $P(s,s') > 0$. A *path* of M is a finite or infinite sequence $\pi = s_0 s_1 \ldots$ of states $s_i \in S$ such that $P(s_i, s_{i+1}) > 0$ for all i. We call the transitions (s_i, s_{i+1}) to be *contained* in the path π, written $(s_i, s_{i+1}) \in \pi$. We write π^i for the ith state on path π; its position is called *depth*. The *length* of a finite path $\pi = s_0 \ldots s_n$ is the number n of its transitions.

We write $\mathit{Paths}^M_{\mathit{inf}}$ for the set of all infinite paths of M, and $\mathit{Paths}^M_{\mathit{inf}}(s)$ for those starting in $s \in S$. Analogously, $\mathit{Paths}^M_{\mathit{fin}}$ is the set of all finite paths of M, $\mathit{Paths}^M_{\mathit{fin}}(s)$ of those starting in $s \in S$, and $\mathit{Paths}^M_{\mathit{fin}}(s,t)$ of those starting in $s \in S$ and ending in $t \in S$. A state $t \in S$ is called *reachable* from another state $s \in S$ iff $\mathit{Paths}^M_{\mathit{fin}}(s,t) \neq \emptyset$.

The *cylinder set* of a finite path π of M is defined as $\mathit{Cyl}(\pi) = \{\pi' \in \mathit{Paths}^M_{\mathit{inf}} \mid \pi \text{ is a prefix of } \pi'\}$. To each state $s \in S$ of M we associate the smallest σ-algebra that contains all cylinder sets of all finite paths in $\mathit{Paths}^M_{\mathit{fin}}(s)$. This yields a unique probability measure Pr^M_s (or short Pr) on the σ-algebra where the probabilities of the cylinder sets are given by

$$\mathit{Pr}\big(\mathit{Cyl}(s_0 \ldots s_n)\big) = \prod_{i=0}^{n-1} P(s_i, s_{i+1}) \ .$$

For finite paths π we set $\mathit{Pr}_{\mathit{fin}}(\pi) = \mathit{Pr}\big(\mathit{Cyl}(\pi)\big)$. For sets of finite paths $R \subseteq \mathit{Paths}^M_{\mathit{fin}}(s)$ we define $\mathit{Pr}_{\mathit{fin}}(R) = \sum_{\pi \in R'} \mathit{Pr}_{\mathit{fin}}(\pi)$ with $R' = \{\pi \in R \mid \forall \pi' \in R.\ \pi' \textit{ is not a prefix of } \pi\}$.

The syntax of *probabilistic computation tree logic (PCTL)* [19] is given by[1]

$$\varphi \ ::= \ p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbb{P}_{\sim\lambda}(\varphi \; U \; \varphi)$$

for (state) formulae with $p \in AP$, $\lambda \in [0,1] \subseteq \mathbb{R}$, and $\sim \ \in \{<, \leq, \geq, >\}$. We define the "finally"-operator $\Diamond$ and the "globally"-operator $\Box$ in the usual way.

For a property $\mathbb{P}_{\leq\lambda}(\varphi_1 \, U \, \varphi_2)$ refuted by M, a *counterexample* is a set $C \subseteq \mathit{Paths}^{M}_{\mathit{fin}}(s_I)$ of finite paths starting in the initial state and *satisfying* $\varphi_1 \, U \, \varphi_2$ such that $\mathit{Pr}_{\mathit{fin}}(C) > \lambda$. For $\mathbb{P}_{<\lambda}(\varphi_1 \, U \, \varphi_2)$, the probability mass has to be at least λ. We consider only upper probability bounds; see [8] for the reduction of lower bounds to this case.

The model checking and counterexample generation problems for $\mathbb{P}_{\leq\lambda}(\varphi_1 \, U \, \varphi_2)$ can be recursively reduced to a reachability problem as follows: We transform the DTMC $M = (S, s_I, P, L)$ to a DTMC $M' = (S, s_I, P', L)$ by removing all outgoing transitions from states satisfying $\neg\varphi_1 \vee \varphi_2$, i. e., $P'(s,s') = 0$ if s satisfies $\neg\varphi_1 \vee \varphi_2$ and $P'(s,s') = P(s,s')$ otherwise. Then M satisfies $\mathbb{P}_{\leq\lambda}(\varphi_1 \, U \, \varphi_2)$ iff M' satisfies $\mathbb{P}_{\leq\lambda}(\Diamond \, \varphi_2)$. In the following we concentrate on this reduced problem.

Consider a DTMC $M = (S, s_I, P, L)$, a set of target states $T \subseteq S$ and an upper bound $\lambda \in [0,1]$ on the allowed probability to reach one of these target states from the initial state s_I. For notational convenience we write $\mathcal{P}_{\leq\lambda}(\Diamond \, T)$ for the property that the probability of reaching a target state from the initial state is less or equal λ. We assume this property to be violated, i. e., the actual probability of reaching T exceeds λ.

In [13] we proposed to represent counterexamples as so-called *critical subsystems* instead of large, possibly infinite sets of paths. Intuitively, a critical subsystem is a part of the original system in which the given probability bound is already exceeded.

Definition 2. *A* subsystem *of a DTMC $M = (S, s_I, P, L)$ is a DTMC $M' = (S', s_I, P', L')$ such that $S' \subseteq S$, $s_I \in S'$, $P'(s,s') \in \{P(s,s'), 0\}$ and $L'(s) = L(s)$ for all $s, s' \in S'$. We call such a subsystem M' of M* critical *for $T \subseteq S$ and $\lambda \in [0,1] \subseteq \mathbb{R}$ iff $S' \cap T \neq \emptyset$ and the probability to reach a state in $S' \cap T$ from s_I in M' is larger than λ.*

Note that the set of all paths leading from the initial state s_I to the set of target states T inside the critical subsystem forms a counterexample.

2.2 Symbolic Representation of DTMCs

In this paper we use symbolic representations of DTMCs and generate symbolic critical subsystems. *Explicit* means that the transition probabilities are represented as a sparse matrix, which contains one entry per transition with non-zero probability. This representation is used, e. g., by the probabilistic model checker Mrmc [20]. A *symbolic* DTMC representation encodes state and transition sets, e. g., as paths in a graph or as solutions of a certain formula. Symbolic representations are often smaller by orders of magnitude than the explicit ones and

[1] In this paper we only consider unbounded properties.

allow to reduce not only the memory consumption but also the computational costs for operations on the data structures.

As a symbolic data structure for the representation of DTMCs we choose *binary decision diagrams* [15] and *multi-terminal binary decision diagrams* [16].

Definition 3. *Let Var be a set of Boolean variables. A* binary decision diagram *(BDD) over Var is a rooted, acyclic, directed graph* $B = (V, n_{\text{root}}, E)$ *with a finite set* V *of nodes, a root node* $n_{\text{root}} \in V$ *and edges* $E \subseteq V \times V$. *Each node is either an* inner *node or a* leaf *node. Leaf nodes* $n \in V$ *have no outgoing edges and are labeled with* $\mathit{label}(n) \in \{0, 1\}$. *Inner nodes* $n \in V$ *have exactly two* successor *nodes, denoted by* $\mathit{hi}(n)$ *and* $\mathit{lo}(n)$, *and are labeled with a variable* $\mathit{label}(n) \in \mathit{Var}$.

A multi-terminal binary decision diagram *(MTBDD) is like a BDD but it labels leaf nodes* $n \in V$ *with real values* $\mathit{label}(n) \in \mathbb{R}$.

Let B be a BDD over Var and $\mathcal{V}(\mathit{Var}) = \big\{\nu : \mathit{Var} \to \{0,1\}\big\}$ the set of all variable valuations. Each $\nu \in \mathcal{V}(\mathit{Var})$ induces a unique path in B from the root to a leaf node by moving from each inner node n to $\mathit{hi}(n)$ if $\nu(\mathit{label}(n)) = 1$ and to $\mathit{lo}(n)$ otherwise. A BDD B represents a function $f_B : \mathcal{V}(\mathit{Var}) \to \{0,1\}$ assigning to each $\nu \in \mathcal{V}(\mathit{Var})$ the label of the leaf node reached in B by the path induced by ν. We often identify B with f_B and write $B(\nu)$ instead of $f_B(\nu)$. Analogously, each MTBDD B represents a function $f_B : \mathcal{V}(\mathit{Var}) \to \mathbb{R}$.

An (MT)BDD is *ordered* if there is a linear order $< \subseteq \mathit{Var} \times \mathit{Var}$ on the variables such that for all inner nodes n either $\mathit{hi}(n)$ is a leaf node or $\mathit{label}(n) < \mathit{label}\big(\mathit{hi}(n)\big)$, and the same for $\mathit{lo}(n)$. An (MT)BDD is *reduced* if all functions rooted at the different nodes of the (MT)BDD are different. For a fixed variable order, they are canonical data structures for representing functions $f : \mathcal{V}(\mathit{Var}) \to \{0,1\}$ resp. $f : \mathcal{V}(\mathit{Var}) \to \mathbb{R}$ [15]. In the following we assume all (MT)BDDs to be reduced and ordered with respect to a fixed variable order.

By Var' we denote the variable set Var with each variable $x \in \mathit{Var}$ renamed to some $x' \in \mathit{Var}'$ such that $\mathit{Var} \cap \mathit{Var}' = \emptyset$. Our algorithms use the standard (MT)BDD operations union $B_1 \cup B_2$, intersection $B_1 \cap B_2$, variable renaming $B[x \to x']$, and existential quantification $\exists x.\ B$ for $x \in \mathit{Var}$, $x' \in \mathit{Var}'$.

BDDs and MTBDDs can be used to represent DTMCs symbolically as follows: Let $M = (S, s_I, P, L)$ be a DTMC and Var a set of Boolean variables such that for each $s \in S$ there is a unique binary encoding $\nu_s : \mathit{Var} \to \{0,1\}$ with $\nu_s \neq \nu_{s'}$ for all $s, s' \in S$, $s \neq s'$. For $s, s' \in S$ we also define $\nu_{s,s'} : \mathit{Var} \cup \mathit{Var}' \to \mathbb{R}$ with $\nu_{s,s'}(x) = \nu_s(x)$ and $\nu_{s,s'}(x') = \nu_{s'}(x)$ for $x \in \mathit{Var}$, $x' \in \mathit{Var}'$. A target state set $T \subseteq S$ is represented by a BDD $\hat{T}$ over Var such that $\hat{T}(\nu_s) = 1$ iff $s \in T$. Similarly for the initial state, $\hat{I}(\nu_s) = 1$ iff $s = s_I$. The probability matrix $P : S \times S \to [0,1] \subseteq \mathbb{R}$ is represented by an MTBDD $\hat{P}$ over $\mathit{Var} \cup \mathit{Var}'$ such that $\hat{P}(\nu_{s,s'}) = P(s,s')$ for all $s, s' \in S$. For an MTBDD B over Var we use B_{bool} to denote the BDD over Var with $B_{\mathit{bool}}(\nu) = 1$ iff $B(\nu) > 0$ for all valuations ν.

This formalism is used, e. g., by the stochastic model checker PRISM [21], whose benchmark set [22] is standard for DTMCs. These test-cases are modeled in a guarded command language describing system *components*; the global state space and the transition probabilities are generated by parallel composition. The transition matrices are usually sparse and well-structured with relatively few different probabilities; therefore the symbolic MTBDD representation is in many cases more compact by several orders of magnitude than explicit representations. For more details we refer to documentation of PRISM.

3 Symbolic Counterexample Generation Framework

In this section we present our framework for the generation of probabilistic counterexamples with symbolic data structures. We give an algorithm that computes, for a symbolically represented DTMC as input, a critical subsystem, which is again symbolically represented. As the most significant ingredient, this algorithm needs a *symbolic path search* method, which returns paths of the input DTMC. The critical subsystem is initially empty and is incrementally extended with the states along found paths and with transitions between them. Implementations of the path search method will be described in Sections 4 and 5.

Algorithm 1. Finding a critical subsystem

```
FindCriticalSubsystem(MTBDD $\hat{P}$, BDD $\hat{I}$, BDD $\hat{T}$, double $\lambda$)
begin
    BDD $States = \emptyset$;   BDD $NewStates = \emptyset$;   MTBDD $SubSys = \emptyset$;
    if (ModelCheck($\hat{P}, \hat{I}, \hat{T}$) $> \lambda$)
      while (ModelCheck($SubSys, \hat{I}, \hat{T}$) $\leq \lambda$)
        $NewStates :=$ FindNextPath($\hat{P}, \hat{I}, \hat{T}, SubSys$);
        if ($NewStates \neq \emptyset$)
          $States := States \cup NewStates$;
          $SubSys :=$ ToTransitionBDD($States$) $\cap \hat{P}$
        end if
      end while
    end if
    return $SubSys$
end
```

The algorithm for finding a symbolic counterexample is depicted in Algorithm 1. The parameters specify the input DTMC symbolically by the MTBDD $\hat{P}$ for the transition probabilities, the BDD $\hat{I}$ for the initial state and the BDD $\hat{T}$ for the target states, as well as a probability bound λ which shall be exceeded by the resulting critical subsystem. The local variable *States* is used to symbolically represent the set of states which are part of the current subsystem, while *NewStates* is used to store the states occurring on a path which shall extend the current subsystem. The MTBDD *SubSys* stores the transition MTBDD of the current subsystem. The algorithm uses the following methods:

`ModelCheck(MTBDD` $\hat{P}$`, BDD` $\hat{I}$`, BDD` $\hat{T}$`)` performs symbolic probabilistic model checking [17,18] and returns the probability of reaching states in $\hat{T}$ from states in $\hat{I}$ via transitions from $\hat{P}$.

`FindNextPath(MTBDD` $\hat{P}$`, BDD` $\hat{I}$`, BDD` $\hat{T}$`, MTBDD` *SubSys*`)` computes a path leading through the DTMC induced by the transition MTBDD $\hat{P}$, the initial state $\hat{I}$, and the set of target states $\hat{T}$. Which path is found next depends on the current subsystem *SubSys* and therefore on the set of previously found paths. Implementations of this method will be discussed in Sections 4 and 5.

`ToTransitionBDD(BDD` *States*`)` computes first the BDD States' by renaming each variable $x \in \mathit{Var}$ occurring in *States* to $x' \in \mathit{Var}'$ and returns the transition BDD $\mathit{States} \cap \mathit{States}'$ in which there is a transition between all pairs of states occurring in *States*, i. e., $(\mathit{States} \cap \mathit{States}')(\nu_{s_1,s_2}) = 1$ iff $\mathit{States}(\nu_{s_1}) = \mathit{States}(\nu_{s_2}) = 1$.

The algorithm proceeds as follows. First, the three empty objects *States*, *NewStates*, and *SubSys* are created in line (1). If `ModelCheck(`$\hat{P}, \hat{I}, \hat{T}$`)` in line (2) reveals that λ is exceeded then the reachability property is violated and the search for a counterexample starts. Otherwise the algorithm just terminates. The condition of the `while`-loop in line (3) invokes model checking for the current subsystem described by *SubSys* and the original initial states and target states. The loop runs until `ModelCheck(`$\mathit{SubSys}, \hat{I}, \hat{T}$`)` returns a value which is greater than λ. In this case, the current subsystem is *critical.* Please note, that in our implementation we do not invoke model checking in every iteration. Depending on the input system, we search for a certain number of paths until we invoke this method. In every iteration, first the method `FindNextPath(`$\hat{P}, \hat{I}, \hat{T}, \mathit{SubSys}$`)` in line (4) returns a set of states which occur on a path through the system. If this set is not empty, the current set of states is extended by these new states in line (6). Afterwards, the current subsystem is extended in line (7): `ToTransitionBDD(`*States*`)` generates a transition relation between all states found so far. The intersection of the resulting BDD and the original transition MTBDD $\hat{P}$ represents a probability matrix $P' \subseteq P$ which is restricted to transitions between the states in *States*. These induced transitions define the updated subsystem *SubSys*.

4 Searching Paths Using SAT Solving

In this section we present two implementations for the path searching method (Algorithm 1) using bounded model checking and SAT solving. First, an existing method which searches paths with certain lengths is adapted to our symbolic framework. Second, we present a new method which searches for path fragments that extend the subsystem. Finally, we describe a new SAT-solving heuristic which guides the SAT solver to prefer more probable path fragments.

4.1 Adapting Bounded Model Checking for Global Search

In [5], a bounded model checking (BMC) approach for DTMCs was developed. Starting with a symbolic representation of a DTMC by an MTBDD $\hat{P}$ and

BDDs $\hat{I}$ and $\hat{T}$ as described before, first Tseitin's transformation [23] is applied to generate formulae in conjunctive normal form (CNF) from the BDDs. We will denote the resulting CNF predicates by $\check{P}$, $\check{I}$, and $\check{T}$, respectively.

The BMC formula built from the symbolic representation of a DTMC is parameterized in $k \in \mathbb{N}$ and has the following structure:

$$BMC(k) \;=\; \check{I}(Var_0) \;\wedge\; \bigwedge_{i=0}^{k-1} \check{P}(Var_i, Var_{i+1}) \;\wedge\; \check{T}(Var_k) \tag{1}$$

where k is the length of the paths considered.

This formula depends on sets $Var_i = \{\sigma_{i,1}, \ldots, \sigma_{i,m}\}$ of Boolean variables which encode the ith state of a path of length k through the DTMC starting in an initial state and ending in a target state. Each satisfying assignment ν of formula (1) corresponds to such a path. If there is no satisfying assignment, there is no such path with length k. We identify the assignment $\big(\nu(\sigma_{i,1}), \ldots, \nu(\sigma_{i,m})\big)$ with the state s_i of the DTMC.

Since usually multiple paths need to be found in order to form a counterexample, the solver has to enumerate satisfying solutions for $BMC(k)$, $k = 0, 1, \ldots$, until enough probability mass has been accumulated. To exclude an already found solution from further search, new clauses are added to the SAT solver's clause database. Consider a path $\pi_j = s_0 \ldots s_k$ that was found in the jth iteration of the search process. Let $\nu : \bigcup_{i=0}^{k} Var_i \to \{0, 1\}$ be the corresponding satisfying assignment. The path π_j is uniquely described by the following formula:

$$\bigwedge_{i=0}^{k} \sigma_{i,1}^{\nu(\sigma_{i,1})} \wedge \sigma_{i,2}^{\nu(\sigma_{i,2})} \wedge \cdots \wedge \sigma_{i,m}^{\nu(\sigma_{i,m})} \;, \tag{2}$$

where $\sigma_{i,j}^{1} = \sigma_{i,j}$ and $\sigma_{i,j}^{0} = \neg\sigma_{i,j}$. To exclude π_j from the solution space of $BMC(k)$, its negation is built and added to the solver's clause database:

$$\bigvee_{i=0}^{k} \bigvee_{j=1}^{m} {\sigma_{i,j}}^{1-\nu(\sigma_{i,j})} \;. \tag{3}$$

This ensures that for a new path at least one state variable has to be differently assigned than for path π_j.

Every time the SAT solver returns a new satisfying assignment, the probability of the underlying path is computed and the path is saved. This proceeds until the probability of all paths found exceeds the bound λ. The resulting counterexample is therefore a set of explicitly represented paths whose cumulated probability mass exceeds the probability bound. If no further satisfying assignment can be found, the path length k is increased by one and the search process gets restarted.

We adopt this procedure for our framework for generating a symbolically represented critical subsystem. Instead of computing the probability of single paths, the BDD state representation of each new path is computed and returned to Algorithm 1. This is done in form of a callback, as we do not want to restart the solver after each iteration. If model checking reports that the probability mass of the generated subsystem is high enough, the procedure stops.

In general, termination is guaranteed as the SAT solver finds all possible paths of length k. Eventually, the subsystem will consist of all states that are part of paths from initial to target states. This subsystem induces the whole probability mass of reaching a target state in the original system. As the algorithm only starts if the probability bound is exceeded, the probability mass of this system will also exceed the bound. Therefore, the algorithm always terminates.

4.2 Adapting Bounded Model Checking for Fragment Search

The previously described approach of using the SAT solver to find paths leading from the initial state of the DTMC to the target states is now extended according to the *local search* approach described in [13]. We aim at finding *path fragments* that extend the already found system iteratively.

The intuition is as follows: In the first search iteration, the CNF formula given to the SAT solver is satisfied if and only if the assignment corresponds to a path of *maximal* length n through the input DTMC leading from the initial state s_I to a target state $t \in T$. This path induces the initial subsystem. Subsequently, this system is extended by paths whose first and last states are included in the current subsystem, while all states in between are fresh states.

For this we need to consider already found states for all possible depths $0 \leq d \leq n$. For a state s let $\nu_s^d : Var_d \to \{0, 1\}$ be the unique assignment of Var_d corresponding to state s.

We introduce a flag f_s^d for each state s and each depth d. This flag is assigned 1 if and only if the assignment of the state variables at depth d corresponds to the state s:

$$f_s^d \leftrightarrow \left(\sigma_{d,1}^{\nu_s^d(\sigma_{d,1})} \wedge \cdots \wedge \sigma_{d,m}^{\nu_s^d(\sigma_{d,m})}\right) . \tag{4}$$

The next variable K_j^d describes the whole set of states which have been found in the iterations $0, 1, \ldots, j$ of the search process (again in terms of the variables Var_d for depth d). Note, that these are exactly the states of the current subsystem *SubSys* after iteration j. We set $K_{-1}^d :=$ false. Assume that in iteration j of the search process path $\pi_j = s_0 s_1 \ldots s_n$ is found. We then define

$$K_j^d \leftrightarrow \left(K_{j-1}^d \vee \bigvee_{i=1}^{n} f_{s_i}^d\right) . \tag{5}$$

In the first search iteration we need a formula which is true iff the variable assignment corresponds to a path of maximal length n leading from the initial state to a target state of the DTMC:

$$\check{I}(Var_0) \; \wedge \; \bigvee_{i=0}^{n} \check{T}(Var_i) \wedge \tag{6a}$$

$$\bigwedge_{i=0}^{n-1} \Big[\big(\neg\check{T}(Var_i) \to \check{P}(Var_i, Var_{i+1})\big) \; \wedge \; \big(\check{T}(Var_i) \to (Var_i = Var_{i+1})\big) \Big] . \tag{6b}$$

Assume that ν is an assignment corresponding to the path $\pi = s_0 s_1 \dots s_n$. Formula (6a) states, that the first state s_0 is the initial state and that one of the states $s_0, \dots s_n$ is a target state. Formula (6b) ensures, that if a state s_i is not a target state, a transition will be taken to the next state. Contrary, if s_i is a target state, all following state variables will be assigned s_i which creates an implicit self loop on this state. In the context of the original system, this path ends with a target state s_n.

For the following iterations $j > 1$, we need the previously defined variables K_d^j:

$$K_{j-1}^0 \;\wedge\; \check{P}(\mathit{Var}_0, \mathit{Var}_1) \;\wedge \neg K_{j-1}^1 \;\wedge\; \bigvee_{d=2}^{n} K_{j-1}^d \tag{7a}$$

$$\wedge \bigwedge_{d=1}^{n-1} \left[\left(\neg K_{j-1}^d \rightarrow \check{P}(\mathit{Var}_i, \mathit{Var}_{i+1}) \right) \;\wedge\; \left(K_{j-1}^d \rightarrow \mathit{Var}_i = \mathit{Var}_{i+1} \right) \right] \;. \tag{7b}$$

Formula (7a) ensures that the first state s_0 of a solution path $\pi_j = s_0 \dots s_n$ is contained in the set K_{j-1}^0 of previously found states, that a transition is taken from this state to a not yet found state s_1 and that one of the following states s_d, $d \geq 2$, is again contained in K_{j-1}^d. Formula (7b) enforces transitions from all not yet found states s_i to s_{i+1}. If s_i was already included in previous paths then all following states are assigned as s_i.

Termination is guaranteed, as the length of the paths is bounded by n. If no further satisfying assignments are found, this number has to be increased. However, the diameter, i. e., the longest cycle-free path of the underlying graph, is an upper bound on the length of loop-free paths from s_{init} to target states. Therefore, n needs to be increased only finitely many times, such that a critical subsystem is always determined in finite time.

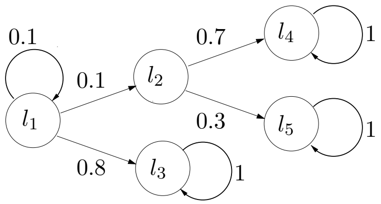
4.3 SAT Heuristic for Finding More Probable Paths

A drawback of the SAT-based search strategies is that paths are found without considering their probability beforehand. If paths or transitions with higher probabilities are preferred, the process can be accelerated. We therefore try to modify the variable selection of the SAT solver.

SAT solvers have efficient variable selection strategies, i. e., strategies to decide which variable should be assigned next during the search process. We adjust the choice of the *value* the solver assigns to the selected variable, in order to prefer paths with higher probabilities.

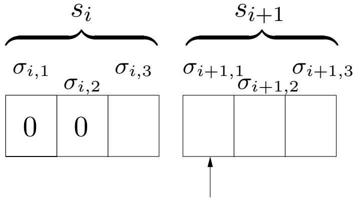
The decision how to assign a variable is based on the transition probabilities. If a variable $\sigma_{i+1,j}$ is to be assigned at depth $0 < i+1 \leq n$, its value partly determines s_{i+1}, being the target state of a transition from s_i. We choose the value for $\sigma_{i+1,j}$ which corresponds to the state s_{i+1} to which the transition with the highest probability can be taken (under the current assignment).

Example 1. Assume the following DTMC:



Let the states of the DTMC be encoded by three propositional variables and assume that the solver partially assigned the state variables for the ith and the $(i+1)$th time instance as follows:

	$\sigma_{j,1}$	$\sigma_{j,2}$	$\sigma_{j,3}$
l_1	0	0	0
l_2	0	0	1
l_3	0	1	0
l_4	0	1	1
l_5	1	0	0



The ith state s_i is determined to be l_1 or l_2. For s_{i+1} still all states are possible. The first bit of the $(i+1)$th state, as indicated by the arrow, should be assigned next. We would choose to set the bit to 0, because in this way we do not exclude the most probable eligible transition from l_1 to l_3.

5 Searching Paths Symbolically

In this section we use symbolic graph algorithms to implement the path search (Algorithm 1) by which a critical subsystem of a DTMC is built. We first recall how one can find the k most probable paths through a symbolically represented DTMC. We call this the *symbolic global search* as the most probable paths through the whole system are found. We embed this procedure into our symbolic counterexample search. Afterwards we present a new search method which symbolically searches for the most probable path fragments that extend the current subsystem. We call this approach the *symbolic fragment search*.

5.1 Symbolic Global Search

The goal of this procedure is to find paths leading from the initial state to a target state ordered by their probability, starting with the most probable path. As usually classical graph algorithms are used, this is also referred to as the k shortest path search, although this corresponds to the k most probable paths. Utilized for a counterexample search, the value of k is not fixed beforehand but the search terminates if enough probability mass is accumulated [8].

In [7], a symbolic version of the k shortest path search was presented. The core components are the calculation of the actual shortest path and a transformation of the DTMC such that the shortest path in the altered system corresponds to the second shortest path in the original system. We adapt this method for

Algorithm 2. The global search algorithm for symbolic DTMCs

```
SymbolicGlobalSearch(MTBDD $\hat{P}$, BDD $\hat{I}$, BDD $\hat{T}$, MTBDD $SP$)
begin
    if ($SP \neq \emptyset$)                                          (1)
        ($\hat{P}$, $\hat{I}$, $\hat{T}$) := Change($\hat{P}$, $\hat{I}$, $\hat{T}$, $SP$);     (2)
    endif                                                             (3)
        $SP$ := ShortestPath($\hat{P}$, $\hat{I}$, $\hat{T}$);                      (4)
    return $SP$;                                                      (5)
end
```

symbolic counterexample computation for DTMCs. The resulting algorithm is depicted in Algorithm 2.

Parameters are as usual $\hat{P}$, $\hat{I}$ and $\hat{T}$ as well as an MTBDD SP to store the path computed in the last iteration. The following methods are used (for details on the MTBDD operations we refer to the appendix of [7]):

`ShortestPath(MTBDD` $\hat{P}$, `BDD` $\hat{I}$, `BDD` $\hat{T}$`)` computes the most probable path leading from a state of $\hat{I}$ to a state of $\hat{T}$ via transitions from $\hat{P}$ and returns the MTBDD representation SP of this path. For this method, a set-theoretic variant of Dijkstra's algorithm is used.

`Change(MTBDD` $\hat{P}$, `BDD` $\hat{I}$, `BDD` $\hat{T}$, `MTBDD` SP`)` changes the DTMC $(\hat{P}, \hat{I}, \hat{T})$ to a new one such that the shortest path in the new DTMC corresponds to the second shortest path of the original DTMC(for the basic algorithm cf. [24]). The core idea of the symbolic implementation is to add an additional state variable that indicates a copy (when set to 1) or the original state (when set to 0). The MTBDD $\hat{P}$ is therefore extended by two variables: One for the source and one for the target state.

The original algorithm works with a fixed number k of search iterations. We modified this method to be incremental, i. e., the resulting $\hat{P}$ and SP after an iteration step are input for the next iteration. If the SP parameter is empty, the first shortest path is computed (line 4). Otherwise, the system is modified to exclude the previously found path (line 2). On the modified system, a new search is performed, yielding the next shortest path (line 4).

As in our framework the termination condition lies inside the symbolic counterexample algorithm (see Algorithm 1), we call Algorithm 2 as often as needed to form a counterexample. We use the MTBDD SP as a parameter in order to determine, what the *next* shortest path is. Note that in this case Algorithm 1 has to call the search method with the last shortest path instead of the current subsystem and it also has to transform the resulting shortest path MTBDD SP to a state set BDD (see the method `ToStateBDD(·)` on page 147).

Finally, this procedure yields a critical subsystem induced by a finite number k of paths. The paths are ordered w. r. t. to their probability. Note that the MTBDD resulting from the iterative application of the `Change()`-method grows rapidly and renders this method not applicable to systems which require a large number of paths, as our test cases will show.

5.2 Symbolic Fragment Search

In contrast to the previous approach, where we search for whole paths through the system, we aim now at finding most probable *path fragments*. Intuitively, first a *base path* is found being the most probable path from the initial state to one of the target states of the input system. This path forms the initial subsystem. Afterwards, the subsystem is incrementally extended by finding the most probable path fragment that connects states from the current subsystem. This approach was successfully implemented for explicit graph representations [13] and is now adapted to symbolic representations. The algorithm is depicted in Algorithm 3.

Algorithm 3. The fragment search for symbolic DTMCs

```
SymbolicFragmentSearch(MTBDD $\hat{P}$, BDD $\hat{I}$, BDD $\hat{T}$, MTBDD $SubSys$)
begin
    MTBDD $SP$;
    BDD $SubSysStates$;
    if ($SubSys = \emptyset$)
      $SP :=$ ShortestPath($\hat{P}$, $\hat{I}$, $\hat{T}$);
    else
      $SubSysStates :=$ ToStateBDD($SubSys$);
      $SP :=$ ShortestPath($\hat{P} \setminus SubSys$, $SubSysStates$, $SubSysStates$);
    end if
    return ToStateBDD($SP$)
end
```

We need an MTBDD *SP* to store the path which is computed and a BDD *SubSysStates* which stores the states of the current subsystem. The following methods are used:

`ShortestPath(MTBDD` $\hat{P}$`, BDD` $\hat{I}$`, BDD` $\hat{T}$`)` uses a set-theoretic variant of Dijkstra's algorithm as in Section 5.1.

`ToStateBDD(MTBDD` *SubSys*`)` computes for the transition MTBDD *SubSys* a BDD describing all states that occur as source state or target state for one of the transitions of *SubSys*. When *SubSys* is defined over the variables $Var = \{x_1, \ldots, x_n\}$ and $Var' = \{x'_1, \ldots, x'_n\}$, this is done by first building the set $OUT := \exists x'_1, \ldots, x'_n.\ SubSys_{bool}$ of all states with an outgoing transition. Afterwards, the set $IN' := \exists x_1, \ldots, x_n.\ SubSys_{bool}$ of states with ingoing transitions is built. These resulting BDDs have to be defined over the same variable set, therefore we perform a variable renaming for the set of states with ingoing transitions: $IN := IN'[x'_1 \to x_1] \ldots [x'_n \to x_n]$. Building the union $IN \cup OUT$ yields the needed BDD.

The symbolic fragment search checks whether the parameter *SubSys* is empty, which means, whether this is the first search iteration. If this is the case then the base path leading from the initial state $s_I \in \hat{I}$ to one of the target states $t \in \hat{T}$ is computed by invoking the shortest path search. The resulting path, stored in the BDD *SP*, is transformed into a state BDD and returned to the symbolic model checking framework (see Algorithm 1). If *SubSys* is not empty then a part of the subsystem has already been determined. In this case, we compute the state BDD *SubSysStates* by invoking `ToStateBDD(`*SubSys*`)`. The shortest path algorithm is called to find the most probable path from a state in *SubSysStates* to a state in *SubSysStates* inside the DTMC induced by $\hat{P}$ without using direct transitions from *SubSysStates* to *SubSysStates*. Note that, since we seach for the most probable such path, this path will not contain any *SubSysStates* states between the starting and ending ones.

6 Case Studies

We developed prototypes in C++ for all approaches described in this paper using the BDD package CUDD [25] and the SAT solver MiniSat [26]. All experiments were performed on a QuadCore Intel CPU (2.66 GHz) with 8 GB RAM. We present results for the Probabilistic Contract Signing protocol [27] and the CROWDS protocol [28]. We used the PRISM models [22] of both protocols.

Probabilistic Contract Signing is a network protocol targeting the *fair* exchange of critical information between two parties A and B. In particular, whenever B has obtained A's commitment to a *contract*, B shall not be able to prevent A from getting B's commitment. The PCTL property $\mathbb{P}_{\leq 0.5}\big(\lozenge\,[knowA \wedge \neg knowB]\big)$ we are investigating describes an unfair situation where A knows B's secrets while B doesn't know A's secrets. The target states in our model carry corresponding labels. The model size is scaled by the number of data pieces to exchange and the size of each data piece.

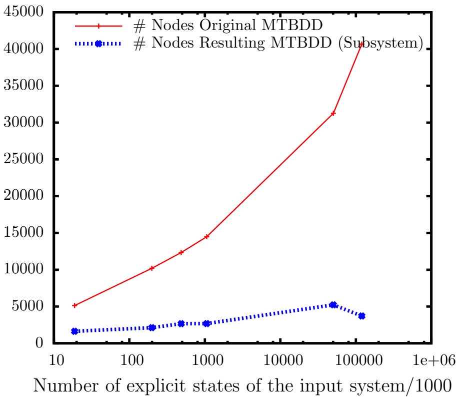


Fig. 1. BDD sizes

The CROWDS protocol aims at anonymous communication in networks, where a crowd of n users is divided in *good members* and *bad members*. A good member delivers a message to its destination with probability $1 - p_f$ and forwards it to another member, randomly chosen, with probability p_f. This guarantees that no bad member knows the original sender of the message. Each *session* describes the delivery of a message to a sender. If a user is identified twice by a bad member, anonymity is no longer guaranteed. This is called *positively identified* (*Pos*).

		Crowds protocol					Contract Signing protocol		
# states		18817	198199	485941	1058353	50445495	33790	156670	737278
model checking		0.426153	0.716089	0.807731	0.871703	0.85054	0.515625	0.515625	0.503906
probability threshold		0.25	0.35	0.4	0.4	0.2	0.5	0.5	0.5
Symb global	# states	630	622	622	622	1013	**6804**	24006	13222
	# paths	1019	978	977	979	738	**512**	326	733
	prob.	0.149138	0.14843	0.14843	0.14843	0.117311	**0.5**	0.318359	0.0447388
	time (s)	TO	TO	TO	TO	TO	**1871.82**	TO	TO
Symb fragment	# states	**600**	**1611**	**2415**	**2884**	**10239**	**6927**	**38247**	**139980**
	# paths	**201**	**1359**	**555**	**835**	**2641**	**521**	**521**	**8192**
	prob.	**0.25659**	**0.350066**	**0.401258**	**0.400333**	**0.201197**	**0.508789**	**0.508789**	**0.5**
	time (s)	**12.18**	**169.93**	**276.41**	**413.15**	**2830.55**	**26.61**	**740.15**	**972.57**
BMC classic	# states	1241	1205	1241	1241	1558	**6684**	**37464**	**139302**
	# paths	140822	127845	126318	129960	43250	**513**	**513**	**8193**
	prob.	0.175123	0.173481	0.173651	0.1746	0.0994408	**0.500977**	**0.500977**	**0.500061**
	time (s)	TO	TO	TO	TO	TO	**20.17**	**410.61**	**367.1**
SAT global	# states	**908**	997	997	997	1583	**6825**	**38025**	**139302**
	# paths	**231359**	295240	258860	253733	238894	**520**	**520**	**8193**
	prob.	**0.250057**	0.261859	0.26189	0.261859	0.179294	**0.507812**	**0.507812**	**0.500061**
	time (s)	**3492.98**	TO	TO	TO	TO	**23.1**	**449.1**	**411.42**
SAT fragment	# states	**6757**	9079	8581	16038	10158	**6684**	9131	11875
	# paths	**1973**	2446	2211	4434	2728	**3074**	1956	604
	prob.	**0.250548**	0.165949	0.0764908	0.0866818	0.0653038	**0.500977**	0.0715447	0.0378418
	time (s)	**805.68**	TO	TO	TO	TO	**3584.47**	TO	TO
SAT fragment + H	# states	**2489**	**7535**	19132	19662	5898	**6684**	8254	7009
	# paths	**700**	**2166**	5573	5556	1704	**3073**	1807	537
	prob.	**0.2858817**	**0.350044**	0.0971835	0.0648511	0.0562423	**0.500977**	0.092741	0.038967
	time (s)	**192.33**	**4172.44**	TO	TO	TO	**5152.11**	TO	TO

Fig. 2. Results for crowds and contract signing (TO $> 2h$)

The PCTL property we consider is $\mathbb{P}_{\leq p}(\Diamond Pos)$. The models are parameterized in their size by the number of sessions and the size of the crowd.

In Figure 2 we have collected a number of results we achieved on different instances of the described case studies. For the input data, we list the number of states (# states), the actual model checking result of reaching target states (model checking) and the probability threshold. We tested the methods for symbolic counterexample generation described in this paper as well as the bounded model checking approach, which computes a set of paths [5]:

- Symb global: The symbolic global search approach, Section 5.1
- Symb fragment: The symbolic fragment search approach, Section 5.2
- BMC classic: The standard bounded model checking approach for DTMCs as described in [5]
- SAT global: The global search approach using SAT solvers, Section 4.1
- SAT fragment: The fragment search approach using SAT solvers, Section 4.2
- SAT fragment + H: The SAT-based fragment search approach together with the SAT heuristic preferring more probable paths, Section 4.3.

For the resulting critical subsystems we present the number of states, the number of performed path searches (# paths), the probability of this system (prob), and the computing time in seconds (time (s)). The timeout (TO) was defined as 2 hours. All results which were finished within this time are printed in **boldface**. For unfinished cases we give the results that were achieved so far. Note that the probability for these unfinished benchmarks lies under the probability threshold.

In Figure 1 we present the number of MTBDD nodes for original instances of the CROWDS-protocol w. r. t. the number of explicit nodes presented by these MTBDDs. The figure shows, that the number of nodes highly increases while the number of nodes for the subsystems stay relatively constant.

The results show that the symbolic fragment search outperforms all other approaches by far on our benchmarks sets. We can compute critical subsystems for benchmarks consisting of millions of states. A result could still be computed for a system having over $1.2 \cdot 10^8$ states in about 3 hours. The explicit counterexample algorithms described in [13,29] were faster on small benchmarks but explicit approaches are not applicable to benchmarks as large as presented here.

7 Conclusion and Future Work

In this paper we presented a new framework for the generation of probabilistic counterexamples for symbolic DTMC representations. We suggested several methods, while the symbolic fragment search turned out to be the best alternative. Our experiments showed that using our framework the size of possible input systems for counterexample generation is increased by orders of magnitude.

In the future we want to integrate this symbolic framework into the COMICS tool [29] for counterexample generation for DTMCs. The adaption of the hierarchical abstraction techniques presented in [13] would increase the usability of counterexamples even for very large systems. It would also be interesting to see if using an SMT solver instead of a SAT solver would accelerate the search process.

References

1. Hermanns, H., Wachter, B., Zhang, L.: Probabilistic CEGAR. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 162–175. Springer, Heidelberg (2008)
2. Chadha, R., Viswanathan, M.: A counterexample-guided abstraction-refinement framework for Markov decision processes. ACM TOCL 12(1), 1–45 (2010)
3. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6(5), 512–535 (1994)
4. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)
5. 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)
6. Braitling, B., Wimmer, R., Becker, B., Jansen, N., Ábrahám, E.: Counterexample Generation for Markov Chains Using SMT-Based Bounded Model Checking. In: Bruni, R., Dingel, J. (eds.) FMOODS/FORTE 2011. LNCS, vol. 6722, pp. 75–89. Springer, Heidelberg (2011)
7. Günther, M., Schuster, J., Siegle, M.: Symbolic calculation of k-shortest paths and related measures with the stochastic process algebra tool CASPA. In: Proc. of DYADEM-FTS 2010, pp. 13–18. ACM Press (2010)
8. Han, T., Katoen, J.P., Damman, B.: Counterexample generation in probabilistic model checking. IEEE Trans. on Software Engineering 35(2), 241–257 (2009)
9. Kattenbelt, M., Huth, M.: Verification and refutation of probabilistic specifications via games. In: Proc. of FSTTCS 2009. LIPIcs, vol. 4, pp. 251–262. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2009)

10. Fecher, H., Huth, M., Piterman, N., Wagner, D.: PCTL model checking of Markov chains: Truth and falsity as winning strategies in games. Performance Evaluation 67(9), 858–872 (2010)
11. 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)
12. 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)
13. Jansen, N., Ábrahám, E., Katelaan, J., Wimmer, R., Katoen, J.-P., Becker, B.: Hierarchical Counterexamples for Discrete-Time Markov Chains. In: Bultan, T., Hsiung, P.-A. (eds.) ATVA 2011. LNCS, vol. 6996, pp. 443–452. Springer, Heidelberg (2011)
14. Wimmer, R., Jansen, N., Ábrahám, E., Becker, B., Katoen, J.-P.: Minimal Critical Subsystems for Discrete-Time Markov Models. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 299–314. Springer, Heidelberg (2012)
15. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers 35(8), 677–691 (1986)
16. Fujita, M., McGeer, P.C., Yang, J.C.Y.: Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. Formal Methods in System Design 10(2/3), 149–169 (1997)
17. Baier, C., Clarke, E.M., Hartonas-Garmhausen, V., Kwiatkowska, M.Z., Ryan, M.: Symbolic Model Checking for Probabilistic Processes. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256, pp. 430–440. Springer, Heidelberg (1997)
18. Parker, D.: Implementation of Symbolic Model Checking for Probabilistic Systems. PhD thesis, University of Birmingham (2002)
19. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6(5), 512–535 (1994)
20. Katoen, J.P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. Performance Evaluation 68(2), 90–104 (2011)
21. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of Probabilistic Real-Time Systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011)
22. Kwiatkowska, M., Norman, G., Parker, D.: The PRISM benchmark suite. In: Proc. of QEST. IEEE CS (September 2012)
23. Tseitin, G.S.: On the complexity of derivations in the propositional calculus. Studies in Mathematics and Mathematical Logic (Part II), 115–125 (1968)
24. Schmid, W.: Berechnung kürzester Wege in Straßennetzen mit Wegeverboten. PhD thesis, Universität Stuttgart, Fakultät für Bauingenieur- und Vermessungswesen (2000)
25. Somenzi, F.: Cudd: Cu decision diagram package release 2.4.1 (2005)
26. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
27. Norman, G., Shmatikov, V.: Analysis of probabilistic contract signing. Journal of Computer Security 14(6), 561–589 (2006)
28. Reiter, M.K., Rubin, A.D.: Crowds: Anonymity for web transactions. ACM Trans. on Information and System Security 1(1), 66–92 (1998)
29. Jansen, N., Ábrahám, E., Volk, M., Wimmer, R., Katoen, J.-P., Becker, B.: The COMICS Tool – Computing Minimal Counterexamples for DTMCs. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, vol. 7561, pp. 349–353. Springer, Heidelberg (2012)