DTMC Model Checking by SCC Reduction

Erika Ábrahám*, Nils Jansen†, Ralf Wimmer‡, Joost-Pieter Katoen*, and Bernd Becker†

*RWTH Aachen University, Germany
abraham|nils.jansen|katoen@cs.rwth-aachen.de

†Albert-Ludwigs-University Freiburg, Germany
wimmer|becker@informatik.uni-freiburg.de

Abstract—Discrete-Time Markov Chains (DTMCs) are a widely-used formalism to model probabilistic systems. On the one hand, available tools like PRISM or MRMC offer efficient model checking algorithms and thus support the verification of DTMCs. However, these algorithms do not provide any diagnostic information in the form of counterexamples, which are highly important for the correction of erroneous systems. On the other hand, there exist several approaches to generate counterexamples for DTMCs, but all these approaches require the model checking result for completeness.

In this paper we introduce a model checking algorithm for DTMCs that also supports the generation of counterexamples. Our algorithm, based on the detection and abstraction of strongly connected components, offers abstract counterexamples, which can be interactively refined by the user.

I. INTRODUCTION

An important advantage of model checking in general is the supply of diagnostic information. The result contains not only the information whether a system is correct regarding a certain property, but in case of incorrectness also a run of the system which leads to the erroneous behavior. This is crucial for the correction of a faulty system. For example, by the usual approach for LTL model checking (see e.g. [1]) a path is found that violates a certain property, this path can help the designer to reproduce and to localize the error when trying to fix the design. Such kind of diagnostic information is called a counterexample for the property. Edmund Clarke, who received the Turing award in 2007 for his pioneer work in verification, also stresses the importance of counterexamples for the correction of erroneous systems [2]:

“It is impossible to overestimate the importance of the counterexample feature. The counterexamples are invaluable in debugging complex systems. Some people use model checking just for this feature.”

A further application for counterexamples arises from abstraction refinement [3], [4]. The idea is to generalize states of a system (abstraction step) and try to prove the required property for this abstracted system. If the proof fails, this can be due to a too coarse abstraction. With the help of a counterexample those abstract states can be identified which have to be concretized for the property to hold (refinement step).

In this paper we deal with probabilistic model checking, i.e., the verification of systems where random choices are made. The classical approach for PCTL (Probabilistic Computation Tree Logic) model checking [5] for DTMCs (Discrete Time Markov Chains) is based on numerical matrix operations. The probabilities that a PCTL path formula holds when starting in a certain state are obtained as the solution of a linear equation system [5]. These algorithms are quite efficient; there exist several tools implementing them, e.g., MRMC [6] and PRISM [7], that can verify rather large models very fast. A main weakness of these numerical algorithms, however, is that they do not provide any diagnostic information but only the pure probabilities that a given property holds.

Alternative methods for checking the validity of a PCTL formula are based on Monte Carlo simulation [8], [9]. But – as far as we know – these methods do not provide counterexamples either.

Besides PCTL model checking for DTMCs we also address the generation of counterexamples. Recently, several approaches for the computation of counterexamples in the probabilistic setting were developed, e.g., [10], [11], [12], [13], [14]. All but the last of these approaches share the problem, that in order to guarantee completeness the model checking result has to be on hand, because, if a property is not refuted, the algorithm searching for counterexamples may not terminate. Whereas we focus on PCTL, the last work [14] deals with counterexamples for probabilistic LTL properties.

Contrary to LTL model checking for non-probabilistic systems where a counterexample is a single path, in probabilistic model checking a counterexample is formed by a set of paths whose probability measure exceeds a given threshold. In general, the number of paths needed to exceed the threshold can be prohibitively large [12]. To reduce their size, the detection of loops [13], the usage of regular expressions [12], and the abstraction of strongly connected components (SCCs) [11], thereby extending an approach presented in [15] and [16], where it is referred to as stochastic complementation, have been suggested.

Most related to our work is the latter approach [11]. Tarjan’s SCC algorithm [17] is executed on the underlying digraph of the DTMC which yields the SCC decomposition of this graph. For every non-bottom SCC, every input state of the SCC by which the SCC can be entered, and every output state outside the SCC
to which the SCC can be exited, the probabilities of reaching
the output state from the input state are computed. Therefore
standard PCTL model checking for unbounded reachability
properties is provided. Afterward, the non-bottom SCCs are
abstracted by transitions leading from the input states to the
output states having the computed probabilities. The result
is an acyclic DTMC which preserves unbounded reachability
probabilities for absorbing states. This digraph can be used to
determine an abstract counterexample.

In order to achieve PCTL model checking for unbounded
properties supporting stepwise refinable abstract counter-
examples we use the following approach. First we identify
the SCCs of the DTMC. Inside of each SCC we perform
again an SCC search whereby all input states are ignored.
This is done recursively up to trivial SCCs without loops.
After this top-down SCC detection the probabilities of leaving
the SCCs through the different output states are computed
in a bottom-up manner. Thereby the special restricted form
of the SCCs through the different output states are computed
again an SCC search whereby all input states are ignored.

The final result is a DTMC with transitions leading from the initial state to the target states
whose unbounded reachability probabilities we are interested
in. The transitions carry exactly the probability mass for
unbounded reachability to these target nodes. This induces
the designated model checking result.

During our model checking algorithm all information for all
stages of the recursion is stored. This allows a user who needs to
obtain a counterexample for the given reachability property
to formulate an abstract counterexample. This set of abstract
paths can be arbitrarily refined with respect to the recursion
structure. In the end this yields very tight counterexamples
which are reasonably built according to the structure of the
DTMC.

We implemented our algorithm and show its applicability
using the well-known case studies of the leader election
protocol [18] and the crowds protocol [19]. We did model
checking for several instances of these protocols and received
very promising results.

This paper is structured as follows: In Section II we give
some preliminaries about probabilistic model checking. In
Section III we introduce our novel model checking method,
give a proof of its correctness, and explain it on an example. In
Section IV we describe how to use thestored information
to extract counterexamples. We present results for some
benchmarks in Section V. Conclusion and outlook on future
work is the content of the last Section VI.

II. PRELIMINARIES

In this section we give a brief introduction to discrete-time
Markov chains and to probabilistic temporal logic.

Discrete-time Markov chains are a frequently used modeling
formalism for stochastic systems with a discrete state space
and a discrete model of time.

Definition 1: A discrete-time Markov chain (DTMC) is a
tuple \( M = (S, s_{init}, P, L) \) with \( S \) a non-empty finite set of
states, \( s_{init} \in S \) the initial state, \( P : S \times S \rightarrow [0,1] \) a function
with \( \sum_{s' \in S} P(s, s') = 1 \) for all \( s \in S \), and \( L : S \rightarrow 2^{AP} \)
a labeling function assigning to each state a set of atomic
propositions from a denumerable proposition set \( AP \).

For simplicity, we restrict ourselves to a unique initial state,
but the concepts introduced in this paper are also applicable to
initial distributions. The function \( P \), known as the transition
probability matrix, gives the probabilities \( P(s, s') \) for taking a
transition from a state \( s \) to another state \( s' \), forming a
discrete probability distribution for each state. A state \( s \)
is called absorbing iff \( P(s, s) = 1 \).

A path of a DTMC \( M \) 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 \( i \).
Let \( Paths(M, s) \) denote the set of all paths of \( M \) starting in
state \( s \) and let \( Paths_{fin}(M, s) \) denote the set of all finite
paths of \( M \) starting in \( s \). We also use \( Paths(M) \) and \( Paths_{fin}(M) \)
to denote all paths resp. all finite paths of \( M \) starting in an
arbitrary state. We say that a state \( s' \) is reachable from another
state \( s \) iff there is a finite path from \( s \) to \( s' \). The length \( |\pi| \)
of a path \( \pi \) is \( \infty \) if the path is infinite and for finite paths
\( \pi = s_0 s_1 \ldots s_n \) it is defined as \( n \).

The cylinder set of a finite path \( \pi \) of \( M \) is defined as
\( Cyl(\pi) = \{ \pi' \in Paths(M) | \pi' \text{ is infinite and } \pi \text{ is a prefix of } \pi' \} \). As usual, we associate to each DTMC \( M \) the smallest
\( \sigma \)-algebra that contains all cylinder sets of all finite paths of
\( M \). This gives us a unique probability measure \( Pr^M \) (or short
\( Pr \)) on the \( \sigma \)-algebra where the probabilities of the cylinder
sets are given by

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

For finite paths \( \pi \in Paths_{fin}(M, s_0) \) we set \( Pr_{fin}(\pi) = Pr(Cyl(\pi)) \). For sets of paths \( R \subseteq Paths_{fin}(M, s) \) we define
\( Pr_{fin}(R) = \sum_{\pi \in R} Pr_{fin}(\pi) \) with \( R' = \{ \pi \in R | \forall \pi' \in R, \pi' \text{ is not a prefix of } \pi \} \). Note that for an infinite path set
the definition may involve an infinite sum, but it always defines
a probability mass between 0 and 1. Note furthermore that
\( Pr_{fin}(\pi s_1 s_2 \pi) = Pr_{fin}(\pi s_1) \cdot Pr_{fin}(s_2) \). Similarly for sets \( R_1 \) and \( R_2 \) of paths, if \( R_1' \subseteq R_1 \) and \( R_2' \subseteq R_2 \) are the sets of
those paths in \( R_1 \) resp. \( R_2 \) that have no prefixes in \( R_1 \) resp.
\( R_2 \), then if all paths in \( R_1' \) end in the same state \( s \) and all
paths in \( R_2' \) start in \( s \), then \( Pr_{fin}(\{s_1 s_2 | s \in R_1' \wedge s s_2 \in R_2' \}) = Pr_{fin}(R_1) \cdot Pr_{fin}(R_2) \).

The Probabilistic Computation Tree Logic (PCTL) [5] is an
adaptation of CTL to probabilistic systems with the abstract
syntax

\[
\varphi ::= p \mid \neg \varphi \mid \varphi \land \varphi \mid \exists \lambda (\varphi U \varphi)
\]

with \( p \in AP \) an atomic proposition, \( \lambda \in \{0,1\} \subseteq \mathbb{R} \) a
probability threshold, and \( \sim \in \{<,\leq,\geq,>\} \) is a comparison
operator. The \( U \) is the classical “until” operator. Syntactic
sugar like \( \Diamond \varphi ::= \text{true U } \varphi \) can be defined as usual. The
“always” operator is derived by \( P_{\leq p}(\Box \varphi) = P_{\geq 1-p}(\Diamond \neg \varphi) \). The
probability operator \( P \) allows to express probability thresholds on
the probability mass of paths satisfying a formula. In PCTL
one can formulate properties like “an erroneous state will be
reached with a probability less than 0.01”.

Given a DTMC $M$ and a property $\mathbb{P}_{\preceq \lambda} (\varphi_1 \cup \varphi_2)$, a counterexample consists in a set $C$ of finite paths such that all paths from their associated cylinder sets satisfy $\varphi_1 \cup \varphi_2$. The finite paths are not prefixes of each other, i.e., they have disjoint cylinder sets. The total probability mass of $C$ has to be larger than $\lambda$. For formulae of the form $\mathbb{P}_{\preceq \lambda} (\varphi_1 \cup \varphi_2)$ the case is similar but the probability mass has to be at least $\lambda$.

Counterexamples for $\mathbb{P}_{\geq \lambda} (\Diamond \varphi)$ can be reduced to the former cases. Here, a counterexample consists of a set of paths violating $\Diamond \varphi$ and having a probability measure larger than $1 - \lambda$. This corresponds to a counterexample for $\mathbb{P}_{\leq \lambda} (\Box \lnot \varphi)$. The case of $>$ instead of $\geq$ can be handled analogously.

The remaining cases – a lower bound on the probability for $\varphi_1 \cup \varphi_2$ or an upper bound on the probability for $\Diamond \varphi$ – have to be handled differently. Their violation cannot be explained by the cylinder sets of a finite set of finite paths, but by the complement thereof. Han et al. [12] point out a way how these properties can nevertheless be reduced to the former cases by a simple transformation of the Markov chain and its labeling.

Therefore we restrict ourselves in the following to formulae $\mathbb{P}_{\sim \lambda} (\varphi_1 \cup \varphi_2)$ with $\sim \in \{\leq, <\}$.

The usual approach to check properties of the form $\mathbb{P}_{\sim \lambda} (\varphi_1 \cup \varphi_2)$ is to (1) generate a labeling for the subformulae $\varphi_1$ and $\varphi_2$, possibly by recursively invoking probabilistic model checking for subformulae, (2) reduce the DTMC such that all states satisfying $\lnot \varphi_1 \land \lnot \varphi_2$ or $\varphi_2$ get absorbing, and (3) compute the probability of reaching a $\varphi_2$ state from an initial state in the reduced DTMC. We also follow this approach, and concentrate in the following on the algorithm for point (3), i.e., to check the validity of formulae of the form $\mathbb{P}_{\sim \lambda} (\Diamond \, p)$ with $p \in AP$ and $\sim \in \{\leq, <\}$ for DTMCs with all $p$-states being absorbing.

### III. SCC-based Abstraction

Assume in the following a DTMC $M$ and a formula $\mathbb{P}_{\sim \lambda} (\Diamond \, p)$ with $p \in AP$ and $\sim \in \{\leq, <\}$, such that all $p$-states of $M$ are absorbing. In this section we introduce our approach to compute the probability of reaching a $p$-state from the initial state of $M$ (and thus decide whether the formula holds for $M$). Instead of using the standard model checking algorithm, which is based on the solution of a linear equation system, we use a technique which can be intuitively described as follows.

We detect the strongly connected components (SCCs) of $M$ and for each SCC we abstract the set of all path fragments through the SCC by some abstract “meta”-transitions.

This abstraction is computed in a recursive manner: to do the abstraction for an SCC we consider the SCC without the states by which we can enter it, determine the SCCs within this restricted system, and apply the algorithm recursively to them. For the example DTMC in Figure 1, to abstract the SCC $S_1$ we would first abstract the SCCs $S_{1.1}$ and $S_{1.2}$, whereby for the latter we would first compute the abstraction for $S_{1.2.1}$.

For the inner-most SCCs (in the example $S_{1.1}$ and $S_{1.2.1}$) we observe that paths entering them will also exit them with probability 1. Furthermore, each path fragment moving through an SCC and finally exiting it consists of a (possibly empty) prefix of cycles leading back to the first state and a postfix not containing the first state. For the inner-most SCCs those postfixes are cycle-free, and thus it is easy to compute their probability mass for each starting state and exit state pair. Having that information, in a first abstraction level we abstract paths through the SCC by (1) an abstract “meta”-self-loop on the starting state of the SCC and (2) abstract “meta”-transitions leading from the starting state to the exit states, where the probability for the self-loop (1) is 1 minus the total probabilities of (2). In a second abstraction level we eliminate the self-loop, yielding a completely cycle-free sub-system (see Figure 2).

After completing this procedure for, e.g., $S_{1.2.1}$, the SCC $S_{1.2}$ becomes an inner-most SCC for which we repeat the above procedure. After the algorithm has terminated, the resulting DTMC has only “meta”-transitions from the initial state to the absorbing states, labeled with the corresponding reachability probabilities. This procedure is closely related to the state elimination approach for obtaining regular expressions from finite automata [20], [21]. This method was applied to Markov chains in [22], [23], [24]. Note that these approaches eliminate states randomly or according to some heuristics [25].
To formalize our algorithm, we first introduce some more concepts. Let $M = (S, s_{\text{init}}, P, L)$ be a DTMC. Similarly to the notation in [11], for a set $K \subseteq S$ we define the set $\text{Inp}^M(K) \subseteq K$ (or short $\text{Inp}(K)$) of input states of $K$ in $M$ to be the set of all states $s' \in K$ such that there exists a path $s_0 \ldots s_m$ in $M$ with $s_0 = s$, $s_m = s'$, and $s_i \in K$ for all $0 \leq i < m$. A bottom connected subgraph $K$ is a connected subgraph with no transitions leading out of it, i.e., $\text{Out}(K) = \emptyset$. We call a connected subgraph consisting of a single state without a self-loop trivial. A DTMC is acyclic iff it has trivial connected subgraphs only. A maximal (with respect to $\subseteq$) connected subgraph is called a strongly connected component (SCC). An SCC is called a bottom1 resp. trivial SCC if it is a bottom resp. trivial connected subgraph.

Now we formalize the abstraction of all paths leading through a connected subgraph by “meta”-transitions from the input to the output states. We keep the statements general and formalize them for arbitrary sets of states instead of connected subgraphs, though the main application will be for connected subgraphs.

Definition 2: Let $M = (S, s_{\text{init}}, P, L)$ be a DTMC and $K \subseteq S$. Let for all $s_0 \in \text{Inp}(K)$ and $s_{\text{out}} \in \text{Out}(K)$ be $P^M_{\text{fin}}(s_0, s_{\text{out}}) = P^M_{\text{fin}}(\{s_0 \ldots s_m \in \text{Paths}_{\text{fin}}(M) \mid s_0 = s_{\text{in}} \land s_m = s_{\text{out}} \land \forall 0 \leq i < m. s_i \in K\})$. We define the path abstraction $M_K = (S_K, s_{\text{init}}, P_K, L_K)$ of $M$ by

$$S_K = (S \setminus K) \cup \text{Inp}(K),$$

$$P_K(s, s') = \begin{cases} P^M(s, s') \quad &\text{if } s \in \text{Inp}(K) \land s' \in \text{Out}(K) \\ 0 \quad &\text{if } s \in \text{Inp}(K) \land s' \not\in \text{Out}(K) \\ P(s, s') \quad &\text{otherwise}, \end{cases}$$

$$L_K(s) = L(s) \text{ for all } s \in S_K.$$

For a finite path $\pi = s_0 \ldots s_m$ of $M$ we denote by $\pi|_{M_K}$ the state sequence resulting from $\pi$ when replacing each maximal subsequence $s_is_{i+1} \ldots s_j$ of states from $K$ in $\pi$ by $s_i$. Note that $\pi|_{M_K}$ is a finite path of $M_K$.

Since the computation of the probabilities is clearly easier for cycle-free state sets, we use the notation $\text{Reduce}(M,K)$ for the path abstraction $M_K$ if $K$ is cycle-free.

The following lemma states that the abstraction of $M$ to $M_K$ preserves reachability probabilities outside $K$.

Lemma 1: Let $M = (S, s_{\text{init}}, P, L)$ be a DTMC, $K \subseteq S$, and $M_K = (S_K, s_{\text{init}}, P_K, L_K)$ the path abstraction of $M$ for $K$. For all $s \in S_K$ and all $\pi_K \in \text{Paths}_{\text{fin}}(M_K, s)$ we have $P^M_{\text{fin}}(\pi_K) = P^M_{\text{fin}}(\{\pi \in \text{Paths}_{\text{fin}}(M, s) \mid \pi|_{M_K} = \pi_K\})$.

Proof: By induction on the number $n$ of transitions from an input state of $K$ to an output state of $K$ in $\pi_K = s_0 \ldots s_m$, i.e., on $n = \{i \in \mathbb{N} \mid 0 \leq i < m \land s_i \in \text{Inp}^M(K) \land s_{i+1} \in \text{Out}^M(K)\}$.

For $n = 0$, let $\pi_K = s_0 \ldots s_m \in \text{Paths}_{\text{fin}}(M_K, s)$ for an $s \in S_K$ such that $s_i \not\in \text{Inp}(K)$ or $s_{i+1} \not\in \text{Out}(K)$ for all $i = 0, \ldots, m-1$. Note that in $\pi_K$ only $s_m$ may be contained in $\text{Inp}(K)$. The set $\{\pi \in \text{Paths}_{\text{fin}}(M, s) \mid \pi|_{M_K} = \pi_K\}$ consists of just $\pi_K$ if $s_m \not\in \text{Inp}(K)$ and otherwise of all paths of the form $\pi_K \pi'$ with $\pi'$, being possibly the empty sequence, having states from $K$ only. In both cases, $\pi_K$ is a prefix of all paths in the set, and thus $P^M_{\text{fin}}(\pi_K) = P^M_{\text{fin}}(\pi_K) = P^M_{\text{fin}}(\pi_K) = P^M_{\text{fin}}(\pi_K) = P^M_{\text{fin}}(\pi_K)$ by the definitions of $P^M_{\text{fin}}$ and $M_K$.

Assume now that the lemma holds for all $s \in S_K$ and all paths from $\text{Paths}_{\text{fin}}(M_K, s)$ with $k$ transitions from an input state of $K$ to an output state of $K$. Let $s \in S_K$ and $\pi_K = s_0 \ldots s_m \in \text{Paths}_{\text{fin}}(M_K, s)$ a path with $k+1$ such transitions, and let $i$ be the smallest index with $s_i \in \text{Inp}^M(K)$ and $s_{i+1} \in \text{Out}^M(K)$.

Let $R_K = \{s_0' \ldots s_{m'}' \in \text{Paths}_{\text{fin}}(M) \mid s_0' = s_i \land s_{m'}' = s_{i+1} \land \forall 0 \leq j < m' \land s'_j \in K\}$ be the set of those paths whose probability mass defines the probability of the transition from $s_i$ to $s_{i+1}$ in $M_K$. Note that none of the paths in $R_K$ has a prefix in $R_K$, since all states but the last are in $K$, and the last states $s_{m'} = s_{i+1} \not\in K$.

Let furthermore $R' = \{\pi \in \text{Paths}_{\text{fin}}(M, s_{i+1}) \mid \pi|_{M_K} = s_{i+1} \ldots s_m \}$ be the set of finite paths of $M$ that behave outside $K$ as $\pi_K$ after the first transition from an input to an output state of $K$. Let $R' \subseteq R$ be the subset of those paths in $R$ that does not have a prefix in $R$. Note that a path in $R$ can have a prefix in $R$ only if $s_m \in \text{Inp}(K)$, and in this case the extension consists of states from $K$ only. Thus $R' = \{s_0' \ldots s_{m'}' \in R \mid s_m \not\in \text{Inp}(K) \lor m' = 0 \lor s_{m'-1} \not\in K\}$.

We have

$$P^M_{\text{fin}}(\pi_K) = P^M_{\text{fin}}(s_0 \ldots s_m) =$$

$$= \prod_{j=0}^{m} P^M(s_j, s_{j+1}) \cdot P^M_{\text{fin}}(s_i, s_{i+1}) \cdot \prod_{j=i+1}^{m} P^M(s_j, s_{j+1})$$

$$= P^M_{\text{fin}}(s_0 \ldots s_{i-1}) \cdot P^M_{\text{fin}}(R_K) \cdot P^M_{\text{fin}}(R)$$

$$= P^M_{\text{fin}}(s_0 \ldots s_i) \cdot \left(\sum_{\pi \in R_K} P^M_{\text{fin}}(\pi)\right) \cdot \left(\sum_{\pi \in R'} P^M_{\text{fin}}(\pi)\right)$$

$$= \sum_{\pi \in \text{Paths}_{\text{fin}}(M, s_0)} P^M_{\text{fin}}(s_0 \ldots s_i \pi_{1+i+1}) \cdot P^M_{\text{fin}}(s_i \pi_{1+i+1})$$

$$= \sum_{\pi \in \text{Paths}_{\text{fin}}(M, s_0)} P^M_{\text{fin}}(s_0 \ldots s_i \pi_{1+i+1}) \cdot P^M_{\text{fin}}(s_i \pi_{1+i+1})$$

Equation (1) is by definition of $P^M_{\text{fin}}$. Equation (2) uses

$$P^M_{\text{fin}}(s_{i+1} \ldots s_m) = P^M_{\text{fin}}(s_0 \ldots s_i)_1$$

and, from the minimality of $i$ we conclude that $M$ and $M_K$ have the same probability distributions for $s_0, \ldots, s_{i-1}$ and thus $P^M_{\text{fin}}(s_0 \ldots s_i) = P^M_{\text{fin}}(s_0 \ldots s_i)$. Furthermore, since $s_i \in \text{Inp}(K)$ and $s_{i+1} \in \text{Out}(K)$, $P^M_{\text{fin}}(s_i, s_{i+1})$ equals $P^M_{\text{fin}}(R_K)$ by the definition of $P_K$. For the third term, $P^M_{\text{fin}}(R')$ by induction. Equation (3) is due to the definition of $P^M_{\text{fin}}$. For Equation (4) note that all paths in $R_K$ start in $s_i$ and end in $s_{i+1}$, and all paths in $R'$ start in $s_{i+1}$. For Equation

1Note that all bottom connected subgraphs are bottom SCCs.
(5) we use the definition of $Pr_{in}$, and for Equation (6) the definitions of $R_{K}$ and $R'$.

Thus given a DTMC $M = (S, s_{init}, P, L)$, we can define a general model checking algorithm shown in Algorithm 1 (which we refine in the following). The algorithm uses path abstraction to construct from $M$ a cycle-free DTMC $M'$. For the latter it is easy to compute the path abstraction $Reduce(M', K)$ with $K$ the set of all non-absorbing states of $M'$. The result is a DTMC that has transitions from the initial state of $M$ to the absorbing states of $M$ only. The probability distribution of the initial state $s_{init}$ gives for each absorbing state $s$ the total probability of reaching $s$ from $s_{init}$ in $M$.

Algorithm 1

Model_check(DTMC $M$)
begin
  Determine the set $C$ of all non-trivial SCCs of $M$;\hspace{1cm} (1)
  while $C \neq \emptyset$ do
    Choose $K \in C$; $C := C\setminus \{K\}$;\hspace{1cm} (2)
    $M := Path_{abstraction}(M, K)$;\hspace{1cm} (3)
  end while
  Let $K$ be the set of all non-absorbing states of $M$;\hspace{1cm} (4)
  $M := Reduce(M, K)$;\hspace{1cm} (5)
  $R := Path_{abstraction}(M, K)$;\hspace{1cm} (6)
  Return $M$;\hspace{1cm} (7)
end

Path_abstraction(DTMC $M$, connected subgraph $K$)
begin
  $M := MK$;\hspace{1cm} (1)
  Return $M$;\hspace{1cm} (2)
end

Though the computation of the path abstraction $Path_{abstraction}(M, K)$ is straightforward if $K$ is cycle-free, the question arises how to compute $M_K$ if $K$ is not cycle-free, e.g., an SCC or a connected subgraph. One possibility would be to use the standard algorithm based on the solution of a linear equation system. However, this method would not directly support the generation of intuitive counterexamples. We use a recursive approach instead, as depicted by the refined Algorithm 2.

Algorithm 2

Path_abstraction(DTMC $M$, connected subgraph $K$)
begin
  $C := $ set of all non-trivial maximal connected subgraphs of $K'$ of $M$ with $K' \subseteq K\setminus Inp(K)$;\hspace{1cm} (1)
  while $C \neq \emptyset$ do
    Choose $K' \in C$; $C := C\setminus \{K'\}$;\hspace{1cm} (2)
    $M := Path_{abstraction}(M, K')$;\hspace{1cm} (3)
  end while
  $M := MK$;\hspace{1cm} (4)
  Return $M$;\hspace{1cm} (5)
end

Algorithm 2 defines a refined Path_abstraction method: it computes the path abstraction for a connected subgraph $K$ by recursively computing the path abstraction for all maximal connected subgraphs of $K\setminus Inp(K)$ (see again Figure 1). After this recursive abstraction, $K$ is in general not yet cycle-free: it may still contain cycles on the input states of $K$. However, since all connected subgraphs of $K\setminus Inp(K)$ are abstracted, $K$ has much less transitions in the resulting system than before the recursive abstraction.

Let $M = (S, s_{init}, P, L)$ be the DTMC after the abstraction of all maximal connected subgraphs of $K\setminus Inp(K)$. To finalize the abstraction, we need to compute the probabilities of reaching an output state of $K$ from an input state of $K$ by paths through $K$. We distinguish between the following cases:

a) $|Out(K)| = 1$, i.e., there is a single output state of $K$, or
b) $|Out(K)| > 1 \land |Inp(K)| = 1$, or
c) $|Out(K)| > 1 \land |Inp(K)| > 1$.

a) Single output: In the first case, as $K$ is not a bottom connected subgraph, all paths entering $K$ will leave it with probability 1. That means, the probability of reaching the output state of $K$ from any of the input states of $K$ is 1.

b) Single input/multiple output: For the second case, let $Inp(K) = \{s_{in}\}$ and $Out(K) = \{s_{out}^1, \ldots, s_{out}^n\}$. Note that all finite paths starting in $s_{in}$, moving in $K$, and ending in one of the output states $s_{out}^i$ consist of a prefix of cycles on $s_{in}$ and a postfix not containing $s_{in}$. We define $R_{in} = \{s_0 \ldots s_m \in Paths_{in}(M) | s_0 = s_{in} \land \forall 0 \leq j \leq m, s_j \in K\}$ for the set of possible prefixes and for all $i = 0, \ldots, n$ $R_{out}^i = \{s_0 \ldots s_m \in Paths_{out}(M) | s_0 = s_{in} \land s_m = s_{out}^i \land (\forall 1 \leq j < m, s_j \in K \land s_j \neq s_{out}^i)\}$ for the set of possible postfixes ending in $s_{out}^i$. As $M$ is the result of the path abstraction for all connected subgraphs in $K\setminus Inp(K)$, $R_{out}^i$ consists of a finite number of cycle-free postfixes for each $i$; we can efficiently compute the probabilities $p_{out} = Pr_{fin}(R_{out}^i)$ for them. Note also that no path in $R_{out}^i$ has a prefix in $R_{out}^j$. The following holds for the probabilities of the abstraction:

$$\begin{align*}
  p_{out}^K(s_{in}, s_{out}^i) &= Pr_{fin}(\{\pi_1s_{in}\pi_2 | \pi_1s_{in} \in R_{in} \land s_1s_2 \in R_{out}^i\}) = \\
  &= \sum_{\pi_1s_{in} \in R_{in}} \sum_{s_2 \in R_{out}^i} Pr_{fin}(\pi_1s_{in}) \cdot Pr_{fin}(s_2) = \\
  &= \sum_{\pi_1s_{in} \in R_{in}} \sum_{s_2 \in R_{out}^i} Pr_{fin}(\pi_1s_{in}) \cdot Pr_{fin}(s_2) = \\
  &= \sum_{\pi_1s_{in} \in R_{in}} \sum_{s_2 \in R_{out}^i} Pr_{fin}(\pi_1s_{in}) \cdot p_{out}^i.
\end{align*}$$

Since $K$ is non-bottom, paths entering $K$ will leave $K$ with probability 1:

$$1 = \sum_{i=0}^{m} p_{out}^K = \sum_{i=0}^{m} p_{in} \cdot p_{out}^i = p_{in} \cdot \sum_{i=0}^{m} p_{out}^i.$$ 

Thus if we know the probabilities $p_{out}^i$, then $p_{in}$ can be determined by $1/(\sum_{i=0}^{m} p_{out}^i)$. We make use of this to build the path abstraction for $M$ as follows:

- Compute $p_{out}^i$ for all $i = 0, \ldots, m$;
- Compute $p_{in} = 1/(\sum_{i=0}^{m} p_{out}^i)$;
- Define $p_{out}^K(s_{in}, s_{out}^i) = p_{in} \cdot p_{out}^i$.
c) Multiple input/multiple output: For the last case when $K$ has several input states $\text{Inp}(K) = \{s_{in}^0, \ldots, s_{in}^n\}$ and several output states $\text{Out}(K) = \{s_{out}^0, \ldots, s_{out}^m\}$, we could also apply the above approach for each input state. However, the computation costs would be polynomial in the number of input and output states. Instead, we use a parametrized version of the standard model checking approach based on solving a linear equation system: We define for each state $s \in K$ an equation
\[
p_s = \sum_{P(s,s') > 0} P(s,s') \cdot p_{s'}
\]
with a variable $p_s$ for each $s \in K \cup \text{Out}(K)$. We eliminate the variables $p_s$ for all $s \in K \setminus (\text{Inp}(K) \cup \text{Out}(K))$ from the equation system. The result is a set of equations, one for each input state $s_{in}^i$, of the form
\[
p_{s_{in}^i} = \sum_{s_j \in \text{Inp}(K) \cup \text{Out}(K)} \left( c_{s_{in}^i,s_j} \cdot p_{s_j} \right).
\]
Each of these equations gets reformulated to
\[
p_{s_{in}^i} = \sum_{s_j \in (\text{Inp}(K) \cup \text{Out}(K)) \setminus \{s_{in}^i\}} \left( c_{s_{in}^i,s_j} \cdot p_{s_j} \right)
\]
and we eliminate $p_{s_{in}^i}$ from the right-hand sides of all other equations. This gives us a set of equations, one for each input node, of the form
\[
p_{s_{in}^i} = \sum_{j = 0}^m \left( c_{s_{in}^i,s_{out}^j} \cdot p_{s_{out}^j} \right)
\]
where $c_{s_{in}^i,s_{out}^j}$ has the value of $p_{K}^M(s_{in}^i,s_{out}^j)$, defining the needed probabilities for each input-output state-pair.

The final algorithm for the computation of the path abstraction is given in Algorithm 3.

**Algorithm 3**

Path_abstraction(DTMC $M$, connected subgraph $K$)
begin
$C :=$ set of all non-trivial maximal connected subgraphs(1) $K'$ of $M$ with $K' \subseteq K \setminus \text{Inp}(K);$ (2)
while $C \neq \emptyset$ do
Choose $K' \in C$; $C := C \setminus K'$; (3)
$M := \text{Path_abstraction}(M, K')$; (4)
end while
if $|\text{Out}(K)| = 1$ then
$M := M_K$ applying method a); (5)
else if $|\text{Inp}(K)| = 1$ then
$M := M_K$ applying method b); (6)
else
$M := M_K$ applying method c); (7)
end if
Return $M$; (8)
end

The soundness of this algorithm is given by Lemma 1 and the subsequent argumentation, as $K$ is built by the union of all SCCs of the input DTMC. Completeness is given by the fact, that the number of connected subgraphs is bounded by the number of nodes and therefore termination is always guaranteed.

**Example 1:** We apply our approach to the DTMC $M$ depicted in Figure 1. The Model_check($M$) method performs an SCC search on the whole DTMC which contains a single SCC $S_1 = \{s_1, s_2, s_3, s_4, s_5, s_7, s_8\}$. Path_abstraction($M, S_1$) will be invoked. The search for maximal non-trivial connected subgraphs in $S_1 \setminus \text{Inp}(S_1) = S_1 \setminus \{s_1\} = \{s_2, s_3, s_4, s_6, s_7, s_8\}$ yields $S_{1,1} = \{s_2, s_3, s_4\}$ and $S_{1,2} = \{s_6, s_7, s_8\}$.

Assume that first the abstraction Path_abstraction($M, S_{1,2}$) is invoked. The method detects $S_{1,2.1} = \{s_7, s_8\}$ as the only maximal non-trivial connected subgraph of $S_{1,2} \setminus \{s_6\} = \{s_7, s_8\}$.

As a result, the abstraction Path_abstraction($M, S_{1,2.1}$) is recursively called for $S_{1,2.1}$.

There is no non-trivial connected subgraph in $S_{1,2.1} \setminus \text{Inp}(S_{1,2.1}) = \{s_8\}$. Since $|\text{Inp}(S_{1,2.1})| = |\{s_7\}| = 1$ and $|\text{Out}(S_{1,2.1})| = |\{s_5, s_6, s_9\}| = 3$, method b) applies (see Figure 3(a) and 3(b)):

\[
p_{s_{out}}^{S_{1,2}} = 0.3, \quad p_{s_{out}}^{S_{1,2}} = 0.7 \cdot 0.1 = 0.07, \quad p_{s_{out}}^{S_{1,2}} = 0.7 \cdot 0.4 = 0.28,
\]

and with $p_{in} = 1/(p_{s_{out}}^{S_{1,2}} + p_{s_{out}}^{S_{1,2}} + p_{s_{out}}^{S_{1,2}})$ we get (see Figure 3(c)):

\[
p_{s_{in}}^{S_{1,2}}(s_7, s_5) = 0.3/0.65 \approx 0.46153846,
p_{s_{in}}^{S_{1,2}}(s_7, s_6) = 0.07/0.65 \approx 0.10769231,
p_{s_{in}}^{S_{1,2}}(s_7, s_9) = 0.28/0.65 \approx 0.43076923,
\]

and the control returns to the path abstraction of $S_{1,2}$.

As $|\text{Inp}(S_{1,2})| = |\{s_8\}| = 1$ and $|\text{Out}(S_{1,2})| = |\{s_5, s_6, s_9\}| = 3$, again method b) applies:

\[
p_{s_{out}}^{S_{1,2}} = 0.2 \cdot 0.13/0.65, \quad p_{s_{out}}^{S_{1,2}} = 0.8 \cdot (0.3/0.65) = 0.24/0.65, \quad p_{s_{out}}^{S_{1,2}} = 0.8 \cdot (0.28/0.65) = 0.224/0.65,
\]

and with $p_{in} = 1/(p_{s_{out}}^{S_{1,2}} + p_{s_{out}}^{S_{1,2}} + p_{s_{out}}^{S_{1,2}})$ we get

\[
p_{S_{1,2}}^{S_{1,2}}(s_6, s_1) = (0.13/0.65) \cdot (0.65/0.594) = 0.13/0.594 \approx 0.21885522,
p_{S_{1,2}}^{S_{1,2}}(s_6, s_5) = (0.24/0.65) \cdot (0.65/0.594) = 0.24/0.594 \approx 0.4040404,
p_{S_{1,2}}^{S_{1,2}}(s_6, s_9) = (0.224/0.65) \cdot (0.65/0.594) = 0.224/0.594 \approx 0.37710438.
\]

The recursive invocation Path_abstraction for $S_{1,1}$ for the other connected subgraph of $S_1$ can not detect any further non-trivial connected subgraphs in $S_{1,1} \setminus \text{Inp}(S_{1,1}) = S_{1,1} \setminus \{s_2, s_3\} = \{s_4\}$. Since $|\text{Inp}(S_{1,1})| = |\{s_2, s_3\}| = 2$ and $|\text{Out}(S_{1,1})| = |\{s_1, s_5, s_6\}| = 3$, method c) is used to compute $M_{S_{1,1}}$. From the equation system

\[
p_2 = 0.4p_1 + 0.4p_3 + 0.2p_6, \quad p_3 = 0.5p_4 + 0.5p_5, \quad p_4 = 1p_2
\]

we eliminate $p_4$ overall yielding

\[
p_2 = 0.4p_1 + 0.4p_3 + 0.2p_6, \quad p_3 = 0.5p_2 + 0.5p_5.
\]
Finally we eliminate $p_2$ and $p_3$ on the right-hand sides, resulting in

$$p_2 = 0.5p_1 + 0.25p_5 + 0.25p_6,$$
$$p_3 = 0.25p_1 + 0.625p_5 + 0.125p_6.$$

I.e., the needed probabilities for the abstraction are

$$p_{s_1}^{M}(s_2, s_1) = 0.5, \quad p_{s_1}^{M}(s_3, s_1) = 0.25,$$
$$p_{s_1}^{M}(s_5, s_2) = 0.25, \quad p_{s_1}^{M}(s_3, s_5) = 0.625,$$
$$p_{s_1}^{M}(s_5, s_6) = 0.25, \quad p_{s_1}^{M}(s_3, s_6) = 0.125.$$

After the abstraction of $S_{1.1}$ is completed, the control returns to the path abstraction of $S_1$.

Since $|\text{Input}(S_1)| = \{s_1\} = 1$ and $|\text{Output}(S_1)| = \{s_5, s_9\} = 2$, we again apply method b) yielding

$$p_{\text{out}}^{\text{refined}} = 0.2464875/0.594, \quad p_{\text{out}}^{\text{refined}} = 0.2058/0.594$$

and finally after scaling

$$p_{S_1}^{M}(s_1, s_5) \approx 0.54497969, \quad p_{S_1}^{M}(s_1, s_9) \approx 0.45502031$$

and the control returns to the Model checking method. For this example the DTMC is already reduced after the abstraction of the SCC, and the reduction does not cause any change. Figure 3(d) illustrates the result.

**IV. INTERACTIVE COUNTEREXAMPLE REFRENCMENT**

In this section we give an overview of how our model checking approach can be used for the generation of counterexamples. In general, the size of a counterexample, i.e., the number of finite paths contributing to the violation of a property for a given DTMC, may be very large and therefore the counterexample not useful in praxis. To cope with this problem, in [26] the definition of *smallest* counterexamples was given: The number of paths needed to reach a certain probability is minimal, and amongst counterexamples with this minimal number of paths the probability mass is maximal. Even when using this definition, the number of paths may be too large, such that it is hard to use them for the correction of erroneous systems. We therefore use another criterion based on the structure of the system. The background is the following observation: Many of the paths of a counterexample may be similar with respect to the visited connected subgraphs and differ only in the number of walks through them. As in [13] it therefore seems useful to see a path as a composition of an acyclic path with some loops attached to its states. This way the probability mass of all paths consisting of the acyclic path and an arbitrary number of walks through the loops can be computed easily.

In order to extend our abstraction-based model checking approach to obtain counterexamples, we save all concrete and abstract information we see during the abstraction algorithm. We use the abstract, acyclic structure of DTMCs resulting from our abstraction algorithm to define *abstract counterexamples*. The basic idea is to choose one or more abstract paths which carry enough probability mass to suffice as a counterexample, i.e., to violate a certain property. These abstract paths induce a (possibly infinite) set of concrete paths, which is a counterexample for the given property. The user can choose the level of abstraction of a counterexample, as he or she can stepwise *refine* it by choosing abstract nodes and replacing them by the concrete input nodes of the underlying SCC and its contained abstract and concrete nodes. This enables the user to choose a more "tight" counterexample and furthermore to have a very clear look on the contribution of certain states and labels to the violation of a property.

In detail we have two stages of abstraction for every abstract node $S$ as depicted in Figure 4. Coming from the abstraction, firstly one can see the abstract node $S$ itself and its outgoing probabilities $p_1', \ldots, p_n'$. As these probabilities are scaled, this corresponds to the probability mass of all paths which walk an arbitrary number of times through the SCC. Secondly, one can see the probability of coming back to the input node of the SCC, represented by a self-loop with the previously saved
probability \( p_r \). The outgoing probabilities \( p_1, \ldots, p_n \) are now the unscaled versions which can be obtained by
\[
p_i = (1 - p_r) \cdot p'_i,
\]
where \( 1 \leq i \leq n \). These two abstraction stages are helpful to decide if returning to the input node, i.e., loops in the SCC, are necessary to reach a certain probability threshold.

Example 2: We use the example DTMC on Figure 1 and the result of the path abstraction applied to it as visualized on Figure 3(d) to illustrate this procedure. Consider the property “The probability of reaching \( s_5 \) is at most 0.3”, which can be expressed in PCTL as \( \mathbb{P}_{\leq 0.3} (\Diamond s_5) \) if we introduce state names as atomic propositions. The counterexample refinement process is depicted in Figure 5. The abstract path \( 1 \rightarrow s_5 \) in (1) is an abstract counterexample for this property and induces an infinite number of paths including those which return arbitrarily often to the input state of SCC 1. By considering (2) one can see that no returning to the input state is necessary, as the outgoing probability itself is higher than 0.3. Expanding SCC 1 (3) leads to three abstract nodes, namely 1.1\(_{s_3}\), 1.1\(_{s_2}\) and 1.2. The indices \( s_2 \) and \( s_3 \) indicate the corresponding input state of SCC 1.1. As the path \( s_1 \rightarrow 1.2 \rightarrow s_5 \) has sufficient probability, one can decide that no visit of nodes 1.1\(_{s_2}\) and 1.1\(_{s_3}\) and their underlying SCC 1.1 is needed to exceed the probability bound. This path is depicted in (4). Note that for readability reasons some transitions and nodes of the DTMC are omitted. (5) shows, that even returning to 1.2 is not needed. Expanding 1.2 (6) leads to the abstract counterexample \( s_1 \rightarrow s_6 \rightarrow 1.2 \rightarrow s_5 \) which carries probability mass 0.33. If the user is interested in a concrete counterexample, he firstly sees, that returning to the input state of 1.2.1 is needed (7). Expansion of 1.2.1 leads to the concrete system (Figure 1), and the user can choose paths that form a counterexample using the following information: No paths leading to the input state of SCC 1.1 are needed, all paths have the prefix \( s_1 \rightarrow s_6 \rightarrow s_7 \), at least for one path state \( s_7 \) is to be revisited and for no path returning to states \( s_1 \) and \( s_6 \) is necessary. A counterexample built from these facts is, e.g., \( C = \{ s_1 \rightarrow s_6 \rightarrow s_7 \rightarrow s_5, s_1 \rightarrow s_6 \rightarrow s_7 \rightarrow s_8 \rightarrow s_7 \rightarrow s_5, s_1 \rightarrow s_6 \rightarrow s_7 \rightarrow s_8 \rightarrow s_7 \rightarrow s_7 \rightarrow s_5 \} \) which has probability mass 0.31806.

V. Case Studies

We will demonstrate the practical use of our previously described theoretical concepts and ideas by some common case studies. Therefore we follow the way of [12] and choose the synchronous leader election protocol [18] and the crowds protocol [19] to show the applicability of the model checking approach by SCC-reduction and compare it with other tools for probabilistic model checking. To test the method, we developed a prototype implementation in C++, which is based on the algorithms given in Section III and which stores all information of all abstraction levels to permit the later generation of counterexamples. We chose MRMC [27] for comparison, because both MRMC and our tool work on explicit model representations and the input format is the same. In order to generate benchmarks, we used the PRISM [7] models of the two protocols, which are provided on the PRISM web page [28]. All experiments were done on a 2.4 GHz Intel Core2 Duo CPU with 4 GB RAM.

In the synchronous leader election protocol, \( N \) processes are connected in a one-way ring and they want to elect a unique leader. They therefore randomly choose a natural number, their \( id \), out of the range \( 1, \ldots, K \), which leads to a uniform probability distribution. These numbers are synchronously passed along the whole ring such that every process can see all other \( ids \). The leader is the process with the highest unique \( id \). If there is no unique highest \( id \), a new selection round is started. This goes on until a leader is elected, which will happen with probability 1 at some point in time. Hence, modeled as a DTMC, the model checking result of reaching a state “leader elected” from the initial state has to be 1. In our tests, we considered models containing processes with the parameters \( K \) and \( N \). Some results are depicted in Table I. The running
times MrMC and our tool – called SCC-MC – needed for model checking, is given in seconds.

The SCC based abstraction took no measurable amount of time. That is due to the special structure of the leader election protocol. As one can see in Figure 6 there is only one big SCC, given by the edge from the next-node to the initial node, representing the case that no leader is elected. This SCC is immediately eliminated, all remaining paths lead to the target state and therefore the expected and correct result, a transition with probability 1 leading from the initial state to the target state, is computed very fast.

The crowds protocol was designed to provide anonymous communication between users in networks. The idea is to take a “crowd” of users and route messages randomly through this group. If a group member wants to send a message, it picks a random member from the whole group and sends the message. The receiver then chooses randomly whether to send the message to the destination directly or to act as a “router”, i.e., to forward it to another random member. By this procedure, this group. If a group member wants to send a message, it picks a random member from the whole group and sends the message. The receiver then chooses randomly whether to send the message to the destination directly or to act as a “router”, i.e., to forward it to another random member. By this procedure, all so-called “bad members” can be sure if a message and its sender belong together. In the models we use there are n crowd members among which are g·n “good members”. A normal user delivers a message to its destination with probability (1 − p_f) and therefore forwards it with probability p_f. One delivery of a message (to the destination) is called a session of the protocol and the number of sessions is denoted by r. If a user has been identified twice by a bad member, it is marked as positively identified, and thus anonymous sending is no longer preserved for this user. In the DTMC that represents the protocol w.r.t. all parameters, such states are labeled with Pos.

The property we want the tools to check is the probability of reaching a state labeled with Pos, i.e., Pr[ Pos]. We chose fixed values for most of the variables: n = 5, g = 0.833, p_f = 0.8. The models are therefore only parametrized by r. A part of the results for the crowds protocol are shown in Table II. Additionally to the time amount also the memory consumption is depicted.

The standard method based on the solution of an equation system was slightly slower than our method for nearly all models. We observe that if a large number of target states, i.e., states labeled with Pos is to be considered, our method could compete in matters of time while additionally diagnostic information that leads toward counterexamples is provided. The memory consumption is comparatively low.

The two main fragments of our implementation are the detection of SCCs and the abstraction itself. In Table III one can find the time both fragments took in detail.

VI. CONCLUSION

In this work, we investigated the problem of a PCTL model checking algorithm with simultaneous counterexample generation. We introduced a method that computes the model checking result for a DTMC and an unbounded reachability property by recursive abstraction of SCCs. During this computation, sufficient information is stored to create counterexamples on different levels of abstraction with concrete counterexamples at the bottom, whose size is reasonable due to the structure of the input DTMC. We proved both correctness and completeness of this procedure and described, how to interactively refine an abstract counterexample until a set of concrete paths is obtained which serves as a counterexample. Our approach was shown to be efficient and competitive with the standard model checking algorithms (which do not provide diagnostic information) by applying our prototype implementation to some widely-used benchmarks.

In future we are going to optimize our implementation and in particular to construct a graphical user interface especially for interactive counterexample refinement. Furthermore, we are going to use the stored information to automatically extract reasonable counterexamples. Reasonable can, e.g., mean that...
they involve only a small part of the system. A formal definition of “reasonable counterexamples” has to be found.

The experimental results show that the detection of the SCCs take a relatively large percentage of the running times of our tool. It would be very helpful to develop an adaptation of Tarjan’s algorithm to find not only SCCs but to detect also connected subgraphs in the SCCs.

Another direction of research is the handling of parametric Markov chains, in which some of the transition probabilities are unspecified parameters. In [24] a state elimination approach for model checking such parametric Markov models was given. Using our approach instead, following the way of using rational functions, seems promising.

REFERENCES